Capability Models and Their Applications in Planning

Yu Zhang
Dept. of Computer Science
Arizona State University
Tempe, AZ
yzhan442@asu.edu

Sarath Sreedharan
Dept. of Computer Science
Arizona State University
Tempe, AZ
ssreedh3@asu.edu

Subbarao Kambhampati
Dept. of Computer Science
Arizona State University
Tempe, AZ
rao@asu.edu

ABSTRACT

One important challenge for a set of agents to achieve more efficient collaboration is for these agents to maintain proper models of each other. An important aspect of these models of other agents is that they are often not provided, and hence must be learned from plan execution traces. As a result, these models of other agents are inherently partial and incomplete. Most existing agent models are based on action modeling and do not naturally allow for incompleteness. In this paper, we introduce a new and inherently incomplete modeling approach based on the representation of capabilities, which has several unique advantages. First, we show that the structures of capability models can be learned or easily specified, and both model structure and parameter learning are robust to high degrees of incompleteness in plan traces (e.g., with only start and end states partially observed). Furthermore, parameter learning can be performed efficiently online via Bayesian learning. While high degrees of incompleteness in plan traces presents learning challenges for traditional (complete) models, capability models can still learn to extract useful information. As a result, capability models are useful in applications in which traditional models are difficult to obtain, or models must be learned from incomplete plan traces, e.g., robots learning human models from observations and interactions. Furthermore, we discuss using capability models for single agent planning, and then extend it to multi-agent planning (with each agent modeled separately by a capability model), in which the capability models of agents are used by a centralized planner. The limitation, however, is that the synthesized "plans" (called c-plans) are incomplete, i.e., there may or may not be a complete plan for a c-plan. This is, however, unavoidable for planning using partial and incomplete models (e.g., considering planning using action models learned from partial and noisy plan traces).

Categories and Subject Descriptors

I.2.11 [**Multiagent systems**]; I.2.6 [**Knowledge acquisition**]; I.2.8 [**Plan execution, formation, and generation**]

Keywords

Capability models; Agent theories and models; Teamwork in human-agent mixed networks

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

1. INTRODUCTION

One important challenge for a set of agents to achieve more efficient collaboration is for these agents to maintain proper models of others. These models can be used by a centralized planner (e.g., on a robot) or via a distributed planning process to perform task planning and allocation, or by the agents themselves to reduce communication and collaboration efforts. In many applications, these models of other agents are not provided and hence must be learned. As a result, these models are going to be inherently partial and incomplete. Thus far, most traditional agent models are based on action modeling (e.g., [18, 7]). These models are not designed with partial information in mind and hence are complete in nature. In this paper, we introduce a new and inherently incomplete modeling approach based on the representation of capabilities. We represent a capability as the ability to achieve a partial state given another partial state. A capability can be fulfilled (or realized) by any action sequence (or plan) that can implement a transition between the two partial states. Each such action sequence is called an *operation* in this paper. A capability model can encode all possible capabilities for an agent in a given domain, as well as capture the probabilities of the existence of an operation to fulfill these capabilities (to implement the associated transitions). These probabilities determine which capabilities are more likely to be used in planning.

Compared to traditional agent models which are complete in nature, capability models have their unique benefits and limitations. In this aspect, capability models should not be considered as a competitor to complete models. Instead, they are useful when complete models are difficult to obtain or only partial and incomplete information can be retrieved to learn the models. This is often true when humans must be modeled.

The representation of a capability model is a generalization of a two time slice dynamic Bayesian network (2-TBN). While a 2-TBN is often used to represent a single action (e.g., [19]), a capability model can encode all possible capabilities for an agent in a given domain. In a capability model, each node in the first time slice (also called a *fact node*) represents a variable that is used in the specification of the initial world state. For each fact node, there is also a corresponding node in the second time slice, which represents the fact node in the eventual state (i.e., after applying a capability). These corresponding nodes are called eventual nodes or e-nodes. The state specified by the fact nodes is referred to as the *initial state*, and the state specified by the e-nodes is referred to as the *eventual state*. The edges between the nodes within the initial and eventual states, respectively, encode the correlations between the variables at the same time instance (i.e., variables in synchrony). The edges from the fact nodes to e-nodes encode *causal relationships*. Both types of edges can be learned (e.g., using learning techniques in [24, 25] for causal relationships). In the case that no prior infor-

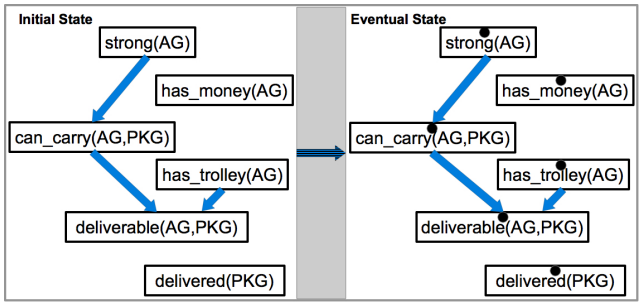

Figure 1: Capability model (as a generalized 2-TBN) for a human delivery agent (denoted as AG) in our motivating example with 6 fact node and e-node pairs. This model can encode all possible capabilities of the agent in this domain. Each fact node corresponds to a variable in the specification of the initial state. e-nodes are labeled with a dot on top of the variable. Edges from the fact nodes to the corresponding e-nodes are simplified for a cleaner representation. Any partial initial state coupled with any partial eventual state may imply a capability.

mation is provided, the safest approach is to not assume any independence. Consequently, we can specify a total ordering between the nodes within the initial and final states, and connect a node at a given level to all nodes at lower levels, as well as every fact node to every e-node. After model construction, parameter learning of a capability model can be performed efficiently online via Bayesian learning.

