Learning to communicate more efficiently in human-agent teams

Yuqing Tang
Graduate Center
City University of New York
New York, NY 10016, USA
ytang@cs.gc.cuny.edu

David Emele
Dept of Computing Science
The University of Aberdeen
Aberdeen, AB24 3UE, UK
c.emele@abdn.ac.uk

Timothy J. Norman
Dept of Computing Science
The University of Aberdeen
Aberdeen, AB24 3UE, UK
t.j.norman@abdn.ac.uk

Simon Parsons
Brooklyn College,
City University of New York
Brooklyn, NY 11210 USA
parsons@sci.brooklyn.cuny.edu

Abstract—**We are developing software agents that can help support the work of human teams. In order to be useful, these agents must be able to adapt their behavior, learning over time how to better serve the human members of the teams. In previous work, we have demonstrated how agents can use techniques from machine learning to improve the performance of agents that identify organisational policy contraints of coalition partners. We have also identified mechanisms that agents can use for planning the communication that supports team activities. In this paper we describe how to combine these lines of work, showing how agents can use learning to improve their performance, both to identify more efficient communication patterns, and to construct communication policies more efficiently.**

I. Introduction

This paper is concerned with managing collaboration in a team [1]. In particular, we are interested in teams engaged in military missions, and teams in which members may come from different parts of an international coalition. In such situations, effective coordination can be problematic, with units unable to communicate easily, and handicapped by having been trained to operate under rather different doctrines. Software agents can support effective collaboration in teams, and can overcome some of the problems with coalition forces. In particular, agents can filter messages [13]; coordinate activities [6], ensure timely delivery of crucial data [17]; and enforce the correct protocol for team behavior, [11], [12].

In our work, we are particularly interested in how software agents can help in the planning and execution of missions. Previously, we have described how to manage appropriate communication between agents during plan execution [18], and how to integrate this communication into team plans [19], developing a mechanism for simultaneously constructing plans for a team and the communication necessary to carry out the plan.

In addition, we are interested in the question of how software agents can improve their performance over time. In any collaborative environment, we can do better once we are used to our teammates' behavior — we can anticipate their actions and needs and modify our own behavior to better fit with theirs. In [10], [9] we have been investigating how software agents can learn the policy constraints that apply to their teammates, showing that techniques from machine learning can significantly improve performance and, in the context of negotiation, reduce the time taken to reach agreement.

In this paper we look at combining these two lines of work, considering how learning can be incorporated into planning for teams. The paper is structured as follows. Section II describes our previous work on team planning and introduces the formal model that is used in the paper. Section III describes our previous work on learning policy constraints. Section IV then describes how the formal model is extended to incorporate dealing with policy constraints, and we show how these can be integrated into the planning process, making it possible to use our prior work on machine learning in this context. Finally, Section V concludes.

II. Multiagent Systems and Dialogues on Symbolic Techniques

This work uses a state-space model as a basis for the formalisation. This model is an adaptation of a model commonly used in non-deterministic planning [7]. *States* are objects that capture some aspect of a system, and *actions* are transitions between states. States and actions together define a *state-space*. When action effects are non-deterministic then what one seeks for any state-space is a *policy*: i.e. a state-action table to specify which actions one should take in a given state. We define a non-deterministic state-transition domain (NSTD) to be a tuple $\mathcal{M} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ where:

- $\mathcal{P} = \mathcal{P}_S \cup \mathcal{P}_A$ is a finite set of propositions;
- $S \subseteq 2^{\mathcal{P}_S}$ is the set of all possible states;
- $A \subseteq 2^{\mathcal{P}_A}$ is the finite set of actions; and
- $R \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the state-transition relation.

We model how the agents can influence the external world as a *policy*, and we consider it to simply be a set of state-action pairs,

$$\pi = \{\langle s_i, a_i \rangle\}$$

where $s_i \in \mathcal{S}$ and $a_i \in \mathcal{A}(s)$ with $\mathcal{A}(s) = \{a | \exists \langle s, a, s' \rangle \in \mathcal{R}\}$ that is the set of actions that are applicable in s. A policy π is a *deterministic policy*, if for a given state s, there is no more than one action specified by π, otherwise it is a *non-deterministic policy*. What we are calling a policy is the state-action table of [7]. While the current model is concerned with the logical structure of states and state transitions, it is our future work to extend the model with probability and utility values attached to states or state transitions. With this extension in mind, the concept of policy in this model is also related to what the literature on MDPs calls a policy [3].

