Acting on Norm Constrained Plans

Nir Oren[1], Wamberto Vasconcelos[1], Felipe Meneguzzi[2], and Michael Luck[3]

[1] School of Computing Science, University of Aberdeen
Aberdeen AB24 3UE, UK
wvasconcelos@acm.org and n.oren@abdn.ac.uk
[2] Robotics Institute, Carnegie Mellon University
Pittsburgh PA 15213, USA,
meneguzz@cs.cmu.edu
[3] Department of Informatics , King's College London
London WC2R 2LS, UK
michael.luck@kcl.ac.uk

Abstract. The behaviour of deliberative agents is often guided by a plan library designed to achieve goals given certain environmental conditions. Plans in these plan libraries are designed to achieve individual goals, and cannot possibly account for all possible variations of restrictions in the societies within which these agents must operate. These restrictions, captured through *norms*, include obligations, prohibitions, and permissions. Unlike traditional planning restrictions, norms can often be contradictory and impossible to achieve simultaneously, necessitating some form of compromise. In this paper we describe a technique for taking norms into consideration when deciding how to execute a plan. Our norms are constraint based, allowing for fine-grained control over actions. Our technique allows for reasoning about the interactions between norms, and resolves conflict by selecting actions where the cost of violating one set of norms is outweighed by the reward obtained in complying with another.

Keywords: Norms, BDI, Constraints

1 Introduction

Most agent architectures (e.g. BDI based approaches such as AgentSpeak(L) [11]) make use of *offline* planning, where a plan library is created before execution in the environment begins. An agent utilising an offline plan library selects a plan for execution based on the state of the environment. A problem when using pre-generated plan libraries involves how to select plans appropriate to the current situation. A pre-generated plan is often conditional, identifying the environmental *context* in which it is applicable. However, it is difficult for the plan designer to envisage all the situations in which a plan could be considered for execution at design time. In particular, the society in which an agent operates may impose a given set of *norms*, which restrict the acceptable behaviour of the agent in that society. Norms do not have to be fixed, and those which apply to the agent may vary over time, and even emerge from multi-agent interaction, so

it is often infeasible for the designer to take account of them in pre-generated plans. For example, consider a plan to build a refugee camp following some disaster. Such a plan may take the terrain in which the refugee camp is to be located into account, but may not have considered some other logistical, social or operational restrictions, such as fuel availability. If the original plan assumed that fuel is freely available, then in the context of fuel limits, the original plan may be unusable. However, by limiting the amount of fuel use, for example by introducing a prohibition on driving large distances once the camp is built (assuming that long drives are necessary for setting up the camp), the plan can still be used.

An advantage in taking norms into account when selecting a plan for execution involves the possibility of norm violation. In some situations, an agent may ignore a norm in order to achieve a critical goal. By basing plan selection not only on some context and invocation condition, but also on the norms that would be complied with and violated during plan execution, more flexible (and robust) behaviour can be achieved.

Clearly, the ability to adapt an existing plan library to cater for norms, and thus function in a variety of different situations, greatly promotes plan reuse. Given this, the main contribution of this paper lies in specifying how an agent should execute a plan, while deciding which norms to adhere to, in such a way so as to maximise its utility.

[13] created a constraint and predicate based norm representation, and we extend this representation to actions found within plans. Constraints allow for fine-grained control of the value of variables, increasing the expressiveness of our notation, and the sophistication of the mechanisms to manipulate them. In order to act in the presence of norms, we adopt a utility based approach. Informally, given a plan represented as an AND/OR tree, with actions specified as constrained predicates, we recursively compute the effects of executing the next action from the plan, identifying what constraints would appear given that the agent decides to comply, or violate a set of norms. Such norm compliance or violation affects the ultimate utility of the plan. Executing an action with specific bindings can trigger rules creating, or deleting additional norms, further constraining future action. Therefore, different sets of constraints may lead to plans with different utilities, and we must consider all possible sets of constraints. Our goal is thus to identify the set of constraints on action that result in maximal utility. These constraints are then used to guide plan execution.

The plans we consider for our approach are similar to a Hierarchical Task Network (HTN) [5]. However, the exploration of these plans by our approach differs from HTN planning with preferences [12] in that active norms, unlike preferences, change dynamically during planning. The defeasible nature of norms further complicates the determination of an optimal plan.

The remainder of this paper is structured as follows. Section 2 describes our approach's underlying data structures. Sections 3 and 4 then detail how these components are combined to reason about plans in the context of normative restrictions. We evaluate our approach in Section 5, and place our approach

in the context of existing work in Section 6, before concluding and identifying future work in Section 7.

2 Plans and Norms

We begin this section by describing the basic building blocks of our system. We then explain how plans, built up of actions, are represented. Section 2.3 then provides details regarding the specification of norms, following which we describe *normative rules* that identify when norms begin, and when they cease, to affect the scope of an action. Finally, we describe *enactment states*, which denote the norms affecting execution at a single point in time. In the remainder of this paper, we denote first-order terms generically as τ; variables are represented as $X, Y, \ldots$ and constants as a, b etc.