One of the unique advantages of capability models is that learning for both model structure and parameters is robust to high degrees of incompleteness in plan execution traces. This incompleteness occurs commonly. For example, the execution observer may not be constantly monitoring the executing agent, and the executing agent may not always report the full traces [26]. Capability models can be learned even when only the initial and final states are partially observed in plan traces for training. While high degrees of incompleteness in plan traces presents learning challenges for traditional models, capability models can still learn to extract useful information out of them, e.g., probabilities of the existence of an operation to achieve certain eventual states given certain initial states.

Furthermore, compared to traditional models for planning, capability models only suggest transitions between partial states (as capabilities), along with the probabilities that specify how likely such transitions can be implemented. Hence, a "plan" synthesized with capability models is incomplete, and we refer to such a plan as a *c-plan* (i.e., involving the application of capabilities). More specifically, we show that "planning" with capability models occurs in the belief space, and is performed via Bayesian inference. At any planning step, the current planning state is a belief state. Applying a capability to this belief state requires the planner to update this state using Bayesian inference on the capability model. After the discussion of single agent planning, we then extend to multi-agent planning.[1] In such cases, we can consider planning as assigning subtasks to agents without understanding precisely how these subtasks are to be handled. Moreover, multi-agent planning can be performed when part of the agents are modeled by capability models, and the other agents are modeled by complete models. This situation can naturally occur in human-robot teaming scenarios, in which human models need to be learned and the robot models are given. Similarly, the synthesized multi-agent c-plans are incomplete (unless all plan steps use robot actions). c-plans are useful when we have to plan using partial and incomplete models (e.g., considering planning using action models learned from partial and noisy plan traces), since they inform the user how likely complete plans can be realized by following their "guidelines".

The rest of the paper is organized as follows. First, we provide a motivating example in Section 2. In Section 3, we discuss capability models in detail. We discuss how to use capability models in planning in Sections 4 and 5. The relationships between capability models and other existing approaches to modeling the dynamics of agents are discussed as related work in Section 6. We conclude afterwards.

2. MOTIVATING EXAMPLE

We start with a motivating example for capability models. In this example, we have a set of human delivery agents, and the tasks involve delivering packages to their destinations. However, there are a few complications here. An agent may be able to carry a package if this agent is strong enough. While an agent can use a trolley to load a package, this agent may not remember to bring a trolley initially. An agent can visit an equipment rental office to rent a trolley, if this agent remembers to bring money for the rental. A trolley increases the chance of an agent being able to deliver a package (i.e., how likely the package is deliverable by this agent). Figure 1 presents the capability model for a human delivery agent, which is a generalized 2-TBN that can encode all possible capabilities for the agent in this domain. Similar to 2-TBN, fact nodes (e-nodes) in a capability model are in synchrony in the initial (eventual) state. Note that any partial initial state coupled with any partial eventual state may imply a capability.

In this example, the correlations between the variables are clear. For example, whether an agent can carry a package is dependent on whether the agent is strong; whether an agent can deliver a package may be influenced by whether the agent has a trolley. These correlations are represented as edges between the variables within the initial and eventual states, respectively and identically. Assuming that no prior information is provided for the causal relationships, we connect every fact node with every e-node (denoted as a single edge in Figure 1).

After obtaining the model structure, to perform parameter learning for the capability model of an agent, this agent can be given delivery tasks spanning a period of time for training purpose. However, the only observations that a manager has access to may be the number of packages that have been successfully delivered by this agent during this period. While this presents significant challenges for learning the parameters of traditional complete models, the useful information for the manager is already encoded: the probability that this agent can deliver a package. Capability models can learn this information from the incomplete traces.

Now, suppose that the manager also has a set of robots (with action models) to use that can fetch items for the delivery agents. Since it is observed that the presence of a trolley increases the probability of successful delivery based on the capability models, the manager can make a multi-agent c-plan, which ensures that the robots deliver a trolley at least to some of the delivery agents. This also illustrates that it is beneficial to combine capability models with other complete models (i.e., action models) when they are available (e.g., for robots in this example).

[1]In this paper, we consider sequential multi-agent plans; synchronous or joint actions of agents are not considered.

3. CAPABILITY MODEL

In our settings, the environment includes a set of agents Φ working inside it. For each agent, for simplicity, we assume that the state of the world and this agent is specified by a set of boolean variables $X_\phi (\phi \in \Phi)$. This implies that an agent can only interact with other agents through variables that pertain to the world state.

