Revisiting Asimov's First Law: A Response to the Call to Arms

David V. Pynadath and Milind Tambe

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
{pynadath,tambe}@isi.edu

Abstract. The deployment of autonomous agents in real applications promises great benefits, but it also risks potentially great harm to humans who interact with these agents. Indeed, in many applications, agent designers pursue adjustable autonomy (AA) to enable agents to harness human skills when faced with the inevitable difficulties in making autonomous decisions. There are two key shortcomings in current AA research. First, current AA techniques focus on individual agent-human interactions, making assumptions that break down in settings with teams of agents. Second, humans who interact with agents want guarantees of safety, possibly beyond the scope of the agent's initial conception of optimal AA. Our approach to AA integrates Markov Decision Processes (MDPs) that are applicable in team settings, with support for explicit safety constraints on agents' behaviors. We introduce four types of safety constraints that forbid or require certain agent behaviors. The paper then presents a novel algorithm that enforces obedience of such constraints by modifying standard MDP algorithms for generating optimal policies. We prove that the resulting algorithm is correct and present results from a real-world deployment.

1 Introduction

Agent applications, such as intelligent homes [6], smart organizations [8], and space missions [3], offer a vision of agent teams deployed to support critical human activities. While such deployment promises great benefits, it also runs a great risk of harm or cost to the humans involved. To mitigate this risk, researchers have pursued two general approaches. The first is inspired by Asimov's First Law of Robotics, "*A robot may not injure a human being or through inaction allow a human being to come to harm*" [1]. Here, researchers have emphasized tractable means of guaranteeing that agents obey *safety conditions* in planning and learning — unfortunately, despite the dramatic call to arms to ensure such safety [11], this approach enjoys only a small following.

Instead, a second, alternative approach of *adjustable autonomy* (AA) has gained more popularity. AA enables an agent to act with varying degrees of autonomy (e.g., in uncertain situations, the agent may transfer decision-making control to a more skilled, more knowledgeable human supervisor). Unfortunately, this research has, to date, focused on individual agent-human interactions. It has failed to address multi-agent team settings, such as our Electric Elves (henceforth E-Elves) application, where software assistants employ AA in helping human users coordinate with their colleagues. In particular, existing AA strategies can result in a rigid transfer of decision-making control to humans. In a team setting, such rigidity can cause costly miscoordination when the humans fail to make timely decisions.

This paper presents a marriage of both approaches, while addressing their key shortcomings in complex multiagent environments. For the foreseeable future, harnessing human skills via AA will be important when deploying agents in real-world settings. However, inaccuracies in an agent's world model can cause its seemingly optimal plans to violate safety, so additional guarantees are important. For instance, in E-Elves, modeling inaccuracies once caused a co-author's software assistant to autonomously cancel a critical meeting with his supervisor! Since these agents interact with different humans with varying preferences, the designers cannot provide them with a single model that represents everyone's safety requirements with complete accuracy. On the other hand, it is unreasonable (if not impossible) to have humans specify sufficient safety conditions to completely determine correct agent behavior. Thus, the synergy of AA reasoning and safety conditions is critical in multiagent domains.

This paper begins by describing our novel AA framework, which uses Markov Decision Processes (MDPs) [7] that encode individual and team costs and uncertainties. The use of MDPs allows the agents to compute optimal policies for flexible transfer of control and changes of coordination in AA, thus alleviating team miscoordination. The paper then integrates safety guarantees within the MDP framework. In particular, we enable humans to explicitly specify four different types of safety conditions: *forbidden actions*, *forbidden states*, *required actions*, and *required states*, realized as symbolic constraints on MDP states and actions, eliminating the need to determine sufficiently high-magnitude reward values to guarantee safety. We provide clear probabilistic semantics for these constraints, and present a novel algorithm that generates optimal MDP policies subject to these symbolic constraints. One key result presented is the correctness proof for the algorithm. By incorporating such explicit safety conditions, agents can guarantee safety despite potential MDP model inaccuracies that may arise. An addi-

tional advantage of using symbolic constraints is the support for constraint propagation as a preprocessing step, restricting the space of policies that standard value iteration must consider and thus providing significant gains in the efficiency of policy generation. Finally, while we present results of these algorithms as implemented in the context of MDPs for AA, they are applicable to MDPs in other domains as well.

2 Electric Elves

E-Elves [8] is a real-world experiment, where an agent team of 16 agents, including 12 proxies (assisting 12 people) have been running 24/7 for the past seven months at USC/ISI. Each proxy, called "Friday" (after Robinson Crusoe's servant, Friday), acts on behalf of its user in the agent team. Each Friday uses a teamwork model, STEAM [10] to manage its teamwork. If a user is delayed to a meeting, Friday can reschedule the meeting, relaying the delay to other Fridays, who in turn inform their human users of the delay. If there is a research presentation slot open, Friday responds to the invitation on behalf of its user. Friday can also order its user's meals and track the user's location. Friday communicates through its user's workstation display and wireless devices (e.g., PDAs, mobile phones).