We define a propositional language $\mathcal{L}$ with quantification extension over the proposition variables by allowing standard connectives $\wedge, \vee, \rightarrow, \neg$ and quantifiers $\exists, \forall$. The resulting language is a logic of quantified boolean formulae (QBF) [4]. A *symbol renaming operation*, which we use below, can be defined on $\mathcal{L}$. For a formula $\xi \in \mathcal{L}$, if $\vec{x}$ and $\vec{x}'$ are two vectors of propositional variables, then a variable renaming operation can be defined by $\xi[\vec{x}/\vec{x}']$ which means that all the appearances of variables $\vec{x}$ are substituted by $\vec{x}'$. In QBF, propositional variables can be universally and existentially quantified: if ξ is a QBF and x is one of its variables, the existential quantification of x in ξ is defined as $\exists x \xi = \xi[x/FALSE] \vee \xi[x/TRUE]$; the universal quantification of x in ξ is defined as $\forall x \xi = \xi[x/FALSE] \wedge \xi[x/TRUE]$. Here $FALSE$ and $TRUE$ are two propositional constants representing "true" and "false" in the logic. Quantification over a vector $\vec{x} = x_1 x_2, \ldots, x_n$ of variables is defined as sequential quantifications over each variable x_i in the vector: $Q_{\vec{x}}\xi = Q_{x_n} Q_{x_{n-1}} \ldots Q_{x_1}\xi$ where Q is either $\exists$ or $\forall$. Similarly, quantifications over a set $X = \{x_1, x_2, \ldots, x_n\}$ of variables is defined as $Q_X \xi = Q_{x_n} Q_{x_{n-1}} \ldots Q_{x_1}\xi$. For the convenience of doing quantifications, given a set X and a vector $\vec{x}$ of proposition variables, we define $-X$ and $-\vec{x}$ to denote the set of variables that are not in X and $\vec{x}$ respectively.

Multiagent Systems

A multiagent system is composed of N agents, $AGS = \{T_1, \ldots, T_N\}$. We model each agent T_i with a NSTD $\mathcal{M}_i = \langle \mathcal{P}_i, \mathcal{S}_i, \mathcal{A}_i, \mathcal{R}_i \rangle$ where $\mathcal{P}_i = \mathcal{P}_{i,\mathcal{S}} \cup \mathcal{P}_{i,\mathcal{A}}$, and is associated with a local policy π_i. The multiagent system as a whole is modeled as a joint NSTD $\mathcal{M} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ along with with a joint policy π. The set of proposition variables in the joint model is the union of those of individual agents: $\mathcal{P}_{\mathcal{S}} = \bigcup_{i=1}^{N} \mathcal{P}_{i,\mathcal{S}}$ and $\mathcal{P}_{\mathcal{A}} = \bigcup_{i=1}^{N} \mathcal{P}_{i,\mathcal{A}}$. The joint states, actions, and state transitions are the result of interpreting $\mathcal{P}_{\mathcal{S}}$ and $\mathcal{P}_{\mathcal{A}}$ as in the basic NSTD. The individual transition relation and the policy can be projected from the joint system: $\mathcal{R}_i = \exists_{-\mathcal{P}_i}\mathcal{R}$ and $\pi_i = \exists_{-\mathcal{P}_i}\pi$. To distinguish from the dialogue model below, we call the individual models $\mathcal{M}_i$s, the joint model $\mathcal{M}$s, the associated polices π_is and the *external* NSTDs and policies π.

Dialogue Systems

In addition to the external transition system $\mathcal{M}_i$, each agent T_i is also associated with dialogue transition system $\mathcal{M}_{i,D} = \langle \mathcal{P}_{i,D}, \mathcal{S}_{i,D}, \mathcal{A}_{i,D}, \mathcal{R}_{i,D} \rangle$ and a dialogue policy $\pi_{i,D}$. The joint dialogue system is then defined as, $\mathcal{M}_D = \langle \mathcal{P}_D, \mathcal{S}_D, \mathcal{A}_D, \mathcal{R}_D \rangle$, along with a joint dialogue policy π_D. The set of propositional variables in the joint model is the union of those of individual agents: $\mathcal{P}_{\mathcal{S},D} = \bigcup_{i=1}^{N} \mathcal{P}_{i,\mathcal{S},D}$ and $\mathcal{P}_{\mathcal{A},D} = \bigcup_{i=1}^{N} \mathcal{P}_{i,\mathcal{A},D}$. The joint dialogue states, actions, and state transitions are the result of interpreting $\mathcal{P}_{\mathcal{S},D}$ and $\mathcal{P}_{\mathcal{A},D}$ as in the basic NSTD. The individual transition relation and the policy can be projected from the joint dialogue system: $\mathcal{R}_{i,D} = \exists_{-\mathcal{P}_{i,D}}\mathcal{R}_D$ and $\pi_{i,D} = \exists_{-\mathcal{P}_{i,D}}\pi_D$. Separate from the external policy π, the requirement for a dialogue policy π_D is that the individual agent can execute the projected $\pi_{i,D}$ without knowing the other agents' dialogue states so that the dialogue can help the multiagent system coordinate the joint external policy execution.