More specifically, for all $X_i \in X_\phi$, X_i has domain $D(X_i) = \{true, false\}$. To specify partial state, we augment the domains of variables to include an unknown or unspecified value as in [1]. We write $D^+(X_i) = D(X_i) \cup \{u\}$. Hence, the (partial) state space is denoted as $S = D^+(X_1) \times D^+(X_2) \times ... \times D^+(X_N)$, in which $N = |\bigcup_\phi X_\phi|$.

3.1 Capability

First, we formally define capability for an agent $\phi \in \Phi$. The state space of agent ϕ is denoted as S_ϕ.

DEFINITION 1 (CAPABILITY). *Given an agent ϕ, a capability specified as a mapping $S_\phi \times S_\phi \rightarrow [0, 1]$, is an assertion about the probability of the existence of complete plans (for ϕ) that connect an initial state (i.e., the first component on the left hand side of the mapping) to an eventual state (i.e., the second component).*

A capability is also denoted as $s_I \Rightarrow s_E$ (i.e., the initial state $\Rightarrow$ the eventual state) when we do not need to reference the associated probability value. The probability value is denoted as $P(s_I \Rightarrow s_E)$, which we will show later is computed based on the capability model via Bayesian inference. There are a few notes about Definition 1. First, both s_I and s_E can be partial states: the variables that have the value u are assumed to be unknown (in the initial state) and unspecified (in the eventual state). In this paper, we refer to a complete plan that fulfills (or realizes) a capability as an *operation*, which is going to be decided and implemented by the executing agent during execution. Although capabilities seem to be similar to high-level actions in HTN, the semantics is different from angelic uncertainty in HTN [12]. While the existence of a concretization is ensured in HTN planning via reduction schemas, it is only known with some level of certainty that a concretization (i.e., operation) exists for a capability in a capability model, and the capability model does not provide specifics for such a concretization. More discussions on this are provided in Section 6 when we discuss the relationships between capability models and existing approaches to modeling agent dynamics.

Capability models also address the qualification and ramification problems, which are assumed away in STRIPS planning (and planning with many complete models). More specifically, an operation for a capability $s_I \Rightarrow s_E$ may be dependent on variables with unknown values in s_I, and updating variables with unspecified values in s_E. This is a unique characterization of capability and critical for learning capability models with incomplete observations.

For example, a capability may specify that given coffee beans, an agent can make a cup of coffee. Thus, we have a capability {*Has coffee beans = true*} $\Rightarrow$ {*Has coffee = true*}. An operation for this capability to make a coffee may be dependent on the availability of water initially, which is not specified in the capability. Similarly, this operation may negate the fact that the kitchen is clean, which is not specified in the capability either. Note that a capability may be fulfilled by operations with different specifications of the initial and eventual states, as long as these specifications all satisfy the specification of this capability.

A *capability model* of an agent is capable of encoding all capabilities of this agent given a domain; it is designed to encode the following probability distribution:

$$P(X_\phi, \dot{X}_\phi) = \int_0^T P(X_\phi, \dot{X}_\phi, t)\, dt \quad (1)$$

in which T represents the maximum length of any operation (i.e., number of actions) for ϕ.[2] X_ϕ represents the initial state and $\dot{X}_\phi$ represents the eventual state. Furthermore, $P(\dot{X}_\phi, t | X_\phi)$ is the probability of any operation resulting in $\dot{X}_\phi$ in exact time t given X_ϕ. Hence, the probability that is associated with a capability $s_I \Rightarrow s_E$ (i.e., $P(s_I \Rightarrow s_E)$) encodes:

$$P(\dot{X}_\phi = s_E \,|\, X_\phi = s_I) = \int_t P(\dot{X}_\phi = s_E, t \,|\, X_\phi = s_I)\, dt \quad (2)$$

We construct the capability model of an agent as a Bayesian network. As an inherently incomplete model, it not only allows the initial and eventual states to be partially specified for any capability (and hence the fulfilling operations) that it encodes, but also allows the correlations between the variables within the initial and eventual states, as well as the causal relationships between them to be partially specified (e.g., when learning from plan traces). However, there are certain implications in this (e.g., the modeling can lose some information), which we will discuss in Section 3.4. Along the line of the qualification and ramification problems, a capability model also allows certain variables to be excluded completely from the network (i.e., related variables that are not captured in X_ϕ) due to incomplete knowledge. For example, whether an agent can drive a car (with a manual transmission) to a goal location is dependent on whether the agent can drive a manual car, even through the agent has a driver license. In this case, the ability to drive a manual car may have been ignored when creating the model.

3.2 Model Construction

We construct the capability model of each agent as an *augmented Bayesian network* [14]. Any partial initial state coupled with any partial eventual state may imply a capability; the probability that a capability actually exists (i.e., it can be fulfilled by an operation) is computed via a Bayesian inference in the network. We use augmented Bayesian network since it allows prior beliefs of conditional relative frequencies to be specified before observations are made, as well as enables us to adjust how fast these beliefs should change.