AA is critical in Friday agents. The more autonomous Friday is, the more effort it saves its user. However, in our early implementation, Friday sometimes made costly mistakes when acting autonomously (e.g., unwarranted meeting cancellation). Such mistakes arose, in part, because of the uncertainty in the domain and the real-world costs of actions. Given such costs and uncertainties, AA aims to enable each Friday to intelligently decide between acting autonomously and transferring control to a human user.

Unfortunately, the team context of domains like E-Elves raises a novel AA challenge in the potential for miscoordination when transferring control to human users. In another example from our early implementation, Friday agents would sometimes rigidly transfer control to their users (e.g., to ask whether they planned to attend a meeting). In some cases, a user would be unable to respond, sometimes due to getting stuck in traffic. While waiting for a response, Friday's inaction would cause miscoordination with its teammates (other Friday agents), since it failed to inform them whether its user would attend. This, in turn meant that the other attendees (humans) wasted their time waiting for their teammate. Later, under a different set of circumstances, Friday did not ask the user and instead, following its best guess, autonomously informed the other Fridays that its user would not attend the meeting. However, the user *did* plan to attend, so the human team suffered a serious cost from receiving this incorrect information from Friday. Friday must instead plan a course of action that makes the best tradeoff possible between the possible costs, both present and future, of inaction and of incorrect action.

3 General Approach to AA in Teams

We consider the general problem of AA in teams as follows. We assume a team with a joint activity, α, and a human team member who performs role, ρ, in α. A software assistant must aid the user in performing ρ (e.g., by securing more resources from the

team) and must keep the team informed about whether the user will perform ρ. Given the uncertainty in whether its user can perform ρ, the AA challenge for the software assistant is to decide when to act autonomously and when to transfer decision-making control to the user (e.g., when informing the team that the user cannot perform ρ). For instance, in E-Elves, α is a meeting, ρ is the user's attendance at the meeting, and the key resource is time. Friday attempts to ensure that its user can attend the meeting. One difficulty here is that rigid transfer-of-control regimes cause team miscoordination (as discussed in Section 2). Another difficulty is balancing the user's needs in performing ρ against the team's needs for α to succeed.

We provide a general approach to this AA problem in teams, based on Markov Decision Processes (MDPs) [7]. In this paper, we use reward functions over both state-action pairs ($R_A : S \times A \rightarrow [0, 1]$) and states ($R_S : S \rightarrow [0, 1]$). The key elements of our approach are as follows. First, the MDP reward functions factor in the individual benefits due to performing ρ, team costs due to changes in α, costs of consulting a user, etc. Second, an agent does not rigidly transfer decision-making control to a user; instead, it defines an MDP policy for flexibly transferring control back and forth to ensure team coordination. Thus, if an agent transfers control to a user and the user fails to respond, then the MDP policy guides the agent to act autonomously, to avoid team miscoordination. Third, when an agent is required to act autonomously to avoid miscoordination, the MDP policy mitigates the significant risks by guiding the agent to change its coordination arrangements, postponing or reordering activities to "buy time" to lower decision cost/uncertainty. For instance, in E-Elves, an agent may delay a meeting, improving the user's chances of attending the meeting, and also buying more time for user input.

To generate such an MDP, the set of actions, A, includes: wait (do nothing), reduce autonomy (ask the user for information), autonomously inform other agents that the user is performing (or not performing) ρ, or change the coordination of α (e.g., asking for delays in performing ρ). Each state, $s \in S$, is a tuple: {*rel-time-*ρ, *user-status*, α*-status*, *user-response*, other α attributes}. Here, *rel-time-*ρ measures time left until performance of ρ in α. Similarly, *user-status* enables an agent to (probabilistically) estimate the time the user will actually need before performing ρ in α.

One example of such an AA MDP is the *delay MDP*, covering all meetings for which Friday may act on behalf of its user. Friday can take any of 12 actions, specializations of the general actions already described. For example, informing other agents that the user is performing ρ corresponds to announcing that the user will attend a meeting. Changing the coordination of α corresponds to delaying the meeting. When Friday reduces its autonomy and asks, its user can respond with any of the 10 options from Figure 1a. In the delay MDP's state representation, *rel-time-*ρ becomes *rel-time-meeting*, *user-status* becomes *user location*, α*-status* becomes *status of meeting*, etc. Figure 1b shows a portion of the state space, showing the *rel-time-meeting*, *user-response*, and *user location* features. The figure also shows some state transitions (a transition labeled "delay n" corresponds to the action "delay by n minutes"). Each state contains other features (e.g., *previous-delays*), not pictured, relevant to the overall joint activity, for a total of 2760 states in the MDP for each individual meeting.