Coordination Dialogues

Using the above general model of multiagent systems and dialogues, in [19] we proposed a mechanism to coordinate agents' execution of a joint plan. This approach takes information about external states, state transitions and the joint policy and generates an appropriate dialogue between agents. These dialogues are then used to communicate the necessary information between agents. Individual dialogue actions communicate the values of external state and action variables to limit the number of dialogue state transitions. A dialogue policy can be planned by setting the goal dialogue states in which the external joint states and the policy decisions are

- fully communicated with each other; or
- partially communicated to a point at which every agent can locally compute a unique joint action from the joint policy locally.

Assume that the joint policy π is known by all the agents in the system. The task is to coordinate any joint decision $\langle s, a \rangle \in \pi$ of the pre-agreed or planned policy based on any joint perception $s = \langle s_1, \ldots, s_N \rangle$ of the whole system. As each agent T_i only holds its local view s_i, in order to conform with the joint plan, each agent might need to obtain additional information from the other agents so that a unique local action a_i can be computed. In order to achieve this, the effective execution of this coordination dialogue is as the following:

- On perceiving a new state s_i, agent T_i updates its local information store IS_i with newly perceived external state s_i by:
$$IS_i \quad = \quad s_i \wedge \pi$$
- On receiving a message $exp_{j,i}$ from another agent T_j, T_i updates its information store IS_i by:
$$IS_i \quad = \quad update(IS_i, exp_{j,i})$$
$update$ is a function to update the information store from the message received.
- At the end, every agent will need to communicate to a level such that a unique local action a_i can be drawn from IS_i.

This, then, provides us with a mechanism by which software agents can build a plan for a team, and know what information needs to be passed between team members in order to make sure that the plan is executed as intended. However, there is a major assumption that underlies this work — that the team members will freely share resources and information. In a coalition scenario this is typically not the case, and so our next step is to consider the impact of agents being less open with each other.

III. Argumentation-derived Evidence Learning

In a coalition, members of different organisations will typically operate under different policies for the sharing of information and resources (a specific form of the general policy defined above on information and resource allocations). Such policies can hamper coalition operations, especially in the planning stage where collaborative planning [2], [14] may involve extended negotiation about the use of resources. To address this problem, our previous work [10] presents an efficient approach for identifying, learning and modeling the policies of others during collaborative problem solving activities.

The mechanisms we presented in [10] enable agents to build more effective approaches to constructing plans by keeping track of who might have, and be willing to provide, the resources required for the enactment of a plan. In [10], we argues that agents can improve their communication strategies by building accurate models of others' policies regarding resource use, information provision, and this is backed up by a series of experiments. In these experiments, we demonstrate that more accurate models of others' policies (or norms) can be developed more rapidly using various forms of evidence from inter-agent communication. In particular, the approach exploits the fact that the dialogues are *argument-based*, that is the agents pass justifications — arguments — for the information that they exchange. We learn these justifications, and use them to predict future responses.

Negotiation dialogues

In argumentation-based dialogues, we assume that agent T_i has a plan (a subset of the joint plan) requiring the use of a set of resources R in order to achieve a goal G.

Definition 1: A resource allocation, denoted as Λ_i^t is a collection R of resources that an agent T_i has at its disposal at time t, where t denotes the time step in the dialogue.

- $\Lambda_i^t \subseteq R$, and $t = \{0, 1, \ldots, n\}$

The negotiation dialogue, as shown in Figure 1, starts with an agent, T_i (A in the figure), sending a request to another agent, T_j (B in the figure), for the use of some resources needed to fulfill a plan. The other agent can then respond with an agree or refuse based on the policy constraints defined below.