DEFINITION 2. *An augmented Bayesian network (ABN) (G, F, ρ) is a Bayesian network with the following specifications:*

- *A DAG $G = (V, E)$, where V is a set of random variables, $V = \{V_1, V_2, ..., V_n\}$.*
- *F is a set of auxiliary parent variables for V.*
- *$\forall V_i \in V$, an auxiliary parent variable $F_i \in F$ of V_i, and a density function ρ_i associated with F_i. Each F_i is a root and it is only connected to V_i.*
- *$\forall V_i \in V$, for all values pa_i of the parents $PA_i \subseteq V$ of V_i, and for all values f_i of F_i, a probability distribution $P(V_i|pa_i, f_i)$.*

A capability model of an agent ϕ is then defined as follows:

DEFINITION 3 (CAPABILITY MODEL). *A capability model of an agent ϕ, as a binomial ABN (G_ϕ, F, ρ), has the following specifications:*

[2]We assume in this paper that time is discretized.

- $V_\phi = X_\phi \cup \dot{X}_\phi$.
- $\forall V_i \in V_\phi$, *the domain of* V_i *is* $D(V_i) = \{true, false\}$.
- $\forall V_i \in V_\phi$, $F_i = \{F_{i1}, F_{i2}, ...\}$, *and each* F_{ij} *is a root and has a density function* $\rho_{ij}(f_{ij})$ *(*$0 \leq f_{ij} \leq 1$*). (For each value* pa_{ij} *of the parents* PA_i*, there is an associated variable* F_{ij}*.)*
- $\forall V_i \in V_\phi$, $P(V_i = true|pa_{ij}, f_{i1}, ...f_{ij}, ...) = f_{ij}$.

in which j in pa_{ij} indexes into the values of PA_i. j in f_{ij} indexes into the variables in F_i. Note that defining partial states (i.e., allowing variables to assume the value u in a state) is used to more conveniently specify the distribution in Equation 1. Variables in the capability model do not need to expand their domains to include u.

For edge construction, we can learn the correlations and causal relationships from plan traces. Note that the correlations between variables must not form loops; otherwise, they need to be broken randomly. When no training information is provided, we can specify a total ordering between the nodes within the initial and final states, and connect a node at a given level to all nodes at lower levels, as well as every fact node to every e-node. Denote the set of edges as E_ϕ. We then have constructed the capability model $G_\phi = (V_\phi, E_\phi)$ for ϕ. Figure 1 provides a simple example of a capability model.

3.3 Parameter Learning

In this section, we describe how the model parameters can be learned. The parameter learning of a capability model is performed online through Bayesian learning. The initial model parameters can be computed from existing plan traces by learning the density functions (i.e., f_{ij}) in Definition 3. These parameters can then be updated online as more traces are collected (i.e., as the agent interacting with the environment).

Plan execution traces can be collected each time that a plan (or a sequence of actions) is executed, whether succeeds or fails.

DEFINITION 4 (COMPLETE PLAN TRACE). *A complete plan trace is a continuous sequence of state observations over time, denoted as* $\mathcal{T} = \langle s_1^*, s_2^*, ..., s_L^* \rangle$*, in which L is the length of the plan and* s_i^* *denotes a complete state (i.e.,* $s_i^* \in D(X_1) \times D(X_2) \times ... \times D(X_N)$*).*

However, in real-world situations, plan traces may be incomplete. The incompleteness can come from two aspects. First, the observed state may be partial. Second, the observations may not be continuous. Hence, we are going to have partial plan traces.

DEFINITION 5 (PARTIAL PLAN TRACE). *A partial plan trace is a discontinuous sequence of partial state observations over time, denoted as* $\mathcal{T} = \langle s_i, s_{i+k_1}, s_{i+k_2}, ... \rangle$*, in which i denotes the time step in the complete plan and* s_i *denotes a partial state (i.e.,* $s_i \in D^+(X_1) \times D^+(X_2) \times ... \times D^+(X_N)$*).*

Note that the only assumption that is made in Definition 5 is that at least two different partial states must be observed during the plan execution. This means that even the start and end states of a plan execution do not necessarily have to be observed or partially observed, which is especially useful in real-world situations where a plan trace may only be a few samplings of (partial) observations during a plan execution. Note also that since the observations are in discontinuous time steps, the transition between contiguous state observations is not necessarily the result of a single action. Instead, it is the result of a plan (i.e., operation) execution. Henceforth, when we refer to plan traces, we always intend to mean partial plan traces, unless otherwise specified.

When more than two states are observed in a plan trace, it can be considered as a set of traces, with each pair of contiguous states as a separate trace. When the states are partially observed in a plan trace, it can be considered as a set of compatible traces with complete state observations.[3] For simplicity, we assume in the following that the plan execution traces used in learning have complete state observations. We denote this set of traces as D.

To learn the parameters of a capability model, a common way is to model F_{ij} using a beta distribution (i.e., as its density function ρ). Denote the parameters for the beta distribution of F_{ij} as a_{ij} and b_{ij}. Then, we have:

$$P(X_i = \text{true}|pa_{ij}) = \frac{a_{ij}}{a_{ij} + b_{ij}} \tag{3}$$

Suppose that the initial values or the current values for a_{ij} and b_{ij} are given. The remaining task is to update a_{ij} and b_{ij} from the given traces. Given the training set D, we can now follow Bayesian inference to update the parameters of F_{ij} as follows:

Initially or currently,

$$\rho(f_{ij}) = beta(f_{ij}; a_{ij}, b_{ij}) \tag{4}$$

After observing new training examples D, we have:

$$\rho(f_{ij}|D) = beta(f_{ij}; a_{ij} + s_{ij}, b_{ij} + t_{ij}) \tag{5}$$

in which s_{ij} is the number of times for which X_i is *true* while PA_i assuming the value of pa_{ij}, and t_{ij} is the number in which it equals X_i is *false* while PA_i assuming the value of pa_{ij}.

