Prognostic Normative Reasoning

Jean Oh[a], Felipe Meneguzzi[b], Katia Sycara[a], Timothy J. Norman[c]

[a]*Robotics Institute*
Carnegie Mellon University
Pittsburgh, PA, USA
{jeanoh,katia}@cs.cmu.edu
[b]*School of Computer Science*
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, RS, Brazil
felipe.meneguzzi@pucrs.br
[c]*Dept. of Computing Science*
University of Aberdeen
Aberdeen, UK
t.j.norman@abdn.ac.uk

Abstract

Human users planning for multiple objectives in complex environments are subjected to high levels of cognitive workload, which can severely impair the quality of the plans created. This article describes a software agent that can proactively assist cognitively overloaded users by providing normative reasoning about prohibitions and obligations so that the user can focus on her primary objectives. In order to provide proactive assistance, we develop the notion of prognostic normative reasoning (PNR) that consists of the following steps: 1) recognizing the user's planned activities, 2) reasoning about norms to evaluate those predicted activities, and 3) providing necessary assistance so that the user's activities are consistent with norms. The idea of PNR integrates various AI techniques–namely, user intention recognition, normative reasoning over a user's intention, and planning, execution and replanning for assistive actions. In this article, we describe an agent architecture for PNR and discuss practical applications.

Keywords:

1. Introduction

Human users planning for multiple objectives in highly-complex environments are subjected to high levels of cognitive workload, which can severely impair the quality of the plans created. The cognitive workload is significantly increased when a user must not only cope with a complex environment, but also with a set of complex rules (or norms) that prescribe how the planning process must be carried out. For example, military planners during peacekeeping operations have to plan to achieve their own unit's objectives while following standing operating procedures that regulate how interaction and collaboration with Non-Governmental Organizations (NGOs) must take place. These procedures generally prescribe conditions upon which the military should perform escort missions, for example, to ensure that humanitarian NGO personnel are kept safe in conflict areas. To assist cognitively overloaded human users, we develop a software assistant agent that can proactively provide timely reasoning support.

In this article, we specifically aim to assist the user's *normative reasoning*. Norms generally define a set of rules that is enforced among the members of a society. In this article, normative reasoning refers to the reasoning about prohibitions and obligations specified in those rules of a society. A recent study shows that dynamically changing normative stipulations hinder human ability to plan to both accomplish goals and abide by the norms [1]. This result is not surprising, and the difficulty is intensified in a multi-national or multi-cultural team context that is common in various operations today. To help the users cope with cognitive overload, there have been growing interests in automated normative reasoning.

We acknowledge ITMAS 2011 as the forum in which the main ideas behind this paper were preliminary discussed.

Existing work on automated norm management relies on a deterministic view of the planning model [2], where norms are specified in terms of classical logic; in this approach, violations are detected only after they have occurred, consequently assistance can only be provided after the user has already committed actions that caused the violation [1]. By contrast, our approach aims to predict potential future violations and proactively take actions to help prevent the user from violating the norms.

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 that the assistance is ready by the time the user actually needs it. This desideratum imposes several technical challenges for the assistant agent 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* task, the key challenge we address here is to *identify* a new set of tasks for the agent–*i.e.*, the agent needs to figure out when and what it can do for the user. 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 tasks for the agent to prevent the occurrence of such norm violations. After identifying new tasks, the agent plans, executes, and replans a series of actions to perform the tasks. As the user's environment changes the agent continuously updates its predictions. Subsequently, the agent must frequently revise its plans 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.

We have published abstract descriptions of our approach in [3, 4]. Several readers of our previous publications expressed interests in the parts that were omitted due to space limitation. This article provides the detail of our approach that has not been fully published earlier. The main contributions of this article 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 normative 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. In order to avoid the agent's interference with the user's actions, although the agent can observe the variables representing the user's state, we assume that the user and the agent can act upon a disjoint set of variables. Thus, although the assistant does not directly enforce the norms, it tries to steer the user towards compliance through communication. 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 the 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 norms 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 in Section 8, and conclude the paper in Section 9.

2. Related Work

We develop the notion of prognostic normative reasoning by bridging two significant branches of AI research: plan recognition and normative reasoning. Here, we discuss only the work closely related to ours and point the readers to survey articles for broader background.

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. In order to recognize a user plan, the agent should have some model of

how the user typically performs certain tasks; for instance, given two locations the user may have a set of preferred routes to drive between the two locations. Such a model is referred to as a *plan library*, and it represents a set of alternative ways to solve a domain-specific problem. The majority of existing work in plan recognition relies on plan libraries; that is, plan recognition algorithms aim to find a plan in the library that best explains the observed behavior. For specific techniques, we refer the readers to survey articles such as [5].

Constructing a plan library is, however, an elaborate process. In order to facilitate the cumbersome step of building a plan library, recent work proposed the idea of formulating plan recognition as a planning problem. Notably, one approach uses classical planners [6] whereas the other approach uses decision-theoretic planners [7]. Following the plan recognition as planning principle, our approach utilizes a decision-theoretic planner, specifically a Markov Decision Process (MDP) [8]. The decision-theoretic planners aim to find an optimal plan with respect to the objective of maximizing a discounted long-term reward (or minimizing a cost). The work presented in [7] uses an MDP to represent how people recognize the plans of others. During their experiments, human subjects watch an agent moving in a 2-dimensional space and are asked to predict the agent's goal positions. Their results show that decision-theoretic predictions match well with those of the human subjects. In this respect, our approach is to design a software assistant to make predictions in a similar manner to a human assistant.