Resourcing policy

A *resourcing policy* governs how resources are deployed to others. The policies in this framework are based on a number of features which characterise the prevailing circumstances under which an agent operates. These features can take on different values, and are defined as follows:

- Organization, denoted by $\mathbf{O}$. This refers to the country/organization/affiliation of the requesting agent. In this framework, an agent is associated with the organization it represents.

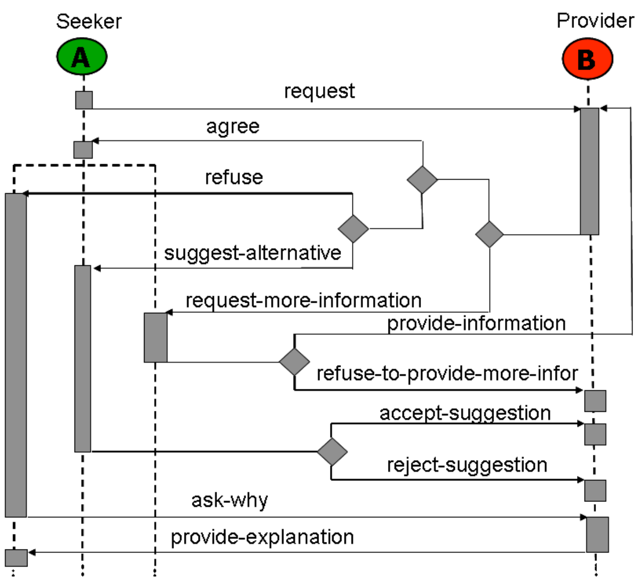

Fig. 1. The negotiation protocol.

You are **prohibited** from releasing a UAV to a coalition member if the affiliation of the coalition partner is $Holistan$, and the UAV is to be used on the *second* day of the mission.

TABLE I
An Example of a Resourcing Policy

- Resources, denoted by $\mathbf{R}$. This generally denotes the physical equipment, capabilities or information that are required to carry out a plan.
- Purpose, denoted by $\mathbf{P}$. This indicates the purpose for which the resources are requested
- Location, denoted by $\mathbf{L}$. This denotes the particular location or zone where the resource is to be deployed
- Day, denoted by $\mathbf{D}$. This refers to the day the resource is to be deployed

For every resource $r \in \mathbf{R}$ required to enact a plan, the resource policy features that determine the decision of the provider are denoted as the following

$$\mathbf{RSF} = \langle \mathbf{O}, \mathbf{P}, \mathbf{L}, \mathbf{D} \rangle$$

A resourcing policy is

$$\pi_{RS} : \mathbf{O} \times \mathbf{P} \times \mathbf{L} \times \mathbf{D} \times \mathcal{R} \rightarrow \{grant|deny\}$$

In other words it maps the above given issues and the resource profile into the decision *grant* or *deny*. The resourcing policy cited in the example given in Table I can be written as:

$$\pi_{RS}(Holistan, Any, Any, Day2, UAV) = deny$$

The term *Any* denotes that the attribute in question can be any of the possible values. There are two kinds of policies: the coalition-wide policy that is known to all the agents, and an individual policy that is only known to the individual agent.

TABLE II
EXAMPLE OF THE TRAINING DATA FOR LEARNING POLICY

Affiliation	Purpose	Location	Day	Resource	Decision
UK	Recon	Tersa	Day2	UAV	Grant
GER	Transport	Bortez	Day3	JEEP	Grant
US	Engagement	Holistan	Day1	TANK	Grant
Holistan	Recon	Tersa	Day2	UAV	Deny
UK	Food	Holistan	Day2	UAV	Grant

Learning Resourcing Policy

We have explored a range of techniques for learning the resourcing policies of other agents in this argumentation-based context. The techniques we have used include decision tree learning (C4.5), instance-based learning (k-Nearest Neighbours, abbreviated as k-NN) and rule-based learning (Sequential Covering, abbreviated as SC).

C4.5 [16] builds decision trees from a set of training data, using the concept of information entropy [15] (beyond the scope of this paper). Generally, the training data is a set $S = s_1, s_2, ..., s_n$ of already classified samples. Each sample $s_i = x_1, x_2, ..., x_m$ is a vector where $x_1, x_2, ..., x_m$ represents attributes of the sample. The training data is augmented with a vector $C = c_1, c_2, ..., c_n$ where $c_1, c_2, ..., c_n$ represent the class to which each sample belongs.

