# Prognostic agent assistance for norm-compliant coalition planning

Jean Oh[1], Felipe Meneguzzi[1], Katia Sycara[1], and Timothy J. Norman[2]

[1] Robotics Institute, Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213, USA
{jeanoh,meneguzz,katia}@cs.cmu.edu
[2] Dept. of Computing Science, University of Aberdeen
Aberdeen, UK
t.j.norman@abdn.ac.uk

**Abstract.** In this paper we describe a software assistant agent that can proactively assist human users situated in a time-constrained environment. We specifically aim at assisting the user's normative reasoning–reasoning about prohibitions and obligations–so that the user can focus on her planning objectives. In order to provide proactive assistance, the agent must be able to 1) recognize the user's planned activities, 2) reason about potential needs of assistance associated with those predicted activities, and 3) plan to provide appropriate assistance suitable for newly identified user needs. To address these specific requirements, we develop an agent architecture that integrates user intention recognition, normative reasoning over a user's intention, and planning, execution and replanning for assistive actions. This paper presents the agent architecture and discusses practical applications of this approach.

## 1 Introduction

Human planners dealing with multiple objectives in a complex environment are subjected to a high level of cognitive workload, which can severely impair the quality of the plans created. For example, military planners during peacekeeping operations must plan to achieve their own unit's objectives while following standing policies (or norms) that regulate how interaction and collaboration with Non-Governmental Organizations (NGOs) must take place. As the planners are cognitively overloaded with mission-specific objectives, such normative stipulations hinder their ability to plan to both accomplish goals and abide by the norms. We develop an assistant agent that takes a proactive stance in assisting cognitively overloaded human users by providing prognostic reasoning support. In this paper, we specifically aim to assist the user's *normative reasoning*–reasoning about prohibitions and obligations.

In order to provide a user with a timely support, it is crucial that the agent recognizes the user's needs in advance so that the agent can work in parallel with the user to ensure the assistance is ready by the time the user actually needs it. This desideratum imposes several technical challenges to: 1) *recognize* the user's planned activities, 2) *reason* about potential needs of assistance for those predicted activities to comply with norms as much as possible, and 3) *plan* to provide appropriate assistance suitable for newly identified user needs.

Our approach to tackle these challenges is realized in a proactive planning agent framework. As opposed to planning for a *given* goal state, the key challenge we address here is to *identify* a new set of goal states for the agent–*i.e.*, a set of tasks for which the user will need assistance. After identifying new goals, the agent plans, executes, and replans a series of actions to accomplish them. Specifically, we employ a probabilistic plan recognition technique to predict a user's plan for her future activities. The agent then evaluates the predicted user plan to detect any potential norm violations, generating a set of new goals (or tasks) for the agent to prevent those norm violations from happening. As the user's environment changes the agent's prediction is continuously updated, and thus agent's plan to accomplish its goals must be frequently revised during execution. To enable a full cycle of autonomy, we present an agent architecture that seamlessly integrates techniques for plan recognition; normative reasoning over a user's plan; and planning, execution and replanning for assistive actions.

The main contributions of this paper are the following. We present a principled agent architecture for prognostic reasoning assistance by integrating probabilistic plan recognition with reasoning about norm compliance. We develop the notion of prognostic norm reasoning to predict the user's likely normative violations, allowing the agent to plan and take remedial actions before the violations actually occur. To the best of our knowledge, our approach is the first that manages norms in a proactive and autonomous manner. Our framework supports interleaved planning and execution for the assistant agent to adaptively revise its plans during execution, taking time constraints into consideration to ensure timely support to prevent violations. For a proof of concept experiment, our approach has been fully implemented in the context of a military peacekeeping scenario.

The rest of this paper is organized as follows. After reviewing related work in Section 2, we describe a high-level architecture of our agent system in Section 3. The three main components are described in detail in the following sections: Section 4 describes the agent's plan recognition algorithm for predicting the user's future plan; Section 5 describes how the agent evaluates the norm rules to maintain a normative state and to detect potential violations; and Section 6 presents how the agent plans and executes actions to accomplish identified goals. We present a fully implemented system in a peacekeeping problem domain, followed by other potential applications of this work in Section 7, and conclude the paper in Section 8.

## 2 Related work

This paper builds on previous work on assisting the user with information management [Oh et al., 2011] where the main discussion was on the algorithms for plan recognition and information management. In this paper, we specifically aim at assisting the user with normative reasoning and autonomous planning and execution. Our paper is related to work in the literature on: 1) plan recognition, 2) assistant agents, 3) normative reasoning, and 4) norm monitoring.

