Ann Math Artif Intell
DOI 10.1007/s10472-013-9363-9

Group planning with time constraints

Meirav Hadad · Sarit Kraus · Irith Ben-Arroyo Hartman · Avi Rosenfeld

© Springer Science+Business Media Dordrecht 2013

Abstract Embedding planning systems in real-world domains has led to the necessity of *Distributed Continual Planning* (DCP) systems where planning activities are distributed across multiple agents and plan generation may occur concurrently with plan execution. A key challenge in DCP systems is how to coordinate activities for a group of planning agents. This problem is compounded when these agents are situated in a real-world dynamic domain where the agents often encounter differing, incomplete, and possibly inconsistent views of their environment. To date, DCP systems have only focused on cases where agents' behavior is designed to optimize a global plan. In contrast, this paper presents a temporal reasoning mechanism for self-interested planning agents. To do so, we model agents' behavior based on the Belief-Desire-Intention (BDI) theoretical model of cooperation, while modeling dynamic joint

This work was supported in part by ERC grant number 267523, the Google Inter-University Center for Electronic Markets and Auctions and MURI grant number W911NF-08-1-0144. Preliminary results appeared in CIA-01 [30] and in CIA-02 [31].

M. Hadad (✉) · S. Kraus
Department of Computer Science, Bar-Ilan University, Ramat-Gan, 52900, Israel
e-mail: meirav.had@gmail.com

S. Kraus
Institute for Advanced Computer Studies, University of Maryland,
College Park, MD 20742, USA
e-mail: sarit@umiacs.umd.edu

I. Ben-Arroyo Hartman
Caesarea Rothschild Institute for Interdisciplinary Applications of Computer Science,
University of Haifa, Mount Carmel Haifa, 31905, Israel
e-mail: irith@cs.haifa.ac.il

A. Rosenfeld
Department of Industrial Engineering, Jerusalem College of Technology,
Jerusalem 91160, Israel
e-mail: rosenfa@jct.ac.il

Published online: 28 June 2013

plans with group time constraints through creating hierarchical abstraction plans integrated with temporal constraints network. The contribution of this paper is threefold: (i) the BDI model specifies a behavior for self interested agents working in a group, permitting an individual agent to schedule its activities in an autonomous fashion, while taking into consideration temporal constraints of its group members; (ii) abstract plans allow the group to plan a joint action without explicitly describing all possible states in advance, making it possible to reduce the number of states which need to be considered in a BDI-based approach; and (iii) a temporal constraints network enables each agent to reason by itself about the best time for scheduling activities, making it possible to reduce coordination messages among a group. The mechanism ensures temporal consistency of a cooperative plan, enables the interleaving of planning and execution at both individual and group levels. We report on how the mechanism was implemented within a commercial training and simulation application, and present empirical evidence of its effectiveness in real-life scenarios and in reducing communication to coordinate group members' activities.

Keywords Artificial intelligence · Multiagent system · Planning · Cooperation · Coordination · Time constraints

Mathematics Subject Classification (2010) 68T42

1 Introduction

Embedding planning systems in real-world domains is a challenging problem that has wide-ranging importance and applications [11, 18, 19, 21, 44, 70, 82, 86]. Examples of these settings are problems that are inherently distributed, such as training and educational settings, Internet information sharing, interactive entertainment, search and rescue missions and robotic space missions [44, 82]. Additionally, using multiple agents to create a plan often yields increased performance and task reliability even in situations where the task can theoretically be performed by a single agent [19].

A key challenge in these planning systems is the ability to address real-world dynamics, typically done through interleaving planning and execution [21]. This type of planning could potentially address the following types of dynamics: the world changes in ways that are beyond the agent's control; the features of the world are revealed incrementally; temporal constraints force execution to begin before a complete plan can be generated; new goals evolve over time [18]. It is impractical to plan for all possible eventualities in such scenarios, particularly due to the highly dynamic nature of multiple agents systems [70]. Moreover, as planning systems are implemented in real-world applications, they raise the issue of temporal constraints in the environments in which they operate [11, 86].

The first contribution of this paper lies in presenting a temporal reasoning mechanism for multiple self-interested planning agents that must coordinate their actions in order to accomplish a joint action under temporal constraints. The mechanism utilizes a temporal constraint network technique to guarantee the temporal consistency of a cooperative plan. The major novelty of the reasoning mechanism is its integration within a distributed planning system based on a well-grounded BDI theoretical model of cooperation, namely the SharedPlan [26] model. SharedPlan

is generally used to define essential characteristics of teamwork for supporting the design and construction of collaborative systems. This includes allowing agents in a group to plan a joint action, perform the action or carry out activities in order to help facilitate their cooperative plan.

This paper also contains two key contributions about how the BDI model is implemented, which are equally applicable for cases of selfless or self-interested agents. First, we present how abstraction can be used to implement teamwork, allowing for joint action to be defined without explicitly describing all possible states in advance, as is done in former frameworks of teamwork [41, 82], making it possible to reduce the number of states which need to be considered.[1] Second, as each agent may reason by itself about the best time to perform its activities, the mechanism reduces coordination messages among the group members. Keeping the search space as small as possible is critical for implementing a working application, especially one capable of running in real-time even as it handles dynamics. Keeping the number of messages small is important for environments where communication is costly, noisy or otherwise problematic.

Applying our mechanism in a real-world application raises new questions regarding the order of plan generation and the commitment of the group to partial plans. We discuss these questions in the next sections and explore them by implementing different methods in a synthetic rescue environment. The results demonstrate that for environments which we have tested, it is better to commit as late as possible. Furthermore, when group members use a similar order to plan parts of their joint action, they have a better chance of succeeding in finding a cooperative plan that satisfies all of the temporal constraints. We have implemented the mechanism as a part of the SharedPlan system and integrated it within a military commercial training and simulation application. The mechanism enables the successful simulation of real-life scenarios where a group of agents has to jointly achieve military missions under temporal constraints. We present empirical evidence for the effectiveness of the mechanism in reducing communication to coordinate the group members' activities.

This work is strongly based on the SharedPlan framework [26]. In Section 2, we briefly describe this framework. In Section 3 we provide formalization of temporal constraint networks in the context of the SharedPlan. Then, we present the temporal reasoning algorithm including methods for exchanging and merging temporal information among the group members. The algorithm ensures temporal consistency of the plans, determines times for executing actions, and coordinates the agents' activities in such a way that all of the temporal constraints are satisfied. In Section 4 we prove the correctness of the temporal reasoning algorithm and discuss its complexity. In Section 5 we discuss methods for exchanging and merging temporal information among the agents, and we study the behavior of these methods in a synthetic rescue environment. An additional problem that we explore relates to the order in which the group members should plan the subsidiary actions of their joint activity. In this section we also discuss how the mechanism was successfully implemented within a commercial training and simulation system for a military domain. In Section 6, we survey related research fields on planning systems with temporal reasoning and scheduling and compare them with our work. Finally, in Section 7, we conclude and present possible directions of future research.

[1] The empirical evidence is beyond the scope of this paper and can be found at [32].

2 Overview of the SharedPlan model

In this section we overview basic concepts and planning processes of the theoretical SharedPlan model [26]. We then describe the major component of the *SharedPlan system* which we developed in [29] that we use in this paper. To motivate the discussion we start with an informal example of "*rescue disaster survivors*" by two rescue robots. We refer to this example throughout the paper.

2.1 A motivating example

Assume two rescue robots A_1 and A_2 must jointly perform the action "*rescue disaster survivors*" denoted by α. Suppose that A_1 and A_2 have different capabilities. Robot A_1 is small and flexible and is able to maneuver deep into crevices in the rubble, flatten itself to crawl through tight spaces, or rear up to climb and look over objects. The second robot, A_2, is large and strong. It is designed to pick up concrete slabs, pipes, broken boards, etc., in the collapsed area. Thus, A_1 and A_2 must collaborate on some activities in order to succeed in performing action α. Robots A_1 and A_2 may have various temporal constraints. For instance: A_1 and A_2 arrive at the collapsed area at 4:00 P.M.; the batteries of A_2 are restricted for 150 min.; A_1 and A_2 must finish the joint action by 7:30 P.M. In this paper we wish to address queries such as: "When should A_1 inform A_2 that it has completed its activities?"; "Which temporal information do A_1 and A_2 exchange?"; "Which robot checks the temporal consistency of a joint action?"; "Can the robots complete their activities on time, or do they perhaps need some help?", and so on.

Consider in the previous example that A_1 and A_2 have agreed to the following plan in order to perform action α, "*rescue disaster survivors*": Suppose that action α is divided into subactions $\beta_1, \beta_2, \ldots, \beta_5$ which are respectively defined as follows: "*find and identify the victims in area A*", "*find and identify the victims in area B*", "*clear obstructions blocking doorways in area A*", "*rescue the victims outside the building in area A*" and "*rescue the victims outside the building in area B*". Also assume that they divide the responsibility among themselves as follows: A_1 performs subactions $\beta_1, \beta_2, \beta_4$ and β_5, and A_2 performs subactions β_3, β_4 and β_5. Note that some subactions are performed by a single agent and others (i.e., β_4 and β_5) are multi-agent actions that require cooperation between A_1 and A_2.

These subactions may include several "*precedent constraints*". For example, assume that A_2 must know the location of the "victims" before it begins to clear blocked doorways in the area (i.e., β_1 must be carried out before β_3). Also, since area B is less dangerous than area A, area B will be scanned by A_1 only after area A is scanned (i.e., β_1 will be performed before β_2). In addition, they will "*rescue the victims outside the building in area A*" only after A_2 clears the blocked doorways in area A (i.e., β_3 will be performed before β_5), and after A_1 "*finds and identifies the victims in area B*" (i.e., β_2 will be performed before β_4). They will "*rescue the victims outside the building in area B*" after A_1 "*finds and identifies the victims in area B*" (β_2 is performed before β_5).

The directed graph of Fig. 1 illustrates the precedence relations between subactions $\beta_1, \beta_2, \ldots, \beta_5$ above. We denote the start and finish time of action β_i by the variables s_{β_i} and f_{β_i}, respectively, and they are represented as vertices in the graph. Note that agent A_2, for example, cannot perform β_3 without knowing the finish time of β_1 (because β_1 precedes β_3).

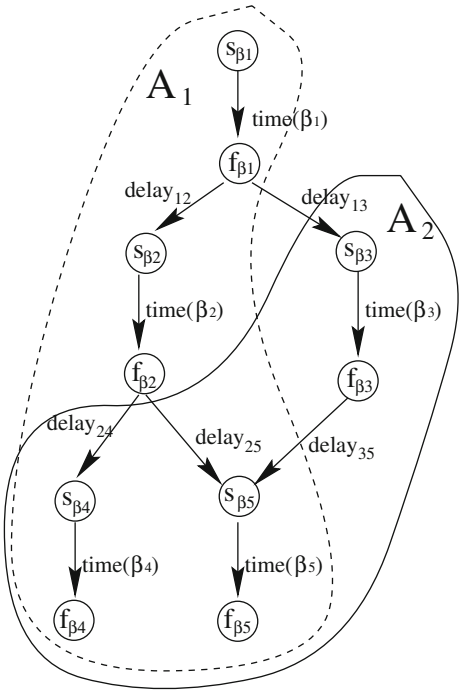

Fig. 1 An example of a precedence graph for multi-agent action α; $time(\beta_i)$ within which β_i must begin and finish; s_{β_j} and f_{β_j} are variables representing start and finish time points of β_j, respectively; $delay_{ij}$ is the interval that denotes a possible delay between the finish time of action β_i and the start time of action β_j

This example raises the following key questions: At what stage should an agent commit to a time for performing an action, and inform the rest of the group members of its commitment? If the individual agent commits to a specific time early on and announces this commitment to the other agents, it may need to negotiate with the other agents if it needs to change its schedule later. Alternatively, if the commitment is made and announced as late as possible, e.g., only when requested by other group members, it may delay the other group members' planning. Another question refers to the order in which each individual in the group plans its subactions in the joint plan and identifies the values of the time variables. For example, suppose that there is no precedent relation between β_1 and β_2. Then, A_1 can either plan β_1 before β_2 or vice versa. We explore these questions empirically in Section 5.1.

2.2 Basic definitions of a collaborative plan

In this section we briefly describe basic definitions of the SharedPlan model that are the formal basis of the temporal reasoning mechanism. The definitions are based on the original formalization but augmented to include time constraints in an explicit way. The planning approach contains many similarities to the previous Hierarchical Task Network (HTN) [23, 25, 37, 65] but includes extensions for joint action planning in a multi-agent environment.

An *action*, in the model, is an abstract entity which has various properties associated with it such as action type, agent, time of performance, and other objects

involved in performing the action. An action may be either a *basic action* or a *complex action*. A *basic action* is an action that can be directly executed by the agent and cannot be subdivided into subactions (e.g., crawl, look over objects, pick-up). A (higher-level) *complex action* is one that cannot be executed directly and is decomposed into subactions (e.g., find and identify the victims). In addition, an action may be either a *single-agent action* or a *multi-agent action*. A single-agent action can be executed by a single agent and a multi-agent action requires two or more cooperative agents to complete the action jointly. To execute a high-level complex action the agents must identify a *recipe* for it. They may know several recipes for the same action. The model assumes that each agent has a library of recipes. A *recipe* for action α, denoted by R_α, refers to a set of actions, which we denote by β_i ($1 \leq i \leq n$), and appropriate *constraints*, denoted by ρ_j ($1 \leq j \leq m$), specifying how the actions can be performed and which agents can perform which action. A constraint ρ_k, contains variables X_i defined in a certain domain Dom_i. Examples of recipe constraints are *agents constraints*, *precedence constraints* and *metric constraints*. The agents constraints specify the capabilities that are required of the agents to perform specific actions in the recipe. For example, the number of agents required to perform an action in a recipe may be between 2 and 5—or formally, $2 \leq X_{\text{agentNun}} \leq 5$. Alternatively, these constraints may specify the type of agent that can perform a certain action, e.g., the agent must be a small robot. Precedence constraints refer to the execution order of the actions. Metric constraints indicate specific times for executing actions in the recipe.

In order to perform some complex action β_i, the agents have to identify a recipe R_{β_i} for it. There may be several recipes for β_i. The recipe R_{β_i} may include subactions δ_{iv}. Each δ_{iv} may similarly be either basic or complex. An illustration of decomposition of α when the plan is fully initiated is depicted in a *complete recipe tree* for α of agent A_k. Formally, *recipe tree* for α of agent A_k is represented by an acyclic digraph $T_\alpha^k = (V_\alpha^k, E_\alpha^k)$ in which V_α^k is the nodes set, E_α^k is the edge set, and each node $v \in V_\alpha^k$ contains an action. We assert that $T_\alpha = (V_\alpha^1 \cup \ldots \cup V_\alpha^n, E_\alpha^1 \cup \ldots \cup E_\alpha^n)$ is the *union recipe tree* of a group of agents $\mathcal{A}_\alpha = \{A_1, \ldots, A_n\}$ that jointly execute a multi-agent action α.

Figure 2 demonstrates an example of possible recipe trees for the action "*rescue disaster survivors*" in a specific world-state. The bold edges represent the actions performed by both robots and the dashed edges are the actions performed by a single agent (A_1, or A_2). The trees differ with respect to single-agent actions but are identical with respect to the first level of each multi-agent action. For example, action α is a multi-agent action and must be performed by A_1 and A_2; thus, the trees of both robots consist of the first level of α (i.e., $\beta_1, \ldots, \beta_5$). On the other hand, β_1 is a single-agent action which has to be performed by A_1; thus, A_2 does not know about actions γ_{11} and γ_{12} which are selected by A_1 in order to perform action β_1. Similarly, β_3 has to be performed by A_2 and thus A_1 does not know about A_2's plan to perform β_3.

The SharedPlan formalism proposes several collaborative planning processes to identify a plan for a joint action α. The formalism distinguishes between five different types of plans. A *full individual plan* specifies those conditions under which an individual agent can be said to have a fully initiated plan to perform a single-agent action α. A *partial individual plan* deals with both partiality of knowledge and partiality of intention. A *SharedPlan* representing that a group of agents has a collaborative

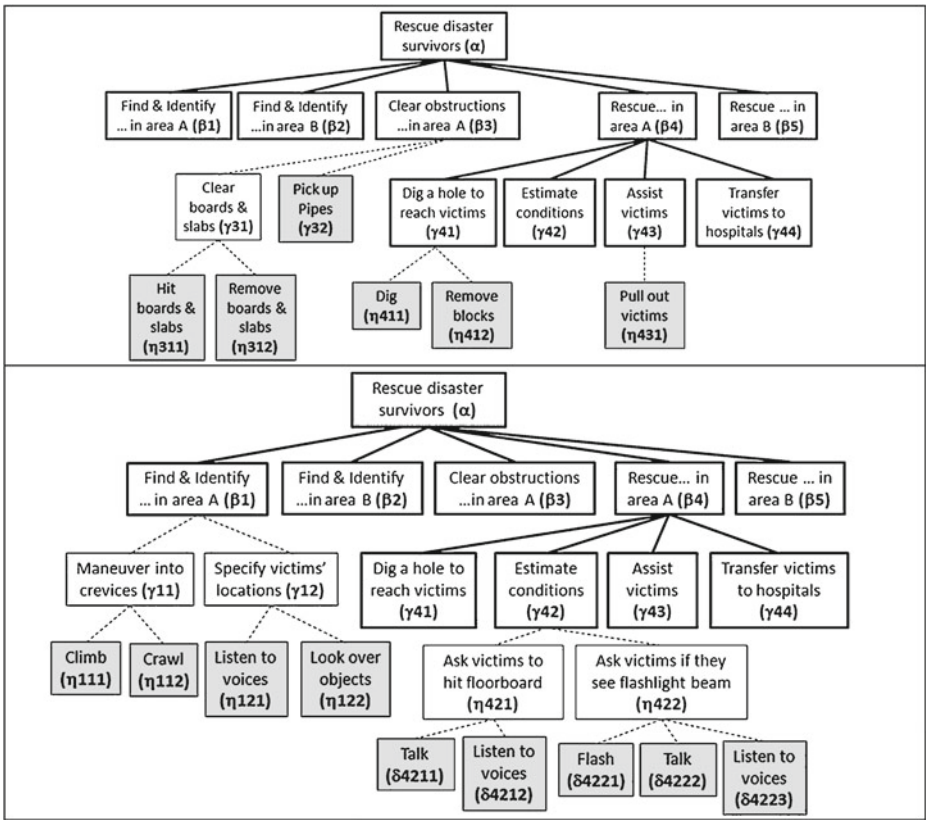

Fig. 2 Part of possible recipe trees for the action "*rescue disaster survivors*", the top tree is planned by the large robot and the bottom is planned by the small robot. The *dashed edges* represent individual plans. The *gray boxes* represent basic actions.

plan to perform together some action α is defined recursively in terms of a *full SharedPlan* and a *partial SharedPlan*. A *full SharedPlan* is the collaborative correlate of a full individual plan and includes full individual plans among its constituents. A *partial SharedPlan* is the collaborative correlate of a partial individual plan. A principal way in which the SharedPlan differs from the individual plan is that knowledge about how to act, ability to act and commitment to act are distributed in the SharedPlan.

The SharedPlan model presents several planning processes for the individual and group to expand partial plans into more complete ones. Even though all of planning processes are implemented in the system, in order to simplify the presentation of the temporal reasoning mechanism we omit the details. More discussion of the SharedPlan definitions and of the planning processes is given in [26, 27].