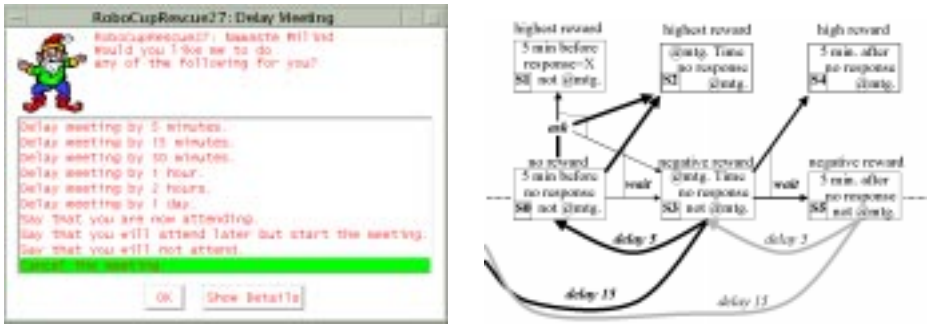

Fig. 1. (a) Dialog box for delaying meetings in E-Elves.(b) A small portion of the *delay MDP*.

In general, our AA MDP framework uses a reward function $R(s,a) = f(\textit{rel-time-}\rho(s), \textit{user-status}(s), \alpha\textit{-status}(s), a)$, where the first three components reflect the value of the team activity in its current status (e.g., active but without user performing ρ), while the action component reflects the cost of changing the coordination of the team activity, α. This reward function has a maximum when the user performs ρ (e.g., the delay MDP encodes a maximum reward in the state where the user is at the meeting location when the meeting starts). The *delay MDP* divides this function further into more specific components. One such component, denoted r_{user}, focuses on the user attending the meeting at the meeting time. r_{user} gives the agent incentive to delay meetings when its user's late arrival is possible, but large delays incur a team cost from rearranging schedules. The team cost is considered by incorporating a negative reward, denoted r_{repair}, with magnitude proportional to the number of delays so far and the number of attendees, into the delay reward function. However, without a delay, the other attendees may waste time waiting for the agent's user to arrive. Therefore, the reward function includes a component, r_{time}, that is negative in states after the start of the meeting if the user is absent, but positive otherwise. The overall reward function for a state, *s*, is a weighted sum: $r(s) = \lambda_{user} r_{user}(s) + \lambda_{repair} r_{repair}(s) + \lambda_{time} r_{time}(s)$.

The delay MDP's transition probabilities represent the likelihood that a user movement (e.g., from office to meeting location) will occur in a given time interval. Figure 1b shows multiple transitions due to "ask" and "wait" actions, with the thickness of the arrows reflecting their relative probability. The designer encodes the initial probabilities, which a learning algorithm may then customize. Other state transitions correspond to uncertainty associated with a user's response (e.g., when the agent performs the "ask" action, the user may respond with specific information or may not respond at all, leaving the agent to effectively "wait").

Given the MDP's state space, actions, transition probabilities, and reward function, an agent can use *value iteration* to generate a policy $P : S \rightarrow A$ that specifies the optimal action in each state [7]. These policies allow user-assistant agents to avoid rigidly committing to transfer-of-control decisions. Thus, if the agent decides to give up autonomy, it does not wait indefinitely for a user response. Instead, the agent continuously reassesses the developing situation, possibly changing its previous autonomy decisions. For instance, one possible policy, for a subclass of meetings, specifies "ask" in state **S0** of Figure 1b (i.e., the agent gives up some autonomy). If the world reaches state **S3**, the policy specifies "wait". However, if the agent then reaches state **S5**, the policy chooses

"delay 15", which the agent then executes autonomously. Thus, the agent's AA is an ongoing process, rather than a single decision.

In addition, when an agent must act autonomously to avoid miscoordination, it can minimize the risk of error by carefully planning a change in coordination (e.g., delaying the meeting). The delay MDP is especially suitable for producing such a plan, because it generates policies by looking ahead at the future costs of team miscoordination and erroneous actions. For instance, through the delay MDP, an agent can reason that a short delay will buy more time for a user to respond to its query, incurring a small communication cost but potentially reducing the uncertainty of a costly decision. Thus, before deciding to autonomously cancel a meeting, an agent might choose a 15-minute delay to give time for an absent user to arrive, or at least respond.

4 Avoiding Harm: Safety Constraints

The MDP-generated policies ensure that an agent's actions are optimal with respect to its current model. However, they do not automatically ensure that agents will avoid harm and obey the *safety constraints* of interest to a human interacting with it. The MDP model of costs and likelihoods may contain inaccuracies, perhaps from the agent's changing environment. Furthermore, in domains such as ours, where agents must interact with different people, the costs and likelihoods may vary significantly from person to person. As an example, students may want to enforce a safety constraint that their Fridays never cancel a meeting with a faculty member, but faculty may not wish to enforce the same (a purely hypothetical case). An agent could potentially learn an individualized MDP model with enough user feedback. However, learning may require an arbitrarily large number of interactions with the humans involved. In the meantime, the agent's incorrect model can still lead to catastrophic results.