Integrating this algorithm into our system with the intention of learning policies is appropriate since the algorithm supports concept learning and policies can be conceived as concepts/properties of an agent. Agent policies are represented as a vector of attributes (e.g. resource, purpose, location, etc.) and these attributes are communicated back and forth during negotiation. The C4.5 algorithm is then used to classify each set of attributes (policy instance) into a class — grant or deny. Grant means that the *provider* agent will possibly provide the resource that is requested while deny implies that the *provider* agent will potentially refuse. The leaf nodes of a decision tree hold the class labels of the instances while the non-leaf nodes hold the test attributes. In order to classify a test instance, the C4.5 algorithm searches from the root node by examining the value of test attributes until a leaf node is reached and the label of that node becomes the class of the test instance.

We have also used Instance-based Learning (k-NN) and Rule-based Learning (Sequential Covering) to learn the resourcing policies. The k-nearest neighbours algorithm (k-NN) [8] is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The universal set of all the policies an agent may be operating within could be conceived as a feature space (or a grid) and the various policy instances represent points on the grid. Using k-NN, a policy instance is classified by a majority vote of its neighbours, with the policy instance being assigned to the class most common amongst its k nearest neighbours, where k is a positive integer, typically small. The k-NN algorithm is incremental, which means all the training examples need not exist at the beginning of the learning process. This is a good feature because the policy model could be updated as new knowledge is learned.

Since policies guide the way entities within a community (or domain) act by providing rules for their behaviour it makes sense to learn policies as rules. The sequential covering algorithm [15], [5] is a rule-based learning technique, which constructs rules by sequentially covering the examples. The sequential covering algorithm, SC for short, is a method that induces one rule at a time (by selecting attribute-value pairs that satisfy the rule), removes the data covered by the rule and then iterates the process. SC generates rules for each class by looking at the training data and adding rules that completely describe all tuples in that class. For each class value, rule antecedents are initially empty sets, augmented gradually for covering as many examples as possible.

Argument-derived Evidence

Using the negotiation dialogue showed in Figure 1, we can add more information into the learned resourcing policy. Without the information derived from the arguments, the agents can only build their model of the other agents' resourcing policy by mapping the resource allocation profile $r \in \mathcal{R}$ to $\{grant|deny\}$ as a *pure resource policy*

$$\pi_\Lambda : \mathcal{R} \rightarrow \{grant|deny\}$$

The underlying assumption of a pure resource policy π_Λ is that the other factors, such as organizations, locations, purpose, and day, can be any values.

With the negotiation protocol, additional issues, such as those mentioned above (that is, policy features), are communicated with each other. For example, a UK agent may request a UAV to be used for $Reconnaissance$ at $Holistan$ on $Day3$. If this request is denied it may assume that the other agent's policy does not allow UAVs to be provided for $Reconnaissance$ but that is not entirely true (as you can see from the training data in Table II). By exchanging arguments about the features and characteristics of the agent's request, the UK agent may gather further evidence about the other agent's policy. For instance, the UK agent may learn that (1) the UAV was not available at the time of request, (2) the other agent only provides UAVs for $Reconnaissance$ at $Tersa$ and not at $Holistan$, or (3) the other agent can provide UAVs on $Day3$ for any purpose other than $Reconnaissance$. As a result, agents can build a more accurate model of the other agents' resourcing policy by mapping the resource allocation profile and the resource policy features $\mathbf{RSF}$ to decisions $\{grant|deny\}$

$$\pi_{RS} : \mathcal{R} \times \mathbf{RSF} \rightarrow \{grant|deny\}$$

In combination with the aforementioned learning techniques, agents are able to learn more accurate model of each other's resourcing policy so that the system as whole can reach the mutual agreed resource allocation to resource the plan more efficiently.

IV. INTEGRATING LEARNING, NEGOTIATION, AND COORDINATION DIALOGUES

In this section we extend the formal model of Section II with the features we need to incorporate negotiation about

Set representation	QBF implementation
$EXEC(s,a)$	$\xi(s) \wedge \xi(a) \wedge \xi(\mathcal{R})[\vec{x'}/\vec{x}]$
$StatesOf(\pi)$	$\exists \vec{a} \xi(\pi)$
$GetAction(s,\pi)$	$\xi(s) \wedge \xi(\pi)$
$ComputeWeakPreImage(S)$	$\exists \vec{x}' \xi(S)[\vec{x}/\vec{x}'] \wedge \xi(\mathcal{R})$
$ComputeStrongPreImage(S)$	$\forall \vec{x}' (\xi(\mathcal{R}) \rightarrow \xi(S)[\vec{x}/\vec{x}']) \wedge \exists \vec{x}' \xi(\mathcal{R})$
$ComputeNextImage(S)$	$\exists \vec{x} \xi(S) \wedge \xi(\mathcal{R})$
$PrunStates(\pi, S)$	$\xi(\pi) \wedge \neg \xi(S)$