2.3 An example of a collaborative plan with time constraints

A collaborative plan of a multi-agent action can be illustrated by the rescue robots example introduced above. In this example, a multi-agent action α, "*rescue disaster*

survivors", that needs to be executed by rescue robots A_1 and A_2 jointly is a complex action. Since this action is a multi-agent action, A_1 and A_2 need to agree on how they are going to perform it. This is accomplished by the process which is responsible for identifying a recipe for this action. Suppose that the robots have agreed upon the plan in Section 2.1. In this case, the identified recipe R_α consists of subactions $\{\beta_1, \ldots, \beta_5\}$. Figure 3 presents the structure of recipe R_α as implemented in the SharedPlan system. As shown in this figure, the recipe consists of: action-type, name, apply-condition, subactions, agents constraints and temporal constraints. Each subaction β_i, $(i = 1, \ldots, 5)$, is associated with variables of agents, denoted by $?A_k$ $(k = 1, 2)$, and with variables of a temporal interval, denoted by $[?s_{\beta_1}, ?f_{\beta_1}]$, $[?s_{\beta_2}, ?f_{\beta_2}]$, etc. The variables s_{β_i} and f_{β_i} represent the start time and the finish time of the corresponding subaction β_i, respectively. The recipe includes two types of temporal constraints, *precedence* constraints and *metric* constraints. In our example, the identified recipe is associated with precedence constraints $\{\beta_1$ before β_2; β_1 before β_3; β_2 before β_4; β_2 before β_5; β_3 before $\beta_5\}$. We assume that R_α is associated with the following metric constraints: A_1 and A_2 must start β_4 within 40 min. after A_1 has started β_1; A_1 and A_2 must start β_5 within 60 min. after A_2 has started β_3; A_2 must start β_3 after 5:00 P.M. The *apply-condition* refers to a set of propositions such as: a building has collapsed in area B (i.e., "(BUILDING24, B, collapsed)") and a victim is buried under the destruction in area B (i.e., "(VICTIM1, B, under-destruction)"). The necessary capabilities for performing a complex level action are given in the *agent-constraints*.

In this phase of their planning process, the agents have a *partial SharedPlan* for the shared action α, "*rescue disaster survivors*". Also, A_1 has a *partial individual plan* for subaction β_1, "*find and identify the victims in area A*". While in partial plans the values of the variables may not be identified, for the agents to achieve a *full plan* the values of the variables must be set such that the appropriate constraints are satisfied. The problem with reasoning about the temporal values is a result of partial knowledge and the uncertain environment of the agents. When a high-level action is broken up into sequences of subactions and finally into basic actions, the

(make-recipe **:action** α

:action-type	rescue_disaster_survivors
:name	R^1_α
:apply-condition	( )
:subactions	β_1: (find_and_identify_victims_A $?A_1$ $[?s_{\beta_1}, ?f_{\beta_1}] \cdots$)
	β_2: (find_and_identify_victims_B $?A_1$ $[?s_{\beta_2}, ?f_{\beta_2}] \cdots$)
	β_3: (clear_obstructions_A $?A_2$ $[?s_{\beta_3}, ?f_{\beta_3}] \cdots$)
	β_4: (rescue_victims_A $(?A_1, ?A_2)$ $[?s_{\beta_4}, ?f_{\beta_4}] \cdots$)
	β_5: (rescue_victims_B $(?A_1, ?A_2)$ $[?s_{\beta_5}, ?f_{\beta_5}] \cdots$)
:agents-constraints	( )
:precedence-constraints	$?f_{\beta_1}$ before $?s_{\beta_2}$ $?f_{\beta_1}$ before $?s_{\beta_3}$
	$?f_{\beta_2}$ before $?s_{\beta_4}$ $?f_{\beta_2}$ before $?s_{\beta_5}$
	$?f_{\beta_3}$ before $?s_{\beta_5}$
:metric-constraints	$(?s_{\beta_4} - ?s_{\beta_1} \leq 40\ minutes)$
	$(?s_{\beta_5} - ?s_{\beta_3} \leq 60\ minutes)$
	$(?s_{\beta_3}$ after $17:00)$

...

Fig. 3 An example of a possible recipe for the complex action "*rescue disaster survivors*" (α)

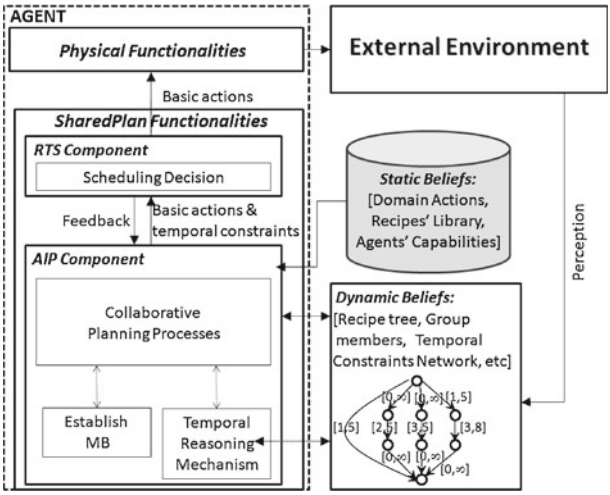

Fig. 4 The architecture of a single-agent. The AIP, RTS and the beliefs components are part of the SharedPlan system. The physical functionalities are responsible for the execution of basic actions

time available to achieve the high-level action must also be split into intervals for each subaction. Doing this correctly requires the modified SharedPlan system to have a mechanism of how long it takes to accomplish the subactions. However, as a result of the dynamic nature of plans, any of the components of the agent's plan may be incomplete and thus it may be impossible to know how long it will take to solve the subactions. Furthermore, in some cases the agent must interleave planning and execution. In the next section we suggest a mechanism for the agent to reason autonomously and in a dynamic fashion to identify the temporal values of the appropriate variables.

2.4 The SharedPlan system

The SharedPlan system implements a group of agents that interact with each other and with an external environment where the communication channels of the system are reliable and synchronized. Figure 4 illustrates the high-level architecture of an agent in the SharedPlan system.[2] Each agent is comprised of two separate components that interact with each other: (i) Physical functionalities and (ii) SharedPlan functionalities. The physical functionalities refer to a predefined set of basic actions that are designed within the core of a single agent and which can be directly executed in the environment. Examples include: maneuver to a specific destination; crawl; climb; look over objects; pick up and so on. We did not implement these functionalities but used existing implementation of the applications we explored (e.g., the military training and simulation system). The SharedPlan functionalities include two major components. One of them is an Artificial Intelligence Planning (AIP) component. The second is a Real-Time Scheduling (RTS) component. The temporal reasoning mechanism presented in this paper is implemented as part of

[2]The full specification of the system's architecture and details of the algorithms can be found at the project site: http://homedir.jct.ac.il/~rosenfa/research/amai.htm.

the AIP component. The AIP component includes collaborative planning processes which we previously discussed.

In addition to the above functionalities, each agent may access the data of beliefs. The beliefs may be either static or dynamic. Static beliefs are predefined and include a library of recipes, description of domain actions, knowledge about the agent capabilities and so on. The recipe tree is a part of the dynamic beliefs as the agent modifies it during the execution. The beliefs about the states of the other group members are also dynamic. The external environment (e.g., the simulator) is designed to test the validity of a predefined set of propositions in the agent arena (e.g, 'is there an obstacle on the left side?') and to answer a predefined set of queries (e.g., 'what is the level of my energy?'). The answers are also referred to as dynamic beliefs.

The AIP component plans the agent's activities while interacting with the environment (including other agents). It identifies, incrementally, a set of basic actions, and a set of temporal requirements associated with the basic actions, to perform α without conflict. During the planning process each basic action β is sent to the RTS component, along with its temporal requirements $\langle D_\beta, d_\beta, r_\beta, p_\beta \rangle$ where D_β is the *Duration time*, i.e., the time necessary for the agent to execute β without interruption; d_β denotes the *deadline*, i.e., the time by which β must be completed; r_β refers to the *release time*, i.e., the time at which β is ready for execution; p_β denotes the *predecessor actions*, i.e., the set $\{\beta_j | 1 \leq j \leq n\}$ of basic actions whose execution must terminate before beginning β. The RTS component receives the basic actions set with the associated requirements and inserts these actions into the agent's schedule as described in the following section. The RTS component is responsible for the scheduling and dispatching of basic actions for execution. The scheduling problem that our RTS component faces is NP-hard [24]. We describe the heuristic algorithm which is used in the RTS component in [33].

3 Mechanism for group temporal reasoning

We consider a problem where a group of agents $\mathcal{A}_\alpha = \{A_1, \ldots, A_n\}$ must jointly execute a multi-agent action α. We assume that α is a complex action. The agents are acquainted with a set of actions (either basic or complex), library of recipes and an initial set of beliefs. We present a mechanism for $\mathcal{A}_\alpha$ to identify a full shared plan in order to execute α without violating temporal constraints. Each agent in $\mathcal{A}_\alpha$ should perform its own part in the plan by identifying a set of basic actions along with their temporal requirements $\langle D_\beta, d_\beta, r_\beta, p_\beta \rangle$ and dispatching them for execution under these requirements. We begin the section with basic definitions. A summary of the notations used in the temporal reasoning mechanism is given in Table 1.

3.1 Supporting definitions and notations

As described in previous sections, a contribution of this paper is how we model the agents' behavior based on the theoretical SharedPlan model of cooperation yet extend this framework to allow for self-interested agents to plan joint activities with temporal constraints. A second novel contribution of this papers lies within how we implemented this extended framework. Specifically, we model dynamics by using

Table 1 Summary of notations used for special variables and constants

Notation	Meaning	Comments
α	Action	Also: β, β_i
R_α	Recipe for α	
s_α	Start time of α	
f_α	Finish time of α	
A_k	A single agent	Also: A_i
$\mathcal{A}_\alpha$	The group of agents that performs α	
$G^p_\alpha = (V^p_\alpha, E^p_\alpha)$	Precedence graph of α	Definition 3.1
$G^k_\alpha = (V^k_\alpha, E^k_\alpha)$	A_k's Temporal Constraints Graph of action α	Definition 3.3
$H(\beta)$	Set of actions which hinder β's performance	Definition 3.4
D_β	Duration time of β	Section 2.4
r_β	Release time of β	Section 2.4
d_β	Deadline of β	Section 2.4
p_β	Predecessor actions of β	Section 2.4
ENABLED	Enabled vertex	Section 3.3.1
EXPLORED	Enabled and explored vertex	Section 3.3.1
UNEXPLORED	Unexplored vertex	Section 3.3.1
ES	Explored single-agent vertex	Section 3.3.1
EM	Explored multi-agent vertex	Section 3.3.1
ENPT	Explored non-participant terminated vertex	Section 3.3.1
ENPW	Explored non-participant wait vertex	Section 3.3.1
$\mathcal{E}$	Set of enabled explored vertices	Section 3.3.1
$\mathcal{U}$	Set of enabled unexplored vertices	Section 3.3.1
$\mathcal{W}$	Set of explored non-participant wait vertices	Section 3.3.1

hierarchical abstraction plans and a temporal constraints network. To do so, we define a new structure, called a *Temporal Constraints Graph*. The main structure which is used in building the Temporal Constraints Graph is the *Precedence Graph*. The precedence graph represents a recipe in the form of a constraints network where the vertices of the graph represent the subactions in the recipe and the edges represent the order relationship between the subactions.

Definition 3.1 (Precedence graph of α, G^p_α) Let α be a complex action, and let R_α be a recipe for α. Let $\beta_1, \ldots, \beta_n$ be the subactions of R_α with precedence constraints defined by the relation $\theta_\alpha = \{(\beta_i, \beta_j) | \ \beta_i \text{ before } \beta_j; i \neq j\}$. The Precedence Graph of α, $G^p_\alpha = (V^p_\alpha, E^p_\alpha)$ with reference to R_α and its precedence constraints is a directed graph, defined as follows: The vertex set is $V^p_\alpha = \{s_{\beta_1}, \ldots, s_{\beta_n}, f_{\beta_1}, \ldots, f_{\beta_n}\}$ where s_{β_i} and f_{β_i} represent variables of the start and finish time of β_i, $1 \leq i \leq n$, respectively and are associated with the action β_i. The edge set, E^p_α, consists of two types of edges:

1. For each β_i, $1 \leq i \leq n$, there is an edge $(s_{\beta_i}, f_{\beta_i})$ representing the time required to perform β_i.
2. For each pair $(\beta_i, \beta_j) \in \theta_\alpha$, there is an edge $(f_{\beta_i}, s_{\beta_j})$ denoting that subaction β_i must terminate before β_j starts. The edge represents the delay between β_i and β_j.

The vertices $s_{\beta_i} \in V^p_\alpha$ with an in-degree zero are called *initial vertices*. The vertices $f_{\beta_i} \in V^p_\alpha$ with an out-degree zero are called *terminal vertices*.

Example 3.1 The gray vertices and edges of the graph in Fig. 5 illustrate the Precedence Graph, G^p_α, of a possible recipe R_α where $V^p_\alpha = \{s_{\beta_1}, \ldots, s_{\beta_5}, f_{\beta_1}, \ldots, f_{\beta_5}\}$ and $E^p_\alpha = \{ (s_{\beta_1}, f_{\beta_1}), (s_{\beta_2}, f_{\beta_2}), (s_{\beta_3}, f_{\beta_3}), (s_{\beta_4}, f_{\beta_4}), (s_{\beta_5}, f_{\beta_5}), (f_{\beta_1}, s_{\beta_2}), (f_{\beta_1}, s_{\beta_3}), (f_{\beta_2}, s_{\beta_4}), (f_{\beta_2}, s_{\beta_5}), (f_{\beta_3}, s_{\beta_5}) \}$. The vertex s_{β_1} is an initial vertex, and vertices f_{β_4}, f_{β_5} are terminal vertices.

Using a recipe to create the temporal constraints graph forms an abstract hierarchical structure of a partial plan where several vertices are associated with different agents. The precedence relationship between subactions which are not associated with the same group of agents is called a *Multi-Precedence Constraint*.

Definition 3.2 (Multi-precedence constraint) A precedence relation "β_i before β_j" is called a *Multi-Precedence Constraint* if subactions β_i and β_j are performed by different groups of agents, i.e. $\mathcal{A}_{\beta_i} \neq \mathcal{A}_{\beta_j}$.

Example 3.2 As shown in the Precedence Graph G^p_α in Fig. 5, actions β_1 and β_2 are single-agent actions which have to be performed by A_1. Similarly, action β_3 is a single-agent action which has to be performed by A_2. Actions β_4 and β_5 are multi-agent actions which have to be performed by A_1 and A_2 jointly, that is, $\mathcal{A}_{\beta_4} = \{A_1, A_2\}$ and also $\mathcal{A}_{\beta_5} = \{A_1, A_2\}$. The precedence relations: (β_1, β_3), (β_2, β_4), (β_2, β_5) and (β_3, β_5) are multi-precedence constraints. Agent A_2, for example, cannot decide on the start time of β_3 without knowing the temporal requirements of its preceding action β_1, which is planned by A_1.

When group $\mathcal{A}_\alpha$ works on the action α, each agent $A_k \in \mathcal{A}_\alpha$ maintains a *Temporal Constraints Graph*. The definition of the *Temporal Constraints Graph* is a combination of a temporal constraints network with hierarchical abstraction of a joint plan. Hence, it utilizes the SharedPlan model to provide collaboration, Hierarchical Task Network (HTN) to enable the construction of abstract plans and a network of binary constraints to enable applying existing techniques in order to resolve temporal constraints. While we assume that the reader is broadly familiar with

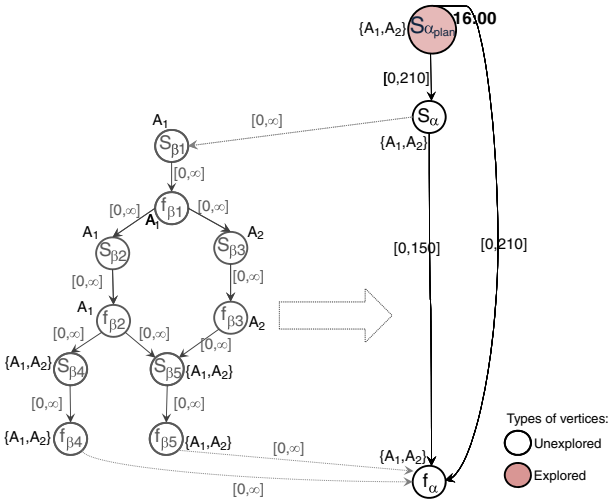

Fig. 5 An example of a Temporal Constraints Graph G^k_α, constructed from a Precedence Graph G^p_α, which is maintained by an agent A_k

Temporal Constraint Satisfaction Problem (TCSP) [16, inter alia], a description of our application of this problem is as follows:

Definition 3.3 (Temporal constraints graph of α, G^k_α) A *Temporal Constraints Graph* $G^k_\alpha = (V^k_\alpha, E^k_\alpha)$ of an agent A_k is a weighted graph constructed from a Precedence Graph $G^p_\alpha = (V^p_\alpha, E^p_\alpha)$ where $V^k_\alpha = V^p_\alpha \cup \{s_{\alpha_{\text{plan}}}, s_\alpha, f_\alpha\}$ and $s_{\alpha_{\text{plan}}}$ represents the time point at which A_k starts to plan action α. E^k_α consists of the edges $E^p_\alpha \cup \{(s_{\alpha_{\text{plan}}}, s_\alpha), (s_\alpha, f_\alpha), (s_{\alpha_{\text{plan}}}, f_\alpha)\}$ with additional edges from s_α to each initial vertex in G^p_α and from each terminal vertex of G^p_α to f_α. For any complex subaction β_i which is associated with $G^p_\alpha = (V^p_\alpha, E^p_\alpha)$ and in which A_k participates, the *Temporal Constraints Graph* is updated recursively (i.e., V^k_α grows to be $V^k_\alpha \cup V^p_{\beta_i}$ and E^k_α grows to be $E^k_\alpha \cup E^p_{\beta_i}$ with additional edges from s_{β_i} to each initial vertex in $G^p_{\beta_i}$ and from each terminal vertex of $G^p_{\beta_i}$ to f_{β_i}).

Each edge in G^k_α is labeled by an interval $[a, b]$ which denotes an upper and lower bound on a time gap. If $e = (s_{\beta_i}, f_{\beta_i})$ then $[a, b]$ denotes the time gap for the duration required to complete subaction β_i, and if $e = (f_{\beta_i}, s_{\beta_j})$ then $[a, b]$ denotes a possible delay between the end of subaction β_i and the beginning of subaction β_j. Initially, all of the edges are labeled $[0, \infty]$.

Each action β which is associated with any vertex in V^k_α contains the following information: (a) whether β is basic or complex; (b) whether β is a multi-agent or a single-agent action; (c) whether a plan for β has been completed; (d) whether β has already been executed by the agent(s); and (e) the agent(s) that is (are) assigned to perform β.

An example of a *Temporal Constraints Graph* G^k_α, constructed from the Precedence Graph G^p_α in Fig. 1, is given in Fig. 5. A more complex example, where $G^1_\alpha \neq G^2_\alpha$, can be seen in the next sections in Figs. 7 and 8. Each single agent A_k in the system runs the algorithm independently in order to construct its Temporal Constraints Graph G^k_α. The information maintained by the graph is determined incrementally by the algorithm which also expands the graph.