4.1 Definition of Constraints

To prevent such catastrophes, it is important to allow humans to directly specify important safety conditions. One way to ensure that agents avoid harm is to forbid them from entering specific states or performing specific actions in specific states. For instance, a user may forbid an agent from autonomously cancelling a meeting in some states (e.g., any state involving a meeting with a supervisor). We define such **forbidden-action** constraints to be a set, C_{fa}, where each element constraint is a boolean function, $c_{fa} : S \times A \rightarrow \{t, f\}$. Similarly, we define **forbidden-state** constraints to be a set, C_{fs}, with elements, $c_{fs} : S \rightarrow \{t, f\}$. If a constraint returns t for a particular domain element (either state or state-action pair, as appropriate), then the constraint applies to the given element. For example, a forbidden-action constraint, c_{fa}, forbids the action a from being performed in state s if and only if $c_{fa}(s, a) = t$.

To provide probabilistic semantics, suitable for an MDP context, we first provide some notation. Denote the probability that the agent will ever arrive in state s_f after following a policy, P, from an initial state s_i as $\Pr(s_i \xrightarrow{*} s_f | P)$. Then, we define the semantics of a forbidden-state constraint c_{fs} as requiring $\Pr(s_i \xrightarrow{*} s_f | P) = 0$. The semantics given to a forbidden-action constraint, c_{fa}, is a bit more complex, requiring

$\Pr(s_i \xrightarrow{*} s_f \wedge P(s_f)=a|P) = 0$ (i.e., c_{fa} forbids the agent from entering state s_f *and* then performing action a). In some cases, an aggregation of constraints may forbid *all* actions in state s_f. In this case, the conjunction allows the agent to still satisfy all forbidden-action constraints by avoiding s_f (i.e., the state s_f itself becomes forbidden). Once a state, s_f, becomes indirectly forbidden in this fashion, any action that potentially leads the agent from an ancestor state into s_f likewise becomes forbidden. Hence, the effect of forbidding constraints can propagate backward through the state space, affecting state/action pairs beyond those which cause immediate violations.

The forbidding constraints are powerful enough to express a variety of safety constraints. For instance, some E-Elves users have forbidden their agents from rescheduling meetings to lunch time. To do so, the users provide a feature specification of the states they want to forbid, such as *meeting-time*=12 PM. Such a specification generates a forbidden-state constraint, c_{fs}, that is true in any state, s, where *meeting-time*=12 PM in s. Similarly, some users have forbidden cancellations by providing a specification of the action they want to forbid, *action*="cancel". This generates a forbidden-action constraint, c_{fa}, that is true for any state/action pair, (s, a), with $a =$"cancel". Users can easily create more complicated constraints by specifying values for multiple features, as well as by using comparison functions other than $=$ (e.g., $\neq$, $>$). We have also noticed synergistic interactions between AA and forbidden constraints, such as forbidding an agent from taking autonomous action in some states (i.e., the agent must instead reduce its level of autonomy and ask the user for input).

While powerful, the forbidden constraints are unable to express all safety constraints of interest. In particular, some safety constraints *require* that an agent to visit certain states states or to perform a specific action in certain states. For instance, in our E-Elves domain, we may require our agents to, at some point during their execution, notify our teammates about our state, whether *attending*, *not attending*, or *cancelled*, as we want to avoid team miscoordination through complete inaction. We do not require this notification property to be true in any particular state, or in all states. The intent is only to ensure that the teammates are eventually notified, so it is difficult to express this property as a forbidden-state constraint (e.g., forbidding all states without this notification property).

Hence, we introduce **required-state** and **required-action** constraints, defined as sets, C_{rs} and C_{ra}, respectively, with elements analogous to their forbidding counterparts. The interpretation provided to the required-state constraint is symmetric, but opposite to that of the forbidden-state constraint: $\Pr(s_i \xrightarrow{*} s_f|P) = 1$. Thus, from any state, the agent *must* eventually reach a required state, s_f. Similarly, for the required-action constraint, $\Pr(s_i \xrightarrow{*} s_f \wedge P(s_f)=a|P) = 1$. The users specify such constraints as they do for their forbidding counterparts (i.e., by specifying the values of the relevant state features or action, as appropriate). In addition, the requiring constraints also propagate backward. Informally, the forbidden constraints focus locally on specific states or actions, while the required constraints express global properties over all states.

4.2 Value Iteration with Constraint Propagation

We have extended standard value iteration to also consider constraint satisfaction when generating optimal policies. The value of each state is no longer a single value, but a tu-

ple $\langle F, N, U \rangle$, where F is a boolean indicating whether the state is forbidden or not, N is a set of requiring constraints that will be satisfied, and U is the expected value (as in traditional value iteration),. For instance, if the value of a state, $V(s) = \langle t, \{c_{rs}\}, 0.3 \rangle$, then executing the policy from state s will achieve an expected value of 0.3 and will satisfy required-state constraint c_{rs}. However, it is not guaranteed to satisfy any other required-state, nor any required-action, constraints. In addition, s is forbidden, so there is a nonzero probability of violating a forbidden-action or forbidden-state constraint. We do not record *which* forbidding constraints the policy violates, since violating any one of them is equally bad. We *do* have to record which requiring constraints the policy satisfies, since satisfying all such constraints is preferable to satisfying only some of them. Therefore, the size of the value function grows linearly with the number of requiring constraints, but is independent of the number of forbidding constraints.