3.4 Implications

In this section, we discuss several implications of capability models with a simple example. We first investigate how information can be lost during learning when the correlations or causal relationships among the variables are only partially captured. This can occur, for example, when model structure is learned. In this example, we have two blocks A, B, and a table. Both blocks can be placed on the table or on each other. The capability model for an agent is specified in Figure 2. Initially, assuming that we do not have any knowledge of the domain, the density functions can be specified as beta distributions with $a = b$. In Figure 2, we use $a = b = 1$ for all distributions. Suppose that we observe the following plan trace, which can be the result of executing a single action that places A on B:

$$s_1 : OnTable(A) \wedge OnTable(B) \wedge \neg On(A, B) \wedge \neg On(B, A)$$
$$s_2 : \neg OnTable(A) \wedge OnTable(B) \wedge On(A, B) \wedge \neg On(B, A)$$

Based on the learning rules in Equation (5), we can update the beta distributions accordingly. For example, the beta distribution for $\dot{X}_3$ (i.e., $On(A, B)$ in the eventual state) is updated to *beta*($X_1 =$ *true*, $X_4 =$ *true*, $\dot{X}_4 =$ *false*, ...; 2, 1). This means that if both A and B are on the table, it becomes more likely that a capability exists for making $On(A, B) =$ *true*. This is understandable since an action that places A on B would achieve $On(A, B) =$ *true* in such cases. For actions with uncertainties (i.e., when using a robotic arm to perform the placing action), this beta distribution would converge to the success rate as more experiences are obtained.

Meanwhile, we also have that the beta distribution for $\dot{X}_2$ is updated to *beta*($X_1 =$ *true*, $X_3 =$ *false*, $\dot{X}_4 =$ *false*, $\dot{X}_3 =$

[3]There is no need to expand such traces into sets of traces with complete state observations for learning, since it can be equivalently considered using arithmetic operations.

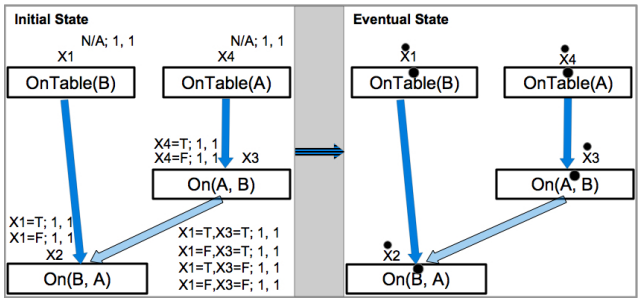

Figure 2: Capability model for an agent in a simple domain with four variables. $OnTable(A)$ means that object A is on the table. $On(A, B)$ means that object A is on B. The correlations that correspond to the light blue arrows distinguish between two scenarios: one with proper model structure and one without (i.e., when certain correlations are missing). For clarity, we only specify the density functions of the variables in the initial state. We also show the two sets of density functions for the augmented variables for X_2 for the two different scenarios, respectively.

true, ...; 1, 2). This means that if A is on B in the eventual state, it becomes less likely that B can also be on A in the eventual state. This is intuitive since we know that achieving $On(A, B) = true$ and $On(B, A) = true$ at the same time is impossible. In this way, capability models can reinforce the correlations between the variables as experiences are observed. The implication is that the edges (capturing the correlations) between these variables must be present; otherwise, information may be lost as described above. If the correlations are not fully captured by the model, for example, when the light blue arrows in Figure 2 are not present, the beta distribution of $\dot{X}_2$ would not be updated as above, since $\dot{X}_3$ would no longer be a parent node of $\dot{X}_2$. A similar implication also applies to causal relationships.

When environment changes, previous knowledge that is learned by the model may no longer be applicable. However, as the parameters grow, the learned model can be reluctant to adapt to the new environment. This issue can be alleviated by performing normalizations occasionally (i.e., dividing all a and b by a constant in the beta distributions), or by weighting previous experiences with a discounting factor.

4. USING CAPABILITY MODELS IN PLANNING

Capability models described in the previous section allow us to capture the capabilities of agents. In this section, we discuss the application of capability models in single agent planning (i.e., with an agent ϕ modeled by a capability model). We extend the discussion to multi-agent planing in the next section. Generally, planning with capability models occurs in the *belief space*, as with POMDPs [10].