2.2. 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 [9]. These concepts represent, respectively, situations that must, must not and can be the case in the world for it to comply with the societal rules. Mathematical study of norms has been carried out in the context of deontic logic [10], 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 [11], as well as practical approaches to reasoning about norms within individual agents [12] and in a society [13]. 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 normative constraints, we leverage the formalisms to create an internal representation of the norms that the assistant must consider when providing assistance.

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* [14] and describes which agents are complying (or violating) which norms. Although various approaches to norm monitoring have been proposed [14, 2, 15], 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. In this regard, deterministic norm representations fail to address an important need of preventions. By contrast, we take a probabilistic approach to be able to specify the agent's belief about potential norm violations so that the agent can proactively take preventive actions for the norms with high probabilities of violation.

3. Agent Architecture

In this section we describe a system architecture for a proactive yet unobtrusive assistant agent. Figure 1 illustrates the agent architecture that is composed of the following main modules: observer; plan recognizer; norm reasoner; planner; executor and presenter. We describe each module and how the components interact, followed by a set of design assumptions and a scenario example.

3.1. Modules

Observer: The agent monitors the user's current activities to identify anything that might indicate changes in the user's current and future plan. Such indicators are referred to as *observations*. The observer is responsible for receiving user input and

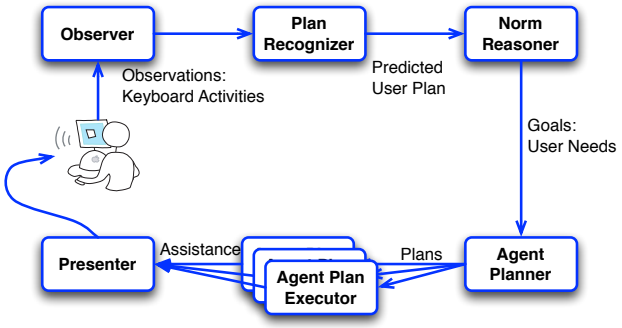

Figure 1: Overview of the proactive assistant.

translating it into observations that the plan recognizer module can utilize. The types of observations are domain specific, and thus must be well defined through a domain engineering process. When a new observation is made it is passed to the plan recognizer.

Plan Recognizer: The agent interprets new observations to make (or update) its predictions on what the user plans to do. When the agent receives a new observation from the observer, the plan recognizer updates the probability distribution over a set of states, known as a *belief state*, that represents the agent's belief about the user's true state. Given a belief state, the plan recognizer identifies most likely plans from the current belief state and constructs a predicted user plan referred to as a *plan-tree*. Nodes in the resulting plan-tree include the expected state induced by the action in the user's plan and a probability estimation that the user will actually visit the state. This plan-tree is fed to the norm reasoner.

Norm Reasoner: The agent evaluates the predicted user plan to detect any potential norm violations ahead of time. Norms specify desirable and undesirable values for state variables under specific circumstances, so violations occur when these state variables are in particular configurations. Thus, once the plan recognizer has computed the set of possible future states the user is likely to reach, given its predicted actions, the norm reasoner evaluates the state variables in each of these states for norm violations. When violations are detected, the norm reasoner tries to find the nearest non-violating states, which are then sent to the planner as the agent's next goals.

Planner: For each predicted plan step where norms would be violated, the agent plans–that is, decides which actions to take–to prevent potential norm violations. Once the norm reasoner detects potential norm violations in the predicted states for the user and finds the nearest compliant states, the agent planner uses a planner to find actions to steer the user away from violating states. These individual plans are then sent to the plan executor for execution.

Plan Executor: The agent executes a prevention plan to avoid the predicted violation from happening. Once plans to avert violations have been generated by the planner, the plan executor is responsible for carrying them out. When the agent's prediction changes because of newly developed observations, *e.g.*, the predicted violation is no longer eminent, the executor marks the correponding plans as inconsistent, aborting their executions.

Presenter: The agent reports to the user by presenting which preventive actions the agent has taken to resolve a certain violation. In many situations, even if the user is under risk of violating a norm, it might be undesirable to allow the agent to autonomously carry out actions to prevent a violation (e.g. when agent actions are costly). In these cases, the agent will be limited to advising the user of the imminence of a violation, and let the user decide on whether to take corrective action or not. Thus, the presenter is responsible for informing the user about imminent norm violations as well as corrective steps that the user may choose to take.

3.2. Design Assumptions

The agent models a user's planning space in terms of a set of task-specific *variables* and their *domains* of valid values, where a *variable* describes an environment and the progress status of certain activities. A set of norms specifies which states must be visited (or avoided) in the user plan using a set of variables and their relationships. In general, such norms introduce additional variables to consider in addition to task-specific ones, adding extra dimensions into the reasoning process. As seen in a recent study [1], when planning involves complex reasoning as in military environments, human users tend to lose track of norms, 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.