2.1 Constraints, Substitution and Unification

Our system makes extensive use of constraints to limit agent action. A constraint γ is an atomic formula of the form $\tau \bowtie \tau'$, where $\bowtie \in \{=, \neq, >, \geq, <, \leq\}$ and τ, τ' are first order terms. We write Γ to denote a generic, possibly empty, set of constraints. We employ the predicate $satisfy(\Gamma)$ for a set of constraints Γ, which holds if and only if the constraints allow at least one solution, i.e. if they are satisfiable.

We also make use of unification and substitution relationships, usually applied to a first order formula and/or a constraint. The application of a substitution σ, which consists of a set of pairs X/τ to some structure β is written $\beta \cdot \sigma$, and consists of replacing all instances of X in β by τ. Finally, two structures β, β' unify according to substitution σ (abbreviated $unify(\beta, \beta', \sigma)$) if and only if $\beta \cdot \sigma = \beta' \cdot \sigma$.

2.2 Actions and Plans

To affect its environment, an agent executes actions, which we represent as ground atomic first order formulae. Plans identify groups of actions that must be taken, and are applicable in different situations. Plans are thus represented using partially ground actions. Applying a plan to a specific situation occurs via the grounding of variables. We call partially ground actions *action specifications*.

Definition 1. ***(Action Specifications, Actions and Entailment)*** *An action specification α is defined as $\psi \circ \Gamma$, where ψ is a first order atomic formula, and Γ is a set of constraints over a subset of the variables in ψ. Act is the set of all action specifications.*

Given an action specification $\alpha = \psi \circ \Gamma$ and a substitution σ, we say that ψ' is an action iff $\psi' = \psi \cdot \sigma$ such that ψ' contains no free variables. An action specification $\alpha = \psi \circ \Gamma$ entails an action specification $\beta = \psi' \circ \Gamma'$ if and only if for all σ such that $unify(\psi, \psi', \sigma)$, whenever $satisfy(\Gamma \cdot \sigma)$, $satisfy(\Gamma' \cdot \sigma)$.

Two norms, $\omega_1 = \mathsf{X}\psi \circ \Gamma, \omega_2 = \mathsf{X}\psi' \circ \Gamma'$, where X is some modality, can also entail each other in an analogous manner to action specifications. If α entails β, we write $\alpha \supset \beta$.

Where obvious, we abbreviate action specifications such as $a(X, Y) \circ X = \tau_1 \wedge Y = \tau_2$, with τ_1, τ_2 terms, as $a(\tau_1, \tau_2)$. Similarly, we write $\psi \circ \emptyset$ simply as ψ.

We represent plans as AND/OR trees, with nodes in the tree generically specifying the actions that must be taken in order to execute the plan. Leaf nodes are associated with *primitive actions* (that is, those actions that an agent executes in order to affect the environment), while other nodes represent *compound actions*, made up of all of the node's children in the case of the node being an *AND* node, or of any one of the node's children in the case of an *OR* node.

Definition 2. ***(Plan)*** *A Plan P is one of*

1. *α, where α is an action specification.*
2. **andN**$(P_1, \ldots, P_n)$ *where $P_1 \ldots P_n$ are plans.*
3. **orN**$(P_1, \ldots, P_n)$ *where $P_1 \ldots P_n$ are plans.*

The α node represents a primitive action within the plan. A node of the form **andN**$(P_1, \ldots, P_n)$ represents an *AND* node in the plan. This node is a compound action, requires all of the actions specified within $P_1, \ldots, P_n$ to be executed. Plan nodes of the form **orN**$(P_1, \ldots, P_n)$ represent *OR* nodes in the plan tree; for this compound action to be executed, one of $P_1, \ldots, P_n$ must have been executed [4].

As an example of such a plan, consider the requirement to establish a refugee camp at position *(X,Y)*. In order to do so, intelligence must first be collected (via a *intel(X,Y)* action). The area must then be cleared, either by the agent executing the plan (using a *selfClear(X,Y)* action, or through some other organisation (via the *outsourceClear(X,Y)* action). Finally, the camp itself must be built by executing the *b_camp* and *b_roads* primitive actions. This plan contains both *AND*, and *OR* nodes, and any constraints on the actions themselves (e.g. stating that $X < 5$) act as hard constraints on variable values. Such a plan (with no hard constraints) can be written as follows

andN(*intel*(X, Y),
  orN(*selfClear*(X, Y),*outsourceClear*(X, Y)),
  b_camp(X, Y), *b_roads*(X, Y))

2.3 Norms

Within our system, norms are obligations, prohibitions and permissions on the possible values of specific variables, in the context of specific actions. An obligation can thus, for example, specify exactly where the refugee camp must be

[4] Compound actions can be associated with an action specification, but this yields no additional representative power.

placed in the previous example, by restricting X and Y to specific values for the $build(X, Y)$ action. In order to create this restriction, norms make use of constraints.

Definition 3. ***(Norms and Constraints)*** *A norm is an obligation, permission or prohibition, written respectively as* $\mathsf{O}\alpha$, $\mathsf{P}\alpha$ *and* $\mathsf{F}\alpha$, *where* $\alpha = \psi \circ \Gamma$ *is an action specification. A norm is interpreted as obliging, permitting, or prohibiting the execution of* ψ *according to constraints* Γ.