In Section 2.2 we defined the *recipe trees* for action α (see an illustration in Fig. 2). This definition was formed by Definition 3.1 of *Precedence Graph* and Definition 3.3 of *Temporal Constraints Graph*. We note that given the Temporal Constraints Graph the recipe tree is *implicit* in the graph and can be easily derived. Thus, similar to recipe trees, the graph of each individual agent in $\mathcal{A}_\alpha$ may be different with respect to its individual actions, but similar with respect to the first level of multi-agent actions.

As mentioned above, in several cases $\mathcal{A}_\beta$'s members are not able to complete β's plan without receiving relevant information about the actions preceding β from the agents performing them. The set of these preceding actions hinders the performance of β; this set is denoted $H(\beta)$. More formally:

Definition 3.4 Hinder members of an action β_j, $H(\beta_j)$, are defined as the set of all minimal members[3] of the set $\{\beta_i | \beta_i$ precedes β_j and $\mathcal{A}_{\beta_j} \nsubseteq \mathcal{A}_{\beta_i}\}$.

[3]We recall that v is a minimal member of S in a partial order if $v \in S$ and no other $w \in S$ exists such that w precedes v. Note that a minimal member is not unique.

3.2 General description of the temporal reasoning algorithm

The Temporal Reasoning Algorithm is used by each $A_k \in \mathcal{A}_\alpha$. The pseudocode is given in Fig. 6. In the initialization phase A_k constructs its initial G^k_α. Then, in the planning and executing loop, A_k expands its G^k_α recursively according to

TEMPORAL_ REASONING_ ALGORITHM(α, $\mathcal{A}_\alpha$)

Initialization phase:

1 $V^k_\alpha \leftarrow \{s_{\alpha_{plan}}, s_\alpha, f_\alpha\}$, $E^k_\alpha \leftarrow \{(s_{\alpha_{plan}}, s_\alpha), (s_\alpha, f_\alpha), (s_{\alpha_{plan}}, f_\alpha)\}$;
2 $\mathcal{E} \leftarrow \emptyset$; $\mathcal{U} \leftarrow$ (the fixed vertices in G^k_α); $\mathcal{W} \leftarrow \emptyset$;
3 **for** each vertex $u \in V^k_\alpha$ **do** status[u] $\leftarrow$ UNEXPLORED;
4 status[$s_{\alpha_{plan}}$] $\leftarrow$ EM;
5 bIsFinishedExecuteAll $\leftarrow$ FALSE; bIsCompletedPlan $\leftarrow$ FALSE; bIsFailureInPlan $\leftarrow$ FALSE;
6 UPDATE_ENABLED_SET($s_{\alpha_{plan}}$);

Planning and executing loop:

7 **while** (not bIsFinishedExecuteAll)

Part I - Planning for a chosen action β from G^k_α:

8 **if** (not bIsCompletedPlan) **then**:
9 select some UNEXPLORED vertex s_β from the $\mathcal{U}$ set;
10 **if** $A_k \notin \mathcal{A}_\beta$ **then**:
11 {
12 **if** the temporal values of β are unknown **then**:
13 status[s_β] $\leftarrow$ ENPW; status[f_β] $\leftarrow$ ENPW;
14 $\mathcal{W} \leftarrow \mathcal{W} \cup \{s_\beta, f_\beta\}$;
15 **else**, the temporal values of β are known **then**:
16 status[s_β] $\leftarrow$ ENPT; status[f_β] $\leftarrow$ ENPT; UPDATE_ENABLED_SET(s_β);
17 }
18 **else**, $A_k \in \mathcal{A}_\beta$ **then**:
19 {
20 **if** β is a single-agent action ($\mathcal{A}_\beta == \{A_k\}$) **then**:
21 {
22 **if** β is a basic action **then**: IDENTIFY_TIMES_OF_BASIC_ACT(β);
23 **else**, if β is a complex action **then:** bIsFailureInPlan $\leftarrow$ EXPAND_TEMOPORAL_GRAPH(β,ES);
24 **if** (not bIsFailureInPlan) **then:** CHECK_NECESSITY_TO_UPDATE_MEMBERS(β);
25 }
26 **else**, β is a multi-agent action ($|\mathcal{A}_\beta| > 1$) **then**:
27 bIsFailureInPlan $\leftarrow$ SELECT_AGREEABLE_RECIPE(β,$\mathcal{A}_\beta$);
28 }
29 **if** (each $v^k_i \in V^k_\alpha$ is EXPLORED) $\wedge$ ($\mathcal{W}$ ==$\emptyset$) **then**:
30 bIsCompletedPlan $\leftarrow$ TRUE;
31 **end if** (not bIsCompletedPlan);

Part II - Obtaining and sending information on activities from and to group's members:

32 sends information to other group members if needed;
33 messages $\leftarrow$ LISTEN_TO_THE_GROUP_MEMBERS;
34 **if** the messages do not include a "failure" message **then**: HANDLE_TEMPORAL_MESSAGES(message);
35 **else if** a "failure" message **then** bIsFailureInPlan $\leftarrow$ TRUE;

Part III - Obtaining and sending messages from and to the RTS on execution:

36 RTSmessages $\leftarrow$ LISTEN_TO_THE_REAL-TIME_COMPONENT;
37 **for** each "succeed to execute an action γ" in RTSmessages **do**
38 status[s_γ] $\leftarrow$ EXECUTED; status[f_γ] $\leftarrow$EXECUTED;
39 **if** (bIsCompletedPlan)$\wedge$(for each $v^k_i \in V^k_\alpha$ associated with a basic action, status[v^k_i]== EXECUTED) **then**:
40 finish_execute[A_k] $\leftarrow$ TRUE;
41 **if** for each $A_i \in \mathcal{A}_\alpha$ finish_execute[A_i] == TRUE;
42 **then** bIsFinishedExecuteAll $\leftarrow$ TRUE;

Part IV - Backtracking, if needed:

43 **if** (bIsFailureInPlan) or (RTS component sent "fail") **then**:
44 if possible, **BACKTRACK**; otherwise **FAIL**;
45 **end while**; (end planning and executing loop)

Fig. 6 The Temporal Reasoning Algorithm for identifying values for temporal variables which is run by each agent $A_k \in \mathcal{A}_\alpha$ during the performance of α

Definition 3.3. As presented in Fig. 6, the planning and executing loop includes four major parts.

In the first part (part I in Fig. 6), A_k chooses a vertex s_β from G^k_α such that the action β is not previously selected by any member in $\mathcal{A}_\alpha$ and all the actions which precede β have been defined. Following the SharedPlan model, A_k distinguishes between single-agent action and multi-agent action. In the case where β is a single-agent action, β may be either a basic action that A_k needs to execute or a complex action whose execution A_k needs to plan. Thus, if β is a basic action, then action β is sent to A_k's RTS component for execution along with β's temporal requirements (i.e., $\langle D_\beta, d_\beta, r_\beta, p_\beta \rangle$). If β is a complex action then A_k plans β individually by looking it up in its recipe library and selecting recipes that satisfy the apply-conditions and the constraints. Then, it uses the selected recipe to expand its graph G^k_α according to Definition 3.3 (see example in Fig. 5). We applied a heuristic algorithm in which the first recipe that satisfies the apply-conditions and the constraints is selected.

If β is a multi-agent action then A_k has to reach an agreement with the other participants of β about the development of β's plan. In our system, we applied a mechanism in which the first agent selects the vertex which is associated with β, plans this action by selecting an appropriate recipe and then incorporates it into the plan. However, first it obtains an agreement to plan β from of all of the other participants of β. If the vertex which is associated with β is selected by more than one agent (in $\mathcal{A}_\beta$) simultaneously, then an agent is drawn randomly, but there is a priority for an agent who cannot continue its plan without planning β.

In addition, the group $\mathcal{A}_\beta$ must agree on their assignment to the subactions in the selected recipe. In the assignment process the agents may be divided into subgroups. In our system, the agents are assigned to perform subactions according to the agent-constraints which are specified in the recipe. The agent-constraints referred to the resources of the agents and to the size of the subgroup. We applied a heuristic algorithm in which the agents which are suitable to a smaller number of subactions are assigned first. In this paper we focus on the identification of the temporal variables. In order to keep the algorithm simple we omit the details of the assignment algorithm. Instead, we assume that the subactions are associated with the agents who should perform them.

We prove in Section 4.1 that if A_k selects an applicable recipe R_β for β, and after the addition of the associated temporal constraints of R_β to G^k_α, G^k_α is consistent, then the Temporal Constraints Graphs of the other members in $\mathcal{A}_\beta$ will also be consistent with the associated temporal constraints. As a result, it is adequate if only one group member selects an applicable recipe and checks the consistency of the graph. This method decreases the computation time of the algorithm, and thus it reduces the load on the system.

The planning and execution loop also includes the exchange of information between the group members (part II in Fig. 6) as well as information exchanged between the AIP and RTS component of A_k (part III in Fig. 6). Also, during the planning and execution loop A_k may fail to achieve some action β and should backtrack (part IV in Fig. 6). A failure occurs in the following cases: (a) when the RTS component cannot find a feasible schedule; (b) when it is impossible to apply a basic action to the environment because its resources were consumed; (c) when it

is impossible to apply a selected recipe to the environment because of unexpected changes; (d) when its group members abandon the joint action. The backtracking mechanism is beyond the scope of this paper.

3.3 The temporal reasoning algorithm

In the following sections we describe in detail the algorithm in Fig. 6 and then we demonstrate the algorithm's operation using the rescue robots example.

3.3.1 Definition and initialization

In the main procedure, agent A_k receives action α along with the group $\mathcal{A}_\alpha$ as an input. The Temporal Constraint Graph is initialized (line 1 in Fig. 6) as well as some variables of the algorithm. The boolean variables *b IsFinishedExecuteAll*, *b IsCompletedPlan* and *b IsFailureInPlan* denote whether all of the basic actions associated with A_k have been executed and the status of the joint plan respectively. Initially, all of the vertices are *UNEXPLORED* (line 3 in Fig. 6). There are four types of *EXPLORED* vertices as follows:

1. A vertex of a single-agent action which is performed by A_k becomes *Explored Single-agent* (ES).
2. A vertex of a multi-agent action in which A_k participates becomes *Explored Multi-agent* (EM).
3. A vertex with unknown temporal values of an action in which A_k does not participate becomes *Explored Non-Participant Wait* (ENPW). In this case A_k waits until it receives the temporal values of the vertex from an appropriate agent.
4. A vertex with known temporal values of an action in which A_k does not participate becomes *Explored Non-Participant Terminated* (ENPT).

A vertex which is associated with an action which temporal values can be defined (because the temporal values of all of its predecessors have been defined) is called an *ENABLED* vertex. We distinguish between two disjointed sets of *ENABLED* vertices: The first set, denoted by $\mathcal{U}$, contains the *UNEXPLORED* vertices, i.e., the vertices which values can be identified but which the algorithm has not yet handled. The second set, denoted by $\mathcal{E}$, contains the *EXPLORED* vertices, i.e., the vertices which values have been identified. According to the algorithm, a vertex $u \in V_\alpha^k$ becomes *ENABLED* when the status of all of the preceding vertices of u becomes *EXPLORED* but not ENPW. All of the vertices that are denoted as ENPW (i.e., the agent waits to receive the values of their temporal variables from others) are maintained in the set $\mathcal{W}$.

A vertex associated with a specific time point is called a *fixed vertex*. Initially, vertex $s_{\alpha_{\text{plan}}}$ is a fixed vertex and is denoted as EM (line 4 in Fig. 6). Then, the $\mathcal{U}$ and $\mathcal{E}$ sets are updated recursively by an appropriate procedure which is called UPDATE_ENABLED_SET and is described in Fig. 17 of Appendix B. In Section 4.1 we prove that there is no deadlock in the system. Hence, at each stage of the algorithm there is at least one agent which may select an *UNEXPLORED* vertex from $\mathcal{U}$. Note

that, in the initialization phase, $\mathcal{U}$ set contains the *ENABLED* vertex s_α. Thus, each agent in $\mathcal{A}_\alpha$ starts its execution and planning loop from s_α.

3.3.2 Planning for a chosen action

During the planning process, a vertex s_β is selected to be planned by A_k if s_β is an *ENABLED* vertex and an *UNEXPLORED* vertex, i.e., s_β is selected from the *UNEXPLORED* ($\mathcal{U}$) set (line 9 in Fig. 6). For this selected vertex which is associated with action β, A_k checks the agents who can participate in the performance of β. In the case that A_k does not participate in β's performance (lines 10–17 in Fig. 6), if the temporal values of β are unknown, it changes the status of the vertices s_β and f_β to ENPW and adds the vertex s_β to $\mathcal{W}$ set (lines 12–14 in Fig. 6). Then, it attempts to select a new *ENABLED* vertex from the $\mathcal{U}$ set. If its $\mathcal{U}$ set is empty but its $\mathcal{W}$ is not empty, it waits for a message with temporal information from the other group members which will allow it to change the status of some ENPW vertices to ENPT vertices and to update its $\mathcal{U}$ set. If A_k does not participate in β's performance but the values of β are already known to A_k, it changes the status of the vertices s_β and f_β to ENPT and then it updates its $\mathcal{U}$ set (lines 15–16 in Fig. 6).

Assume that the selected *ENABLED* vertex is associated with an action β where A_k participates in β's performance (lines 18–25 in Fig. 6). If A_k is the only performer of this action then A_k distinguishes between basic actions and complex actions (the pseudocode is given in Figs. 19 and 24 in Appendix B, respectively). After A_k completes the planning of β's plan, if β is a subaction in a recipe of a multi-agent action, it should send information to its group members by running the CHECK_NECESSITY_TO_UPDATE_MEMBERS procedure (see Fig. 25 in Appendix B). The goal of this procedure is to determine whether β is a subaction in a recipe of a multi-agent action and whether β is in a set of the hinder members of some action β_i (i.e., $\beta \in H(\beta_i)$, see Definition 3.4). If $\beta \in H(\beta_i)$, the procedure checks if all other vertices in $H(\beta_i)$ are *EXPLORED* but not ENPW. If so, the temporal information of β can be sent to the performers of β_i. The methods for exchanging information are discussed in Section 3.4. In the case where β is a multi-agent action and A_k is one of the participants (lines 26–27 in Fig. 6), $\mathcal{A}_\beta$'s members have to reach a consensus on the recipe for β as described in Fig. 18 in Appendix B. Note that the SELECT_AGREEABLE_RECIPE procedure includes a process for selecting a recipe by the group $\mathcal{A}_\beta$ and a process for the assignment of the agents to subactions according to the SharedPlan model.

3.3.3 An illustration of the temporal reasoning algorithm

In Sections 2.1 and 2.3 we described an example from the rescue domain. In this section we illustrate the algorithm using that example. The rescue robots A_1 and A_2 intend to perform a multi-agent action α (i.e., "*rescue disaster survivors*") jointly. Suppose that A_1 and A_2 arrive at the disaster area at 4:00 P.M. Thus, both robots simultaneously begin the collaborative plan for α. Hence, planning and executing α begin after 4:00 P.M. Suppose that the batteries of A_2 are restricted to 150 min.; thus α must be completed within 150 min. We also assume that α must be finished before sunset, at 7:30 P.M. Thus, A_1 and A_2 construct their Temporal Constraints Graph accordingly (see the bold edges in Fig. 5 in Section 3.1).

Next, the agents need to agree on how they are going to perform action α (i.e., agree on a recipe for α). Suppose that the robots have agreed on R_α which is described in Section 2.3 and Fig. 3. At this stage of the planning process, both robots know about the selected recipe R_α and about its associated temporal constraints. Thus, each of the robots can incorporate the associated subactions and constraints into its Temporal Constraints Graph (see Fig. 5 in Section 3.1). Following this, the robots update the metric constraints that are associated with the selected recipe R_α. Then, each robot A_k, $(k = 1, 2)$, tries to continue the planning of its Temporal Constraints Graph by selecting an action to be planned. At this stage, G^1_α is identical to G^2_α but only action β_1 may be selected since this is the only action which is associated with an *ENABLED* vertex (i.e., s_{β_1}).

Since A_1 is the single agent performing β_1, A_2 has to wait until it receives the values of the temporal variables of β_1 from A_1, and the status of s_{β_1} and f_{β_1} are changed in A_2's graph to Explored Non-Participants Wait (ENPW). Suppose that the recipe that A_1 selected for β_1 consists of the basic actions: "*scanning outside area A*" and "*scanning under the rubble in area A*", denoted as γ_{11} and γ_{12}, respectively. Also suppose that the execution time for each of them is exactly 5 min. Figure 7 presents the Temporal Constraints Graphs G^k_α, $(k = 1, 2)$, which are maintained by each of the robots in this stage of their planning process. Thus, A_1 should identify $\langle D_{\gamma_{11}}, r_{\gamma_{11}}, d_{\gamma_{11}}, p_{\gamma_{11}} \rangle$ and $\langle D_{\gamma_{12}}, r_{\gamma_{12}}, d_{\gamma_{12}}, p_{\gamma_{12}} \rangle$ and send them to the Real-Time Scheduling (RTS) component. The decision regarding the exact time in which β_1 will be executed is determined by the RTS component. In this example, we assume that the RTS component of A_1 decides to execute γ_{11} at 4:02 P.M. and γ_{12} at 4:07 P.M.