4.1 Single Agent Planning

First, note that applying a capability $s_I \Rightarrow s_E$ of agent ϕ to a complete state s^* results in a belief state $b(\mathcal{S})$ as long as $s_I \sqsubseteq s^*$ (otherwise, this capability cannot be applied), in which $s_I \sqsubseteq s^*$ denotes that all the variables that are not assigned to u in s_I have the same values as those in s^*. After applying the capability, assuming successfully, we can compute the probability weight of a state s that satisfies $s_E \sqsubseteq s$ in the resulting belief state as follows:

$$P(s) = \frac{P(s^* \Rightarrow s)}{P(s^* \Rightarrow s_E)} = \frac{P(\dot{X}_\phi = s | X_\phi = s^*)}{P(\dot{X}_\phi = s_E | X_\phi = s^*)} \quad (6)$$

For any state s that does not satisfy $s_E \sqsubseteq s$, we have $P(s) = 0$. Denote $\mathcal{S}$ in $b(\mathcal{S})$ as $\mathcal{S} = \{s | s_E \sqsubseteq s \text{ and } s \text{ is a complete state}\}$. Clearly, we have:

$$\sum_{s \in \mathcal{S}} P(s) = 1 \quad (7)$$

Since there can be an exponential number of complete states in a belief state, depending on how many variables are assigned to u, we can use a sampling approach (e.g., Monte Carlo sampling) to keep a set of complete states to represent $b(\mathcal{S})$. We denote the belief state after sampling as $\hat{b}(\mathcal{S})$.

When applying a capability $s_I \Rightarrow s_E$ to a given belief state $\hat{b}(\mathcal{S})$, for each complete state in $\mathcal{S}$, we can perform sampling based on Equation (6), which returns a set of complete states with weights after applying the capability. We can then perform resampling on the computed sets of states for all states in $\mathcal{S}$ to compute the new belief state $\hat{b}(\mathcal{S}')$. In this way, we can connect different capabilities of an agent to create *c-plans*, which are plans in which capabilities are involved. Next, we formally define a planning problem for a single agent with a capability model.

DEFINITION 6 (SINGLE AGENT PLANNING PROBLEM). *A single agent planning problem with capability models is a tuple $\langle \phi, b(\mathcal{I}), G, \rho \rangle$, in which $b(\mathcal{I})$ is the initial belief state, G is the set of goal variables, and ρ is a real value in $(0, 1]$. The capability model of the agent is $G_\phi = (V_\phi, E_\phi)$. When we write ρ^* instead of ρ in the problem, it indicates a variance of the problem that needs to maximize ρ.*

DEFINITION 7 (SINGLE AGENT C-PLAN). *A single agent c-plan for a problem $\langle \phi, b(\mathcal{I}), G, \rho \rangle$ is a sequence of application of capabilities, such that the sum of the weights of the complete states in the belief state which include the goal variables (i.e., $G \sqsubseteq s$), is no less than ρ (or maximized for ρ^*) after the application.*

The synthesized single agent "plan" (i.e., a c-plan) is incomplete: it does not specify which operation fulfills each capability. In fact, it only informs the user how likely there exists such an operation. A single agent c-plan can be considered to provide *likely landmarks* for the agent to follow. For example, in Figure 2, suppose that the initial state is

$$s_I : On(A, B) \wedge \neg On(B, A)$$

and the goal is

$$s_E : \neg On(A, B) \wedge On(B, A)$$

A c-plan created with a capability model may include an intermediate state in the form of $s_I \Rightarrow s_{in} \Rightarrow s_E$,[4] in which s_{in} can include, e.g., $OnTable(A) =$ true and $OnTable(B) =$ true, such that the probability of the existence of a sequence of operations to fulfill the c-plan may be increased. Another example is our motivating example (Figure 1) in which having $has\_trolley(AG) = true$ as an intermediate state helps delivery when $has\_trolley(AG)$ is false at the beginning.

Although a single agent c-plan may seem to be less useful than a complete plan synthesized with action models, it is unavoidable

[4]Note that $P(s_I \Rightarrow s_E)$ is not equivalent to $P(s_I \Rightarrow s_{in}) \cdot P(s_{in} \Rightarrow s_E)$.

when we have to plan with incomplete knowledge (e.g., incomplete action models). This is arguably more common in multi-agent systems, in which the planner needs to reason about likely plans for agents even when it does not have complete information about these agents. A specific application in such cases is when we need to perform task allocation, in which it is useful for the system to specify subtasks or state requirements for agents that are likely to achieve them without understanding how the agents achieve them.

4.2 Planning Heuristic

Given a single agent planning problem $\langle \phi, b(\mathcal{I}), G, \rho \rangle$, besides achieving the goal variables, planning should also aim to reduce the *cost* of the c-plan (i.e., probability of success for the sequence of application of capabilities in the c-plan).

ASSUMPTION: To create a heuristic, we make the following assumption – capabilities do not have variables with *false* values in s_I. With this assumption, we have the following monotonicity properties hold:

$$P(s_I \Rightarrow s_E) \geq P(s'_I \Rightarrow s_E)(T(s'_I) \subseteq T(s_I) \wedge F(s_I) \subseteq F(s'_I)) \tag{8}$$

in which T, F are used as operators on a set of variables to denote the set of variables with *true* and *false* values, respectively. Equation (8) implies it is always easier to achieve the desired state with more *true*-value variables and less *false*-value variables in the initial state; and

$$P(s_I \Rightarrow s_E) \geq P(s_I \Rightarrow s'_E)(T(s_E) \subseteq T(s'_E) \wedge F(s_E) \subseteq F(s'_E)) \tag{9}$$

which implies that it is always more difficult to achieve the specified values for more variables in the eventual state.