We initialize the value function over states as follows:

$$V^0(s) \leftarrow \left\langle \bigvee_{c \in C_{fs}} c(s), \{c \in C_{rs} | c(s)\}, R_S(s) \right\rangle \tag{1}$$

Thus, s is forbidden if any forbidden-state constraints apply and is needed by any required-state constraints that apply.

In value iteration, we must define an updated value function V^{t+1} as a refinement of the previous iteration's value function, V^t. States become forbidden in V^{t+1} if they violate any constraints directly or if *any* of their successors are forbidden according to V^t. States satisfy requirements if they satisfy them directly or if *all* of their successors satisfy the requirement. We define S' to be the set of all successors: $\{s' \in S | M^a_{ss'} > 0\}$. The following expression provides the precise definition of this iterative step:

$$\begin{aligned} V^{t+1}(s) \leftarrow \max_{a \in A} \Bigg\langle & \bigvee_{c \in C_{fs}} c(s) \vee \bigvee_{c \in C_{fa}} c(s,a) \vee \bigvee_{V^t(s') = \langle U', F', N' \rangle, s' \in S'} F', \\ & \{c \in C_{rs} | c(s)\} \cup \{c \in C_{ra} | c(s,a)\} \cup \bigcap_{V^t(s') = \langle U', F', N' \rangle, s' \in S'} N', \\ & R_S(s) + R(s,a) + \sum_{V^t(s') = \langle U', F', N' \rangle, s' \in S} M^a_{ss'} U' \Bigg\rangle \end{aligned} \tag{2}$$

The maximization uses the following preference ordering, where $x \prec y$ means that y is preferable to x:

$$\begin{aligned} \langle t, N, U \rangle &\prec \langle f, N', U' \rangle \\ \langle F, N, U \rangle &\prec \langle F, N' \supset N, U' \rangle \\ \langle F, N, U \rangle &\prec \langle F, N, U' > U \rangle \end{aligned}$$

In other words, satisfying a forbidden constraint takes highest priority, satisfying more requiring constraints is second, and increasing expected value is last. We define the optimal action, $P(s)$, as the action, a, for which the final $V(s)$ expression above is maximized.

Despite the various set operations in Equation 2, the time complexity of this iteration step exceeds that of standard value iteration by only a linear factor, namely the number

of constraints, $|C_{fs}| + |C_{fa}| + |C_{rs}| + |C_{ra}|$. The efficiency derives from the fact that the constraints are satisfied/violated independently of each other. The determination of whether a single constraint is satisfied/violated requires no more time than that of standard value iteration, hence the overall linear increase in time complexity.

Because expected value has the lowest priority, we can separate the iterative step of Equation 2 into two phases: constraint propagation and value iteration. During the constraint-propagation phase, we compute only the first two components of our value function, $\langle F, N, \cdot \rangle$. The value-iteration phase computes the third component, $\langle \cdot, \cdot, U \rangle$, as in standard value iteration. However, we can ignore any state/action pairs that, according to the results of constraint propagation, violate a forbidding constraint ($\langle t, N, \cdot \rangle$) or requiring constraint ($\langle f, N \subset C_{rs} \cup C_{ra}, \cdot \rangle$). Because of the componentwise independence of Equation 2, the two-phase algorithm computes an identical value function as the original, single-phase version (over state/action pairs that satisfy all constraints).

The worst-case time complexity of the two versions is also identical. However, in practice, the two-phase algorithm achieves significant speedup through the rapid elimination of state/action pairs during constraint propagation. While the single-phase version generally requires as much time as standard value iteration (i.e., without constraints), the two-phase algorithm often computes a policy in dramatically *less* time, as the results in Section 5 demonstrate.

4.3 Correctness of Propagation Algorithm

Given a policy, P, constructed according to the algorithm of Section 4.2, we must show that an agent following P will obey the constraints specified by the user. If the agent begins in some state, $s \in S$, we must prove that it will satisfy all of its constraints if and only if $V(s) = \langle f, C_{ra} \cup C_{rs}, U \rangle$. We prove the results for forbidding and requiring constraints separately.