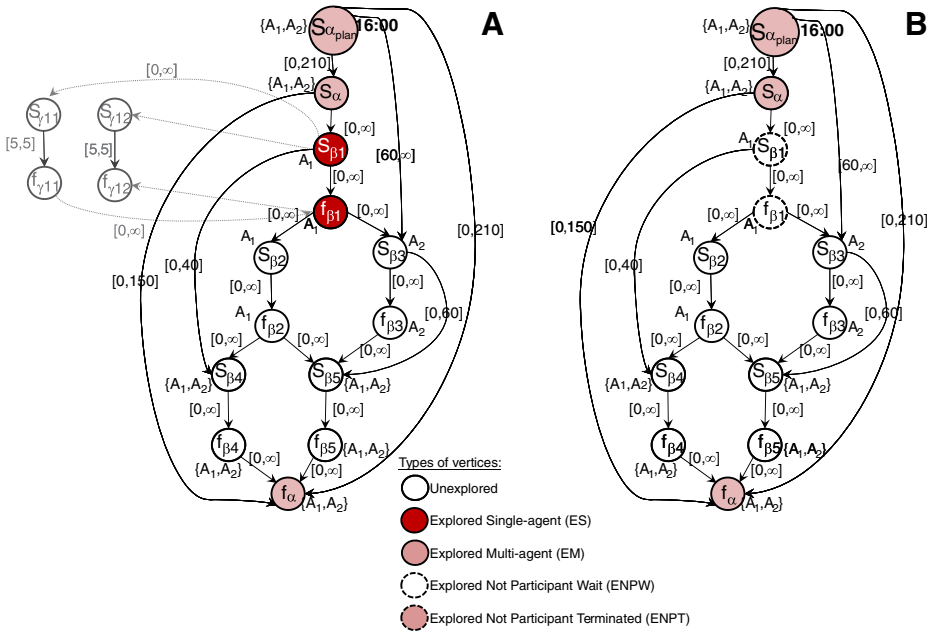

Fig. 7 The Temporal Constraints Graphs G^k_α, $(k = 1, 2)$, maintained by A_1 (graph **A**) and A_2 (graph **B**) after adding subactions γ_{11} and γ_{12} and the appropriate metric constraints by A_1. At this stage of the planning process the graphs of A_1 and A_2 are distinguishable

Thus A_1 will inform A_2 that it intends to terminate the execution of β_1 at 4:12 P.M. A_1 commits to this schedule by adding the edge $(s_{\alpha_{\text{plan}}}, f_{\beta_1})$ with weight [12, 12]. Following this announcement, A_2 changes the status of the vertices s_{β_1} and f_{β_1} from ENPW to Explored Non-Participant Terminated (ENPT) and it uses the temporal information it received from A_1 by adding edge $(s_{\alpha_{\text{plan}}}, f_{\beta_1})$ of weight [12, 12]. Similarly, A_1 commits to its announcement by adding this edge as well. In this stage of the planning process the status of the vertices $s_{\alpha_{\text{plan}}}, s_\alpha, f_\alpha, s_{\beta_1}, f_{\beta_1}, s_{\gamma_{11}}, f_{\gamma_{11}}, s_{\gamma_{12}}$ and $f_{\gamma_{12}}$ are *EXPLORED*, where $s_{\alpha_{\text{plan}}}, s_\alpha, f_\alpha$ are Explored Multi-agent (EM) vertices and $s_{\beta_1}, f_{\beta_1}, s_{\gamma_{11}}, f_{\gamma_{11}}, s_{\gamma_{12}}, f_{\gamma_{12}}$ are Explored Single-agent (ES) vertices . Thus, A_1 can choose to plan the action β_3 or β_2. Suppose A_1 chooses to plan β_3. Since $A_1 \notin \mathcal{A}_{\beta_3}$, it changes the status of s_{β_3} and f_{β_3} to ENPW and it plans β_2. Similarly, A_2 can start planning β_3. We assume that A_2 selected a recipe for β_3 which consists of the basic actions "*pick up pipes*" and "*clear boards and slabs*" which block the entrance, denoted by γ_{31} and γ_{32}. We assume that the execution time of γ_{31} is exactly 2 min. and the execution time of γ_{32} is 1 min. Figure 8 depicts the Temporal Constraints Graph G^k_α which is maintained by each agent in this stage of their collaborative plan after applying the CHECK_CONSISTENCY procedure (see Fig. 16 in Appendix B). Note that the implicit recipe trees of A_1 and A_2, at this planning stage, are part of the trees in Fig. 2 (i.e., $V^1_\alpha = \{\alpha, \beta_1, \beta_1, \beta_1, \beta_1, \beta_1, \gamma_{11}, \gamma_{12}\}$ and $V^2_\alpha = \{\alpha, \beta_1, \beta_1, \beta_1, \beta_1, \beta_1, \gamma_{31}, \gamma_{32}\}$).

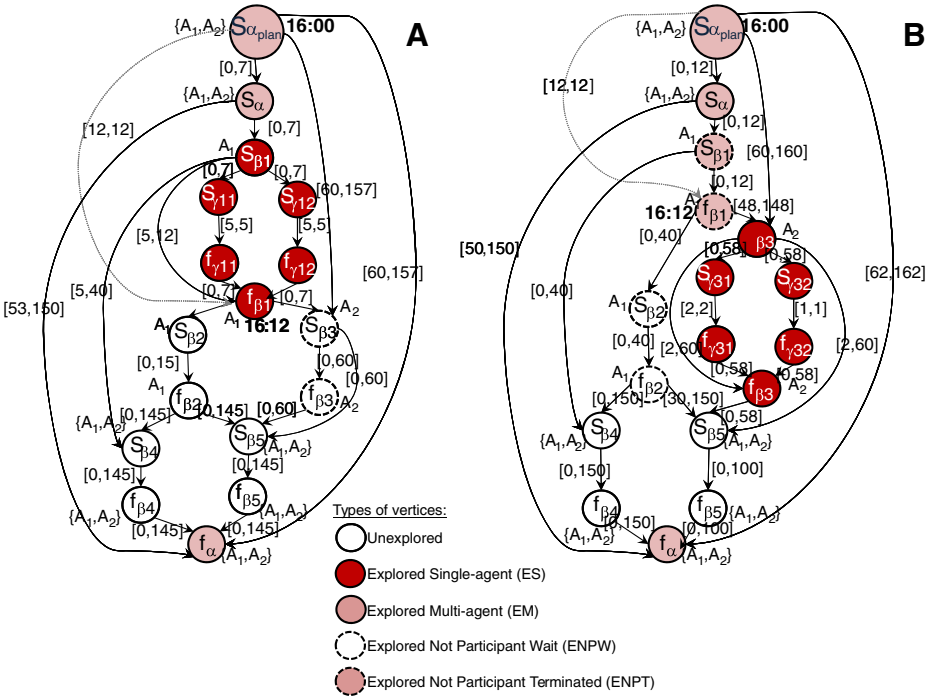

Fig. 8 Graph **A** is built by A_1 and graph **B** is built by A_2 during the collaborative planning. Each agent maintains a different graph according to its plan. The identical vertices represent the subactions which appear in the recipe of their collaborative action

3.4 Information exchange between agents

Identification of values for temporal variables in the collaborative plan for α requires information exchange. Each $A_k \in \mathcal{A}_\alpha$ exchanges information with its group members in the following cases:

(1) When they identify a recipe for their joint action or for any joint subaction in their plan. This exchange of messages may be about possible recipes and may also be part of their group decision-making.
(2) Agent A_k may inform its group members about completing the plan of a subaction in a recipe of a joint action. It informs the group members that its plan for their joint action has been completed.
(3) Agent A_k may inform its group members about the time values that it identified for the set $\mathrm{H}(\gamma)$ of individual actions which hinder γ (see Definition 3.4).
(4) When agent A_k finishes the execution of all of the basic level actions in its complete recipe tree it informs the group members.
(5) If A_k has already sent information to specific group members about some action β but failed to perform it, then A_k backtracks and informs them about the failure of β or about the changes in their plan that were determined as a result of the backtracking.

The information exchange in case (1) above, is done in the SELECT_AGREEABLE_RECIPE (β) procedure (lines 26–27 in Fig. 6). It refers to their agreement on the recipe selection and the assignment to subactions. The information exchanges in cases (2)–(4) are done in part II of the algorithm (lines 32–33 in Fig. 6). The last case is a case of failure (line 44 in Fig. 6) where the agent may announce failure or replan actions. Note that replanning of actions causes changes in the recipe tree as well as in the Temporal Constraints Graph.

Cases (2)–(4) above are described in the HANDLE_TEMPORAL_MESSAGES procedure which is presented in Fig. 9. In case (2) (lines 1–8 of HANDLE_TEMPORAL_MESSAGES

```
HANDLE_TEMPORAL_MESSAGES(message)
1  if the message is "temporal information of subactions_of_multi_recipe" then:
2    for each action γ_m ∈ subactions_of_multi_recipe do:
3      if the message includes temporal value of s_{γ_m} then:
4        status[s_{γ_m}] ← ENPT; W ← W − {s_{γ_m}};
5        weight(s_{α_plan}, s_{γ_m}) ← [s_{γ_m} − s_{α_plan}, s_{γ_m} − s_{α_plan}];
6      if the message includes temporal value of f_{γ_m} then:
7        status[f_{γ_m}] ← ENPT; W ← W − {f_{γ_m}};
8        weight(s_{α_plan}, f_{γ_m}) ← [f_{γ_m} − s_{α_plan}, f_{γ_m} − s_{α_plan}];
9  if the message is "waiting for temporal information" from a sender A_i then:
10   if there is a set H(γ) which is maintained by A_k and A_i ∈ A_γ then:
11     CALCULATE_AND_COMMIT_TIME of each action γ_m ∈ H(γ)
       and send the information to the members in A_γ;
12   else there is no such H(γ) set, then:
13     A_k marks that A_i is "waiting for temporal information";
14 if the message is "finish the execution" from a sender A_i then:
15   finish_execute[A_i] ← true;
```

Fig. 9 In HANDLE_TEMPORAL_MESSAGES $A_k \in \mathcal{A}_\alpha$ listens to the other members in $\mathcal{A}_\alpha$, updates its graph and sends the needed information. The variable *subactions_of_multi_recipe* indicates a set of subactions in a recipe of a multi-agent action. In the procedure CALCULATE_AND_COMMIT_TIME, the AIP component of A_k asks the RTS component to send it the termination times of all basic actions in the set $\mathrm{H}(\gamma)$ and updates the graph accordingly

procedure), A_k receives temporal information from its group members. Thus, A_k changes the relevant vertices in its Temporal Constraints Graph to ENPT. The goal of this message is to enable the agents to know what the status of their joint plan is. The joint plan is completed when all of the vertices in the graphs of all of the group members have been *EXPLORED* but not ENPW. In case (3), A_k may be asked about the temporal information (lines 9–13 of HANDLE_TEMPORAL_MESSAGES procedure). Once A_k sends the temporal values of $\gamma_1, \ldots, \gamma_m$ to its group members it commits to these values. Thus, one of the main questions in a distributed planning environment is at what stage should A_k inform its group members about the times it will take to perform $\gamma_1, \ldots, \gamma_m$, thus committing itself to these times. We consider two methods of answering this question.

In the first method, called *provide-time*, A_k sends the temporal information to its group members immediately when it completes the planning of all of the actions that directly precede γ that has to be performed by its group. Thus, A_k should commit to the temporal values it sends. This method enables the other group members to begin planning γ immediately upon completion of the planning of all actions preceding action γ. Furthermore, they do not need to ask each other for the relevant times since they are informed about them as soon as possible. However, since A_k has to commit to these times it has less flexibility in determining the time performance for its other actions. Having flexibility means that an individual agent has more alternatives to change temporal values while it does not need to coordinate the changes with the other group members. On the contrary, the flexibility is reduced when an agent decides to change the announced temporal values and it needs to negotiate with its group members. Thus, we also consider an alternative mechanism, called *ask-time*. Following this approach, each member plans its individual actions as long as it does not depend on its group members' activities. When such a case occurs, the relevant member asks for the appropriate time values from its group members (lines 8–12 of HANDLE_TEMPORAL_MESSAGES procedure). In this manner, the commitment is left to the latest time possible, but it may delay the agent waiting for an answer to plan its actions and this method results in additional messages being exchanged.

3.5 Group planning order

An additional problem in a distributed planning environment involves the order in which the members of the group should plan the joint action. As described above, during the planning process, each agent A_k in the group selects a vertex s_β in G_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. In a joint activity a selection of a specific vertex by an individual agent may influence the activity of the entire group. For instance, in certain cases group members may need temporal information about a specific action from another agent A_k. But the order in which A_k chooses to complete the plan of this action may influence the temporal information that it will eventually send and when this information will be available.

In this work we consider three methods for determining the planning order of an individual in the group. The first is called *random-order*, where A_k randomly selects one of the several actions that can be planned. In the second method, called *dfs-order*, A_k selects the action according to the depth-first search order of G_α^k. In this order, the agent selects an action from the lowest level of its recipe tree. Thus, it plans a subaction until it reaches the basic action level, then it continues to the next subaction. The third is called *bfs-order*, where A_k selects one of the actions according to the breadth-first search order of G_α^k. In this order, the agent tries to complete the plans of the highest level actions in each stage of the planning process of its recipe tree. Note that in the first method the planning order of the group members differ. In the two latter methods, all of the agents in the group plan their subactions in the same order. Forcing all of the group members to plan subactions in the same order may decrease the flexibility of the individuals, but planning the activities in different orders may delay the plans of some members of the group. Our simulation results, presented in Section 5.1, demonstrate that the planning order influences the success rate of the agents. We show that it is better to commit as late as possible, and thus the bfs-order and ask-time methods perform the best.

4 Correctness and complexity of the temporal reasoning algorithm

In this section we present the correctness of the temporal reasoning algorithm for special cases and discuss its complexity. First, we present lemmas and propositions to prove that no deadlocks occur and that the algorithm always terminates. Then, we show the cases of soundness and completeness. The termination of the algorithm also depends on the following cases: (a) the agreement of the group members to perform the selected recipe; (b) the method used to assign group members to perform the subactions; and (c) other constraints associated with the selected recipe. In our proofs we assume that the members always agree to perform the recipes that satisfy the temporal constraints as well as to perform the subactions that are associated with them. Also, the recipe does not include additional constraints, except for temporal constraints. That is, in our proofs we focus on the consistency of the temporal constraints in the plan.

4.1 Lemmas and propositions

In each stage of the algorithm, A_k selects a vertex v which is an *ENABLED* vertex and tries to identify its temporal value. In the following proposition we prove that, according to the algorithm, the temporal information of the selected vertex can be identified and that this vertex is therefore *ENABLED*.

Proposition 4.1 *Suppose that the AIP component of an agent A_k runs the Temporal Reasoning Algorithm, which builds the Temporal Constraints Graph $G_\alpha^k = (V_\alpha^k, E_\alpha^k)$. Let v be a vertex in V_α^k and S be the set of all minimal members*[4] *of the set $\{u | u$ is a*

[4]We recall that v is a minimal member of S in a partial order if $v \in S$ and no other $w \in S$ exists such that w precedes v.

fixed vertex and u precedes v}. Then, during the building of the Temporal Constraints Graph, if each vertex in the path from all the vertices in S to v are EXPLORED *(but not ENPW), then v is* ENABLED.

Proof By induction (see Appendix A). □

In the following proposition we prove that all of the values of the variables associated with the basic actions which are sent by A_k to its RTS component are not changed later by the AIP component (except for the case of backtracking). Thus, these actions can be scheduled and executed before the agent completes its planning for α. This fact enables the agent to interleave planning and execution.

Proposition 4.2 *Suppose that the AIP component of an agent A_k runs the Temporal Reasoning Algorithm which builds the Temporal Constraints Graph $G_\alpha^k = (V_\alpha^k, E_\alpha^k)$. Let $s_\beta \in V_\alpha^k$ be an* ENABLED *which represents the start time of the basic level action β which has to be performed by A_k.*

Then, the values of the temporal requirements (i.e., $\langle D_\beta, d_\beta, r_\beta, p_\beta \rangle$), which are associated with action β and are sent by A_k to the RTS component, are not changed during the planning process of α (unless A_k backtracks and chooses a different recipe).

Proof Since β is a basic level action it is obvious that the computation time, D_β, is final.[5] Now, we have to show that if v represents a start time point of β, and v is an *ENABLED*, then r_β can be identified. Since v is an *ENABLED*, all of the paths between all of the minimal members of the set $S = \{u | u$ is a fixed vertex and u precedes $v\}$ to v are final, and the weights of all of the edges in these paths are final, thus the final value of r_β can be identified. Similarly, the final value of d_β can be identified. Also, all of the basic edges in the paths between all of the minimal members of the set S to v are final. Thus, all of the basics actions preceding v are final. □

Proposition 4.3 *Suppose that the AIP component of A_k runs the Temporal Reasoning Algorithm which builds the Temporal Constraints Graph G_α^k. Let T_α^k be the implicit recipe tree of G_α^k and T_α be the union of implicit recipe trees. Let β be a node in T_α^k such that $A_k \in \mathcal{A}_\beta$. Then, according to the algorithm, a node β is a leaf in T_α^k if and only if β is a leaf in T_α.*

Proof

($\Rightarrow$) Suppose that β is a leaf in T_α^k. Since $A_k \in \mathcal{A}_\beta$, then either $\{A_k\} = \mathcal{A}_\beta$ or $\{A_k\} \subset \mathcal{A}_\beta$. However, according to the algorithm, if $\{A_k\} = \mathcal{A}_\beta$, A_k is the only planner of this action. Thus, if β is a leaf in T_α^k, then either β is a basic action or β is not planned, and thus β is leaf in T_α. If $\{A_k\} \subset \mathcal{A}_\beta$ according to the algorithm the agent who plans β informs all members in $\mathcal{A}_\beta$ of the recipe selected for performing β and all of them update their Temporal Constraints Graphs.

($\Leftarrow$) It is easy to see that, according to the construction of union recipe trees, if β is a leaf in T_α then β is a leaf in T_α^k. □

[5] We use the term "final" to refer to the values of the variables that will not be changed during the planning process (unless the agent backtracks).

In the following lemma we first prove that no deadlocks occur. In other words, at each stage of the planning process at least one of the agents in the group has at least one *UNEXPLORED* vertex in its $\mathcal{U}$ set. Then, we prove that the algorithm always terminates.

Lemma 4.1 *Let $G_\alpha^k = (V_\alpha^k, E_\alpha^k)$ be the Temporal Constraints Graph of agent $A_k \in \mathcal{A}_\alpha$. Suppose that the AIP component of A_k runs the Temporal Reasoning Algorithm which constructs G_α^k from a given initial graph G_α^k where $V_\alpha^k = \{s_{\alpha_{\text{plan}}}, s_\alpha, f_\alpha\}$ and $E_\alpha^k = \{(s_{\alpha_{\text{plan}}}, s_\alpha), (s_\alpha, f_\alpha), (s_{\alpha_{\text{plan}}}, f_\alpha)\}$. Then, during A_k's execution:*

1. *If G_α^k consists of some* UNEXPLORED *vertex or ENPW vertex, then there is at least one agent (in $\mathcal{A}_\alpha$) in the group whose $\mathcal{U}$ set is non-empty.*
2. *If all of the vertices in G_α^k are* EXPLORED*, then A_k has completed identifying all of the values of the temporal requirements (i.e., $\langle D_\beta, d_\beta, r_\beta, p_\beta \rangle$) of all of the basic actions that should be executed by A_k.*

Proof

1. Since G_α^k is a directed acyclic graph (DAG), we can perform a topological sort on the graph. Let v_i be the first *UNEXPLORED* vertex or the first ENPW in the order of the topological sort. If v_i is *UNEXPLORED*, then it is clear that all of the vertices in the paths from $s_{\alpha_{\text{plan}}}$ to v_i are *EXPLORED* (but not ENPW). Thus, by Proposition 4.1, v_i is an *ENABLED* and $v_i \in \mathcal{U}$ of A_k. If v_i is ENPW, suppose that v_i is associated with action β. By the definition of Temporal Constraints Graph, β must be a subaction in a recipe of a multi-agent action. Since all of the vertices which precede s_β are *EXPLORED*, s_β is *ENABLED* and thus, according the algorithm, the temporal values of $\mathrm{H}(\beta)$ will be sent to the members of $\mathcal{A}_\beta$. Consequently, s_β becomes an *ENABLED* in the graphs of $\mathcal{A}_\beta$ and the $\mathcal{U}$ set of these members is non-empty.
2. Suppose, by contradiction, that A_k has not finished identifying all of the values of the temporal variables of the actions in which performance it participated. Thus, the graph consists of at most one action β for which A_k did not identify its time variables and $A_k \in \mathcal{A}_\beta$. But, according to the algorithm, for each basic level action β where $A_k \in \mathcal{A}_\beta$, the vertices which represent action β have been *EXPLORED* once the temporal variables of β are identified. For each complex level action β where $A_k \in \mathcal{A}_\beta$, when the vertices which represent this action become *EXPLORED*, new *UNEXPLORED* vertices are added to the graph (i.e., the vertices which represent the subactions of β). Thus, the assumption that all of the vertices in the graph are *EXPLORED* is contradicted. □

Corollary 4.1 *When the algorithm terminates, all of the leaves in the union of the implicit recipe trees of individual agents are associated with basic level actions.*

Proof Otherwise, if leaf β of the union recipe trees is a complex level action in which A_k has participated in its performance, then, by Proposition 4.3, β is a leaf in the implicit recipe tree of A_k and, according to the algorithm, the graph includes an *UNEXPLORED* vertex and the algorithm has not terminated. □

In the following lemma we prove that in each stage of the algorithm the performance of the actions in the leaves of the implicit recipe tree is consistent with all temporal constraints of α.