## 2.1 Plan Recognition

Plan recognition refers to the task of identifying the user's high-level goals (or intentions) by observing the user's current activities. The majority of existing work in plan recognition relies on a *plan library* that represents a set of alternative ways to solve a domain-specific problem, and aims to find a plan in the library that best explains the observed behavior [Armentano and Amandi, 2007]. In order to avoid the cumbersome process of constructing elaborate plan libraries of all possible plan alternatives, recent work proposed the idea of formulating plan recognition as a planning problem using either classical planners [Ramírez and Geffner, 2009] or decision-theoretic planners [Baker et al., 2009]. The plan recognition algorithm used in our approach is based on a decision-theoretic planner, namely a Markov Decision Process (MDP) [Bellman, 1957].

## 2.2 Assistant Agents

An assistant agent is commonly modeled as a planning agent in literature. For example, Partially Observable Markov Decision Process (POMDP) models have been used to develop a hand-washing assistant for dementia patients [Boger et al., 2005] or a doorman assistant that helps a user navigate a maze by opening the doors [Fern et al., 2007]. These models are not suitable for a prognostic agent that must provide proactive assistance by predicting a user's future actions in advance due to the following reason. In these approaches, the states of a POMDP are defined in terms of the variables describing both a user's state and the agent's state. Such models, however, do not include user actions (*i.e.*, the actions defined in a POMDP are agent actions). Since the user actions are not represented in the models, it is not possible to predict future user actions. Moreover, as the number of states of a (PO)MDP grows exponentially in the number of state variables, this approach suffers from the curse of dimensionality.

In contrast, we take a modularized approach. While an agent's planning problem is defined using only those variables that the agent is entitled, a plan recognition module keeps track of a user's current and future activities to identify new tasks for the agent to prepare assistance. By separating the plan prediction from the agent's action selection, our approach not only achieves an exponential reduction in the size of state space, but also enables the agent to simultaneously assist the user with multiple tasks because each new task is handled in an independent thread.

## 2.3 Normative Reasoning

In order to ensure that certain global properties of a society or organization are maintained, rules (or norms) that express permissions, prohibitions and obligations have been developed [Jones, 1990]. Mathematical study of norms has been carried out in the context of deontic logic [von Wright, 1968], while computational treatment of these stipulations has been studied recently by the agents community as normative systems. These efforts led to the development of various formal models of norms [Vázquez-Salceda et al., 2005], as well as practical approaches to reasoning about norms within individual agents [Lopez y Lopez and Luck, 2003] and in a society [García-Camino et al., 2009]. The formalisms that allow modeling of norms for agent systems can also be used for the specification of the rules that humans must follow. Since this work is concerned with assisting a user to mitigate the cognitive load of planning under environmental norms, we leverage the formalisms to create an internal representation of the norms that the assistant must consider when providing assistance.

### 2.4 Norm Monitoring

In order for norms to be enforced in a norm-regulated system, various mechanisms were devised to monitor norm compliance within a system. The state of compliance of a set of norms within a system is known as the *normative state* [Farrell et al., 2005] and describes which agents are complying (or violating) which norms. Although various approaches to norm monitoring have been proposed [Farrell et al., 2005,Modgil et al., 2009,Hübner et al., 2010], they all rely on a deterministic logic view of the normative state. Without a probabilistic model of agent behavior, a norm monitoring mechanism can only assert whether a norm is definitely violated or not, lacking a gradual notion of how *likely* an agent is to violate a norm or *when* an agent is *about to* violate a norm. Thus, an assistant aiming to warn a user of potential violations can either *constantly* remind the user of *all* the norms in the system (which can potentially be violated), or inform the user *after* a violation has occurred that some remedial action should be taken. Whereas these approaches fail to address an important issue of prevention, our probabilistic normative reasoning approach allows the agent to detect probable norm violations in advance, *preventing* the user from violating norms.