Theorem 1. *An agent following policy, P, with value function, V, generated as in Section 4.2, from any state $s \in S$ will violate a forbidding constraint with probability zero if and only if $V(s) = \langle f, N, U \rangle$ (for some U and N).*

Proof: We first partition the state space, S, into subsets, S_k, defined to contain all states that can violate a forbidding constraint after a minimum of k state transitions. In other words, S_0 contains those states that violate a forbidding constraint directly; S_1 contains those states that do not violate any forbidding constraints themselves, but have a successor state (following P) that does (i.e., a successor state in S_0); S_2 contains those states that do not violate any forbidding constraints, nor have any successors that do, but who have at least one successor state that has a successor state that does (i.e., a successor state in S_1); etc. There are at most $|S|$ nonempty subsets in this mutually exclusive sequence. To make this partition exhaustive, the special subset, S_∞, contains all states from which the agent will never violate a forbidding constraint by following P. We first show, by induction over k, that $\forall s \in S_k$ ($0 \leq k \leq |S|$), $V(s) = \langle t, N, U \rangle$, as required by the theorem.

Basis step (S_0): By definition, the agent will violate a forbidding constraint in $s \in S_0$. Therefore, either $\exists c \in C_{fs}$ such that $c(s) = t$ or $\exists c \in C_{fa}$ such that $c(s, P(s)) = t$, so we know, from Equation 2, $V(s) = \langle t, N, U \rangle$.

Inductive step ($S_k, 1 \leq k \leq |S|$): Assume, as the induction hypothesis, that $\forall s' \in S_{k-1}, V(s') = \langle t, N', U' \rangle$. By the definition of S_k, each state, $s \in S_k$, has at least one successor state, $s' \in S_{k-1}$. Then, according to Equation 2, $V(s) = \langle t, N, U \rangle$, because the disjunction over S' must include s', for which $F' = t$.

Therefore, by induction, we know that for all $s \in S_k$ ($0 \leq k \leq |S|$), $V(s) = \langle t, N, U \rangle$. We now show that $\forall s \in S_\infty$, $V(s) = \langle f, N, U \rangle$. We prove, by induction over t, that, for any state, $s \in S_\infty$, $V^t(s) = \langle f, N, U \rangle$.

Basis step (V^0): By definition, if $s \in S_\infty$, there cannot exist any $c \in C_{fs}$ such that $c(s) = t$. Then, from Equation 1, $V^0(s) = \langle f, N^0, U^0 \rangle$.

Inductive step ($V^t, t > 0$): Assume, as the inductive hypothesis, that, for any $s' \in S_\infty$, $V^{t-1}(s') = \langle f, N', U' \rangle$. We know that $V^t(s) = \langle f, N^t, U^t \rangle$ if and only if all three disjunctions in Equation 2 are false. The first is false, as described in the basis step. The second term is similarly false, since, by the definition of S_∞, there cannot exist any $c \in C_{fa}$ such that $c(s, P(s)) = t$. In evaluating the third term, we first note that $S' \subseteq S_\infty$. In other words, all of the successor states of s are also in S_∞ (if successor $s' \in S_k$ for some finite k, then $s \in S_{k+1}$). Since all of the successors are in S_∞, we know, by the inductive hypothesis, that the disjunction over V^{t-1} in all these successors is false. Therefore, all three disjunctive terms in Equation 2 are false, so $V^t(s) = \langle f, N^t, U^t \rangle$.

Therefore, by induction, we know that for all $s \in S_\infty$, $V(s) = \langle f, N, U \rangle$. By the definition of the state partition, these two results prove the theorem as required. □

Theorem 2. *An agent following policy, P, with value function, V, generated as described in Section 4.2, from any state $s \in S$ will satisfy each and every requiring constraint with probability one if and only if $V(s) = \langle F, C_{ra} \cup C_{rs}, U \rangle$ (for some U and F).*

Proof Sketch: The proof parallels that of Theorem 1, but with a state partition, S_k, where k corresponds to the *maximum* number of transitions before satisfying a requiring constraint. However, here, states in S_∞ are those that *violate* the constraint, rather than satisfy it. Some cycles in the state space can prevent a guarantee of satisfying a requiring constraint within any fixed number of transitions, although the probability of satisfaction *in the limit* may be 1. In our current constraint semantics, we have decided that such a situation fails to satisfy the constraint, and our algorithm behaves accordingly. Such cycles have no effect on the handling of forbidding constraints, where, as we saw for Theorem 1, we need consider only the *minimum*-length trajectory. □

The proofs of the two theorems operate independently, so the policy-specified action will satisfy all constraints, if such an action exists. The precedence of forbidding constraints over requiring ones has no effect on the optimal action in such states. However, if there are conflicting forbidding and requiring constraints in a state, then the preference ordering causes the agent to choose a policy that satisfies the forbidding constraint and violates a requiring constraint. The agent can make the opposite choice if we simply change the preference ordering from Section 4.2. Regardless of the choice, from Theorems 1 and 2, the agent can use the value function, V, to identify the existence of any such violation and notify the user of the violation and possible constraint conflict.