Lemma 4.2 *Suppose that a group of agents $\mathcal{A}_\alpha$ plans action α. Let T_α^k be the implicit recipe tree of G_α^k and T_α be the union implicit recipe trees. Then, during the planning of the graph G_α^k by each $A_k \in \mathcal{A}_\alpha$, performing (possibly in parallel) all of the actions (possibly complex) in the union implicit recipe trees, T_α, is consistent with α's temporal constraints.*

Proof By induction (see Appendix A). □

Corollary 4.2 *Suppose that during the planning of graph G_α^k by $A_k \in \mathcal{A}_\alpha$, A_k selects a recipe R_β in order to perform β. Assume that after adding the associated temporal constraints of R_β to G_α^k, G_α^k is consistent. Then, the same action makes all G_α^i, $(i \neq k)$ for each $A_i \in \mathcal{A}_\beta$ also consistent.*

Proof Otherwise, the temporal constraints of R_β are not consistent in the union recipe trees of α. □

4.2 Soundness and completeness theorem

The correctness depends on the way in which TCSP is generated and solved as part of the Temporal Reasoning Algorithm. Assume that the algorithm applies only to special cases of TCSP that can be solved by sound and complete methods (such as STP [16]). Then, for these cases we can prove that the Temporal Reasoning Algorithm is sound and complete.

Theorem 4.1 *Suppose that a group of agents $\mathcal{A}_\alpha$ needs to perform a joint action α with a given set of temporal constraints.*

1. **Soundness***: Assume that for each agent, $A_k \in \mathcal{A}_\alpha$, the Temporal Reasoning Algorithm terminates after having identified the set of basic actions along with their temporal requirements $\langle D_\beta, d_\beta, r_\beta, p_\beta \rangle$. Then the execution of these basic actions (possibly in parallel), according to the identified temporal requirements, is consistent with all of the temporal constraints of α.*
2. **Completeness***: If there exists a complete recipe tree for α which satisfies all of the appropriate temporal constraints and for which RTS can find a feasible schedule, then the Temporal Reasoning Algorithm identifies all the basic actions along with their corresponding values of temporal requirements for each agent, $A_k \in \mathcal{A}_\alpha$. Otherwise the algorithm fails.*

Proof

1. By Corollary 4.1, when the algorithm terminates, all of the leaves in the union of the implicit recipe trees of all individual agents are associated with basic level actions.
   By Proposition 4.2, the agent can identify all of the values of the temporal requirements of these actions. Now we have to prove that the execution of

the basic actions under the identified values of the temporal requirements is consistent with all of the temporal constraints of α, but by Lemma 4.2, during the planning of the graph G_α^k by each $A_k \in \mathcal{A}_\alpha$, performing the actions is consistent with the temporal constraints of α.

2. Suppose that during the planning of the implicit recipe tree for α by agent $A_k \in \mathcal{A}_\alpha$ either A_k does not find a recipe for some complex level action which satisfies α's temporal constraints, or the RTS component of A_k cannot find a feasible schedule.[6] Then A_k can use some known backtracking method on the implicit recipe tree for α, which enables it to check all of the options of all the other appropriate available recipes (by Corollary 4.2 it is enough that only one agent checks all of these options). As a result, only if such a recipe does not exist, the agents fail in their plan. □

4.3 Complexity

The planning loop of the algorithm includes four major parts: (I) planning for a chosen action; (II) exchanging messages between group members; (III) dispatching and scheduling basic actions by the Real-Time Scheduling (RTS) component; and (IV) backtracking.

For the complexity analysis of part (I) and part (II), denote m as the number of nodes of the largest partial recipe tree that has been planned by A_k during the performance of the algorithm. Denote s as the number of times the process for selecting a recipe is initiated by a member A_k, and h as the number of messages with temporal information which A_k receives. Similar to the correctness proof, the complexity analysis of resolving the temporal constraints in part (I) depends on the way TCSP is generated and solved. The general TCSP problem is intractable (see [16, inter alia]) but there is a simplified version, *Simple Temporal Problem* (STP), in which each constraint consists of a single interval. This version can be solved by using efficient techniques available for finding the shortest paths in a directed graph with weighted edges such as Floyd-Warshall's all-pairs-shortest-paths algorithm[7] [13]. Thus, in a case of STP, the number of times that A_k runs the Floyd-Warshall algorithm is $(s + h)$. Since the complexity of the Floyd-Warshall algorithm is $O(m^3)$, in this case, resolving the temporal constraints is $O((s + h)m^3)$. Hence, in a case when using STP, if there is a unique possible recipe for each complex action (backtracking is unavailable) resolving the temporal constraints is polynomial in reference to the number of nodes of this tree. However, the scheduling problem that the RTS component faces (part III) is NP-hard [24]. In addition, if there is more than one possible recipe for each complex action and backtracking is available, the complexity analysis is equivalent to the complexity of HTN planning and may be exponential (see the discussion of the HTN's complexity in [25]).

[6] Note that the scheduling problem faced by the RTS component is NP-complete and the RTS component employs a heuristic algorithm. Though the heuristic has been proven to be efficient in our domain (see [33]), the RTS component does not guarantee that it will find a solution when a solution exists.

[7] Floyd-Warshall's algorithm efficiently finds the shortest paths between all pairs of vertices in a graph.

Thus, the overall complexity is exponential and several heuristics should be used to limit the search. The complexity of our solution is limited by the NP-hard component in part III, and potentially by the TCSP formalization in parts I and II. Practically, we overcome this theoretical complexity in parts I and II by using a simplified STP that can be solved through the Floyd-Warshall algorithm, which assumes that the self-interested agents will always accept the plans being proposed to them by other group members. Assuming our simplifying assumptions are not true, the original TCSP NP-hard complexity would need to be addressed through approximations and heuristics. Also, in Part IV, the backtracking needs heuristics to limit the search as the problem is inherently NP-hard or may even contain an exponentially large search space.

5 Experimental analysis and results

The goal of this section is to evaluate elements of the planner within our mechanism. We first study the general behavior of the planning mechanism through a series of experiments involving planning interactions between two self-interested agents. While our work considers self-interested planning, previous leading multi-agent planning work (e.g., [11, 47, 50, 86]) assumes that each agent acts selflessly for the group (also see Section 6 for more about these planners). Thus, any comparison with the previous approach is not relevant. As a result, in the first series of experiments, we focus on elements that are unique to our self-interested planning agents, such as when to communicate and when to commit to an action or schedule. We then study the ability to implement this mechanism in a real-world domain while we consider larger groups of agents (up to 12) as well as issues regarding how our teamwork implementation is superior to previous teamwork models without planning and temporal constraints, such as STEAM and BITE [41, 82].

5.1 Studying distributed planning methods of self-interested agents

Specifically, we studied two questions regarding the planning process of self-interested agents. The first question concerns the stage at which an agent commits itself to the temporal values in its schedule and communicates these values to the relevant members. The second question refers to the planning order of the subactions of a joint activity. We explored these questions by implementing the Temporal Reasoning Algorithm when the agents used the *provide-time* and *ask-time* methods for information exchange (see Section 3.4) and the *random-order*, *dfs-order* and *bfs-order* methods for group planning order (see Section 3.5). The combined methods are called *random-provide*, *dfs-provide*, *bfs-provide*, *random-ask*, *dfs-ask* and *bfs-ask*, respectively. These six different methods of distributed planning were implemented in the SharedPlan system separately.

We ran the SharedPlan system in a rescue-robot domain composed of two agents with different capabilities: a small and flexible robot and a large and strong robot. The goal of the agents was to execute the action "*rescue disaster survivors*" by executing all of the basic actions in such a way that each individual agent had to execute its own part in the plan without violating the temporal constraints. As presented in our mechanism, the agents could interact with each other and each of them planned its

own recipe tree and decided about the values of the temporal variables of subactions in its partial recipe tree while it sent its basic subactions to the RTS component for scheduling and execution. At each stage in the planning process the agents selected an appropriate recipe according to their beliefs about the world-states by looking it up in the recipe library. To study planning elements, we intentionally focused on problems where the duration of the executable actions are short in order to force agents to interleave planning and execution. We created a knowledge base of actions and recipes where each agent could execute 60 different basic level actions. The agent's type as well as its basic capabilities were stated as static beliefs in the agent's knowledge-base. Specific examples of the basic actions in the rescue-robot domain are: "*picking up*", "*cutting*", "*breaking*", "*hitting*", "*chiseling*", "*shaving*", "*peeling*", "*digging*", "*removing*", "*prying*", "*pulling*", etc.

In addition, the rescue-robot domain includes several complex actions such as: "*searching for victims*", "*excavation of debris*", "*clear piles of rubble*", "*find and identify victims*", "*specify locations of victims*", "*rescue victims*", "*clear obstructions*", etc. Section 2 demonstrates a possible recipe (Fig. 3) and recipe trees (Fig. 2) for the action "*rescue disaster survivors*" (as implemented in the SharedPlan system). The union of the recipe trees of both robots represents the joint plan, which we term the *union recipe tree*.

We conducted a set of experiments in which we produced 140 different *union complete recipe trees* for the joint action "*rescue disaster survivors*" (i.e., 140 fully initiated plans considering all actions and subactions but not associated with temporal values and constraints) for random world-states created by the external generator. The external generator created temporal constraints for actions (in each union complete recipe tree) by starting with the lowest levels actions (i.e., basic actions) up to the higher level actions recursively. All temporal constraints have been generated randomly in such a way that they were consistent with the appropriate world-states and the other actions in the tree. This was done by creating a consistent schedule for the basic actions in the complete recipe tree. Then, each complex action was generated recursively by creating a general recipe that was guaranteed to be consistent with all possible world-states and then one constraint was chosen at random from a set of random temporal constraints that were guaranteed to be consistent with the schedule. Hence, for each set of generated constraints, the recipe library did include a union complete recipe tree in which all of the temporal constraints could be satisfied. In particular, the above process identified recipes for the highest level action "*rescue disaster survivors*" and for the appropriate subactions of each recipe and stored them in a recipe library.

We focused on world-states in which the union complete recipe tree included: (1) a total of between 100 and 120 (between 50 and 60 for each agent) *basic actions*; the duration of each basic action could be between 1 to 10 min; (2) an average of between 0.20 and 0.55 *precedence constraints* between actions that were to be performed by the same agent; (3) an average of between 0.7 and 1.1 *metric constraints* with respect to all of the actions in the recipe tree; and (4) a total of 3 complex *multi-agent actions*.

We then ran the *random-provide*, *dfs-provide*, *bfs-provide*, *random-ask*, *dfs-ask* and the *bfs-ask* methods as part of the temporal reasoning mechanism and we tested the *success rate* of each method in a given range of multi-precedence constraints. In each test we supplied the agents with a different set of generated world-states. As we focused on testing the distributed affect in group planning, we made the

simplified assumption that the agents were acquainted with only one recipe for performing a complex action in a specific world-state and backtracking was not allowed. Consequently, the recipe library included a unique union complete recipe tree for a given world-state. Note that the agents may become aware of new world-states during the planning and the execution of subactions. Thus, although all of the recipe trees with their appropriate constraints were generated by us in advance, the agents did not know them in advance, and the trees evolve incrementally while new world-states are revealed over time.

We assert that a group of agents, $\mathcal{A}_\alpha$, *succeeds* in performing the joint action α if: (1) The AIP component of each agent $A_k \in \mathcal{A}_\alpha$ has completed the planning for α and by building a complete recipe tree for α which satisfies all ofthe appropriate temporal constraints; (2) The RTS component of each $A_k \in \mathcal{A}_\alpha$, has found a feasible schedule, which consists of all of the basic level actions in A_k's complete recipe tree for α. A failure is defined when either the AIP component of at least one agent (in $\mathcal{A}_\alpha$) has failed to build a complete recipe tree, or the RTS component of at least one agent (in $\mathcal{A}_\alpha$) has failed to find a feasible schedule. Accordingly, the *success rate* of a method is obtained by dividing the quantity of cases that $\mathcal{A}_\alpha$ *succeeds* in performing α by the total cases where $\mathcal{A}_\alpha$ tries to perform α.

We also compared the performance of the temporal reasoning mechanism in its distributed planning mode, in which each agent planned its own recipe tree, with another mode in which a union recipe tree has been planned centrally by a central planner where each agent was equivalent to the RTS component (see Fig. 10). The computational power of the central planner was equivalent to the computational power of the AIP component of one agent in the distributed mode. During the planning process, the central planner sent the basic actions with their associated temporal constraints to the appropriate agent for scheduling and execution. Similar to the distributed planning mode, each agent was able to execute 60 different basic level actions according to its type. The union recipe tree has been planned according to the *bfs-order* planning method. When the central planner chose to plan action γ which had to be performed by agent A_j, but γ had precedential basic actions $\beta_1, \ldots, \beta_m$ that were assigned to be performed by another agent A_i, the central

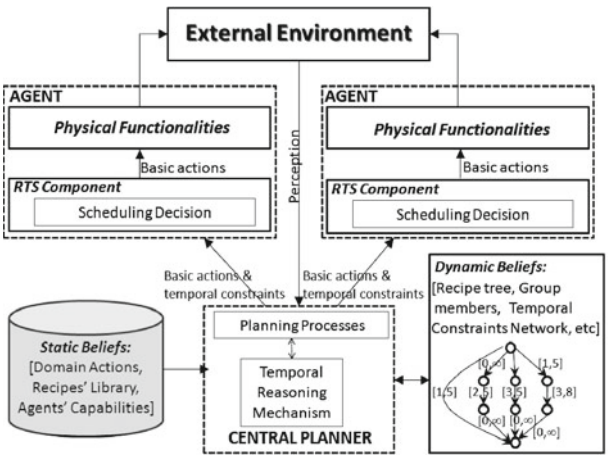

Fig. 10 An illustration of the system with a central planner

Table 2 A comparison between the success rate of the different methods

Constraints	random-provide (%)	dfs-provide (%)	bfs-provide (%)	random-ask (%)	dfs-ask (%)	bfs-ask (%)	Aggregated (%)	Central (%)
0	75	94	95	86	92	98	90	80
1–2	79	75	82	87	88	88	83	91
3–4	64	66	70	84	75	83	74	72
5–6	58	53	88	84	88	84	76	66

planner requested the times of $\beta_1, \ldots, \beta_m$ from A_i who became committed to these times. Note that this is relevant only for the execution time of basic actions. All other information regarding the joint plan was locally available to the central planner. The success rate is defined in a similar way to the distributed planning mode but here the central planner has to complete the planning for α under the appropriate temporal constraints.

5.1.1 The effects of multi-precedence constraints between actions

First, we tested how the number of *multi-precedence constraints* between subactions affect the performance of the system. Our hypothesis was that a large number of multi-precedence constraints would decrease the flexibility of the agents in deciding about the temporal values to perform their individual activities. Thus, it would reduce the success rate. The success rate of each method is given in Table 2. The first column specifies the number of *multi-precedence constraints* and the other columns specify the success rate of each method in a given number of *multi-precedence constraints*. The aggregated column refers to the success rate of the aggregation of all of the distributed methods and the last column refers to the success rate of the central planner. As shown in Table 2, in some cases the behavior is non-monotonic. This was in contrast to our hypothesis, something we discussed below.

To study our first question, we checked whether the *ask-time* method was better than the *provide-time* method. The results in Fig. 11 show that, in the general case (i.e., if we consider all the instances regardless of the number of precedence

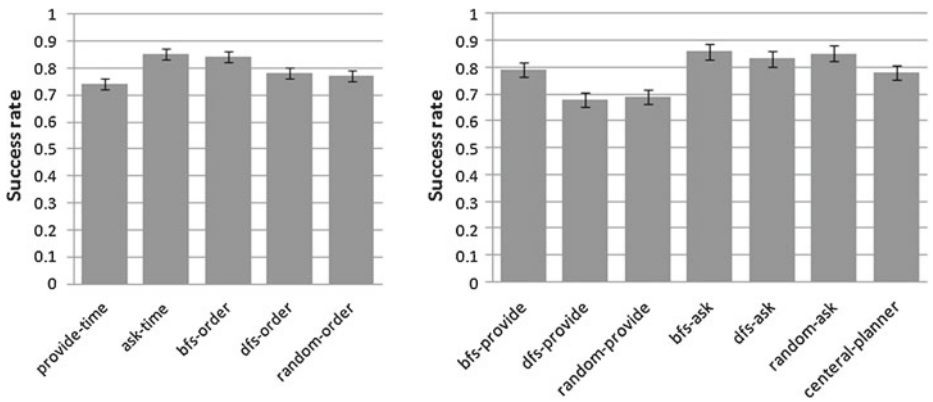

Fig. 11 A comparison between the success rate of each method in the general case

constraints) the *ask-time* method is significantly better[8] than the *provide-time* method (p-value < 0.01). The comparison between the *dfs-ask* and *dfs-provide* methods also shows that the *ask-time* method is significantly better than the *provide-time* method (p-value $= 0.025$). Similarly, when we consider the general case with all the instances, the *bfs-ask* method is significantly better than the *bfs-provide* method (p-value < 0.01). The reason that the success-rate is not equal for all methods when the number of the multi-precedence constraints is "0" results from the failures of the heuristic algorithm which is employed by the RTS component (see Section 2.4). Nevertheless, if we consider the specific case of "0" multi-precedence constraints, the difference between the *dfs-provide* and *dfs-ask* as well as the difference between the *bfs-provide* and *bfs-ask* is not significant.

Thus, as we hypothesized, we can conclude that in the general case (regardless of the number of precedence constraints), when the agents make their commitments as late as possible, their performance is better. This is also the reason that the success rate of the *dfs-provide* method decreases as the number of multi-precedence constraints increases (see Table 2). 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 schedule and commits to it at an early stage. As the number of multi-precedence constraints increases, commitments are made earlier. In the other methods the success rate does not change monotonically as a function of the number of multi-precedence constraints. We assume 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 direct the scheduler and the other group 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 group since they cause them to make more commitments in their schedule. Lowered flexibility leads to a lower success rate in cases of 3–4 multi-precedence constraints. We believe that these results come about because if a problem is weakly constrained, a complete algorithm will easily find a solution, and if the problem is very strongly constrained, the problem is again easy since the complete algorithm can prune most of the branches in the search tree. Therefore, the most difficult problems will exist in the middle. Thus, this problem follows easy-hard-easy pattern noted in other constraint satisfaction problems [53, 57, 91].

To study our second question, we explored how the planning order affects the success rate. A comparison of *dfs-provide* to *bfs-provide* shows that when the number of precedence constraints is "0" then the success rate of *dfs-provide* is equal to the success rate of *bfs-provide*. However, if we consider the general case with all of the instances (regardless of the number of precedence constraints) then the *bfs-provide* method is significantly better than the other methods of distributed planning (p-value < 0.01). These results are not surprising since the stage at which the agent makes its commitments according to the *dfs-provide* method is earlier than the *bfs-provide*

[8] In the rest of this section, we used the *chi-squared* test for a comparison between several values. In this case a standard t-test is not appropriate as there is more than one pair of results to compare. The *likelihood-ratio* test is selected to compare two models. In the other cases we used the standard t-test.

method, and the *random-provide* method is a combination of the *bfs-provide* and *dfs-provide* methods.