## 3 Agent architecture

Figure 1 provides a high-level overview of our agent system for proactive yet unobtrusive assistance. The observer module monitors the user's current activities in the environment to identify new observations that might indicate any changes in the user's current and future plan. Given a new observation, the plan recognizer module uses a probabilistic algorithm to update its prediction for the user's plans. From the predicted user plan, the norm reasoner module evaluates each plan step (actually the state resulting from these steps) to detect any potential norm violations. For each state in which norms are violated, the reasoner generates a new assistant task for the agent to carry out. The agent planner

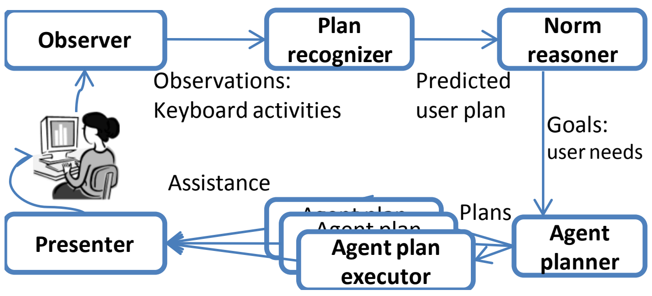

**Fig. 1.** An autonomous assistant system

module receives the new planning task that is to find a series of actions to prevent potential norm violations. When a prognostic plan is generated, the agent executes the plan until either the goal is reached or the goal becomes irrelevant to the user's planning context. The effects of the successful plan execution can be presented to the user, *e.g.*, notifying which actions the agent has taken to resolve a certain violation.

**Design assumptions:** A user's planning state space is defined in terms of a set of *variables* and their *domains* of valid values, where a *variable* describes an environment and the progress status of certain activities. The variables are also used to specify regulating rules in the norm representation, *e.g.*, a norm rule may define a relationship among a subset of variables. In general, such norm rules introduce additional variables to consider, adding extra dimensions into the reasoning process. As seen in a recent study [Sycara et al., 2010], when planning involves complex reasoning as in military environments, human users tend to lose track of policies (or norm rules), resulting in plans with significant norm violations. By developing an assistant agent that manages norm-related variables, our approach aims to relieve the user from having to deal with both task-specific variables and norm-related variables. We make a specific assumption that task-specific *user variables* and norm-specific *agent variables* are independent and thus changing an agent variable does not affect the values of user variables. For representation, let $((\textit{user-variables}), (\textit{agent-variables}))$ denote a state composed of user variables and agent variables.

**Example scenario:** We use a simple example of peacekeeping scenario to illustrate the approach throughout the paper. We develop an assistant agent for a humanitarian NGO teamed with a military coalition partner. Consider a norm rule that an NGO must have an armed escort when operating in conflict areas. An escort can be arranged through a well-defined communication protocol, *e.g.*, sending an escort request to and receiving a confirmation from a military party. Here, a state space can be defined in terms of two variables: *area* specifying the user's geographic coordinates and *escort* indicating the status of an armed escort in each region. In our approach, a user can focus on reasoning about variable *area* only since the agent manages variable *escort* to assure that the user plan complies with norms. Note that variable *escort* is a simplified representation as it is defined for each value of variable *area*, *i.e.*, it is a function *escort*(*area*) to be precise.

In the following sections, technical details will be described for three main components: namely, plan recognizer, norm reasoner, and agent planner and executor. Due to space limitation, we omit the description for the observer and presenter modules that are responsible for interacting with the user.

## 4 Probabilistic plan recognition

Our plan recognition algorithm is leveraged from previous work [Oh et al., 2011]. We assume that a user's planning problem is given as an MDP. Based on the assumption that a human user generally reasons about consequences and makes decisions to maximize her long-term rewards, we utilize an optimal stochastic policy of the MDP to predict a user's future activities.

We compute an optimal stochastic policy as follows. Let $G$ denote a set of possible goal states. For each potential goal $g \in G$, we compute policy $\pi_g$ to achieve goal $g$ using a dynamic programming algorithm known as *value iteration* [Bellman, 1957]. Instead of a deterministic policy that specifies only the optimal action, we compute a stochastic policy such that probability $p(a|s,g)$ of taking action $a$ given state $s$ when pursuing goal $g$ is proportional to its long-term expected value $v(s,a,g)$ such that:

$$p(a|s,g) \propto \beta\ v(s,a,g),$$

where $\beta$ is a normalizing constant. The intuition for using a stochastic policy is to allow the agent to explore multiple likely plan paths in parallel, relaxing the assumption that a human user always acts to maximize her expected reward.

The plan recognition algorithm is a two-step process. In the first step, the algorithm estimates a probability distribution over a set of possible goals. We use a Bayesian approach that assigns a probability mass to each goal according to how well a series of observed user actions is matched with the optimal plan toward the goal. We assume that the agent can observe a user's current state and action. This assumption can be relaxed such that a sequence of user states and actions can be derived from primitive observations, but the detail is omitted here. Let $O_t = s_1, a_1, s_2, a_2, ..., s_t, a_t$ denote a sequence of observed states and actions from time steps 1 through $t$ where $s_t$ and $a_t$ denote the user state and action, respectively, at time step $t$.