We assume independence of variables on the agent side as a means of preventing the agent from acting concurrently and possibly competitively with the user. In the agent architecture, whereas the state space of the (user) plan recognizer is defined in terms of user variables that of the (agent) planner is in terms of agent variables. The norm reasoner aligns the two components since the norms are defined in terms of both user and agent variables.

3.3. 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 stating 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 ensure 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.

The foci of this article are on three main components within the agent, namely: plan recognizer, norm reasoner, and agent planner and executor. We plan to study the observer and presenter modules in our future work.

4. PROBABILISTIC PLAN RECOGNIZER

This section describes our plan prediction algorithm that enables the assistant agent to predict the user's future plans. The basic idea is to develop a computational model that resembles a human user's decision-making process so that an assistant agent can use this model to predict the human user's future decisions. We use a Markov Decision Process (MDP) [8] to represent a user's decision-making model. The use of an MDP is justified as follows. In the problem domain of our interest, *e.g.*, military operations, the users have a strong objective of accomplishing a set of goals that are clearly defined. Thus, we can assume that a user is executing a sequence of planned actions; that is, the user must have *planned* the observed actions. In order to model the user's planning process we consider an AI planner. Instead of constructing a plan library (a typically cumbersome process), we can generate a set of alternative plans by solving a user's planning problem. At the same time, we need a model that can capture the *nondeterministic* nature of real-life applications. Since an MDP is a stochastic planner it suits both of our purposes.

An MDP is formally defined as:

Definition 1 (MDP). *A Markov Decision Process (MDP) is represented as a tuple $\langle S, A, r, T, \gamma \rangle$, where S denotes a set of states; A, a set of actions; $r : S \times A \rightarrow \mathbb{R}$, a function specifying a reward of taking an action in a state; $T : S \times A \times S \rightarrow \mathbb{R}$, a state transition function; and γ, a discount factor indicating that a reward received in the future is not worth as much as an immediate reward.*

Solving an MDP generally refers to a search for a *policy* that maps each state to an optimal action with respect to a discounted long-term expected reward.

Without loss of generality we add a simplifying assumption that the reward function depends only on the states, such that given state s reward $r(s) = r(s, a)$ for all actions a in A. Although the MDP literature sometimes refers to a goal state as being an absorbing or terminal state, that is, state s with $T(s, a, s') = 0$ for all a and for all s', that is a state with no possibility of leaving it, we mean a goal state to be a state with a positive reward, that is any state s with $r(s) > 0$.

Example 1. *Consider a user who is navigating a maze to reach a destination. In this example, we can design an MDP representing the user's decision making problem as follows: the* state *space can be defined in terms of location coordinates; the user can take* actions *to move to a new location while the new location after taking an action is nondeterministic (e.g., the user may unsuccessfully attempt to move to the left); and the user is awarded* rewards *when she arrives at the destination. Solving this MDP finds a policy that prescribes which move the user should make at each location.*

Let $\Phi = \langle S, A, r, T, \gamma \rangle$ denote an MDP representing the user's planning problem. The plan recognition

algorithm is a two-step process. The agent first estimates which goals the user is trying to accomplish and then predict a sequence of possible plan steps that the user is most likely to take to achieve those goals. We first describe the algorithm assuming that the user's current state is fully observable and that the user does not change goals. These assumptions will later be relaxed as described in Section 4.3 and Section 4.4, respectively.

4.1. Goal prediction

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.

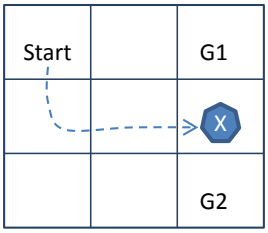

Figure 2: An example of goal prediction

Example 2. *The basic idea can be illustrated using a simple example shown in Figure 2 where the user in a* 3×3 *grid has moved from the top left corner to the current state denoted by* X *that is adjacent to two goal states* $G1$ *and* $G2$*. If only the current state is considered, the user may move up or down with equal probability because both actions are optimal in the current state. However, the observed trajectory is closer to the optimal behavior for aiming at* $G2$ *rather than* $G1$ *(since there is a shorter path to* $G1$ *from the starting position). In other words, the conditional probability of the user pursuing goal* $G2$ *is higher than that of* $G1$ *given the observed path. Our algorithm is thus to update the conditional probability of each goal given a sequence of observations.*

We define set G of possible goal states as all states with positive rewards such that $G \subseteq S$ and $r(g) > 0, \forall g \in G$. The algorithm initializes the probability distribution over the set G of possible goals, denoted by $p(g)$ for each goal g in G, proportionally to the reward $r(g)$: such that $\sum_{g \in G} p(g) = 1$ and $p(g) \propto r(g)$.

Algorithm 1 An algorithm for plan recognition

```
1: function PREDICTFUTURESTEPS(goals G, observations O)
2:     plan-tree t ← createNewTree()
3:     root-node n ← getRootNode(t)
4:     current-state s ← getCurrentState(O)
5:     for all goal g ∈ G do
6:         w_g ← p(g|O_t) /* Equation (1) */
7:         BLDPLANTREE(t, n, π_g, s, w_g, 0)
8:     end for
9:     return t /* predicted plan-tree */
10: end function
```

The algorithm then computes an optimal policy π_g for every goal g in G, considering the positive reward only from the specified goal state g and zero rewards from any other states $s \in S \wedge s \neq g$.