A generic norm is represented by the symbol ω.

Norms are intended to constrain the values assigned to some variables within an action specification. Critically, and unlike most work on norms (e.g. [1],[6]), an obligation $\mathsf{O}\psi \cdot \Gamma$ thus does not specify that ψ *should be* executed, but instead states that *if* action ψ is executed, it should be done in a way that is consistent with constraints Γ. Given an action specification $\alpha = \psi \cdot \Gamma$ and a norm $\omega = \mathsf{X}\beta$, where $\beta = \phi \cdot \Gamma'$, we say that α is *in the scope* of ω if $\psi = \phi$. Furthermore, if $unify(\psi, \beta, \sigma)$, $\mathsf{X} \in \{\mathsf{O}, \mathsf{P}\}$, and $satisfy(\Gamma' \cdot \sigma)$, then we say that α *complies with* ω. Alternatively, if $\mathsf{X} = \mathsf{F}$, then the norm is complied with if, for all σ such that $unify(\psi, \beta, \sigma)$, $\neg satisfy(\Gamma' \cdot \sigma)$.

A norm $\omega = \mathsf{X}\psi \circ \Gamma_\omega$ is said to be applicable for an action specification $\alpha = \phi \circ \Gamma$, written $applicableNorm(\omega, \alpha)$ if and only if $unify(\phi, \psi, \sigma)$ for some σ.

Consider a norm $\omega_c = \mathsf{O}selfClear(X, Y) \circ X \leq 8 \wedge Y = 2$. An action specification $selfClear(A, B) \circ \Gamma$ for any Γ is in the scope of ω_c . Similarly, $selfClear(5, 2)$ complies with ω_c.

2.4 Permissions and Conflicts

We treat permissions as exceptions to obligations and prohibitions, so they do not have meaning in isolation. Thus, for example, the norm $\mathsf{O}selfClear(X, Y) \circ \{X < 30, Y = 20\}$ imposes an obligation, when executing the *selfClear* action, that X is bound to a value less than 30, and Y is equal to 20. The permission $\mathsf{P}selfClear(X, Y) \circ \{X < 40\}$ allows X to be less than 40 *if* the obligation is present, while still complying with the obligation.

Violations apply in specific cases where an obligation or prohibition is not complied with, and no permission exists that permits this non-compliance to occur. Given a norm $\omega = \mathsf{X}\psi \circ \Gamma$ where $\mathsf{X} \in \{\mathsf{O}, \mathsf{F}\}$, and a permission $\omega' = \mathsf{P}\psi \circ \Gamma'$, we refer to ω' as a *mitigating permission*. An action specification β violates an obligation or prohibition ω if β is in the scope of ω, it does not comply with ω and there is no mitigating permission $\mathsf{P}\phi \circ \Gamma'$ such that $satisfy(\Gamma' \cdot \sigma)$. Norm violation must thus be considered with regards to a set of permissions.

Finally, multiple obligations or prohibitions may conflict, requiring contradictory behaviour. Informally, a set of norms over an action is in conflict if no substitution of variables can be made that is consistent with the obligations and prohibitions found within the norm set, and no mitigating permissions exist allowing this substitution. Given a set of norms Ω, let $\Omega^O, \Omega^P, \Omega^F$ represent the subsets of obligations, permissions and prohibitions within Ω. Similarly,

$$
\begin{aligned}
R &::= LHS \Rightarrow RHS \\
LHS &::= \alpha | NLHS \\
NLHS &::= \omega | NLHS \wedge NLHS \\
RHS &::= RHS \wedge RHS | \oplus \omega | \ominus \omega
\end{aligned}
$$

Fig. 1. BNF for normative rules.

$\Gamma^O/\Gamma^P/\Gamma^F$ represents the set of constraints found in $\Omega^O/\Omega^P/\Omega^F$. Then, given a set of norms Ω of the form $\omega_i = \mathsf{X}\psi \circ \Gamma_i$ (i.e. referring to the same action ψ), the set Ω is in conflict, iff there is no substitution σ such that the following holds

$$
\bigwedge_{\gamma^O \in \Gamma^O} \gamma^O \cdot \sigma \bigwedge_{\gamma^F \in \Gamma^F} \neg\gamma^F \cdot \sigma \bigvee_{\gamma^P \in \Gamma^P} \gamma^P \cdot \sigma \tag{1}
$$

When conflicting norm sets occur, a reasoner must choose which subset of norms to comply with, and which to ignore.

2.5 Normative Rules

The norms imposed on agents are situation dependent, and we make use of a simple rule language to specify *normative rules* that identify the cases in which a norm starts, and ceases, to exist. Normative rules are written in the form $LHS \Rightarrow RHS$, where LHS contains conjunctions of actions and norms, and RHS identifies which obligations, permissions and prohibitions should be added to, or removed from, the set currently affecting the agent. Informally, if an action in the LHS of such a rule has been executed, then the set of norms must be modified according to the RHS of the rule. Similarly, if a norm ω exists in a rule's LHS, then the set of norms is modified as per the rule's RHS. The LHS is formed of a maximum of one action specification, together with a conjunctive combination of zero or more norms. The RHS of the rule then identifies the norm to be added or removed. The BNF for normative rules is shown in Figure 1.