The surprising results are shown in the comparison between *random-ask*, *dfs-ask* and *bfs-ask* methods because in this case it is unclear which method is better (see the right graph in Fig. 11). The reason for these results is the order in which recipes with the multi-precedence constraints are selected affects the success-rate. If such recipes are selected early, the commitment by the agents comes at an early stage, thus reducing their flexibility. However, because the recipes trees were created randomly, in certain cases the *dfs-order* adds these recipes to the plan before the *bfs-order* while in other cases such a recipe first appears according to the *bfs-order*. Thus, there is no one method that is significantly better than the others. Consequently, for certain world-states *dfs-ask* is better, and for other world-states *bfs-ask* is better. The *random-ask*, which is a random combination of *bfs-ask* and *dfs-ask*, drew the wrong order in some cases, which caused it to fail more than the *dfs-ask*. But, in certain cases, it drew the best order for a specific world-state resulting in good outcomes. The reason for the low success rate of the random order method, in particular in the case where there were no multi-precedence constraints, is the heuristic algorithm for scheduling which is employed by the RTS component (see Section 2.4). The heuristic assumes that the AIP component first sends all the basic actions which are constituents of a specific complex action and only then sends the constituents of another complex action. The random order does not satisfy this assumption.

The last column in Table 2 shows the success rate of the central planner. As indicated in Fig. 11, the distributed *bfs-ask* is significantly better than the central planner (p-value = 0.025). The central planner attained worse results than the *bfs-ask* for two major reasons. First, the high processing time (even on the high-end Sun Workstation that we used) to conduct planning (which is more substantial than the overhead caused by the exchanged messages in the distributed method) leads to a delay in sending some basic actions for execution and causes certain basic actions to miss their deadline. This low success rate can be improved by increasing the computational power of the central planner. Second, the central planner plans the actions of all of the group members and a high number of precedence constraints causes the central planner to ask the agents to make more commitments in their schedule. Thus, similar to the *provide-time* method, these commitments reduce the flexibility of the scheduling process, particularly when the multi-precedence constraints are in the highest levels of the recipe tree and are done at an early stage. However, if we compare the central planner to the *provide-time* method when using distributed planning, the central planner is better in some cases. Note that since the processing time to conduct planning by the central planner is high, the *ask-time* method which increases the delay of this time is inefficient for the central planner.

5.1.2 The effects of multi-precedence constraints between complex actions

In our experiments we notice that the multi-precedence constraints between basic actions significantly decreases the success rate of the system. This is because the RTS schedules basic actions and not complex actions. As a result, when the agent commits to the time of a basic action, it causes the RTS component to commit the agent to precise time frame values. However, in the case of multi-precedence constraints between complex actions, the agent estimates the time frame (based on the basic actions that constitute the complex action and have been scheduled). Thus,

while the multi-precedence constraints between basic actions cause the agent to be committed to a tight time frame, the multi-precedence constraints between complex actions allow the agent to be committed to a wide range of possible values. Also, when the constraints are between basic actions in the high level of the recipe tree, the commitment is made at a very early stage of the planning process. In this section we study the implications of the multi-precedence constraints between complex actions and basic actions separately. For this purpose, we selected the planning problems whose solutions included multi-precedence constraints only between complex actions; the total number of these problems was 111.

As we hypothesized, if we remove the cases with multi-precedence constraints between basic actions, the performance of the system is significantly higher (p-value < 0.01). The results are given in Table 3. As indicated from the table in *bfs-provide*, the success rate is 100 % when the recipe consists of 1–2 and 5–6 multi-precedence constraints (vs. 82 % and 88 % where multi-precedence constraints between basic actions are allowed). The lowest success rate is 80 % with 3–4 constraints. The lowest success rate of the *bfs-ask* which is significantly better than the *bfs-provide* (p-value < 0.01) is 89 %. Also, the *random-ask* is significantly better than the *random-provide* (p-value < 0.01) and the *dfs-ask* is significantly better than the *dfs-provide* (p-value $= 0.0385$). These methods are also attained higher results than the case of multi-precedence constraints between basic actions. The results also show that the *bfs-order* planning method is significantly better for all the cases (p-value < 0.05). Thus, for environments without multi precedence constraints between basic actions, we can conclude that it is more efficient for a group to perform their plan in the same order and to first plan the highest level actions.

Overall, these results demonstrate that planners for self-interested agents should commit to the group as late as possible. Furthermore, when group members use a similar order to plan the sub-actions of their joint activity, they have a better chance of succeeding in executing the joint action under all the temporal constraints. We have tested six different methods of distributed planning with different numbers of multi-precedence constraints; the success rate of the aggregation of all the distributed methods was about 80 %. Nevertheless, the results may be improved by allowing backtracking and more choices of recipes.

5.2 Effectiveness of the mechanism in a real-world simulator

While the above results are important for testing the limits of the planner, we also focused on how to practically implement this mechanism and how our teamwork model compares with other implementations. Towards this goal, we present how

Table 3 A comparison between the success rate of different methods when the multi-precedence constraints are only between complex actions

Constraints	random-provide (%)	dfs-provide (%)	bfs-provide (%)	random-ask (%)	dfs-ask (%)	bfs-ask (%)	Aggregated (%)	Central (%)
0	75	94	95	86	92	98	90	80
1–2	91	83	100	96	92	100	94	100
3–4	74	77	80	89	75	89	81	86
5–6	62	64	100	90	95	95	85	78

this mechanism was implemented within a commercial training and simulation system at BVR Systems LTD. BVR is focused on joint force, on-board training and distributed simulation systems. BVR has already planned realistic simulators for airplane cockpits, naval stations and ground forces. We propose a new application that builds upon BVR's existing simulators to simulate more complex group training missions by using the SharedPlan system.

Figure 12 depicts a high level architecture of the BVR's simulation engine and the SharedPlan system integration. As shown in this figure, the simulation engine includes Train Knowledge Base (TKB) with the geographical data about the training scenario and an exercise database with the initial data of the training exercise (e.g., agents types, agents' forces, their initial location, their initial mission). Unique to the SharedPlan system, an Agent Knowledge Base (AKB) is created containing properties about each agent (e.g. aircraft type, *max*, *min* velocity). In addition, it includes various types of groups (e.g., platoon, battalion) and their decomposition methods which describe possible ways of decomposing the groups into subgroups. Also, a Behavior Knowledge Base (BKB) is created containing a predefined set of basic actions that the agent can execute in the simulation and complex actions and their appropriate recipes. Agents' decisions are based on the dynamic and static knowledge that the agents gather from the simulation engine as well as the information in the AKB and the BKB. The Control component enables the human trainer to interact with the simulated arena and to influence the agents' behavior.

In studying the usefulness of the temporal reasoning mechanism in BVR's system, content experts defined real-life scenarios including fixed time points and temporal constraints on actions as may occur in practice. The temporal reasoning mechanism determines times for executing actions and coordinates the agents' activities. Messages have been sent between agents as described in Section 3.4. The information of fixed vertices is not exchanged thus saving the number of exchanged messages. The content experts defined a number of recipes for performing the same action in different world- states, including a default recipe which enabled a 100 % success rate. That is, in any state of the world, the agents were able to define a complete recipe tree for their joint action in which all of the constraints were satisfied. Hence, in a

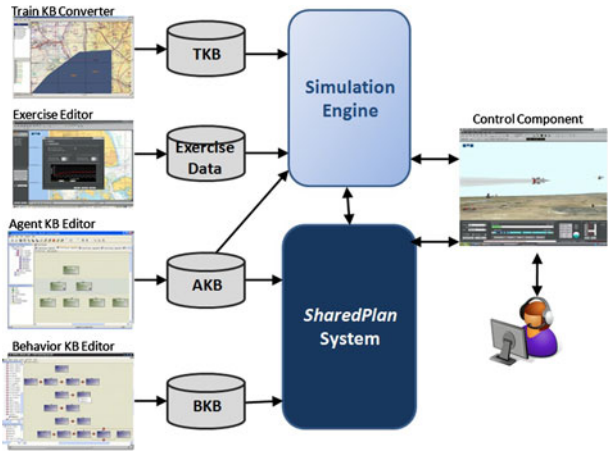

Fig. 12 A high level overview of the simulation system

case of failure, the agents can backtrack and re-plan actions by selecting alternative recipes until a complete plan is achieved.

Practically speaking, we applied scenarios involving fighter jets attempting to "*destroy the enemy target*". Each scenario involved a target that needed to be destroyed, as well as groups of attacking and defending planes. The attacking planes, which form the blue group, consist of bomber and fighter planes (e.g. F16 fighters and Stealth bombers), and the defending group consists exclusively of red fighter planes (F16). The goal of the blue fighters is to disable the enemy's red fighters, after which the blue bombers are able to destroy the ground target. The scenarios focused on different group sizes for the blue groups. A pictorial description of one scenario involving 8 blue and 4 red planes is given in Fig. 13. The scenario included several temporal constraints. For example, the blue bombers should reach the border 5 min after the blue fighters reachit. Using our mechanism, the bombers reason that they should leave the base 3 min. after the fighters. Hence, given that the fighters leave the base on 4:00 P.M, based on our mechanism, the bombers should leave the base at 4:03 P.M. Dynamic changes in this scenario may include unknown issues, such as

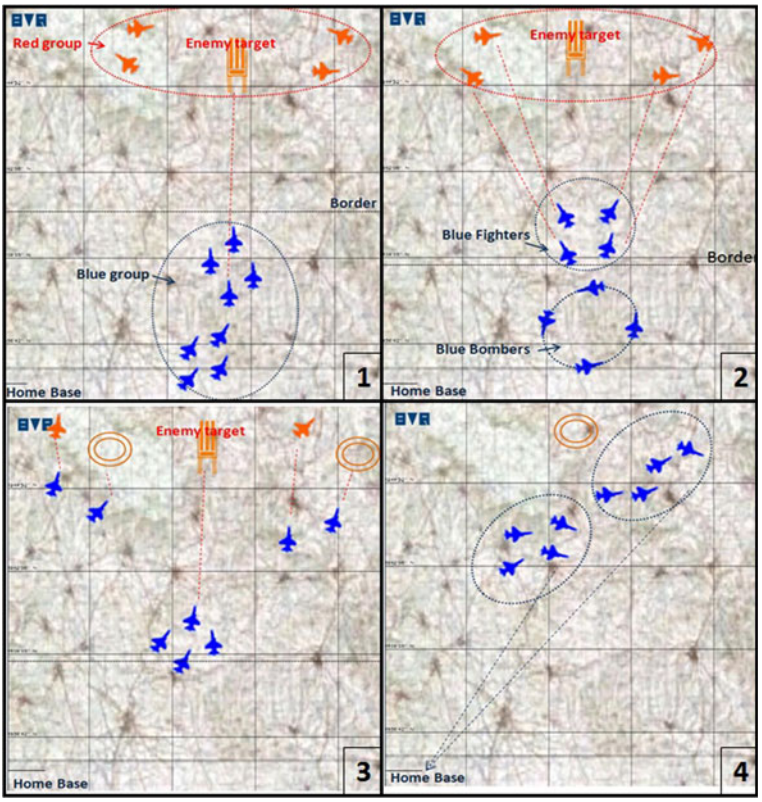

Fig. 13 Simulation view snapshots of "*destroy the enemy target*" action: *1* A blue group (8 agents) approaches the target area; *2* The blue group splits into two subgroups. One blue subgroup (4 fighters) approaches the red group and the second blue subgroup (4 bombers) waits; *3* The red fighters leave the area and the bombers destroy the target; *4* The blue group returns to home base

delays in the performance of actions which lead to time changes. For example, the fighters may encounter an unexpected enemy on their way to the border and arrive at the border later than planned. The bombers should update the times of leaving the base, accordingly.

In order to generate realistic plans for the "*destroy the enemy target*" action, we consulted with a group of professional fighter pilots whose expert knowledge was then directly encoded. We relied on these experts to provide details about how they would perform theoretical missions. We then encapsulated this information to form complex actions and a recipe library. In creating the recipes we utilized 241 existing predicates and 135 atomic actions of the simulation engine. To demonstrate the above scenario we created 103 recipes and 9 types of partitions of the planes into groups (e.g. pairs, triplets, etc.). The total number of complex actions was 79.

An example of a possible temporal constraints graph's implicit recipe tree, maintained by different members in the blue group demonstrated in Fig. 13, is given in Fig. 14. As shown in this figure, the recipe of the "*destroy the enemy target*" (α) action includes the subactions: "*setup*" (β_1), "*fly to border*" (β_2), "*attack target*" (β_3) and "*return to home base*" (β_4). The recipe is associated with precedence constraints $\{\beta_1$ before β_2; β_2 before β_3; β_3 before $\beta_4\}$. In addition, based on the experts' information, we defined four landmarks with fixed times, including: time to be at the border, finish time to disable the red fighters, time to finish waiting at the border, and time to leave the enemy area. Accordingly, R_α was associated with the following metric constraints: the agents must finish β_2 within 20 min. of starting α and β_4 must begin at 5:00 P.M. Also, it was defined that the bomber agents should start to destroy the ground target at 16:40 P.M. We used the mechanism to simulate groups of 2, 4, 8 and 12 blue planes which needed to destroy the target of the red group, consisting of 4

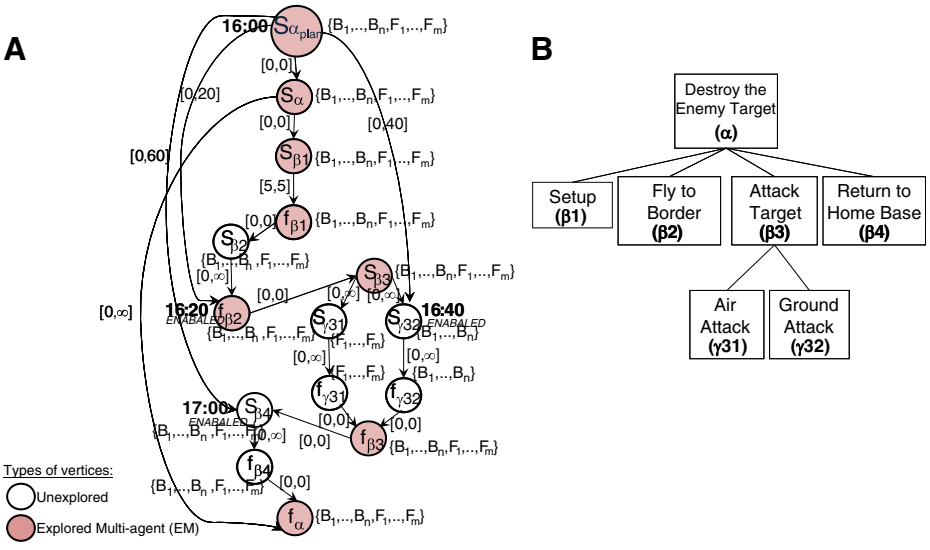

Fig. 14 **A** An example of a possible temporal constraints graph for the action "*destroy the enemy target*". The bomber agents are denoted by B_i $(1 \leq i \leq n)$ and the fighter agents are denoted by F_i $(1 \leq i \leq m)$; **B** The implicit recipe tree

Table 4 Comparing the number of coordination messages using the temporal reasoning mechanism versus other state-of-the art systems

No. of agents	No. of messages		
	BITE/STEAM	Our mechanism	Our mechanism with replanning
2	24	10	12
4	124	62	84
8	320	174	236
12	568	380	480

jets. Based on the results in Section 5.1 above, the agents planned their actions in the same order and used *bfs* planning order whenever it was possible. In addition, the agents used the *provide-time method* in order to save the number of the messages. In all the runs the agents achieved their mission under the temporal constraints with a 100 % success rate, which provided empirical evidence of the effectiveness of the mechanism in real-life scenarios.

We also explored the effectiveness of the mechanism by its ability to reduce the number of coordination messages between the group members. In particular, we compared it with other state-of-the art systems, based on teamwork models, in which the group members coordinate their activities via messages. We recorded the number of messages required to coordinate teamwork in our system, based on the temporal reasoning mechanism, in the blue group. The messages in our system are exchanged according to Section 3.4. We then compared it to the number of messages needed in the same set of problems as examined by previous teamwork approaches, BITE [41] and STEAM [82], where the group members maintain coordination by broadcasting messages. The number of necessary messages observed in previous approaches, are given in column 2 of Table 4. Column 3 describes the number of messages in our system. Nevertheless, in column 3 we assume that no dynamic changes appear in the environment. In column 4 we describe the number of messages needed when the environment includes dynamic changes and the agents had to communicate and change their temporal decisions 7 times. These changes were based on real-life changes. As Table 4 demonstrates, we found that the temporal mechanism yielded a significant[9] (p-value < 0.05) saving in the number of coordination messages between the group members, as well as when the environment includes dynamic changes.

Overall, the results in this section demonstrate the success of our approach in both a rescue domain and within a commercial real-world simulator. The results from the rescue domain show that self-interested planning agents within a group should commit to a schedule as late as possible. These results also demonstrate that if the group members perform their plan in the same order and they first plan the highest level actions, they then have a better chance of succeeding in their joint activity. We also present cases in which it is more efficient to distribute the planning among the group members than to solve the problem via a central planner. Additionally, we have integrated the mechanism within a commercial training and simulation application. These results not only present empirical evidence of its effectiveness in real-life scenarios, but also show their advantage over state-of-the-art implementation models

[9]We used the *two-tailed t-Test*.

for teamwork [41, 82] in reducing the number of messages needed to coordinate group members' activities by a very large amount (between 18 to 50 %). Thus, this mechanism should be considered to be the the state-of-the-art implementation for teamwork.

6 Related work

This work focuses on distributed planning with time constraints for self-interested agents. While this work is novel in how it addresses self-interested agents, distributed planning with time constraints has been well studied previously, in general terms. des-Jardins et al. [18] address two major approaches for the distributed planning problem. The first approach, called *Cooperative Distributed Planning (CDP)*, places emphasis on how to expand a plan in a distributed environment, where the process of formulating or executing a plan involves actions and interactions of multiple agents. One of the earliest versions of the CDP approach was introduced by Corkill [12]. Additional known versions of CDP include the COLLAGE planner which uses localization to partition the planning [46], the Distributed SIPE-2 planner (DSIPE) [89] and the Partial Global Planning (PGP) [20]. The second approach, called *Negotiated Distributed Planning (NDP)*, focuses on how to control and coordinate actions of multiple agents in a shared environment, where planning representations and algorithms are adopted accordingly. The purpose of agents who engage in CDP is different from the purpose of agents in NDP. Agents in CDP aim at executing the distributed parts of the plan in a coherent and effective way. Yet, agents who act according to NDP do not aim at forming a good collective plan but rather at ensuring that their local goals are achieved, when viewed in a global context.

To date, the leading approach for distributed planning that considers time constraints falls into the CDP category in which a group of multiple agents aims to maximize the overall utility accrued by the group. That is, group members exchange information regarding their plans, which are iteratively refined and revised until they fit together well with the objective to optimize a global plan. One example is the Distributed Temporal Planner (DTP) algorithm [86] that distributes over available processors the computation involved in finding a feasible solution to Temporal Plan Networks (TPNs). Another example is the multi-agent executive for scheduling temporal plans, named Chaski [74]. The most notable example is the COORDINATORS program [45] that aimed to create hand-held coordination assistants in order to enable military units to adapt mission plans more rapidly. The COORDINATORS program involved hierarchical distributed planning based on TAEMS framework [47, 72, 81] as well as methods for coordination and scheduling [3, 34, 50, 51, 77]. Nevertheless, the CDP approach, which offers coordination and scheduling methods, assumes that each agent is committed to activities that optimize the global plan—something that cannot be assumed in our problem.