We use a variation of the *value iteration* algorithm [8] to compute an optimal policy [8]. For each potential goal $g \in G$, the algorithm computes a goal-specific policy π_g to achieve goal g. Following the assumption that the user acts more or less rationally, this policy can be computed by solving the MDP to maximize the long-term expected rewards. Instead of a deterministic policy that specifies only the best action that results in the maximum reward, we compute a stochastic policy such that probability $p(a|s,g)$ of taking action a given state a when pursuing goal g is proportional to its long-term expected value $v(s,a,g)$:

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

where β is a normalizing constant. Note that this step of computing optimal policies is performed only once and can be done off-line, and the resulting policies are also used in the second step that will be described in Section 4.2.

Let $O_t = s_1, a_1, s_2, a_2, ..., s_t$ denote a sequence of observed states and actions from time steps 1 through t where $s_{t'} \in S, a_{t'} \in A, \forall t' \in \{1, ..., t\}$. Here, the assistant agent needs to estimate the user's targeted goals.

After observing a sequence of user states and actions, the assistant agent updates 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 the Bayes rule as:

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

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

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

By the MDP problem definition, the state transition probability is independent of the goals. By the Markov assumption, the state transition probability is also independent of any past states except the current state, and the user's action selection depends only on the current state and the specific goal. By using these conditional independence relationships, we get:

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

where the probability $p(a|s, g)$ represents the user's stochastic policy $\pi_g(s, a)$ for selecting action a from state s given goal g that has been computed at the initialization step.

By combining Equation 1 and 2, the conditional probability of a goal given a series of observations can be obtained. We use this conditional probability to assign weights when constructing a predicted plan-tree in the next step.

The algorithmic complexity of solving an MDP using value iteration is quadratic in the number of states and linear in the number of actions. Here, the optimal policies for candidate goals can be precomputed off-line. Thus, the actual runtime complexity of our goal recognition algorithm is linear in the number of candidate goals and the number of observations.

4.2. Plan-step prediction

Based on the predicted goals from the first step, we now generate a set of possible scenarios that the user will follow. Recall that we solved the user's MDP Φ to get stochastic policies for each potential goal. The intuition for using a stochastic policy is to allow the agent to explore multiple likely plan paths in parallel, relaxing the assumption that the user always acts to maximize her expected reward.

Algorithm 2 Recursive building of a plan tree

```
1: function BLDPLANTREE(plan-tree t, node n,
   policy π, state s, weight w, deadline d)
2:     for all action a ∈ A do
3:         w' ← π(s, a)w
4:         if w' > threshold θ then
5:             s' ← sampleNextState(state s, action a)
6:             node n' ← createNewNode(s', w', d)
7:             addChild(parent n, child n')
8:             BLDPLANTREE(t, n', π, s', w', d + 1)
9:         end if
10:    end for
11: end function
```

Using the MDP model and the set of stochastic policies, we sample a tree of most likely sequences of user actions and resulting states from the user's current state, known here as a *plan-tree*. In a predicted plan-tree, a *node* contains the resulting state from taking a predicted user action, associated with the following two features: *priority* and *deadline*. We compute the *priority* of a node from the probability representing the agent's belief that the user will select the action in the future; that is, the agent assigns higher priorities to assist those actions that are more likely to be taken by the user. On the other hand, the *deadline* indicates the predicted time step when the user will execute the action; that is, the agent must prepare assistance before a certain time point by which the user will need help.

The algorithm builds a plan-tree by traversing the most likely actions (to be selected by the user) from the current user state according to the policy generated from the MDP user model. First, the algorithm creates a root node with probability 1 with no action attached. Then, according to the MDP policy, likely actions are sampled such that the algorithm assigns higher priorities to those actions that lead to a better state with respect to the user's planning objective. Note that the algorithm adds a new node for an action only if the agent's belief about the user's selecting the action is higher than some threshold θ; actions are pruned otherwise. The recursive process of predicting and constructing a plan tree from a state is described in Algorithm 2. The algorithmic complexity of plan generation is linear in the number of actions. The resulting plan-tree represents a horizon

of sampled actions and their resulting states for which the agent can prepare appropriate assistance.

Henceforth, we represent a plan-tree node in a tuple $\langle t, s, l\rangle$ denoting 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. A user action is omitted in the representation for simplicity because normative reasoning is performed based on the resulting states regardless of causing actions. In general, other types of assistance can be performed based on user actions, *e.g.*, an action may be associated with information needs.

Example 3 shows a segment of plan-tree for the scenario in Section 3.3 indicating that the user is likely to be in area 16 with probability .8 or in area 15 with probability .17 at time step t_1.