$\oplus\omega$ denotes the addition of ω to the set of currently active norms, while $\ominus\omega$ denotes the removal of ω from this set. *Rules* represents the set of all normative rules in the system.

Thus, the rule $intel(20, 5) \Rightarrow \oplus\omega_c$ states that if action $intel(20, 5)$ is executed, norm ω_c will come into force. Normative rules containing a norm in the LHS can represent norms that come into force due to the presence of other norms, allowing contrary-to-duty obligations to be modelled. For example, an obligation to build a camp, represented by ω_{bc}, might create an obligation to build a road, represented as ω_{br}. The normative rule $\omega_{bc} \Rightarrow \oplus\omega_{br}$ captures this situation.

Before discussing the semantics of norms, we describe the structure used to track the normative status of an executing system. This structure, referred to as an *enactment state*, identifies the norms that exist at any point in time due to the application of normative rules in a previous time point.

2.6 Enactment States

By executing actions, an agent changes the state of the world around it. Under the influence of normative rules, such changes cause new norms to be instantiated, or existing ones to be lifted, affecting future actions. Similarly, past actions can limit future action, by binding values to some of the future action's variables. Following [4], we represent the environment affecting the agent as a transition system between individual *enactment states*, each of which represent the system at a single time point. By executing an action, the system is transitioned to a new enactment state. Such a transition system, starting at an initial state, and transitioning to new enactment states until all agent actions have been executed, represents an entire run of the system.

To capture the portion of the environment affecting the agent, enactment states must track the obligations, permissions and prohibitions that are in force, and the constraints on variable values that have already been committed to.

Definition 4. ***(Enactment State)*** *$\Delta = (\Omega_\Delta, \Gamma_\Delta)$ is an enactment state, with Ω_Δ, a set of norms, and a constraint Γ_Δ.*

We have now described the basic data structures used by an agent in determining how to act in the presence of norms and normative rules. Next, we describe the rules that govern transitions between enactment states in more detail. These rules are then extended to provide an algorithm for acting in the presence of normative rules.

3 Transitioning Between Enactment States

The previous section described the data structures used in our framework, and we now assign an operational semantics to these structures, using them to describe legal transitions between enactment states. Our approach modifies that taken by [4] in two ways. First, we allow only norms and actions in the *LHS* of a rule, simplifying our semantics. Second, [4] was concerned with identifying a new enactment state following the execution of an action. We are concerned with determining the *possible* enactment states following the execution of some action specified by an action specification. Thus, multiple enactment states are possible. For example, consider the rules

$$intel(X, Y) \Rightarrow \oplus\omega_1 \quad intel(5, 6) \Rightarrow \oplus\omega_2 \quad intel(7, 8) \Rightarrow \oplus\omega_3$$

Executing action $intel(2, 2)$ results in norm ω_1 added to the resulting enactment state. Executing $intel(5, 6)$ leads to both ω_1 and ω_2 appearing in the new enactment state. If it is known that the *intel* action is executed, but its parameters are unknown (e.g. due to the action appearing in a partially ground plan), three possible enactment states can be transitioned to, namely one in which ω_1 exists, one where ω_1, ω_2 exist, and one in which ω_1, ω_3 exists. Since our approach considers the possible enactment states resulting from the execution of the action specifications, our semantics, unlike those of [4], must identify a set of possible enactment states rather than a single enactment state.

Algorithm 1 Computing all possible enactment states.

1: **function** PosEnactStates($\Delta, \alpha, Rules$)
2: $\Delta = (\Omega_\Delta, \Gamma_\Delta)$
3: $\alpha = \psi \circ \Gamma_\alpha$
4: $\Delta_N = \{\}$
5: $P = 2^R$ s.t. $R = \{r | r \in Rules$ and $potApp(r, \Omega_\Delta, \alpha)\}$
6: **for all** $p \in P$ **do**
7: $\Gamma = \Gamma_\Delta \wedge \Gamma_\alpha \bigwedge_{r \in p} actionConstraints(r) \bigwedge_{s \in R \setminus p} \neg actionConstraints(s)$
8: **if** satisfy(Γ) **then**
9: $\Omega = \Omega_\Delta$
10: **for all** $RHS \Rightarrow \oplus\omega \in p$ **do**
11: $\Omega = \Omega \cup \{\omega\}$
12: **for all** $RHS \Rightarrow \ominus\omega \in p$ **do**
13: $\Omega = \Omega \backslash \omega$
14: $\Delta_N = \Delta_N \cup \{(\Omega, \Gamma)\}$
15: **for all** $(\Omega, \Gamma) \in \Delta_N$ **do**
16: **if** $\exists(\Omega', \Gamma') \in \Delta_N$ s.t. $\Omega \subset \Omega'$ and $\Gamma' \rightarrow \Gamma$ **then**
17: $\Delta_N = \Delta_N \backslash (\Omega, \Gamma)$
18: **return** Δ_N