In contrast, our work is based on the SharedPlan model and considers self-interested agents who plan their actions in a group [28]. The model supports the reasoning of a group member and the tradeoff between its own cost of performing activities in the context of the collaborative plan and its benefit from the success of the joint goal. While the idea of self-interested planning agents is new to this setting, self-interested group agents have been considered in other settings such as in cooperative

exploration [71]. Additionally, unlike the CDP- based approach in which agents are not inherently involved in the generation of collaborative plans, the SharedPlan model enables the agents to reach an agreement on the method to execute the joint action and various subactions, as well as who is going to perform the subactions in the plan. As the approach may be concerned with inter-agent negotiation it is typified by [18] as *Negotiated Distributed Planning (NDP)*. However, using this structure for resolving temporal constraints in distributed planning is new, and to the best of our knowledge has never been previously applied.

Several other works propose planning techniques for real-world environments, taking into account changes in the environment while executing a plan. Although they suggest an intelligent control system that can dynamically plan its own behavior, they neither consider temporal constraints nor teamwork. Examples of such works include M-SHOP [64], which is focused on domain-independent planning formalization and planning algorithms; the Zeno system [42], which suggests a method for building a decision-making mechanism for a planner in an uncertain environment; and the SGP contingent planning algorithm [87], which handles planning problems with uncertainty in initial conditions and with actions that combine causal and sensory effects. It also includes the planning model of the constraint-based EXCALIBURE planning system [62, 63] and so on. Unlike these works, our work does handle temporal constraints as well as teamwork activity.

Other planners, such as O-plan [14], ZENO [66], ParcPlan [22, 48], Cypress [88] and DCAPS [9] are able to handle temporal constraints. However, they do not interleave planning and execution and are not suitable for continual planning. In other works that combine planning and scheduling such as the Cypress [88], CASPER [10] and SGPlan4 [8] systems, the planner is not concerned with group planning processes that aim to facilitate self-interested teamwork agents.

Clement and Barrett [11] do consider self-interested agents that interleave planning and execution. They present the Shared Activity Coordination (SHAC) framework to provide a decentralized algorithm for negotiating the scheduling of shared activities over the lifetimes of multiple agents. Nevertheless, they suggest communication protocols for the agents but do not show any mechanism for collaborative planning as well as for temporal reasoning, as we do.

Our work uses *continual planning* in which planning and execution are interleaved through interaction between a Real-Time (RT) layer and an Artificial Intelligence (AI) layer as presented in previous systems. Examples of such systems include Miller and Gat's three-layer ATLANTIS system [54], Simmons' Task Control Architecture (TCA) [76] and the CIRCA system [60, 61]. While these systems do include separate RT and AI components which cooperate to achieve the overall desirable behavior, they are not concerned with group planning. In contrast, our work presents a new framework by utilizing the SharePlan model to implement an interaction between the AI components in order to achieve group joint plans.

Representing and reasoning about incomplete and indefinite qualitative temporal information is an essential part of many AI systems. Several formalisms for expressing and reasoning about temporal knowledge have been proposed, most notably Allen's interval algebra [1], Vilain and Kautz's point algebra [85] and Dean and McDermott's time map [15]. Each of these representation schemes is supported by a specialized constraint-directed reasoning algorithm. At the same time, extensive research has been carried out on *Constraint Satisfaction Problems* (CSPs) which

provide a powerful and efficient framework for describing state search problems. Some of these (e.g., [58]) have been extended to problems involving *temporal constraint satisfaction problems* (TCSPs) [4, 16] which are special cases of CSPs.

Our mechanism exploits the TCSP framework in order to resolve the temporal constraints in the system. For that reason, the performance of our system is strongly dependent upon the way that TCSP is generated and solved as a part of the Temporal Reasoning Algorithm. The general TCSP problem is intractable [16, inter alia]. In the experiments presented in this paper, we applied a *simple temporal problem* (STP) [16] and solved it by using Floyd-Warshall's all-pairs-shortest-paths algorithm [13].

Nevertheless, the performance of our system can be improved by applying more efficient and extended algorithms to resolve the STP as well as the general TCSP. For example, Demetrescu and Italiano [17] suggest an efficient algorithm for maintaining all-pairs-shortest-paths in directed graphs. Mohr and Henderson [56] present algorithms for arc and path consistency and show that the arc consistency algorithm is optimal in time complexity. Xu and Choueiry [90] improve the performance of the exponential-time backtrack search presented by [16]. Planken et al. [68] improve the algorithm proposed by Xu and Choueiry and suggest the P^3C algorithm for resolving STP. In [69] they propose the IPPC algorithm that maintains partial path consistency when new constraints are added or existing constraints are tightened. Shu et al. [75] utilize a temporal constraint network in order to provide an *Incremental Temporal Consistency* (ITC) algorithm for continuous planning. In [73] they suggest the ICA-TCSP algorithm which reduces the space necessary to encode the TCSP solution. Vidal et al. [59, 83, 84] extend classical temporal constraint networks to handle all the types of temporal constraints presented by Allen [2] in dynamic environments. While these works consider more efficient algorithms and a wider range of constraints than we do, they do not show how to combine them within a distributed planning environment.

Our work considers a problem in which multiple agents have to find a consistent set of actions under given constraints. This problem is naturally modeled as a *Distributed Constraint Optimization Problem* (DCOP) [35, 36, 49, 52, 55, 67]. In DCOP, variables and constraints are distributed among multiple agents. Solving a DCOP requires that agents not only solve their own constraints, but also coordinate the choice of their values with other agents in order to optimize a global objective function. The global objective function is modeled as a set of constraints, and each agent is familiar with the set of constraints for its variables. Previous work has proposed the DCOP for modeling a wide variety of multi-agent coordination problems such as distributed recourse allocation [79], distributed planning [81] and distributed scheduling [43]. In this work we exploit the SharedPlan model of cooperation to resolve temporal constraints in a distributed fashion. Thus, unlike DCOP approaches that optimize a global objective function, in our work we consider self-interested agents who belong to a group, as has been described above.

The BDI theoretical model of cooperation, which has many similarities with the approach we present, has been applied in previous teamwork systems [40, 41, 82]. Nonetheless, the key novelty of this work, and what differentiates our system from all other state-of-the-art BDI-based systems, is in its implementation. Namely, this work is unique in its ability to apply hierarchical abstraction planning and to incorporate these plans within its temporal reasoning mechanism where plans and temporal networks are built incrementally. Therefore, unlike other systems attributed to the

NDP approach, which either suggest broadcasting messages between the group members in order to maintain the full synchronization [41, 78, 82], or which propose that the team leader be responsible for the timing of the individual actions [39], our mechanism enables each agent to reason about its schedule individually, and thus reduces the necessity for communication. Furthermore, in former systems belonging to the NDP approach, a joint action is defined by explicitly describing all possible states in advance, so they do not keep the search space as we do. Practically, this difference facilitates a significant reduction in the number of messages that agents need to send—a claim that is supported by our result that agents using our mechanism sent far fewer messages (between 18 to 50 %) than those using other implementations.

In our work we focus on the integration of temporal constraints networks into multi-agent planning environments. More recent work makes use of temporal constraint networks in distributed dynamic environments yet does not include abstract joint plans and the BDI model of cooperation as does our work. Hunsberger [38] presents a mechanism whereby distributed agents can each solve different sections of a temporal network. While this framework is novel in that it allows for real-time execution of tasks and a novel mechanism allowing agents to partition the temporal network, it does not specify how it might generalize to planning self-interested agents with these types of constraints. Boerkoel and Durfee [5–7] exploit temporal constraints networks to find a joint schedule by distributed agents. They introduce the Multiagent Disjunctive Temporal Problem (MaDTP), a new distributed formulation of the widely-adopted Disjunctive Temporal Problem (DTP) [80] representation. Similarly, they focus on the scheduling problem but do not generalize it to teamwork with joint plans as we do.

7 Conclusion and future work

In this work we have presented a temporal reasoning mechanism for continual distributed planning by a group of self-interested agents. We have shown how to combine the BDI theoretical model of cooperation with hierarchical abstraction plans and temporal constraints networks. The reasoning mechanism exploits collaborative planning processes of the SharedPlan model [26] to enable each agent to decide about its schedule individually while interacting with the other group members. During the planning, each agent determines the duration and the time windows of the actions it has to perform and it takes into consideration the required temporal constraints of its activities and of collaborators. The times and the periods of the events that occur during the agents' activities need not be known in advance. In addition, the agents can expand their plans in a hierarchical manner without explicitly describing all possible states in advance. As a result, an agent may change its schedule if it identifies new plans and constraints due to changes in the environment, or if it receives new time constraints from other group members. Furthermore, if the agent determines that the course of action it has adopted is not successful, then it can revise its schedule of future actions.

We tested our approach in both a rescue domain and a commercial real-world simulator. The results from the rescue domain check the theoretical behavior of the self-interested planning agents acting within a group. We showed that such agents

should commit to a schedule as late as possible and that if the group members perform their plan in the same order and they first plan the highest level actions, they then have a better chance of succeeding in their joint activity. We also presented cases in which it is more efficient to distribute the planning among the group members than to solve the problem via a central planner. Additionally, we have integrated the mechanism within a commercial training and simulation application. These results not only present empirical evidence of its effectiveness in real-life scenarios, but also show their advantage over existing teamwork models [41, 82] in reducing the number of messages needed to coordinate group members' activities by a very large amount.

For future work, we hope to consider how the results presented may be improved. One option would be to change the heuristic algorithm used by the RTS component. Another improvement may be achieved by enabling the agents to estimate the amount of time needed by others to perform their actions, as is usually done in distributed planning among people. This may result in a significant savings in the amount of communication and negotiation. Such an estimation could be based, for example, on past performance.

Appendix A: Proofs of lemmas and propositions for the correctness of the temporal reasoning algorithm

Proposition 4.1 *Suppose that the AIP component of an agent A_k runs the Temporal Reasoning Algorithm which builds the Temporal Constraints Graph $G_\alpha^k = (V_\alpha^k, E_\alpha^k)$. Let v be a vertex in V_α^k and S be the set of all minimal members of the set $\{u | u$ is a fixed vertex and u precedes $v\}$. Then, during the building of the Temporal Constraints Graph, if each vertex in the path from all of the vertices in S to v are* EXPLORED *(but not ENPW) and then v is an* ENABLED.

Proof The proof is by induction on the length of the longest path from $u \in S$ to v (u is a minimal member of S in a partial order if $u \in S$ and no other $w \in S$ exists such that w precedes u). Let v be a vertex such that the longest path from all of the vertices in S to v are *EXPLORED* but not ENPW. Initially, when the length of the longest path is equal to zero, the vertex v is a fixed time point and the proposition certainly holds. Also, when the length of the longest path is equal to 1, the proposition certainly holds (as all of the vertices that precede v are fixed). Suppose that v is not a fixed vertex and let S' be the set of all minimal members of the set $\{u' | u'$ is a fixed vertex and u' precedes $u\}$. Then, for each $(u, v) \in E_\alpha$, the longest path from the minimal members of the set S' to u is shorter than the longest path from the vertices in S to v. Therefore, according to the inductive hypothesis, if each vertex in the path from all of the vertices in S' to u are *EXPLORED* (but not ENPW), then u is an *ENABLED* vertex. Now, it remains to be shown that the weight of each edge $(u, v) \in E_\alpha$ is known. But according to the algorithm, the weight of an edge $(u, v) \in E_\alpha$ becomes known if the values of all of the vertices in the path from u to v are known. These values are known if the vertex is fixed or if the plan of the action which is associated with the vertex is completed. Suppose, by contradiction, that there is at least one vertex whose value is unknown to A_k. But if A_k participates in the performance of the action which is associated with this vertex, then during the building of the graph,

A_k changes the statues of v and u to *EXPLORED* only after it adds *UNEXPLORED* vertices between v and u. Thus, there is a path of *UNEXPLORED* vertices between u and v. If A_k does not participate in the performance of this action, the vertex becomes ENPT only if the time value of this vertex is known (i.e., the vertex is fixed). Consequently, the weight of the edge from S to v is known and v is therefore an *ENABLED* vertex. □

Lemma 4.2 *Suppose that a group of agents $\mathcal{A}_\alpha$ plans action α. Let T_α^k be the implicit recipe tree of G_α^k and let T_α be the union implicit recipe trees. Then, during the development of the graph G_α^k by each $A_k \in \mathcal{A}_\alpha$, performing (possibly in parallel) all of the actions (possibly complex) in the union implicit recipe trees, T_α, is consistent with α's temporal constraints.*

Proof The proof is by induction from the number of the vertices in the implicit recipe tree of an individual agent. Initially, when the recipe-tree consists of one vertex (i.e., only action α), the proposition certainly holds (see the initialization phase in Fig. 6). Suppose that the implicit recipe tree of A_k consists of n vertices. As an inductive hypothesis we assume that all of the actions in the leaves of the recipe tree with m vertices, where $m < n$, are consistent with α's temporal constraints. Let T_α^k be a partial implicit recipe tree, planned by A_k for α, with m vertices. According to the algorithm A_k expands its recipe tree by selecting a leaf which represents a complex level action β (lines 25 and 27). Suppose that A_k selected the recipe R_β in order to expand the recipe tree for α which consists of b subactions. According to the algorithm (see Fig. 19), when A_k selects the recipe for β and adds it to the constraints graph, it also checks that all of the constraints of β are consistent with temporal constraints of α (see Fig. 20). If these constraints are consistent with the temporal constraints of α (line 7 in Fig. 19), it continues with its plan for α and the subactions of β become the leaves of the recipe tree of α (the "while" loop in Fig. 19). Thus, all of the actions in the leaves of the tree which consists of $n = m + b$ vertices are consistent with α's temporal constraints.

Since, according to Proposition 4.3, the leaves of the union implicit recipe trees, T_α, are leaves of the implicit recipe trees of the appropriate agents, performing the actions in the leaves of these implicit recipe trees is consistent with α's temporal constraints. □

Appendix B: Temporal reasoning mechanism procedures

```
FLOYD_WARSHALL(M, n) [Input:the n x n distance matrix M]
1   M^(0) ← M
2   for k ← 1 to n do
3       for i ← 1 to n do
4           for j ← 1 to n do
5               m_{i,j}^(k) ← min(m_{i,j}^(k−1), m_{i,k}^(k−1) + m_{k,j}^(k−1));
6   for i ← 1 to n do
7       if m_{i,i}^(n) < 0 then return false [Output: Not feasible];
8       else, return M^(n); [Output: The shortest distance matrix M^(n)];
```

Fig. 15 Floyd-Warshall's algorithm

```
CHECK_CONSISTENCY(G_α)
    Assume edge (u, v) has weight range [first(weight(u, v)), second(weight(u, v))]
1   M ← array[|V_α|x|V_α|];
2   for i ← 1 to |V_α| do
3       for j ← 1 to |V_α| do
4           M[i,j] ← ∞;
5   for each (v, u) ∈ E_α
6       M[u,v] ← (second(weight(u, v));
7       M[v,u] ← (-first(weight(u, v));
8   if not FLOYD_WARSHALL(M, |V_α|) return false;
9   else, for each (v, u) ∈ E_α
10      (second(weight(u, v)) ← M[u,v];
11      (-first(weight(v, u)) ← M[v,u];
12      return true;
```

Fig. 16 CHECK_CONSISTENCY procedure finds whether G_α is consistent

```
UPDATE_ENABLED_SET(u)
1   E ← E ∪ u; U ← U − u;
2   for each vertex (u, v) ∈ E_α do
3     bDoesBecomeEnable ← True;
4     for each vertex (a, v) ∈ E_α do
5       if (a ∉ E and a is a fix vertex) then E ← E ∪ a;
6       if a ∉ E then bDoesBecomeEnable ← False;
7     if bDoesBecomeEnable == True then
8       if status[v] == UNEXPLORED then U ← U ∪ v;
9         else, if status[v] ≠ ENPW then UPDATE_ENABLED_SETS(v);
```

Fig. 17 The procedure UPDATE_ENABLED_SET updates the sets of the *ENABLED* vertices. As an input this procedure receives a vertex u for which A_k has either completed identifying its time (if it is a basic vertex) or completed identifying a preliminary plan for it (if it is a complex vertex) or it has received its time values from another agent (if A_k does not participate in the performance of the action which is associated with this vertex). First it adds vertex u to the $\mathcal{E}$ set. Then, for each vertex v adjacent to u (i.e., $(u, v) \in E$) it checks if all of the vertices which enter v (i.e., for each vertex a such that $(a, v) \in E$) belong to $\mathcal{E}$ (lines 4–6). If so, it checks if the status of v is an *UNEXPLORED* vertex. If so, it adds v to $\mathcal{U}$ (lines 7–8); otherwise (if the status of v is *EXPLORED* but is not *ENPW*) the procedure runs again recursively on v (line 9)

```
SELECT_AGREEABLE_RECIPE(β,A_β)
1   bIsAgreementAchieved ← OBTAIN_AGREEMENT("A_k PLANNING β", A_β);
2   if (bIsAgreementAchieved) then:
3     if bIsConsistent ← EXPAND_TEMPORAL_GRAPH(β, EM) then:
4       if bIsAgreementAchieved ← OBTAIN_AGREEMENT("SELECTED R_β",A_β) then:
5         if bIsAgreementAchieved ← OBTAIN_AGREEMENT("ASSIGNMENT TO SUBACTIONS OF R_β",A_β) then:
6           each member A_i ∈ A_β ADD_SELECTED_RECIPE_TO_GRAPH(R_β, EM);
7   return bIsAgreementAchieved ∧ bIsConsistent;
```

Fig. 18 This procedure is based on the planning processes of the SharedPlan model. That is, the procedure includes the process for selecting a recipe by the group $\mathcal{A}_\beta$ and the process for the assignment of the members to subactions. In line **1** A_k informs all other members in $\mathcal{A}_\beta$ about its intention to plan β and receives answers from all of the other members in $\mathcal{A}_\beta$. If there is no member in $\mathcal{A}_\beta$ who objects, then A_k tries to plan β. In line **3** A_k tries to plan β according to Fig. 19. If A_k succeeds in finding a feasible plan for β (bIsConsistent is true), then, A_k sends the information about the selected recipe, R_β, to the other participants and they need to reach an agreement regarding the selected recipe (line **4**) and about their assignment to the subactions of R_β (line **5**). If the other members agree about the selected recipe and their assignment to the subactions, then each of them adds the selected recipe to its graph (line **6**) according to Fig. 20. If some member in $\mathcal{A}_\beta$ objects to A_k or A_k does not find an appropriated recipe in its library, then the procedure returns a failure

```
EXPAND_TEMPORAL_GRAPH(β, vSTATUS)
1  bIsRecipeConsistent ← FALSE; ℜ_{β_tried} ← ∅;
2  while (not bIsRecipeConsistent)∧(ℜ_β − ℜ_{β_tried} ≠ ∅)
3     Select_Rec R_β for β not in R_{β_tried};
4     add R_β to ℜ_{β_tried};
5     if bIsRecipeConsistent ← ADD_SELECTED_RECIPE_TO_GRAPH(R_β, vSTATUS) then:
6        UPDATE_ENABLED_SET(s_β);
7        else, G^k_α is not consistent, REMOVE(R_β);
8  end while;
9  return bIsRecipeConsistent;
```

Fig. 19 Planning the Temporal Constraints Graph according to a complex level action β. $\Re_\beta$ denotes the set of recipes for β. The REMOVE procedure removes the vertices and edges which are associated with the subactions of R_β from G^k_α

```
ADD_SELECTED_RECIPE_TO_GRAPH(R_β, vSTATUS)
1  G^p_β ← CONSTRUCT_PRECEDENCE_GRAPH(R_β);
2  ADD_PRECEDENCE_GRAPH((s_β, f_β), G^p_β);
3  ADD_METRIC_CONSTRAINTS(R_β);
4  if bIsRecipeConsistent ← CHECK_CONSISTENCY(G^k_α) then:
5     status[s_β] ← vStatus; status[f_β] ← vStatus;
6  return bIsRecipeConsistent;
```