Example 3.
$\langle\langle t_1, (\texttt{area} = \texttt{16}), .8\rangle,$
$\langle t_1, (\texttt{area} = \texttt{15}), .17\rangle\rangle$

4.3. Handling partial observability

Hitherto we have described algorithms based on the agent's full observability on user states. We extend our approach to handle a partially observable model for the case when the assistant agent cannot directly observe the user states and actions. Instead of observing the user's states and actions directly, the agent first infers the user's current state from indirect observations about the user's environment. The agent maintains a probability distribution over the set of user states, known as a *belief state*, that represents the agent's belief regarding the user's current state. The agent updates its belief state as it receives indirect observations from the user or the environment, *e.g.*, keyboard and mouse inputs from the user's computing environment or sensory inputs from various devices. For instance, if no prior knowledge is available the initial belief state can be a uniform distribution indicating that the agent believes that the user can be in any state. The fully observable case can also be represented as a special case of belief state where the whole probability mass is concentrated in one state.

To update a belief state, we use a variation of the *forward* algorithm [16], which we briefly sketch here. Let s_t denote the user's state at time t; $b = [b_1, ..., b_{|S|}]$, a belief state where $b(s) = p(s_t = s)$ is the belief probability of that the user is in state s at current time t; and $z_1, ..., z_t$, a series of observations from time step 1 through time step t. We assume that an initial probability $O(z|s)$ of the agent sensing observation z in state s is known (or it can be learned off-line). For each state $s \in S$, the algorithm updates the probability of being in state s given a sequence of observations $z_1, ..., z_t$, denoted by $p(s_t = s|z_1, ..., z_t)$.

In order to compute this value efficiently, the algorithm utilizes the joint probability that the user reaches state s at time t after collecting observations $z_1, ..., z_t$, denoted by $\alpha_s(t) = p(z_1, ..., z_t \wedge s_t = s)$. The algorithm operates as follows. Given the first observation z_1 and initial belief state b, the initial α values at time step 1 can be computed for all states $s \in S$ as: $\alpha_s(1) = O(z_1|s_1 = s)b(s)$. As the agent receives a new observation z, the α values are updated by recursively combining the previous alpha values of all incoming states with the probabilities of sensing the new observation in each state:

$$\alpha_s(t+1) = O(z|s)\sum_{s' \in S} T'(s', s)\alpha_{s'}(t),$$

where we estimate the state transition probability $T'(s'|s)$ by combining state transition function $T(s'|s, a)$ of the MDP user model and the optimal policy π precomputed off-line. By summing up transition probabilities $T(s'|s, a)$ for all the actions dictated by policy π we get:

$$T'(s'|s) = \sum_{a \in A} \pi_s(a)T(s'|s, a). \tag{3}$$

Finally, the belief state can be updated by normalizing the current α values using the following equation:

$$b(s) = \frac{\alpha_s(t)}{\sum_{s' \in S} \alpha_{s'}(t)}.$$

The belief state is updated whenever the agent receives a new observation.

Finally, Algorithm 1 for predicting the future plan can be modified as follows. Line 4 for getting the user's current state from the last observation is replaced with updating a belief state using the observations. Line 7 for constructing a plan tree for each goal is substituted by constructing a plan tree for each goal and for each state; here, the weight parameter is also adjusted by multiplying the weight of the goal by the belief probability of state. Thus, the possible future steps are predicted from those states that the agent strongly believes where the user currently is.

4.4. Handling changing goals

The user may change a goal during execution, or the user may interleave plans for multiple goals at the same time. Our algorithm for handling changing goals is to discount the values of old observations as follows. The likelihood of a sequence of observations given a goal is expressed in a product form such that $p(O_t|g) = p(o_t|O_{t-1}, g) \times ... \times p(o_2|O_1, g) \times p(o_1|g)$. In order to discount the mass from each observation $p(o_t|O_{t-1}, g)$ separately, we first take the logarithm to transform the equation to a sum of products, and then discount each term as follows:

$$\begin{aligned} log[p(O_t|g)] &= \gamma^0 log[p(o_t|O_{t-1}, g)] + \\ &... \quad +\gamma^{t-1} log[p(o_1|g)], \end{aligned}$$

where γ is a discount factor in range $(0, 1)$ such that the most recent observation is not discounted and the older observations are discounted exponentially. Since we are only interested in relative likelihood of observing the given sequence of states and actions given a goal, such a monotonic transformation is valid (although this value no longer represents a probability).

The basic idea is to discount old observations, so naturally the algorithm considers newer observations, which must be consistent with newer goals, with higher weights. This approach can handle gradually changing goals but does not perform well when the user is switching goals frequently or interleaving multiple goals.

5. Norm Reasoner

In this section we specify the component responsible for using normative reasoning to generate new goals for the agent. Norms generally define constraints that should be followed by the members in a society at particular points in time to ensure certain system-wide properties. The user's state space is defined in terms of a set of variables describing the user's environment and the progression of her activities, formally defined as follows:

Definition 2 (State Variables). *Let S be the set of states used in the user planning model described in Section 4. Each state $s \in S$ represents a complete assignment to the set of random variables $\vec{\varphi} = \{\varphi_1 \dots \varphi_n\}$ describing the properties of the environment at any given time.*

We specify our norm representation format, followed by two algorithms for 1) predicting violations and 2) finding the nearest complying state–*i.e.* the agent's new goal state–towards which we can steer the user.