Rules are applied when they are consistent with the action being executed, and the norms found in the system. Since an action specification can encapsulate a large range of actions, we must identify when a rule is *potentially applicable*. This situation occurs when the action specification found in the rule entails the action specification being entailed, and all norms found in the rule are entailed by norms found within the current enactment state. Formally,

Definition 5. ***(Potentially Applicable Rule)*** *A rule $R \equiv \beta \wedge \omega_1, \ldots, \omega_n \Rightarrow RHS$ is potentially applicable with respect to a set of norms Ω and an action specification α if and only if $\alpha \supset \beta \wedge \forall\omega_i \exists\omega \in \Omega$ such that $\omega \supset \omega_i$. The predicate $potApp(R, \Omega, \alpha)$ holds if R is potentially applicable with respect to Ω and α.*

Given an enactment state, an action specification, and a set of normative rules, Algorithm 1 returns the possible enactment states that can result from executing all possible actions represented by the action specification. Within this algorithm, the *actionConstraints* function returns the constraint associated with the action specification found within the rule (or true if no action specification exists).

The algorithm operates by first identifying all combinations of potentially applicable rules, and then evaluating each of these combinations individually (Line 5). Line 7 computes the constraints imposed due to the action specification and the norms under consideration, together with those constraints imposed to ensure that the remaining norms are not applied. If these constraints can be satisfied, the set of rules under consideration will result in a new enactment state, and Lines 9–14 create this new enactment state, based on the old one, by

adding the appropriate norms, and the constraints imposed by the applied rules. Finally, starting at Line 15, we remove all enactment states obtained due to the application of non-maximally consistent sets of potentially applicable rules.

Algorithm 1 provides us with an operational semantics for normative rules[5]. In the next section, we investigate what action specification should be executed given that the current enactment state contains some set of norms.

4 From Plans to Norm Constrained Actions

When executing an action specification, we must reason about the constraints that should be imposed on it. These constraints are obtained from the norms found within the current enactment state. For example, when executing action $a(X)$, given the norm $\mathsf{O}a(X) \circ X < 5$, the agent could constrain the value of X to less than 5 if it decides to comply with the norm. Now, consider a sequence of action specifications, such as $a(X), b(Y), c(X, Z)$. Constraints on the value of X, selected due to $a(X)$, could affect norm compliance when executing $c(X, Z)$. Thus, compliance with norms at one time point can affect later norm compliance choices.

We adopt a utility based model of norm compliance. More specifically, we assume that the execution of a plan results in some base utility, and that different types of norms are associated with different utility measures. Obligations and prohibitions are associated with a a utility gain for compliance, and a utility loss for violation. Permissions are associated with a utility loss for utilisation (for example, obtaining permission to construct the refugee camp further away than is normally allowed might incur a loss of trust within the society, reflected by loss of utility). Actions also have a utility cost. Formally, we represent this via a utility function $cost : Act \times 2^{Norms} \times 2^{Norms} \rightarrow \mathbb{R}$. The *cost* function is a partial function, and its first parameter represents obligations and prohibitions that are complied with, while its second parameter is those obligations and prohibitions that are not complied with, together with the permissions that have been utilised. Thus, if a norm appears in the set passed in as one parameter, it may not appear within the other parameter. Under this model, the problem we are addressing reduces to selecting a path through the plan, and a set of norms (created by the rules as actions are executed) with which to comply, that is conflict free, and which lead to maximal utility.

Our approach undertakes a best-first incremental search in the enactment state space created by selectively expanding plans and selecting a subset of norms for compliance. We define a data structure $\langle \Delta, \mathit{Actions}, \mathit{Utility} \rangle$ to track the execution of a plan. Here, Δ is an enactment state, *Actions* is a sequence of action specifications, and *Utility* is the current utility of the plan. Using this data structure Algorithm 2 describes the process of identifying an optimal plan.

The algorithm first creates a initial structure Υ containing a single element representing the plan with no actions having yet been executed. It also initialises

[5] Note that our algorithms do not explicitly manipulate substitutions, as these are applied to variables during the computations.

Algorithm 2 Finding the optimal plan.