Fig. 20 Add relevant vertices and edges according to the selected recipe R_β

```
CONSTRUCT_PRECEDENCE_GRAPH(R_β)
1  V^p_β ← ∅; E^p_β ← ∅;
2  for each subaction γ_i ∈ R_β
3        do E^p_β ← E^p_β ∪(s_{γ_i}, f_{γ_i}); V^p_β ← V^p_β ∪ {s_{γ_i}, f_{γ_i}}; weight(s_{γ_i}, f_{γ_i}) ← [0, ∞];
4  for each precedence constraint γ_i < γ_j ∈ R_β
5        do E^p_β ← E^p_β ∪(f_{γ_i}, s_{γ_j}); weight(f_{γ_i}, s_{γ_j}) ← [0, ∞];
```

Fig. 21 The algorithm for constructing a precedence constraint graph for a given recipe R_α

```
ADD_PRECEDENCE_GRAPH ((s_β, f_β),G^p_β)
1  V^k_α ← V^k_α ∪ V^p_β;
2  E^k_α ← E^k_α ∪ E^p_β;
3  for each initial vertex u_b ∈ V^p_β do
4        E^k_α ← E^k_α ∪ (s_β, u_b); weight(s_β, u_b) ← [0, ∞];
5  for each terminal vertex u_e ∈ V^p_β do
6        E^k_α ← E^k_α ∪ (u_e, f_β); weight(u_e, f_β) ← [0, ∞];
```

Fig. 22 The ADD_PRECEDENCE_GRAPH PROCEDURE which incorporates the Precedence Graph G^p_β into G^k_α

```
ADD_METRIC_CONSTRAINTS(R_β)
1  for each metric constraint (a_{i,j} ≤ u_j − v_i ≤ b_{i,j}) in R_β do
2        if (v_i, u_j) ∉ E^k_α then E^k_α ← E^k_α ∪ (v_i, u_j); weight(v_i, u_j) ← [a_{i,j}, b_{i,j}];
3        else, weight(v_i, u_j) ← [max(a_{i,j}, first[weight(v_i, u_j)]), min(b_{i,j}, second[weight(v_i, u_j)])];
4  for each subaction γ_i ∈ R_β do
5        for each metric constraint (a_{i,j} ≤ u_i − v_j ≤ b_{i,j}) in R_{γ_i} do
6              if (v_i, u_j) ∉ E^k_α then E^k_α ← E^k_α ∪ (u_i, v_j); weight(u_i, v_j) ← [c_{j,i}, c_{i,j}];
7              else, weight(v_i, u_j) ← [max(a_{i,j}, first[weight(v_i, u_j)]), min(b_{i,j}, second[weight(v_i, u_j)])];
8  for each subaction γ_i ∈ R_β add the appropriate metric constraints;
```

Fig. 23 The ADD_METRIC_CONSTRAINTS procedure for incorporating the metric temporal constraints of R_β into G^k_α

```
IDENTIFY_TIMES_OF_BASIC_ACT(β)
1  status[s_β] ← ES; status[f_β] ← ES;
2  UPDATE_ENABLED_SET(s_β)
3  UPDATE_ENABLED_SET(f_β)
4  D_β ← time_period(β);
5  r_β ← time(s_{α_plan}) + M(s_{α_plan}, s_β);
6  d_β ← time(s_{α_plan}) + M(s_{α_plan}, f_β);
7  p_β ← FIND_PRECEDENCE_ACTIONS(β, G^k_α);
8  transfer action β and ⟨D_β, d_β, r_β, p_β⟩ to RTS component
   which tries schedule β and informs "succeed" or "fail";
```

Fig. 24 Identification of the temporal values variables $\langle D_\beta, d_\beta, r_\beta, p_\beta \rangle$ of basic level action β. Remark: the procedure FIND_PRECEDENCE_ACTIONS (β, G^k_α) finds all basic actions preceding β

```
CHECK_NECESSITY_TO_UPDATE_MEMBERS(β)
1  β_parent ← the parent of β in implicit recipe of G^k_α;
2  if A_k ≠ A_{β_parent} then:
3     bPreOfMulti ← False;
4     for each action β_i such that (f_β, s_{β_i}) ∈ E^k_α do:
5        if A_{β_i} ⊈ A_β then:
6           bPreOfMulti ← True;
7           H[β_i] ← UPDATE_HINDER_MEMBERS(β_i);
8     if not bPreOfMulti then: subactions_of_multi_recipe ← subactions_of_multi_recipe ∪ β;
9  else, A_k == A_{β_parent} then:
10    if the plan of β_parent is completed then:
11       CHECK_NECESSITY_TO_UPDATE_MEMBERS (β_parent)
```

Fig. 25 The CHECK_NECESSITY_TO_UPDATE_MEMBERS procedure for determining whether β is a subaction in a recipe of multi-agent action and whether β is in a set of the hinder members of some action β_i (i.e., $\beta \in H(\beta_i)$, see Definition 3.4). If $\beta \in H(\beta_i)$, the procedure checks if all other members in $H(\beta_i)$ are *EXPLORED*. If so, the temporal information of β can be sent to the performers of β_i

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**, 832–843 (1983)
2. Allen, J.F.: Towards a general theory of action and time. Artificial Intelligence Journal **23**(2), 123–144 (1984)
3. Barbulescu, L., Rubinstein, Z.B., Smith, S.F., Zimmerman, T.L.: Distributed coordination of mobile agent teams: the advantage of planning ahead. In: AAMAS, pp. 1331–1338 (2010)
4. van Beek, P.: Reasoning about qualitative temporal information. Artificial Intelligence Journal **58**(1):297–326 (1992)
5. Boerkoel, J.C. Jr., Durfee, E.H.: Evaluating hybrid constraint tightening for scheduling agents. In: AAMAS, pp. 673–680 (2009)
6. Boerkoel, J.C. Jr., Durfee, E.H.: Distributed algorithms for solving the multiagent temporal decoupling problem. In: AAMAS, pp. 141–148 (2011)
7. Boerkoel, J.C. Jr., Durfee, E.H.: A distributed approach to summarizing spaces of multiagent schedules. In: AAAI, pp. 1742–1748 (2012)
8. Chen, Y., Wah, B.W., Hsu, C.W.: Temporal planning using subgoal partitioning and resolution in SGPlan. Journal of Artificial Intelligence Research **26**, 323–369 (2006)
9. Chien, S., Rabideau, G., Willis, J., Mann, T.: Automating planning and scheduling of shuttle payload operations. Artificial Intelligence Journal **114**(1), 239–255 (1999)
10. Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., Tran, D.: ASPEN—automating space mission operations using automated planning and scheduling. In: Space Ops, pp. 1–10 (2000)
11. Clement, B.J., Barrett, A.C.: Continual coordination through shared activities. In: AAMAS, pp. 57–64 (2003)
12. Corkill, D.: Hierarchical planning in a distributed environment. In: IJCAI, pp. 168–175 (1979)

13. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, London (2001)
14. Currie, K., Tate, A.: O-Plan: the open planning architecture. Artificial Intelligence Journal **52**(1), 49–86 (1991)
15. Dean, T.L., McDermott, D.V.: Temporal data base management. Artificial Intelligence Journal **32**(1), 1–55 (1987)
16. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence Journal **49**(1), 61–95 (1991)
17. Demetrescu, C., Italiano, G.F.: Experimental analysis of dynamic all pairs shortest path algorithms. ACM Trans. Algor. **2**(4), 578–601 (2006)
18. desJardins, M., Durfee, E.H., Ortiz, C., Wolverton, M.: A survey of research in distributed continual planning. AI Mag **1**(4), 13–22 (1999)
19. Dudek, G., Jenkin, M.R.M., Milios, E., Wilkes, D.: A taxonomy for multi-agent robotics. Auton. Robot. **3**(4), 375–397 (1996)
20. Durfee, E.H., Lesser, V.R.: Partial global planning: a coordination framework for distributed hypothesis formation. IEEE Trans. Syst. Man Cybern. **21**(5), 1167–1183 (1991)
21. Durfee, E.H.: Distributed problem solving and planning. In: Weiss, G. (ed.) Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence, pp. 121–164. MIT Press, Cambridge, MA (1999)
22. El-Kholy, A., Richards, B.: Temporal and resource reasoning in planning: the parcPlan approach. In: ECAI, pp. 614–618 (1996)
23. Erol, K., Nau, D., Hendler, J.: HTN planning: complexity and expressivity. In: AAAI, pp. 1123–1128 (1994)
24. Garey, M.R., Johnson, D.S.: Computers and Intractability: a Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco, CA (1979)
25. Ghallab, M., Nau, D., Traverso, P.: *Automated Planning: Theory & Practice*. Morgan Kaufmann, San Francisco, CA (2004)
26. Grosz, B.J., Kraus, S.: Collaborative plans for complex group action. Artificial Intelligence Journal **86**(2), 269–357 (1996)
27. Grosz, B.J., Kraus, S.: The evolution of SharedPlans. In: Rao, A., Wooldridge, M. (eds.) Foundations and Theories of Rational Agency, pp. 227–262. Academic, Boston, MA (1999)
28. Grosz, B.J., Hunsberger, L., Kraus, S.: Planning and acting together. AI Mag **20**(4), 23–34 (1999)
29. Hadad, M., Kraus, S.: Sharedplans in electronic commerce. In: Klusch, M. (ed.) Intelligent Information Agents, pp. 204–231. Springer, New York (1999)
30. Hadad, M., Kraus, S.: A mechanism for temporal reasoning by collaborative agents. In: CIA, pp. 229–234 (2001)
31. Hadad, M., Kraus, S.: Exchanging and combining temporal information by collaborative agents. In: CIA, pp. 279–286 (2002)
32. Hadad, M., Rosenfeld, A.: Adapt: abstraction hierarchies to better simulate teamwork under dynamics. In: Agents for Educational Games and Simulations, pp. 166–182 (2012)
33. Hadad, M., Kraus, S., Gal, Y., Lin, R.: Time reasoning for a collaborative planning agent in a dynamic environment. Ann. Math. Artif. Intell. **37**(4), 331–380 (2003)
34. Harbers, T., Maheswaran, R.T., Szekely, P.: Centralized, distributed or something else? making timely decisions in multi-agent systems. In: AAAI, p. 738 (2007)
35. Hirayama, K., Yokoo, M.: Distributed partial constraint satisfaction problem. In: Principles and Practice of Constraint Programming, pp. 222–236 (1997)
36. Hirayama, K., Yokoo, M.: An approach to over-constrained distributed constraint satisfaction problems: distributed hierarchical constraint satisfaction. In: AAMAS, pp. 135–142 (2000)
37. Horling, B., Lesser, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., Decker, K., Garvey, A.: The TAEMS White Paper. Multi-Agent Systems Lab University of Massachusetts (1999)
38. Hunsberger, L.: Distributing the control of a temporal network among multiple agents. In: AAMAS, pp. 899–906 (2003)
39. Jennings, N.R.: Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. Artificial Intelligence Journal **75**(2), 1–46 (1995)
40. Kamar, E., Gal, Y., Grosz, B.: Incorporating helpful behavior into collaborative planning. In: AAMAS, pp. 875–882 (2009)
41. Kaminka, G.A., Frenkel, I.: Integration of coordination mechanisms in the BITE multi-robot architecture. In: ICRA, pp. 2859–2866 (2007)
42. Karacapilidis, N.I.: Planning under uncertainty: a qualitative approach. In: EPIA, pp. 285–296 (1995)

43. Kim, Y., Krainin, M., Lesser, V.: Effective variants of the max-sum algorithm for radar coordination and scheduling. In: Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, pp. 357–364 (2011)
44. Kitano, H.: Robocup rescue: A grand challenge for multi-agent systems. In: ICMAS, Boston, MA, pp. 5–12 (2000)
45. Kohout, B.: The DARPA COORDINATORS program: a retrospective. In: CTS, pp. 342–342 (2011)
46. Lansky, A., Getoor, L.: Scope and abstraction: two criteria for localized planning. In: IJCAI, pp. 1612–1619 (1995)
47. Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., Nagendra Prasad, M., Raja, A., Vincent, R., Xuan, P., Zhang, X.Q.: Evolution of the GPGP/TEAMS domain independent coordination framework. In: AAMAS, pp. 87–143 (2004)
48. Lever, J., Richards, B.: parcPlan: a planning architecture with parallel actions, resources and constraints. In: Methodologies for Intelligent Systems, pp. 213–222 (1994)
49. Liu, J.S., Sycara, K.: Exploiting problem structure for distributed constraint optimization. In: ICMAS, pp. 246–253 (1995)
50. Maheswaran, R.T., Szekely, P.: Criticality metrics for distributed plan and schedule management. In: ICAPS, vol. 2, p. 2 (2008)
51. Maheswaran, R., Rogers, C.M., Sanchez, R., Szekely, P., Gati, G., Smyth, K., VanBuskirk, C.: Multi-agent systems for the real world. In: AAMAS, pp. 1281–1282 (2009)
52. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: AAMAS, pp. 438–445 (2004)
53. Mailler, R., Lesser, V.: Using cooperative mediation to solve distributed constraint satisfaction problems. In: AAMAS, pp. 446–453 (2004)
54. Miller, D.P., Gat, E.: Exploiting known topologies to navigate with low-computation sensing. In: Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 1383, pp. 425–435 (1991)
55. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence Journal **161**(1), 149–180 (2005)
56. Mohr, R., Henderson, T.C.: Arc and path consistency revisited. Artificial Intelligence Journal **28**(2), 225–233 (1986)
57. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining computational complexity from characteristic 'phase transitions'. Nature **400**(6740), 133–137 (1999)
58. Montanari, U.: Networks of constraints: fundamental properties and applications to picture processing. Inf. Sci. **7**, 95–132 (1974)
59. Morris, P., Muscettola, N., Vidal, T., et al.: Dynamic control of plans with temporal uncertainty. In: IJCAI, pp. 494–502 (2001)
60. Musliner, D.J., Dufree, E.H., Shin, K.G.: CIRCA: a cooperative intelligent real-time control architecture. IEEE Trans. Comput. **23**(6), 1561–1574 (1993)
61. Musliner, D.J., Dufree, E.H., Shin, K.G.: World modeling for dynamic construction of real-time control plans. Artificial Intelligence Journal **74**(1), 83–127 (1995)
62. Nareyek, A.: A planning model for agents in dynamic and uncertain real-time environments. In: AIPS Workshop on Integrating Planning, pp. 7–14 (1998)
63. Nareyek, A.: Open world planning as scsp. In: AAAI Workshop on Constraints and AI Planning, pp. 35–46 (2000)
64. Nau, D., Cao, Y., Lotem, A., Muñoz-Avila, H.: SHOP and M-SHOP: planning with ordered task decomposition. Technical report, University of Maryland (2000)
65. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: an HTN planning system. Journal of Artificial Intelligence Research **20**, 379–404 (2003)
66. Penberthy, J.S., Weld, D.: Temporal planning with continuous change. In: AAAI, pp. 1010–1015 (1994)
67. Petcu, A.: A class of algorithms for distributed constraint optimization. Phd. thesis no. 3942, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland (2007)
68. Planken, L., De Weerdt, M., van der Krogt, R., Rintanen, J., Nebel, B., Beck, J.C., Hansen, E.: P3c: a new algorithm for the simple temporal problem. In: ICAPS, pp. 256–263 (2008)
69. Planken, L.R., de Weerdt, M.M., Yorke-Smith, N.: Incrementally solving stns by enforcing partial path consistency. In: ICAPS, pp. 129–136 (2010)
70. Pynadath, D.V., Tambe, M.: The communicative multiagent team decision problem: analyzing teamwork theories and models. Journal of Artificial Intelligence Research **16**, 389–423 (2002)

71. Rochlin, I., Sarne, D., Laifenfeld, M.: Coordinated exploration with a shared goal in costly environments. In: ECAI, pp. 690–695 (2012)
72. Sarne, D., Grosz, B.J.: Determining the value of information for collaborative multi-agent planning. Auton. Agent. Multi-Agent Syst. **26**(3), 456–496 (2013)
73. Shah, J.A., Williams, B.C.: Fast dynamic scheduling of disjunctive temporal constraint networks through incremental compilation. In: ICAPS, pp. 322–329 (2008)
74. Shah, J.A., Conrad, P.R., Williams, B.C.: Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In: ICAPS, pp. 289–296 (2009)
75. Shu, I., Effinger, R., Williams, B.: Enabling fast flexible planning through incremental temporal reasoning with conflict extraction. In: ICAPS, pp. 252–261 (2005)
76. Simmons, R.: An architecture for coordinating planning, sensing, and action. In: Procs. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control, pp. 292–297 (1990)
77. Smith, S.F., Gallagher, A., Zimmerman, T.: Distributed management of flexible times schedules. In: AAMAS, p. 74 (2007)
78. Sonenberg, E., Tidhar, G., Werner, E., Kinny, D., Ljungberg, M., Rao, A.: Planned team activity. Technical Report 26, Australian Artificial Intelligence Institute, Australia (1992)
79. Stefanovich, N., Farinelli, A., Rogers, A., Jennings, N.R.: Resource-aware junction trees for efficient multi-agent coordination. In: AAMAS, pp. 363–370 (2011)
80. Stergiou, K., Koubarakis, M.: Backtracking algorithms for disjunctions of temporal constraints. Artificial Intelligence Journal **120**(1), 81–117 (2000)
81. Sultanik, E., Modi, P.J., Regli, W.C.: On modeling multiagent task scheduling as a distributed constraint optimization problem. In: IJCAI, pp. 1531–1536 (2007)
82. Tambe, M.: Toward flexible teamwork. Journal of Artificial Intelligence Research **7**, 83–124 (1997)
83. Vidal, T., Fargier, H.: Handling contingency in temporal constraint networks: from consistency to controllabilities. J. Exp. Theor. Artif. Intell. **11**, 23–45 (1999)
84. Vidal, T., Ghallab, M.: Dealing with uncertain durations in temporal constraint networks dedicated to planning. In: ECAI, pp. 48–52 (1996)
85. Vilain, M., Kautz, H.A.: Constraint propagation algorithms for temporal reasoning. In: AAAI, pp. 132–144 (1986)
86. Wehowsky, A., Block, S., Williams, B.: Robust distributed coordination of heterogeneous robots through temporal plan networks. In: ICAPS Workshop on Multiagent Planning and Scheduling, pp. 67–72 (2005)
87. Weld, D., Anderson, C., Smith, D.: Extending graphplan to handle uncertainty and sensing actions. In: AAAI, pp. 897–904 (1998)
88. Wilkins, D.E., Myers, K.L., Lowrance, J.D., Wesley, L.P.: Planning and reacting in uncertain and dynamic environments. J. Exp. Theor. Artif. Intell. **7**(1), 197–227 (1995)
89. Wolverton, M., desJardins, M.: Controlling communication in distributed planning using irrelevance reasoning. In: AAAI, pp. 868–874 (1998)
90. Xu, L., Choueiry, B.: Improving backtrack search for solving the tcsp. In: Principles and Practice of Constraint Programming, pp. 754–768 (2003)
91. Yokoo, M.: Distributed Constraint Statification. Springer, Germany (2001)