TABLE III
THE MAPPING BETWEEN SET REPRESENTATION AND QBF IMPLEMENTATION OF SOME TRANSITION RELATION AND POLICY FUNCTIONS

resources and the learning mechanism described above. In particular, we add a set of resource variables and policy issues to the state variables, and embed negotiation dialogues within the coordination dialogues. With the original model, resources were just a common pool that agents could draw on as required. In the extended model, resources have an owner, and it is necessary to negotiate their use with their owner.

We handle resources in the formal model by assuming that for any joint decision $\langle s, a \rangle$ in the joint policy π, there is a resource profile $RS \subseteq \mathcal{R}$ that can resource it. Among these resource allocations, there is a subset $RS^* \subseteq RS$, acceptable to all the agents according to their constraint policies. $RS = \langle RS_1, \ldots, RS_N \rangle$ where RS_i is the *locally feasible allocation* for individual agent T_i, and RS^*_i is the *globally acceptable allocation* of all the agents to T_i. RS_i and RS^*_i can contain resources from other agents.

As is the case for the local perception s_i, the resource allocation can be used to compute a unique local action of T_i to achieve the joint policy as the output of the negotiation dialogues.

Ontology Extension

To handle negotiation over resources, we need to extend the ontology we use in the planning system with a set of resource variables $\mathcal{P}_{i,\mathcal{R}}$ for agent T_i. Each resource r_i will be represented by a set of resource variables $\vec{xr}_i \in \mathcal{P}_{i,\mathcal{R}}$. There is another set of variables $\mathcal{P}_{i,\mathbf{RSF}}$ for the resource policy features of agent T_i. For each issue, there is a subset of issue variables $\mathcal{P}_{i,*,\mathbf{RSF}}$ where $*$ stands for $\{\mathbf{O}, \mathbf{P}, \mathbf{L}, \mathbf{D}\}$. In addition, we need to introduce a resource decision variable $\mathcal{P}_{RD} = \{dec_R\}$ where dec_R means $grant$ the resource request and $\neg dec_R$ means deny the resource request.

With this setup, we can use the same framework to represent resource allocation and resource policy, and resource decision for each joint decision.

Policies and resources

With this extended formal model, we have the following relationship between policies and resources.

- We start with the policy π that describes the joint plan for the team and its projection π_i for each agent T_i; this requires a set of resources, and they are listed in a:
- Resource requirement table $\Lambda : \mathcal{S} \times \mathcal{A} \times 2^{\mathcal{P}_{\mathcal{R}}}$; for each resource we have a resource policy:

$$\pi_{RS} : \mathcal{S} \times \mathcal{A} \times 2^{\mathcal{P}_{\mathbf{RSF}}} \times 2^{\mathcal{P}_{\mathcal{R}}} \rightarrow 2^{\mathcal{P}_{RD}}$$

We currently assume that the resource requirement is associated with a state-action pair — that is each action requires a resource, and that resource may vary with the state in which the action is carried out. The resource policy π_{RS} can then be learned using the learning methods mentioned in Section III.

Planning joint policy with learned resource policy

The model described so far is sufficient for us to create a team plan. However, we can prune the set of joint state transitions before planning reducing it to:

$$\mathcal{R}^* = \exists_{-(\mathcal{P}_{\mathcal{S}} \cup \mathcal{P}_{\mathcal{A}} \cup \mathcal{P}'_{\mathcal{S}})} \left(\mathcal{R} \wedge \Lambda \wedge \pi_{RS} \right)$$

Using $\mathcal{R}^*$ means we will only consider those state transitions that can possibly be resourced given the teams' knowledge of the constraints on the use of resources by team members, ignoring those resources that might be used if team members were operating under different resource policies. Feeding $\mathcal{R}^*$ to a Algorithm IV.1 will then generate a joint plan that is feasible in the sense that the team is willing to use all the resources that are needed to carry out the plan.