Require: A plan *Plan*, utility function *cost* and set of rules *Rules*

1: **for all** *Action* $\in$ first possible actions of *Plan* **do**
2: Let $\Upsilon = \{\langle(\{\}, \top), [Action], 0\rangle\}$
3: $Best = \emptyset$
4: **while** $\Upsilon! = \emptyset$ **do**
5: $\langle(\Omega, \Gamma), Actions, U\rangle$=removed highest utility element of Υ
6: Let α = The last element of *Actions*
7: **if** $\alpha = \emptyset$ & $U >$ utility of *Best* **then**
8: $Best = \langle(\Omega, \Gamma), Actions, U\rangle$
9: **break**
10: $AN = applicableNorms(\Omega, \alpha)$
11: **for all** $\Omega_{AN} \in 2^{AN}$ **do**
12: $\Gamma_N = \Gamma \wedge constraint(\alpha)$
13: **for all** $\mathsf{X}\psi \cdot \Gamma_\psi \in \Omega_{AN}$ **do**
14: **if** $\mathsf{X} = \mathsf{O}$ **then** $\Gamma_N = \Gamma_N \wedge \Gamma_\psi$
15: **if** $\mathsf{X} = \mathsf{F}$ **then** $\Gamma_N = \Gamma_N \wedge \neg\Gamma_\psi$
16: **if** $\mathsf{X} = \mathsf{P}$ **then** $\Gamma_N = \Gamma_N \vee \Gamma_\psi$
17: **if** $satisfy(\Gamma_N)$ **then**
18: $U_n = U + cost(\alpha, \Omega_{AN}, AN \backslash \Omega_{AN})$
19: $\Delta_N = posEnactStates((\Omega, \Gamma_N), \alpha, Rules)$
20: **for all** $\delta \in \Delta_N$ **do**
21: **for all** β=next possible action of *Plan* **do**
22: insert $\langle \delta, [Actions, \beta], U_n\rangle$ into Υ

the currently found solution (represented by *Best*) to the empty set. The heart of the algorithm starts at Line 4. We begin by selecting the current best action sequence (Line 5) and checking if it satisfies the plan in a manner better than the current best solution. If so, this action sequence replaces the current best solution. Otherwise, all applicable norms are identified. For each possible combination of applicable norms, the constraint of those norms that are applied are added to any existing constraints (Lines 11–16). Γ_N is analogous to (1) from Section 2.4, and if it is satisfiable, then this combination of norms is not in conflict, and can thus executed. The utility for complying with this subset of applicable norms is thus computed (Line 18), and all possible enactment states resulting from this action are then created (Line 19). Finally, all possible next action specifications are obtained from the plan, and the updated action sequences, utilities, and enactment states are added back to Υ (Lines 20–22) allowing the process to continue.

We do not describe how to extract the next actions from an AND/OR tree, as standard algorithms exist to do so [5]. It should be noted that our algorithm can easily be extended to reason over multiple plans by associating each plan with its own base utility, and storing the plan in Υ. Also, note that a fully norm-compliant reasoner (that is, one that will only act if it can comply with all its

norms) can be obtained from Algorithm 2 by modifying Line 11 to consider the set of applicable norms rather than its powerset.

While our algorithm is guaranteed to terminate, and is sound and complete if left to run to termination, its worst case complexity is clearly exponential. However, this complexity is mitigated by two factors. First, our algorithm is anytime, storing incrementally better solutions in *Best* (if they exist) as time progresses. Second, it is possible to use heuristics to improve the algorithm's performance. Before discussing such heuristics, we evaluate our algorithm in a simple domain.

5 Evaluation

We implemented our system in SWI-Prolog [6], and evaluated it on a simple bomb clearing scenario, as shown in Figure 2. The domain consists of a tile world with a single agent. Each tile could be empty, or contain a bomb that is either moderately (grey), or very (black), dangerous. The bomb clearing agent has only one plan available to it, abstractly represented as follows:

$\mathbf{andN}(scanC, moveC, \mathbf{orN}(nothing, pickup, explodeC))$

All except the *nothing* and *pickup* actions are in fact compound actions, made up of a **orN** of primitive actions. For example, the *moveC* action is defined as follows:

$$\begin{aligned} moveC \equiv \mathbf{orN}(&move(X,Y,A,B) \cdot A = X \wedge B = Y, \\ &move(X,Y,A,B) \cdot A = X+1 \wedge B = Y, \\ &move(X,Y,A,B) \cdot A = X-1 \wedge B = Y, \\ &move(X,Y,A,B) \cdot A = X \wedge B = Y+1, \\ &move(X,Y,A,B) \cdot A = X \wedge B = Y-1) \end{aligned}$$

This action thus allows the agent to move to a neighbouring tile, or stay in its current position (X and Y are replaced by the current position in our implementation). Similarly, the *scanC* action scans all four compass points around it to a range of 2, identifying the contents of the tile and its associated scan threat level. The *explodeC* action consists of an *orN* composed of 8 primitive actions, allowing the agent to trigger an explosion up to 2 tiles away from it. The pickup action, used to clear a bomb from the square occupied by the agent, is defined as

$$pickup(C, D), C = A \wedge D = B$$

Note that this action makes use of A and B, whose values are constrained by the *moveC* action.

We defined 10 normative rules for the system. Due to space constraints, we describe most of these only informally. The first normative rule is designed to prevent an agent from wandering out of the area in which bombs may exist.

[6] `http://www.swi-prolog.org`

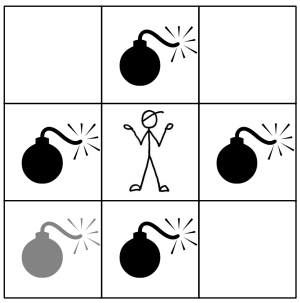

Fig. 2. A scenario in which norm aware and norm compliant agents will behave differently.