When a new observation is made, the assistant agent updates, for each goal $g$ in $G$, the conditional probability $p(g|O_t)$ that the user is pursuing goal $g$ given the sequence of observations $O_t$. The conditional probability $p(g|O_t)$ can be rewritten using Bayes' rule as:

$$p(g|O_t) = \frac{p(s_1, a_1, ..., s_t, a_t|g)p(g)}{\sum_{g' \in G} p(s_1, a_1, ..., s_t, a_t|g')p(g')}. \tag{1}$$

By applying the chain rule, we can write the conditional probability of observing the sequence of states and actions given a goal as:

$$\begin{aligned} p(s_1, a_1, ..., s_t, a_t|g) = &\, p(s_1|g)p(a_1|s_1, g)p(s_2|s_1, a_1, g) \\ & ... \, p(s_t|s_{t-1}, a_{t-1}, ..., g). \end{aligned}$$

We replace the probability $p(a|s, g)$ with the user's stochastic policy $\pi_g(s, a)$ for selecting action $a$ from state $s$ given goal $g$. By the MDP problem definition, the state transition probability is independent of the goals. Due to the Markov assumption, the state transition probability depends only on the current state, and the user's action selection on the current state and the specific goal. By using these conditional independence relationships, we get:

$$\begin{aligned} p(s_1, a_1, ..., s_t, a_t|g) = &\, p(s_1)\pi_g(s_1, a_1)p(s_2|s_1, a_1) \\ & ... \, p(s_t|s_{t-1}, a_{t-1}), \end{aligned} \tag{2}$$

By combining Equations 1 and 2, the conditional probability of a goal given a series of observations can be obtained.

In the second step, we *sample* likely user actions in the current state according to a stochastic policy of each goal weighted by the conditional probability from the previous step. Subsequently, the next states after taking each action are sampled using the MDP's state transition function. From the sampled next states, user actions are recursively sampled, generating a tree of user actions known here as a *plan-tree*. The algorithm prunes the nodes with probabilities below some threshold. A node in a plan-tree can be represented in a tuple $\langle t, s, l \rangle$ representing the depth of node (*i.e.*, the number of time steps away from the current state), a predicted user state, and an estimated probability of the state visited by the user, respectively. Example 1 shows a segment of plan-tree indicating that the user is likely be in area 16 with probability .8 or in area 15 with probability .17 at time step $t_1$.

*Example 1.* $\langle\langle t_1, (area = 16), .8\rangle, \langle t_1, (area = 15), .17\rangle\rangle$

## 5 Norm reasoner

In this section we specify the agent component responsible for evaluating predicted user plan to generate new goals for the agents. For this purpose, we utilize normative reasoning. Norms generally define constraints that should be followed by the members in a society at particular points in time in order for them to be compliant with societal regulations. We specify our norm representation format, followed by two algorithms for 1) predicting violations and 2) finding the nearest complying state towards which we can steer the user.

### 5.1 Norm representation

Inspired by the representation in [García-Camino et al., 2009], we define a norm in terms of its deontic modality, a formula specifying when the norm is relevant to a state (which we call the *context condition*), and a formula specifying the constraints imposed on an agent when the norm is relevant (which we call the *normative condition*). We restrict the deontic modalities to those of *obligations* (denoted **O**) and *prohibitions* (denoted **F**); and use these modalities to specify, respectively, whether the normative condition must be true or false in a relevant state. The conditions used in a norm are specified in terms of state variables and their relationships such as an equality constraint. Formally,

**Definition 1 (Norm).** *A norm is a tuple $\langle \nu, \alpha, \mu \rangle$ where $\nu$ is the deontic modality, $\alpha$ is the context condition and $\mu$ is the normative condition.*

*Example 2.* In the peacekeeping operations scenario, suppose that an intelligence message notifies that regions 3, 16 and 21 are risky areas. The norm, denoted by $\iota_{escort}$, that an NGO is obliged to have an armed escort can be expressed as:

$$\iota_{escort} = \langle \mathbf{O}, \mathit{area} \in \{3, 16, 21\}, \mathit{escort} = \mathit{granted} \rangle.$$