5 Evaluation

We begin with an evaluation of the overall utility of E-Elves and then evaluate the MDP-based AA framework and the safety constraints separately. We have used the E-Elves system (described in Section 2) within our research group, and here are some key facts about our experience so far:

- Running since 6/1/2000, 24 hours/day, 7 days/week (occasionally interrupted for enhancements).
- Mean number of messages/day among agents: 85
- Number of meetings: 689; autonomous rescheduling: 213; user rescheduling: 152
- Number of meeting presenters selected: 10, 8 autonomously and 2 by users
- Number of meals ordered: 54, with mean 1.4 people per order

The fact that E-Elves users were (and still are) willing to use the system over such a long period and in a capacity so critical to their daily lives is a testament to its effectiveness.

Our MDP-based approach to AA has provided much value to the E-Elves users, as attested to by the large number (689) of meetings that the users have allowed their agent proxies to monitor using the delay MDP. The large number (213) of autonomous rescheduling corresponds to a large amount of user effort saved. As for the correctness of the MDP approach, over the entire span of their operation, the agents have performed without any catastrophic mistakes. For example, the policy described in Section 3 avoided the problems with rigid transfer of control, as described in Section 2. When giving up autonomy, the agent's policy specified waiting a certain amount of time to give the user a chance to respond. However, as the meeting drew closer, the policy eventually preferred taking back autonomy, so that the agent can act on its own before any miscoordination takes place. Figure 2 demonstrates how often the agents required this ability. The solid line plots the percentage of meetings vs. the number of transfers of autonomy. A transfer of autonomy takes place either when the agent asks the user for input or else when the agent, having asked the user for input without receiving it, stops waiting and acts autonomously. For some simple meeting instances, the agent was able to act completely autonomously, without ever giving up control (e.g., the user arrives at the meeting early). Ignoring these "easy" cases, the dashed line shows an alternate histogram over only those meetings for which the agent had to ask the user at least once. Although the current agents do occasionally make mistakes, these errors are typically on the order of asking for input a few minutes earlier than desired, etc. Unfortunately, the inherent subjectivity and intrusiveness of user feedback has complicated our determination of the optimality of the agents' decisions.

In evaluating our safety-constraint framework, we have already proven the correctness of our algorithm. The constraints have also been useful. For instance, one typical user has added five safety constraints. Two forbidden-state constraints prevent meetings on Saturday or Sunday (e.g., forbid *meeting-day*=Sunday). A second forbidden-state constraint prevents his agent from rescheduling a meeting to lunch time *if* not originally scheduled at that time (i.e., forbid combination of *meeting-time*=12 PM and *previous-delays*> 0). The user has also specified a forbidden-action constraint preventing cancellations in meetings with his superiors (i.e., forbid *action*="cancel"). Therefore, his agent never performs any erroneous, autonomous cancellations.

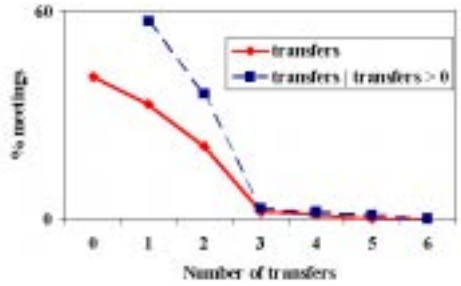

Fig. 2. Histogram of transfers of control per meeting.

To evaluate the synergy between AA and constraints, we merged user-specified constraints from all E-Elves users, resulting in a set of 10 distinct constraints. We started with an unconstrained instance of the *delay MDP* and added these constraints one at a time, counting the policies that satisfied the applied constraints. We then repeated these experiments on expanded instances of the *delay MDP*, where we increased the initial state space by increasing the frequency of decisions (i.e., adding values to the *rel-time-meeting* feature). Figure 3a displays these results (on a logarithmic scale), where line A corresponds to the original *delay MDP* (2760 states), and lines B (3320 states), C (3880 states), and D (4400 states) correspond to the expanded instances. Each data point is a mean over five different orderings of constraint addition. For all four MDPs, the constraints substantially reduce the space of possible agent behaviors. For instance, in the original *delay MDP*, applying all 10 constraints eliminated 1180 of the 2760 original states from consideration, and reduced the mean number of viable actions per acceptable state from 3.289 to 2.476. The end result is a 50% reduction in the size ($\log_{10}$) of the policy space. On the other hand, constraints alone did not provide a complete policy, since all of the plots stay well above 0, even with all 10 constraints. Since none of the individual users were able/willing to provide 10 constraints, we cannot expect anyone to add enough constraints to completely specify an entire policy. Thus, value iteration is still far from redundant.

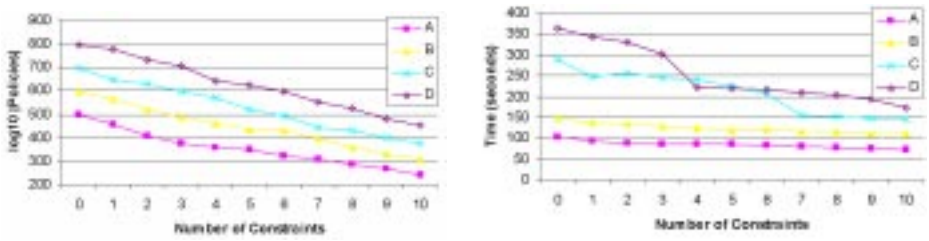

Fig. 3. (a) Number of possible policies (logarithmic). (b) Time required for policy generation.