Additional normative rules are designed to prohibit an agent from moving onto a dangerous bomb, oblige the agent to explode such dangerous bombs, and also to oblige the agent to pick up low threat level bombs. Another rule prohibits explosions within 1 tile of the agent, and is defined as follows:

$$
\begin{aligned}
&move(R4XO, R4YO, R4X, R4Y) \cdot \top \Rightarrow \\
&\quad \oplus \mathsf{F} explode(R4A, R4B) \cdot (R4A = R4X \wedge R4B = R4Y) \vee \\
&\qquad\qquad (R4A = R4X - 1 \wedge R4B = R4Y) \vee \\
&\qquad\qquad (R4A = R4X + 1 \wedge R4B = R4Y) \vee \\
&\qquad\qquad (R4A = R4X \wedge R4B = R4Y - 1) \vee \\
&\qquad\qquad (R4A = R4X \wedge R4B = R4Y + 1)))
\end{aligned}
$$

A similar rule removing this obligation was also created. The order of rule evaluation (removal and then addition) allows these rules to operate correctly. Finally, 4 rules were defined to remove any obligations associated with bombs that have been removed from the environment. Finally, we associated a utility gain of 10 with exploding a bomb, and 5 with picking it up. We associated a utility cost of 100 with exploding a bomb too close to the agent, and a cost of 1 for moving into a square containing a dangerous bomb.

In this environment, an agent using the algorithms described in this paper will perform differently to a fully norm compliant agent. Consider the situation illustrated in Figure 2, where the agent is surrounded by dangerous bombs. A fully norm compliant agent will not move from its starting position as doing so would violate one of the norms imposed by its normative rules. An agent capable of violating norms will move into one of the dangerous squares (violating a less important norm) and explode the bomb opposite it from 2 squares away (complying with a more important norm), thereby freeing it to continue moving around the environment. It should be noted that this is the only situation (outside similar cases when the agent is in a corner or edge of the world) where norm awareness allows an agent to select a different action to one that would be chosen by a norm compliant agent. Given this, we saw only a small improvement in the

performance of the former agent over the latter when evaluated over randomly generated worlds.

Now norm-aware and norm-compliant agents should be contrasted with classic (norm unaware) BDI type agents. The latter type of agent, when operating in the sample domain, would execute some version of the plan at random, often moving into dangerous squares, randomly triggering explosions in tiles near them, and so on, and ultimately perform poorly. The difference between this type of agent, and the ones described previously does not lie in their plans, but rather in their norms. The ability to assign and modify norms in this way thus changes the behaviour of an agent *without requiring any modification to its plan library.*

The improved performance in bomb clearing comes at the cost of additional time; the norm aware and norm compliant agents both took approximately 13 seconds to select an action on a 2.4 GHz computer. This occurs as all possible executions of the plan are evaluated by the system. In the next section, we discuss a number of possible techniques for improving the performance of our algorithms.

6 Discussion

Algorithms such as A* have shown that the addition of a heuristic to estimate the remaining utility gained by executing the rest of a plan can improve the speed at which good solutions can be found. Making use of such a heuristic within our framework is simple, requiring only a change to to Line 18 of Algorithm 2. However, identifying an appropriate heuristic is more difficult. [12] suggests several heuristics usable in HTN planning with preferences, and inspired by these, possible heuristics include assuming that no more norm violations will occur; that all norms will be complied with; or that some norms will be ignored. More complex heuristics include Monte-Carlo sampling of a plan.

Pruning low utility elements from Υ can also improve algorithm runtime. This is achieved by modifying Line 22 of Algorithm 2 to not run if the candidate addition's utility is much worse than the current best solution's. However, this speedup comes at the cost of completeness unless the cost function is monotonic.

The focus of this paper is on the role of norms within plans. While our work can be viewed as a form of HTN planning, the presence of norms provides a differentiator for our work. Norms provide guarantees to open large-scale distributed systems, establishing limits to the autonomy of components/agents [1]. There have been attempts at connecting the computational behaviours of individual components/agents and norms, whether to detect norm violation [4], or with a view to verify if a set of norms can ever be fulfilled by a society of autonomous software agents [13], or to inform agents about changes in their behaviour so as to make the behaviours norm-compliant [10]. However, our problem is distinct, and our approach novel: autonomous agents, with access to a library of plans to choose from, but subject to norms, can make use of our mechanism to choose a plan that will achieve individual and global goals while attempting to abide by

these norms. Our approach was inspired by [13], which presents a mechanism to detect potential normative conflicts before they arise. However, that approach is overcautious, detecting conflicts that may never arise in actual system execution. In contrast, the work in this paper adopts a more accurate representation of agent behaviour, represented as a plan (with non-determinism in the choices of values for variables and choices for *OR* branches). Finally, Dignum et al. [2] propose the idea of an action having potential deontic effects. When reasoning about action execution, the norms resulting from the action are computed, and the norms resulting from those norms (e.g. contrary to duty obligations) are recursively identified. If normative conflict is detected, the action would not be executed. Dignum et al. focus on the deontic effects of a single action in the context of contrary to duty obligations, while we concentrate on the effects of norms on an entire plan.