HEURISTIC: We use A^* to perform the planning. At any time the current planning state is a belief state $\hat{b}(\mathcal{S})$ (i.e., $b(\mathcal{S})$ after sampling); there is also a sequence of application of capabilities (i.e., a c-plan prefix), denoted as π, to reach this belief state from the initial state. We compute the heuristic value for $\hat{b}(\mathcal{S})$ (i.e., $f(\hat{b}(\mathcal{S})) = g(\hat{b}(\mathcal{S})) + h(\hat{b}(\mathcal{S}))$) as follows. First, we compute $g(\hat{b}(\mathcal{S}))$ as the sum of the negative logarithms of the associated probabilities of capabilities in π.

To compute $h(\hat{b}(\mathcal{S}))$, we need to first compute $h(s)$ for each complete state $s \in \mathcal{S}$. To compute an admissible $h(s)$ value, we denote the set of goal variables that are currently *false* in s as G_s. Then, we compute $h(s)$ as follows:

$$h(s) = \underset{v \in G_s, s_{\neg v}}{\operatorname{argmax}} -\log P(s_{\neg v} \Rightarrow s\{v = \textit{true}\}) \tag{10}$$

in which $s_{\neg v}$ denotes a complete state with only v as *false*, and $s\{v = \textit{true}\}$ denotes the state of s after making v *true*. Finally, we compute $h(\hat{b}(\mathcal{S}))$ as:

$$h(\hat{b}(\mathcal{S})) = \sum_{s \in \mathcal{S}} P(s) \cdot h(s) \tag{11}$$

LEMMA 1. *The heuristic given in Eq.* (11) *is admissible for finding a c-plan that maximizes ρ (i.e., with ρ^*), given that $\hat{b}(\mathcal{S})$ accurately represents $b(\mathcal{S})$.*

PROOF. We need to prove that $h(\hat{b}(\mathcal{S}))$ is not an over-estimate of the cost for $\mathcal{S}$ to reach the goal state G while maximizing ρ. First, note that we can always increase ρ by trying to move a non-goal state (in the current planning state) to a goal state (i.e., $G \sqsubseteq s$). Furthermore, for each $s \in \mathcal{S}$, given the monotonicity properties, we know that at least $h(s)$ cost must be incurred to satisfy G. Hence, the conclusion holds. □

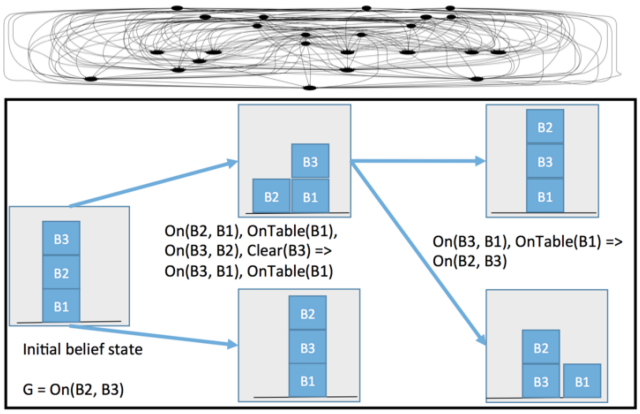

Figure 3: Illustration of solving a single agent planning problem with a capability model. The topology of the capability model used is shown in the top part of the figure.

Lemma 1 implies that the A^* search using the heuristic in Equation (11) would continue to improve ρ given sufficient time. Hence, the heuristic should be used as an anytime heuristic and stop when a desired value of ρ is obtained or the increment is below a threshold. Also, approximation solutions should be considered in future work to scale to large networks.

4.3 Evaluation

We provide a preliminary evaluation of single agent planning with a capability model. We build this evaluation based on the blocksworld domain. First, we use 20 problem instances with 3 blocks[5] and generate a complete plan for each instance. Then, we randomly remove $1-5$ actions from each complete plan to simulate partial plan traces.

The capability model for this domain contains 12 variables and we ignore the $holding$ and $handempty$ predicates to simulate partial state observations. We manually construct the correlations between the variables in the initial and eventual states. For example, $On(A, B)$ is connected with $On(B, A)$ since they are clearly conflicting. For causal relationships, we connect every node in the initial state to every node in the eventual state.

After learning the parameters of this capability model based on the partial plan traces, we apply the capability model to solve a problem as shown in Figure 3. The topology of the capability model constructed is shown in the top part of the figure, which is also the model used in Figure 4. Since we have connected every fact node with every e-node in this case, the model appears to be quite complex. Initially, we have $On(B3, B2)$, $On(B2, B1)$, and $OnTable(B1)$ (a complete state), and the goal is to achieve $On(B2, B3)$. We can see in Figure 3 that $\mathcal{I}$ in $b(\mathcal{I})$ contains a single complete state. The c-plan involves the application of two different capabilities. For illustration purpose, we only show two possible states (in the belief state) after applying the first capability,

For one of the two states, we then show two possible states after applying the second capability. Note that each arrow represents a sequence of actions in the original action model. Also, the possible states must be compatible with the specifications of the eventual states in the capabilities. We see some interesting capabilities being

[5]It does not have to be three blocks but we assume only three blocks to simplify the implementation.

learned. For example, the first capability is to pull out a block that is between two other blocks, and place it somewhere else.