5.1. Norm Representation

Inspired by the representation in [13], 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 3 (Norm). *A norm is a tuple $\langle \nu, \alpha, \mu \rangle$ where ν is the deontic modality; α, the context condition; and μ, the normative condition.*

The conditions used in a norm are specified in terms of a set of *domains* for the random variables that compose a state analogously to the specification of constraint domains in constraint programming. Thus, the norms considered by our assistant refer to states rather than to actions, as alternatively used by recent work on normative reasoning.

Definition 4 (Satisfaction). *Let φ be the set of state variables; α, a context (or normative) condition containing m variables $\varphi_\alpha \subseteq \varphi$ and their valid domain D of m-tuples. We say that condition α is* satisfied *in state s (written $s \models \alpha$) if there exists a tuple in the valid domain that is consistent with the variable assignment in state s; such that $\exists d \in D \ \wedge \ \forall v \in \varphi_\alpha,\ d(v) = s(v)$ where $d(v)$ and $s(v)$ denote the value assignments for variable v in tuple d and state s, respectively.*

Example 4. *Coming back to the peacekeeping operations scenario introduced earlier, consider that there is an obligation that requires the NGO to have an escort to operate at certain high-risk areas designated by grid coordinates. Further, consider that the states over which the NGO plans are defined in terms of two variables: area indicating the location of the user,*

and escort indicating the status of an escort arrangement in the specified area. Therefore, a state can be written as a pair $(area, escort)$*. The norm that an NGO is obliged to have an armed escort when entering unsafe regions, denoted by* ι_{escort}*, can be expressed as:*

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

Thus, the example above denotes that regions 16 *and* 21 *should not be entered without an escort (as they are unsafe). Then, the context condition is satisfied when variable* `area` *(indicating the user's location) has the value of* 16 *or* 21*.*

5.2. Detecting Violations

Given the norm representation of Definition 3, we define a norm violation as consisting of an agent being in a state that is relevant to a norm and that is also violating a normative condition. We say a state is *relevant* to a norm if this state supports the context condition α specified in the norm. Violation of a normative condition depends on the type of norm being evaluated, which in this paper is either an obligation or a prohibition. Obligations are norms that require certain properties of the world to have particular values, thus an agent is violating an obligation if it is in a norm-relevant state and if the normative condition is not supported by this state. Specifically, an obligation is *violated* if the normative condition μ is not satisfied in state s; *i.e.*, $s \not\models \mu$. Conversely, prohibitions are norms that specify properties that *should not* be the case, consequently, an agent is violating a prohibition if it is in a norm-relevant state and if the normative condition is supported by that state. Thus, a prohibition is violated if the normative condition is satisfied in state s such that $s \models \mu$. Formally, a violating state is defined below.

Definition 5 (Violating State). *Let* $s_i \in S$ *be a state and* ι*, a norm* $\langle \nu, \alpha, \mu \rangle$ *with normative stipulation* ν*, context condition* α *and normative condition* μ*. We say that norm* ι *is relevant in state* s *if and only if the condition in* α *is satisfied by the assignment of variables in* s*, so that* $s \models \alpha$*. Furthermore, we say that a state* s *is violating norm* ι *(represented as* $violating(s, \iota)$*), if and only if the activation condition is valid and if the normative condition is either true for prohibitions and false for obligations. We can represent this condition as a function as follows:*

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

For instance, considering norm ι_{escort} in Example 4, given state $s = \{(\texttt{area} = \texttt{16}), (\texttt{escort} = init)\}$ the violation detection function $violation(s, \iota_{escort})$ would return 1, denoting that norm ι_{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. In terms of complexity, the calculation of the algorithmic complexity of the reasoning as put forth in this article depends on two elements. First, the state space searched by the norm reasoner will be exactly the size of the predicted plan tree described in Section 4.2. Thus, the space complexity of the predicted plan is similar to that of iterative deepening search where the number of nodes in the tree is $O(bd)$, where b is the maximum branching factor in the domain, and d is the depth of the deepest node in the tree. We know d is bounded, since at each subsequent tree level, the probability of an action choice monotonically decreases, as it is always multiplied by the probability of an action choice in the previous tree level. Second, the verification of compliance for a single norm can be carried out in constant time, using a canonical representation for the formulas describing state (e.g. using a hash of the valid state attributes or a OBDD representation [17]). Thus, the complexity of norm reasoning will be $O(bdn)$, with n being the number of norms being considered.

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 where $violating(s, \cdot) = 1$, the agent is to find the nearest state g that satisfies $violating(g, *) =$

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.

Norm violations occur as the result of certain variables in the state space being in particular configurations. Thus, finding compliant states can be intuitively described as a search for alternative value assignments for the variables in the normative condition such that norms are no longer violated. This 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 such 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 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* [18], obligations that come into force when a norm violation has occurred (*e.g.*, you must pay a fine if you park in a forbidden area, etc.). For instance in the escort scenario, a contrary-to-duty obligation 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. In this case, the user has initially violated a norm (prohibition to enter an area without escort), but the agent has complied with the contrary-to-duty obligation to alert the user of this requirement. Thus, in cases where the user has violated a norm while agent has complied with its contrary-to-duty norms, we say that the system is in partial compliance, for which 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 5 (Norm Reasoning). *Given a predicted plan-tree in Example 3, 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:

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

6. Planner and Executor

We propose a scalable model where the assistant agent dynamically plans and executes a series of actions to solve smaller problems as they 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. For instance, in the peacekeeping scenario the set of agent actions are the following:

- send-request
- receive-reply
- alert-user

Given a planning problem–*i.e.*, an initial and a goal states–from the norm reasoner, the planner module is responsible for finding a series of actions to accomplish these goals. In Example 5, 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 must plan 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 5, 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, *e.g.*, 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 result 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.