**Definition 2 (Satisfiability).** *A context condition $\alpha$ (alternatively a normative condition $\mu$) is* satisfiable *in state $s$ (so that $s \models \alpha$) if, for each variable $\varphi$ in state $s$, the current value assignment $v_s(\varphi)$ of variable $\varphi$ in state $s$ is within the valid domain $d_{\varphi,\alpha}$ of variable $\varphi$ specified in condition $\alpha$, where the default domain is open if unspecified; such that $\forall \varphi \in s, v_s(\varphi) \in d_{\varphi,\alpha}$.*

## 5.2 Detecting violations

We say that a state is *relevant* to a norm if the norm's context condition is satisfied in the state. When a state is relevant to a norm, a normative condition is evaluated to determine the state's compliance, which depends on the deontic modality of the norm. Specifically, an obligation norm is violated if the normative condition $\mu$ is not supported by state $s$; *i.e.*, $s \not\models \mu$. Conversely, as prohibitions specify properties that *should not* be realized, a prohibition is violated if the normative condition is supported by state $s$ such that $s \models \mu$. Formally,

**Definition 3 (Violating State).** *Given state $s$ and norm $\iota = \langle \nu, \alpha, \mu \rangle$, a function determining the violation of norm $\iota$ in state $s$ is defined as:*

$$\mathit{violating}(s, \iota) = \begin{cases} 1 & \mathit{if}(s \models \alpha) \wedge (s \not\models \mu) \wedge (\nu = \mathbf{O}) \\ 1 & \mathit{if}(s \models \alpha) \wedge (s \models \mu) \wedge (\nu = \mathbf{F}) \\ 0 & \mathit{otherwise.} \end{cases}$$

For instance, considering norm $\iota_{escort}$ in Example 2, given state $s = \{(\mathit{area} = 16), (\mathit{escort} = \mathit{init})\}$ the violation detection function $\mathit{violation}(s, \iota_{escort})$ would return 1, denoting that norm $\iota_{escort}$ is violated in state $s$.

Given a predicted user plan in a plan-tree, the norm reasoner traverses each node in the plan-tree and evaluates the associated user state for any norm violations. Recall from Section 4 that each node in a predicted plan-tree is associated with a user state and an estimated probability of the user visiting the node in the future. Using the estimated probability, the agent selects a set of high-risk norm violations to manage them proactively.

## 5.3 Finding the nearest compliant state

Our assistant agent aims at not only alerting the user of active violations but also proactively steering the user away from those violations that are likely to happen in the future. In order to accomplish this, for each state that violates a norm the agent needs to find a state that is *compliant* with all norms. That is, for each state $s$ that violates *any* norm (*i.e.*, $violating(s, \cdot) = 1$), the agent is to find the nearest state $g$ that satisfies *all* norms (*i.e.*, $violating(g, *) = 0$), where '$\cdot$' and '$*$' are regular expressions denoting any and all, respectively . Here, the distance between two states is measured by the number of variables whose values are different.

Since norm violations occur as the result of certain variables in the state space being in particular configurations, finding compliant states can be intuitively described as a search process for alternative value assignments for the variables in the normative condition such that norms are no longer violated, which is analogous to search in constraint satisfaction problems.

When a norm-violating state is detected, the norm reasoner searches the nearby state space by trying out different value assignment combinations for the agent-variables. For each altered state, the norm reasoner evaluates the state for norm compliance. The current algorithm is not exhaustive, and only continues the search until a certain number of compliant states, say $m$, are found.

When compliant state $g$ is found for violating state $s$, state $g$ becomes a new goal state for the agent, generating a planning problem for the agent such that the agent needs to find a series of actions to move from initial state $s$ to goal state $g$. The goals that fully comply with norms are assigned with *compliance level* 1. When a search for compliant states fails, the agent must proactively decide on remedial actions aimed at either preventing the user from going to a violating state, or mitigating the effects of a violation. In the norm literature these are called *contrary-to-duty obligations* [Prakken and Sergot, 1996]. For instance, a contrary-to-duty obligation in the escort scenario can be defined such that if a user is about to enter a conflict area without an escort, the agent must *alert* the user of the escort requirement. For such partial compliance cases, we assign compliance level 2.

A planning problem can be expressed as a pair of an initial state $s$ and a set of goal states $g_i$ annotated with their compliance levels $c_i$, such that $\langle s, \{(g_1, c_1)..., (g_m, c_m)\}\rangle$.