Of course since π_{RS} contains those constraints on resources that, from experience, the team believes will be applied, it might not capture the real allocation constraints accurately. It might also not provide enough resources to carry out the plan. If no joint plan can be generated using π_{RS}, we can then remove constraints from π_{RS} gradually with respect to what has been learned about the likelihood of them being granted. As these constraints are removed, $\mathcal{R}^*$ grows, and Algorithm IV.1 will search through an expanding set of possible plans until it either finds a plan that will work for some subset of all the possibly available resources, or discovers that the task in question is beyind the ability of the team.

Policy Based Negotiation dialogues

So far we have shown how to incorporate a model of resources into the planning model, and how to flexibly incorporate a model of resource constraints, which has been obtained by machine learning, into the planning process. Now we look at the question of how we might learn the resource constraints from the planning process.

We start by denoting the specification of kth resource requirement of agent T_i as

$$RS_{i,k} \quad \text{which is labeled by} \quad l_{RS,i,k}$$

The the full set of resources, which we call the joint resource, is:

$$CONRS \;=\; \bigwedge_{i=1}^{N} \left(\bigwedge_{k=1}^{K_{RS,i}} (l_{RS,i,k} \rightarrow RS_{i,k}) \right) \wedge SHRSC$$

where $SHRSC$ is the set of publically known resource constraints — constraints that have been revealed by previous

Algorithm IV.1 World policy generation

```
1: function $ComputeWorldPolicy(I, G, CompPreImage)$ {
   (1) $I$: Initial states,
   (2) $G$: Goal states,
   (3) $CompPreImage$ : A pre-image function }
2: $DJMAP \leftarrow \emptyset$
3: $OldSA \leftarrow Fail$
4: $SA \leftarrow \emptyset$
5: $SA_D \leftarrow \emptyset$
6: while $OldSA \neq SA \wedge I \not\subseteq (G \cup StatesOf(SA))$ do
7:    $PreImage \leftarrow ComputePreImage(G \cup StatesOf(SA))$
8:    $NewSA \leftarrow PruneStates(PreImage, G \cup StatesOf(SA))$
9:    if $\exists i |joint(GetAction(proj_i(NewSA)))| > 1$ then
10:      $I_D \leftarrow ComputeDialState(NewSA)$
11:      $G_D \leftarrow ComputeDialGoal(NewSA)$
12:      $NewSA_D \leftarrow ComputePolicy(I_D, G_D, \mathcal{R}_D, ComputePreImage)$
13:      if $NewSA_D = \emptyset$ then
14:         return $Fail$
15:      end if
16:      $SA_D \leftarrow SA_D \cup NewSA_D$
17:   end if
18:   $OldSA \leftarrow SA \cup NewSA$
19: end while
20: if $I \subseteq (G \cup StatesOf(SA))$ then
21:   return $\langle SA, SA_D \rangle$
22: else
23:   return $Fail$
24: end if
25: end function
```

dialogues. We can then write:

$$CONRS^{+} = CONRS \wedge \bigwedge_{i=1}^{N} \left(\bigvee_{k=1}^{K_{RS,i}} (l_{RS,i,k}) \right)$$

where $\bigwedge_{i=1}^{N} \left(\bigvee_{k=1}^{K_{RS,i}} (l_{RS,i,k}) \right)$ denotes the fact at least one resource requirement of each agent should be satisfied. This can be replaced by a more fine grained global resourcing criteria. For example, we can introduce constraint types associated with each item $RS_{i,k}$: must-be-satisfied flg_{MS}, satisfied-one flg_{S1}. As a result, each $RS_{i,k}$ is represented by a tuple $\langle l_{RS,i,k}, \{flg_{MS}|flg_{S1}\}, RS_{i,k} \rangle$ and we get a modified set of constraints:

$$\begin{aligned} CONRS^{+} &= CONRS \wedge \\ &= \bigwedge_{i=1}^{N} \left(\bigvee_{k=1}^{K_{RS,i}} ((flg_{S1} \rightarrow l_{RS,i,k})) \right) \\ &\quad \wedge (flg_{MS} \wedge l_{RS,i,k}) \end{aligned}$$

Given this, we can show that the negotation described above ties in with the planning process:

Proposition 1: Using the aforementioned negotiation dialogues, a participating agent T_i will reveal each $RS_{i,k}$ during the dialogue, and the negotiation dialogue will stop when $CONRS^{+}$ is not empty. Every agent can extract a mutually agreed joint resource allocation from $CONRS^{+}$ because the items $RS_{i,k}$ are revealed and become known by every agent.