The agent's plan can be updated during execution as more recent assessments of rewards arrive 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).

7. Theoretical performance analysis

In this section, we evaluate the theoretic properties of our approach and discuss expected performance.

The assistant agent's goal is to reduce the penalty incurred to a human user due to norm violations. Without loss of generality, let us assume that a penalty for violating any norm is a constant, denoted by c, and that the penalty is accumulative such that if a user violates n norms, the total penalty incurred would be nc. Since the agent acts based on uncertain predictions over user behavior, the agent is also subject to a second type of penalty, denoted by c', due to wrong actions, *e.g.*, there may be a cancelation fee if the agent has requested an escort unnecessarily.

Let tp refer to the number of true-positive instances that an (human or software) agent correctly identify forthcoming norm violations, and fp, the number of false-positive cases where the agent incorrectly predicts norm violations. Similarly, let tn and fn denote true-negative and false-negative cases, respectively. The *recall* rate r is defined as the ratio of true-positives to the total true counts, such that $\frac{tp}{tp+fn}$. Conversely, the *precision* rate p is defined as the ratio of true-positives to the total positive counts, such that $\frac{tp}{tp+fp}$.

Assuming that a human user's errors are mostly due to false-negatives (*i.e.*, by forgetting to call for escort), we can say that cognitively overloaded hu-

man users have extremely low recall rates. Given this assumption, we can simplify our evaluation by dropping the second type of errors in the human's case (*i.e.*, humans request for an escort only when needed). Then, the expected penalty of a human user per decision is $c(1-r_h)$ where r_h is the human's recall rate. On the other hand, the agent's penalty reflects the tradeoffs between precision and recall rates as follows: $c(1-r_a)+c'(1-p_a)$. Generally, when recall is increased precision is decreased, and vice versa. Because penalties of the second type due to norm violation are significantly smaller than the first type such that $c' < c$, improving the recall rate is expected to result in a cut in penalty. Although the agent's actual performance depends on specific problem domains, we can state that the agent's assistance can reduce the penalty incurred as long as the agent's recall rate is higher than that of a human user and the second type of penalty (*e.g.*, cancelation fee) is significantly lower than the norm violation penalty. In the problem domains of our interest, both conditions can be reasonably met as tested in our simplified examples in 8. Empirical study for supporting this claim is in our future work.

8. Applications

Through this research, we aim to make not only scientific contributions but also practical impact on real-life 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.

8.1. Military escort planning in peacekeeping

We have implemented our approach as a proof of concept prototype in the context of planning for peacekeeping operations, in a scenario adapted from [1], 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 regulations.

In this scenario, a user interacts with a user interface simulating the decisions made by the head of a humanitarian mission in the field, including issue movement orders and requests for military escort in the area of operations. The *Observer* component (c.f. Section 3.1) collects all movement orders and communication generated by the human operator, forwarding this information to the *Plan Recognizer*, which

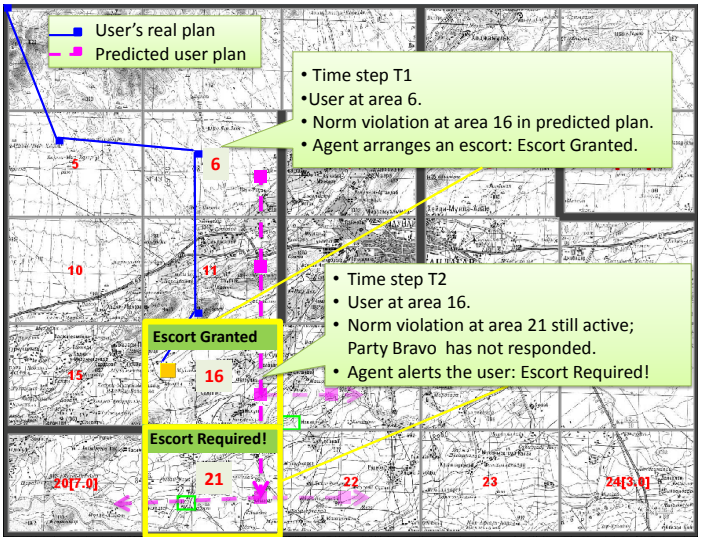

Figure 3: An annotated (and cropped) screenshot of a humanitarian party's planning interface

generates a tree of predicted user actions. This predicted plan tree is sent to the *Norm Reasoner* to determine which future states are norm-compliant and which ones result in violations. If the predicted plan tree includes violating states, these states and their likelihood are sent to the *Agent Planner* to allow the agent to deal with the contrary to duty obligations resulting from the user's violations by planning remedial actions. In the case of our scenario, whenever the user's predicted plans involve movements to areas for which the user has not yet requested an escort, the agent takes on the obligation to warn the user of this violation. These warning actions are queued by the *Agent Plan Executor* and are executed whenever the user's likelihood of entering an area that requires an escort goes above a certain threshold. When the *Agent Plan Executor* determines that a warning should be issued, the warnings are forwarded to the *Presenter*, which cause the User Interface to highlight areas for which escorts are needed.