*Example 3 (Norm Reasoning).* Given a predicted plan-tree in Example 1, if variable *escort* for area 16 has value *init* indicating an escort has not been arranged, the agent detects a norm violation and thus searches for a compliant state as follows. Let us define the domain of agent-variable *escort* to be: $\{init, requested, granted, denied, alerted\}$. By alternating values, we get the following two compliant states:

$$\{(granted, 1), (alerted, 2)\},$$

where state *granted* is fully compliant while state *alerted* is partially compliant from the agent's perspective, as it complies with the contrary-to-duty obligation to warn the user. As a result, a newly generated planning problem is passed to the planner module as follows:

$$\langle \mathit{init}, \{(\mathit{granted}, 1), (\mathit{alerted}, 2)\}\rangle.$$

## 6 Planner and executor

As opposed to precomputing a policy for every possible case, we propose a scalable model where the assistant agent dynamically plans and executes a series of actions as new problems arise. Note that the issues regarding adjustable autonomy are outside the scope of this paper. Instead, we use a cost-based autonomy model where the agent is allowed to execute those actions that do not incur any cost, but is required to get the user's permission to execute costly (or critical) actions.

### 6.1 Planning

The agent has a set of executable actions. In the peacekeeping scenario, for instance, the agent has the following actions: {`send-request`, `receive-reply`, `alert-user`}. Given a planning problem–an initial and goal states–from the norm reasoner, the planner module is responsible for finding a series of actions to accomplish the specified goal. In Example 3, two goal (or absorbing) states have been assigned by the norm reasoner: an escort is granted or the user is alerted of the need for an escort. Thus, the agent needs to find a way to change the value of escort variable from *init* to either *granted* or *alerted*.

Since our representation of planning problems is generic, one may use classical planners in the implementation. Instead, we use an MDP to develop a planner in order to respect uncertainty involved in agent actions, *e.g.*, sending a request may fail due to a communication network failure.

Recall that a predicted user plan from the plan recognizer imposes deadline constraints (specified as the depth of node) to the agent's planning. Specifically, if the user is likely to commit a violation at a certain time step ahead, the agent must take actions to resolve the violation *before* the time step. In the planner, a deadline constraint is utilized to determine the horizon for an MDP plan solver, such that the agent planner needs to find an optimal policy given the time that the agent has until the predicted violation time.

In Example 3, when the violation is predicted far in advance, an optimal policy prescribes the agent to always request an escort from the other party, except if an escort request has been denied by the other party then the agent should alert the user of the denied request. Note that an optimal policy can change as time elapses, *i.e.*, as the future horizon shortens, the expected values of states change. For instance, the user is better off by being warned when there is not enough time left for the agent to arrange an escort. We compare the number of sequential actions in a plan with the depth of node (or the goal's deadline) to determine the plan's feasibility.

The planning problem formulated by the reasoner may not always be solvable; that is, a compliant state can only be accomplished by modifying those variables that the agent does not have access to, or none of the agent's actions has effects that results in the specified goal state. In this case, the agent notifies the user immediately so that the user can take appropriate actions on her own. Otherwise, the agent starts executing its actions according to the optimal policy until it reaches a goal state.

### 6.2 Execution

Execution of an agent action may change one or more variables. For each newly generated plan (or a policy) from the planner module, an executor is created as a new thread. An executor *waits* on a signal from the *variable observer* that monitors the changes in the environment variables to determine the agent's current state. When a new state is observed the variable observer *notifies* the plan executor to wake up. The plan executor then selects an optimal action in the current state according to the policy and executes the action. After taking an action, the plan executor is resumed to wait on a new signal from the variable observer. If the observed state is an absorbing state, then the plan execution is terminated, otherwise an optimal action is executed from the new state.

In order to handle unexpected exceptions during execution time, an executable action has a timeout such that when the execution of an action reaches its timeout the plan is aborted. When a plan is aborted the specific goals of the plan have generally not been achieved. If the goals are still relevant to the user's current plan (according to a newly predicted user plan), then the norm reasoner will generate them as new goals for the agent to accomplish.

The agent's plan can be updated during execution as more recent assessment of rewards arrives from the norm reasoner, forcing the agent to replan. For instance, after the agent requested an escort from the other party, the other party may not reply immediately causing the agent to wait on the request. In the meantime, the user can proceed to make steps towards the unsafe region, imposing a tighter deadline constraint. When the new deadline constraint is propagated to the planner, an optimal policy is updated for the executor, triggering a new action, *e.g.*, to alert the user of the potential violation (instead of trying to arrange an escort).