The constraints' elimination of behaviors also decreases the time required for policy generation. Figure 3b plots the total time for constraint propagation and value iteration over the same four MDPs as in Figure 3a (averaged over the same five constraint orderings). To smooth out any variability in machine load, each data point is also a mean over five separate iterations, for a total of 25 iterations per data point. The values for the zero-constraint case correspond to standard value iteration without constraints. The savings in value iteration over the restricted policy space dramatically outweigh the cost of prepropagating the additional constraints. In addition, the savings increase with the size of the MDP. For the original *delay MDP* (A), there is a 28% reduction in policy-generation time, while for the largest MDP (D), there is a 53% reduction. Thus, the introduction of constraints speeds up policy generation, providing another example of the synergy between the two components of our implementation. We are continuing to conduct experiments to more exhaustively cover the set of possible constraints, constraint orderings, state space sizes, etc.

6 Summary and Related Work

It is a tribute to Asimov's creative intellect that his laws of robotics have not only fascinated generations of readers, but also inspired a research direction in autonomous agents. This paper builds on several key themes from prior work in this direction, as well as previous AA research. Our contribution to this line of research is to explicitly address safety in the context of AA, as well as to address AA in team settings (where agents must simultaneously satisfy team and individual responsibilities). This paper also introduces safety constraints that forbid or require certain states or actions in the context of MDPs. This paper presents an algorithm to propagate such constraints, addressing constraint interactions in optimal policy generation for the MDP and proving its correctness. It also presents experimental results that demonstrate the utility, in both safety and efficiency, of this algorithm in a real-world domain.

Existing research has already addressed the Asimovian concept of safety by emphasizing computationally tractable means of avoiding harm within the context of classical planning [11] and finite-state machines [5]. In other related work, previous AA research, in user control of robots [3] or in mixed-initiative planning [4], has focused on interactions between an individual agent and an individual human. Our research builds on both traditions, contributing the use of MDPs for AA in agent team settings and then providing a novel extension of supporting safety constraints. Our forbidden-action constraints support human-specified partial policies, as developed by other researchers [2]. However, constraints over states, as well as the required-action constraints, raise the novel issue of constraint propagation not addressed earlier. In previous work [9], we presented a specialized version of our MDP approach to AA, that focused on the delay MDP. This paper contributes a generalization of that approach using notions of role, ρ, and team activity, α and integrates four types of safety constraints absent in that work. The success we have achieved by combining and extending these two approaches (safety conditions and AA) within E-Elves demonstrates the value of such an approach in such real-world, multiagent domains.

References

1. Isaac Asimov. Runaround. *Astounding Science Fiction*, pp. 94–103, 1942.
2. Craig Boutilier, Ray Reiter, Mikhail Soutchanksi, and Sebastian Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the National Conference on Artificial Intelligence*, 2000.
3. Gregory A. Dorais, R. Peter Bonasso, David Kortenkamp, Barney Pell, and Debra Schreckenghost. Adjustable autonomy for human-centered autonomous systems on mars. In *Proceedings of the International Conference of the Mars Society*, 1998.
4. George Ferguson, James Allen, and Brad Miller. TRAINS-95 : Towards a mixed initiative planning assistant. In *Proceedings of the Conference on Artificial Intelligence Planning Systems*, pp. 70–77, May 1996.
5. Diana F. Gordon. Asimovian adaptive agents. *Journal of Artificial Intelligence Research*, 13:95–153, 2000.
6. Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling, Anita Raja, Rgis Vincent, Thomas Wagner, Ping Xuan, and Shelley XQ. Zhang. A multi-agent system for intelligent environment control. In *Proceedings of the International Conference on Autonomous Agents*, 1999.
7. Martin L. Puterman. *Markov Decision Processes*. John Wiley & Sons, 1994.
8. David V. Pynadath, Milind Tambe, Yigal Arens, Hans Chalupsky, Yolanda Gil, Craig Knoblock, Haeyoung Lee, Kristina Lerman, Jean Oh, Surya Ramachandran, Paul S. Rosenbloom, and Thomas Russ. Electric Elves: Immersing an agent organization in a human organization. In *AAAI Fall Symposium on Socially Intelligent Agents: The Human in the Loop*, pp. 150–154, 2000.
9. Paul Scerri, David V. Pynadath, and Milind Tambe. Adjustable autonomy in real-world multi-agent environments. In *Proceedings of the Conference on Autonomous Agents*, page to appear, 2001.
10. Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
11. Daniel S. Weld and Oren Etzioni. The first law of robotics (a call to arms). In *Proceedings of the National Conference on Artificial Intelligence*, pp. 1042–1047, 1994.