Algorithm IV.2 General policy generation

```
1: function $ComputePolicy(I, G, ComputePreImage)$ {
   (1) $I$: Initial states,
   (2) $G$: Goal states,
   (3) $ComputePreImage$ : A pre-image function }
2: $OldSA \leftarrow Fail$
3: $SA \leftarrow \emptyset$
4: while $OldSA \neq SA \wedge I \not\subseteq (G \cup StatesOf(SA))$ do
5:    $PreImage \leftarrow ComputePreImage(G \cup StatesOf(SA))$
6:    $SA \leftarrow PruneStates(PreImage, G \cup StatesOf(SA))$
7:    $OldSA \leftarrow SA \cup SA$
8: end while
9: if $I \subseteq (G \cup StatesOf(SA))$ then
10:   return $SA$
11: else
12:   return Fail
13: end if
14: end function
```

In other words, the negotiation process will generate the set of resources necessary to execute the plan.

V. Summary

This paper has described how we are integrating two lines of work in Project 10, Task 1 — work on generating plans for teams and work on negotiating the resources required by teams and learning over time how the policies that govern the use of these resources. We have shown how the formal model that underlies our approach to team planning can be extended with the notion of the resources that are used in the plan. This provides a bridge to the work on resource negotiation — integrating the objects that are being negotiated into the planning process. It is then possible to detect which resources are required, so that their use can be negotiated. The planning model is also extended with a model of the constraints on the use of resources, and this allows the planning model to benefit from the work on learning — learning provides a way to adapt the model of contraints over time, and when this changing model is used in planning, the planning process will adapt to what is learnt.

Acknowledgement

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] J. Allen, A. Mowshowitz, T. Norman, S. Parsons, and A. Preece. Managing collaboration in hybrid-agent teams. In *Proceedings of the First Annual Conference of the ITA*, University of Maryland, 2007.

[2] A. Bahrami, J. Yuan, D. Mott, and C. D. Emele. Collaborative, context-aware and chain of command sensitive planning. In *Proceedings of the Second Annual Conference of the International Technology Alliance*, London, September 2008.

[3] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[4] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[5] J. Cendrowska. Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.

[6] H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. V. Pynadath, T. A. Russ, and M. Tambe. Electric elves: Agent technology for supporting human organizations. *Artificial Intelligence*, 23(2):11–24, 2002.

[7] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[8] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transaction on Information Theory*, 13(1):21–27, 1967.

[9] C. D. Emele, T. J. Norman, F. Guerin, and S. Parsons. Argumentation-based agent support for learning policies in a coalition mission. In *Proceedings of the Third Annual Conference of the International Technology Alliance*, pages 151–154, Maryland, USA, 2009.

[10] C. D. Emele, T. J. Norman, F. Guerin, and S. Parsons. On the benefits of argumentation-derived evidence in learning policies. In *Proceedings of the Seventh International Workshop on Argumentation in Multiagent Systems*, Toronto, May 2010.

[11] M. Esteva, D. Cruz, and C. Sierra. Islander: an electronic institution editor. In *Proceedings of the 1st International Conference on Autonomous Agents and Multi-Agent Systems*, New York, NY, 2002. ACM Press.

[12] M. Esteva, J. A. Rodriguez, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specification of electronic institutions. In *Agent-mediated Electronic Commerce*, number 1991 in Lecture Notes in Artificial Intelligence, pages 126–147. Springer Verlag, Berlin, Germany, 2001.

[13] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, 1994.

[14] T. J. McKearney. Colaborative planning for military operations: Emerging technologies and changing command organizations. In *Proceedings of the Command and Control Research and Technology Symposium*, Naval Postgraduate School, Monterey, CA, June 2000.

[15] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[16] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[17] S. D. Ramchurn, B. Deitch, M. K. Thompson, D. C. de Roure, N. R. Jennings, and M. Luck. Minimising intrusiveness in pervasive computing environments using multi-agent negotiation. In *Proceedings of the 1st International Conference on Mobile and Ubiquitous Systems*, Boston, MA, 2004.

[18] Yuqing Tang, Timothy J. Norman, and Simon Parsons. Agent-based dialogues to support plan execution by human teams. In *Proceedings of the Second Annual Conference of the ITA*, Imperial College, London, 2008.

[19] Yuqing Tang, Timothy J. Norman, and Simon Parsons. A model for integrating dialogue and the execution of joint plans. In *Proceedings of the Eigth International Joint Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, May 10-15 2009.