When alerted by the agent, the user may take certain actions to resolve the violation for herself, or alter her current plan to avoid the violation. In the worst case, the user may still proceed with the current plan and violate the norm. The agent is not penalized for such cases when we evaluate the agent's performance.

## 7 Applications

Through this research, we aim to make not only scientific contributions but also practical impact on realistic applications. The autonomous assistant agent framework that has been presented can be applied to various problem domains. Here, we include some examples of potential applications.

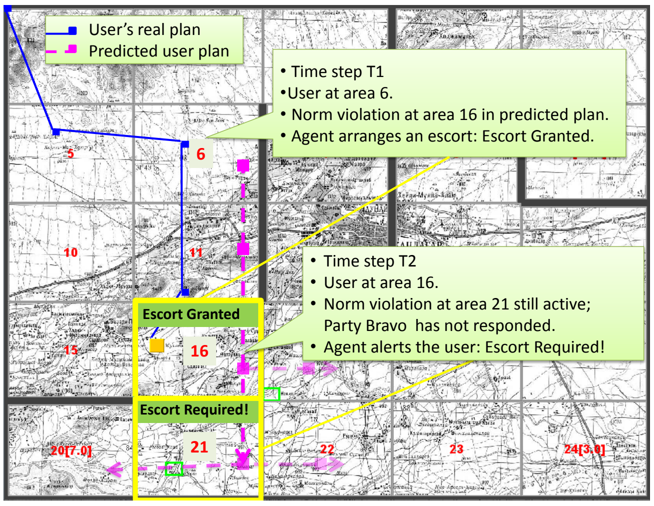
User's real plan
Predicted user plan
• Time step T1
•User at area 6.
• Norm violation at area 16 in predicted plan.
• Agent arranges an escort: Escort Granted.
• Time step T2
• User at area 16.
• Norm violation at area 21 still active; Party Bravo has not responded.
• Agent alerts the user: Escort Required!
Escort Granted
Escort Required!
5
6
10
11
15
16
20[7.0]
21
22
23
24[3.0]

**Fig. 2.** An annotated screenshot of a humanitarian party's planning interface

## 7.1 Military escort planning in peacekeeping

As a proof of concept prototype, our approach has been implemented in the context of planning for peacekeeping operations, in a scenario inspired by the work in [Sycara et al., 2010], where two coalition partners (a humanitarian party Alpha and a military party Bravo) plan to operate in the same region according to each party's individual objectives and a set of regulation rules.

Figure 2 shows a planning interface of a humanitarian party (Alpha). The figure is annotated with labels for illustration. At time step $T1$, the agent identifies a norm violation at area 16 in the predicted user plan, for which the agent sends an escort request to Bravo. When the agent receives a reply from Bravo granting a permission the escort status is displayed in the interface. Similarly, the agent sends an escort request for area 21 for another norm violation, but Bravo does not respond. At time step $T2$, an updated policy prescribes the agent to alert the user, and a warning is displayed in the interface.

We have used a simplified military escort planning scenario throughout this paper to illustrate our approach. In practice, the planning and scheduling of escort services in military peacekeeping operations involve complex norm reasoning due to diverse stakeholders. Through a series of collaborations with the military and various NGO groups, we have identified a significant amount of interest in developing software assistant for this problem domain, and we are currently working on scaling up the system to deal with more realistic settings.

### 7.2 Assistive living applications

It is important to note that the goal of this research is not to guide the user in finding optimal planning solutions, but instead, the agent aims to support the user's plan by identifying and making amends for the plan's weaknesses. As opposed to directing the user to make optimal decisions with respect to a certain objective (as in decision-support systems), we aim to design an agent that can maximize the support to help the user in making decisions based on her own criteria and judgement. From the user's perspective, independent decision making is crucial in many problem domains such as assistive living technologies for the disabled and the elderly.

In this domain, the norm rules can be defined to specify a set of prohibition rules for unsafe activities. When the agent predicts any potential dangers, the agent's new goal becomes restoring a safe state. For instance, if the safe state can be accomplished by taking the agent's available actions, *e.g.*, moving certain objects on the floor, the agent can resolve the issue. When the agent cannot accomplish the goal using its own capabilities, the agent can instead alert the human assistant before an accident happens.

Similarly with the assistive living applications, our approach can also be applied to other care-giving applications.

## 8 Conclusion and Future Work

In this paper, we presented an assistant agent approach to provide prognostic reasoning support for cognitively overloaded human users. We designed the proactive agent architecture by seamlessly integrating several intelligent agent technologies: probabilistic plan recognition, prognostic normative reasoning, and planning and execution techniques. Our approach presents a generic assistant agent framework with which various applications can be built as discussed in Section 7. As a proof of concept application, we implemented a coalition planning assistant agent in a peacekeeping problem domain.