Additionally, there is some similarity with work pursued by Governatori and others (e.g. [7,8]), in that both use an initial specification of possible behaviours, and check the norm-compliance of these behaviours. [7] presented an early form of Governatori's model, which concentrated on manually constructed plans, while [8] appears to be the most fully developed version of their approach. Both our approach and theirs contain conditional norms, which are represented as rules. However, there are also many differences. For example, while they utilise business process descriptions and informally define a mix of predicate and first order logic for their underlying representation, we use a more abstract, and simpler (but fully formalised) plan description. Furthermore, [8] addresses a specific class of norms, namely, reparational obligations, in which violated norms can be repaired or compensated via other norms, but does not address, for instance, the violation of prohibitions, as we have done. Furthermore, the propositional nature of their work makes handling deadlines difficult, and their approach does not support norm removal.

[9] also considered norm compliance. However, this work was at a more abstract level, and while the interaction between an agent's goals and norms was discussed, no computational mechanism for deciding whether to comply with a norm was proposed. Like us, [3] attempts to maximise utility based on the consequence of complying with or violating a norm, with future world states represented as MDPs. However, while norms in our framework act as constraints on the values of parameters, Fagundes consideres norms as affecting the ability to perform an action in a given space. Thus, the space of actions to be considered, and the effects of norm compliance and violation, are very different.

Our work can be compared with the work of Sohrabi *et al.* [12] on the **HTNPlan-P** planner. This planner uses an extended version of PDDL that allows preferences on the decompositions and actions employed in HTN planning in a similar way to which we use norms to restrict action execution. Their extended preferences include temporally extended preferences regarding when a particular action (or goal) should or should not be executable, conditional on a subset of linear temporal logic (LTL). Preferences in LTL do not interact in

the same way as permissions affect obligations and prohibitions, limiting the applicability of their techniques to our domain.

7 Conclusions and Future Work

When utilising offline planning, a plan is selected for execution from a pre-generated plan library, with the selection being based on the goal to be met, and on the current state of the environment in which the plan is to be executed. However, such a plan cannot easily be adapted to operate under normative constraints which were not originally anticipated by the plan designer. By making use of a utility based approach, we have shown how a reasoner can act in an optimal manner, violating, and complying with norms as needed.

We can identify a number of avenues of future work. Our current focus involves investigating the heuristics discussed in Section 6. Additionally, as mentioned previously, the our obligations, prohibitions and permissions can be viewed as a specific type of conditional norm, imposing constraints on the manner in which an action should, should not, or may be executed, but only in the case that the action is executed. We intend to extend our framework to cope with more general norms, for example, obliging an action to be executed subject to some specific constraints. Such an extension would allow us to apply our work to areas outside the practical reasoning domain, such as electronic contracts, where the contracting parties can analyse the contract, and their plans, in order to determine whether they can achieve their own goals in a satisfactory manner while following the contract. We also intend to enrich our representation language in order to allow for constraints over finite sets and inference over rules. Finally, we intend to investigate how our approach can be adapted to domains containing uncertainty.

References

1. Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., van der Torre, L.: The BOID architecture. In: Proc. 5th Int'l Conf. on Autonomous Agents. pp. 9–16 (2001), `citeseer.nj.nec.com/broersen01boid.html`
2. Dignum, F., Morley, D., Sonenberg, E.A., Cavedon, L.: Towards socially sophisticated BDI agents. In: Proc. ICMAS-2000. pp. 111–118 (2000), `citeseer.nj.nec.com/dignum00towards.html`
3. Fagundes, M.S., Billhardt, H., Ossowski, S.: Reasoning about norm compliance with rational agents. In: Proc. ECAI-10. pp. 1027–1028 (2010)
4. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Constraint rule-based programming of norms for electronic institutions. JAAMAS 18(1), 186–217 (2009)
5. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kauffman (2004)
6. Governatori, G., Hulstijn, J., Riveret, R., Rotolo, A.: Characterising deadlines in temporal modal defeasible logic. In: Proc. AI-2007. LNAI, vol. 4830, pp. 486–496 (2007)

7. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: 10th Int'l Enterprise Distributed Object Computing Conf. pp. 221–232 (2006)
8. Governatori, G., Rotolo, A.: How do agents comply with norms? In: Proc. NorMAS. No. 09121 (2009), http://drops.dagstuhl.de/opus/volltexte/2009/1909
9. López y López, F., Luck, M., d'Inverno, M.: A normative framework for agent-based systems. In: Boella, G., van der Torre, L.W.N., Verhagen, H. (eds.) Proc. NorMAS. No. 07122 (2007)
10. Meneguzzi, F., Luck, M.: Norm-based behaviour modification in BDI agents. In: Proc. AAMAS 2009. pp. 177–184 (2009), http://portal.acm.org/citation.cfm?id=1558013.1558037
11. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Proc. MAAMAW 1996. pp. 42–55 (1996)
12. Sohrabi, S., Baier, J.A., McIlraith, S.A.: HTN planning with preferences. In: Proc. IJCAI-09. pp. 1790 –1797 (2009)
13. Vasconcelos, W., Kollingbaum, M., Norman, T.J.: Normative conflict resolution in multi-agent systems. JAAMAS 19(2), 124–152 (2009)