Figure 3 shows the planning interface of a humanitarian party (Alpha), annotated with labels for illustration (This figure is best viewed in color). Annotations are drawn in the light green dialogue boxes, and everything else is part of the actual implemented interface. 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 in-

terface. 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. By meeting with the US military and various NGO representatives, we have identified a significant amount of interest in developing a software assistant for this problem domain, and we are currently working on scaling up the system to deal with more realistic settings.

8.2. Potential 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, to provide support to the user's planning by identifying and making amends for weaknesses in current plans. 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. Critically, the research presented in this paper is intended to help unburden a user from having to deal with a large number of dynamically changing norms. For instance, in military-civilian collaboration planning, each planner is expected to remember and take into account a large number of directives that change dynamically as the security situation evolves in a war zone. From the user's perspective, independent decision making is crucial, as many rules guiding this kind of collaboration might not necessarily be formalized, so a fully automated planning system would not be suitable. One experimental deployment of our intention recognition approach has been done in the context of disaster response management, with preliminary results showing that even a relatively minor level of assistance can lead to improved reaction times [19].

Furthermore, our research can be applied in many other problem domains such as assistive living technologies for the disabled and the elderly. In this domain, the norms can be defined to specify a set of prohibitions 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.

9. Discussion 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 8. 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. As a basis of comparison, the work of Fagundes *et al.*[20] proposes reasoning about norm violations using MDPs by taking the original MDP representing the world dynamics and creating new MDPs, one for each norm, that include a modified reward function to account for the penalties and rewards resulting from violating and fulfilling each norm, as well as a modified transition function to account for potential restrictions imposed to an agent that violates a norm. This necessitates solving a new MDP every time the set of norms in effect changes. Although in its most recent iteration, Fagundes *et al.*[21] mitigates the problem of having to solve multiple MDPs, changing norms still require the entire MDP to be solved again. When compared to other decision-theoretic models such as [20, 21], our approach is significantly more scalable because of the exponential state space reduction due to the isolation of agent variables from user variables. The technique presented in this paper offers the advantage that the MDP needs to be solved only when the user's decision model changes, and not when the set of effective norms changes. 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 an especially important requirement in time-constrained real-life environments.

We made a specific assumption that agent variables are independent from user variables. For example, in the use case of Section 8, if *escort* is a user variable then the agent does not have the ability to make a request for escort on behalf of the user, in which case, the agent can only provide warnings to the user (thus affecting agent-only variables). Dropping this assumption would entail research in another very complex research area, adjustable autonomy [22], and has been left as future work.

Possible future work involves investigating ways to relax this assumption, refine the algorithm for determining a plan's feasibility in Section 6.1 by estimating expected time required for each action. Furthermore, our approach could be extended in multiple ways. First it could be extended 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. Second, the norm representation could be extended to consider actions, as is done in [2, 21].

Acknowledgments

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 hereon.

References

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

[2] S. Modgil, N. Faci, F. Meneguzzi, N. Oren, S. Miles, M. Luck, A framework for monitoring agent-based normative systems, in: Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems, 2009, pp. 153–160.

[3] J. Oh, F. Meneguzzi, K. Sycara, Probabilistic plan recognition for intelligent information assistants, in: ICAART, 2011.

[4] J. Oh, F. Meneguzzi, K. Sycara, T. Norman, An agent architecture for prognostic reasoning assistance, in: Proc. IJCAI, 2011.

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

[6] M. Ramírez, H. Geffner, Plan recognition as planning, in: Proc. IJCAI, 2009, pp. 1778–1783.

[7] C. Baker, R. Saxe, J. Tenenbaum, Action understanding as inverse planning, Cognition 31 (2009) 329–349.

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

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

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

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

[12] F. Lopez y Lopez, M. Luck, Modelling norms for autonomous agents, in: Proceedings of the Fourth Mexican International Conference on Computer Science, 2003, pp. 238–245.

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

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

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

[16] L. Rabiner, A tutorial on HMM and selected applications in speech recognition, Proc. of IEEE 77 (2) (1989) 257–286.

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

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

[19] F. Meneguzzi, S. Mehrotra, J. Tittle, J. Oh, N. Chakraborty, K. Sycara, M. Lewis, A cognitive architecture for emergency response, in: Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems, 2012, pp. 1161–1162.

[20] M. S. Fagundes, H. Billhardt, S. Ossowski, Reasoning about norm compliance with rational agents, in: H. Coelho, R. Studer, M. Wooldridge (Eds.), ECAI, Vol. 215 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2010, pp. 1027–1028.

[21] M. S. Fagundes, S. Ossowski, M. Luck, S. Miles, Using normative markov decision processes for evaluating elec-

tronic contracts, AI Communications 25 (1) (2012) 1–17.
[22] M. Tambe, P. Scerri, D. Pynadath, Adjustable autonomy for the real world, Journal of Artificial Intelligence Research 17 (2002) 171–228.