Our approach has several advantages over existing assistant agent approaches. When compared to other decision-theoretic models, our approach is significantly more scalable because of exponential state space reduction discussed in Section 2. As opposed to assistant agent models where an agent takes turns with the user, our agent has more flexibility in its decision making because the agent can execute multiple plans asynchronously. More importantly, our agent is proactive in that the agent plans ahead of time to satisfy the user's forthcoming needs without a delay. Such proactive assistance is especially an important requirement in time-constrained real-life environments.

We made a specific assumption that agent variables are independent from user variables. We will investigate approaches to relax this assumption. Also, we will refine the algorithm for determining a plan's feasibility in Section 6.1 by estimating expected time required for each action. Furthermore, we plan to extend our approach to work in a multi-user, multi-agent setting where resolving a norm violation may involve multi-party negotiations. In addition, when there are more than one assistant agents, newly generated goals can be shared or traded among the agents. We will address these special issues raised in multi-agent settings in our future work.

## Acknowledgments

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

## References

[Armentano and Amandi, 2007] Armentano, M. G. and Amandi, A. (2007). Plan recognition for interface agents. *Artif. Intell. Rev.*, 28(2):131–162.

[Baker et al., 2009] Baker, C., Saxe, R., and Tenenbaum, J. (2009). Action understanding as inverse planning. *Cognition*, 31:329–349.

[Bellman, 1957] Bellman, R. (1957). A markov decision process. *Journal of Mathematical Mechanics*, 6:679–684.

[Boger et al., 2005] Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., and Mihailidis, A. (2005). A decision-theoretic approach to task assistance for persons with dementia. In *Proc. IJCAI*, pages 1293–1299.

[Farrell et al., 2005] Farrell, A. D. H., Sergot, M. J., Sallé, M., and Bartolini, C. (2005). Using the event calculus for tracking the normative state of contracts. *Int. J. Cooperative Inf. Syst.*, 14(2-3):99–129.

[Fern et al., 2007] Fern, A., Natarajan, S., Judah, K., and Tadepalli, P. (2007). A decision-theoretic model of assistance. In *Proc. of AAAI*.

[García-Camino et al., 2009] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. W. (2009). Constraint Rule-Based Programming of Norms for Electronic Institutions. *Journal of Autonomous Agents & Multiagent Systems*, 18(1):186–217.

[Hübner et al., 2010] Hübner, J., Boissier, O., Kitio, R., and Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400.

[Jones, 1990] Jones, A. J. I. (1990). Deontic logic and legal knowledge representation. *Ratio Juris*, 3(2):237–244.

[Lopez y Lopez and Luck, 2003] Lopez y Lopez, F. and Luck, M. (2003). Modelling norms for autonomous agents. In *Proceedings of the Fourth Mexican International Conference on Computer Science*, pages 238–245.

[Modgil et al., 2009] Modgil, S., Faci, N., Meneguzzi, F., Oren, N., Miles, S., and Luck, M. (2009). A framework for monitoring agent-based normative systems. In *Proc. of AAMAS*, pages 153–160.

[Oh et al., 2011] Oh, J., Meneguzzi, F., and Sycara, K. (2011). Probabilistic plan recognition for intelligent information agents. In *Proc. ICAART*.

[Prakken and Sergot, 1996] Prakken, H. and Sergot, M. J. (1996). Contrary-to-duty obligations. *Studia Logica*, 57(1):91–115.

[Ramírez and Geffner, 2009] Ramírez, M. and Geffner, H. (2009). Plan recognition as planning. In *Proc. IJCAI*, pages 1778–1783.

[Sycara et al., 2010] Sycara, K., Norman, T., Giampapa, J., Kollingbaum, M., Burnett, C., Masato, D., McCallum, M., and Strub, M. (2010). Agent support for policy-driven collaborative mission planning. *The Computer Journal*, 53(5):528–540.

[Vázquez-Salceda et al., 2005] Vázquez-Salceda, J., Aldewereld, H., and Dignum, F. (2005). Norms in multiagent systems: from theory to practice. *International Journal of Computer Systems Science & Engineering*, 20(4):225–236.

[von Wright, 1968] von Wright, G. H. (1968). *An Essay in Deontic Logic and the General Theory of Action.* North-Holland Publishing Company.