5. MULTI-AGENT PLANNING

In this section, we extend our discussion from single agent planning to multi-agent planning. In particular, we discuss how capability models can be used along with other complete models (i.e., action models) by a centralized planner to synthesize c-plans. The settings are similar to those in our motivating example. In particular, we refer to multi-agent planning with mixed models as MAP-MM. This formulation is useful in applications in which both human and robotic agents are involved. While robotic agents are programmed and hence have complete models, the models of human agents must be learned. Hence, we use capability models to model human agents and assume that the models for the human agents are already learned (e.g., by the robots) for planning.

For robotic agents, we assume STRIPS action model $\langle \mathcal{R}, \mathcal{O} \rangle$, in which $\mathcal{R}$ is a set of predicates with typed variables, $\mathcal{O}$ is a set of STRIPS operators. Each operator $o \in \mathcal{O}$ is associated with a set of preconditions $Pre(o) \subseteq \mathcal{R}$, add effects $Add(o) \subseteq \mathcal{R}$ and delete effects $Del(o) \subseteq \mathcal{R}$.

DEFINITION 8. *Given a set of robots $R = \{r\}$, a set of human agents $\Phi = \{\phi\}$, and a set of typed objects O, a multi-agent planning problem with mixed models is given by a tuple $\Pi = \langle \Phi, R, b(\mathcal{I}), G, \rho \rangle$, where:*

- *Each $r \in R$ is associated with a set of actions $A(r)$ that are instantiated from $\mathcal{O}$ and O, which $r \in R$ can perform; each action may not always succeed when executed and hence is associated with a cost.*
- *Each $\phi \in \Phi$ is associated with a capability model $G_\phi = \langle V_\phi, E_\phi \rangle$, in which $V_\phi = X_\phi \cup \dot{X}_\phi$. $X_\phi \subseteq X$, in which X_ϕ represents the state variables of the world and agent ϕ and X represents the joint set of state variables of all agents.*

Note that the grounded predicates (which are variables) for robots that are instantiated from $\mathcal{R}$ and O also belong to X. Human and robotic agents interact through the shared variables.

LEMMA 2. *The MAP-MM problem is at least PSPACE-complete.*

PROOF. We only need to prove the result for one of the extreme cases: when there are only robotic agents. The problem then essentially becomes a multi-agent planning problem, which is more general than the classical planning problem, which is known to be PSPACE-complete. □

5.1 State Expansion for MAP-MM

First, we make the same assumption as we made in single agent planning, such that the monotonicity properties still hold. Given the current planning state $\hat{b}(\mathcal{S})$ (i.e., sampled from $b(\mathcal{S})$), for each $s \in \mathcal{S}$, a centralized planner can expand it in the next step using the following two options in MAP-MM:

- Choose a robot r and an action $a \in A(r)$ such that at least one of the complete states s in $\mathcal{S}$ satisfies $T(Pre(a)) \subseteq T(s)$.
- Choose a capability on human agent ϕ with the specification $s_I \Rightarrow s_E$, such that at least one of the states s in $\mathcal{S}$ satisfies $T(s_I) \subseteq T(s)$.

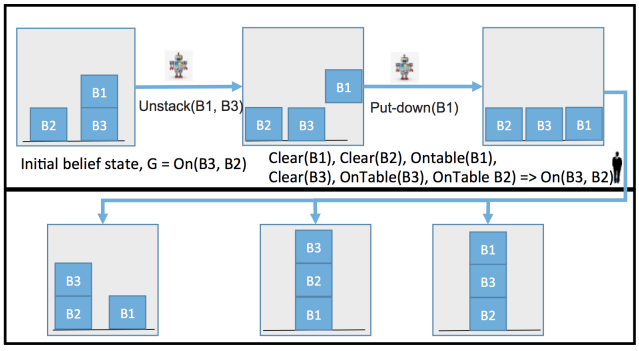

Figure 4: Illustration of solving a MAP-MM problem with a robotic and a human agent, in which the robotic agent has an action model and the human agent has a capability model that is assumed to be learned by the robotic agent.

If we have chosen an action a, for any $s \in \mathcal{S}$ that satisfies $T(Pre(a)) \subseteq T(s)$, the complete state s after applying a becomes s', such that $T(s') = (T(s) \cup Add(a)) \setminus Del(a)$, and $F(s') = (F(s) \cup Del(a)) \setminus Add(a)$. The probability weight of s' in the new belief state after applying a does not change. For states in $\mathcal{S}$ that do not satisfy $T(Pre(a)) \subseteq T(s)$, we assume that the application of a does not change anything. In this way, we can construct the new belief state after applying this action.

If we have chosen a capability in the form of $s_I \Rightarrow s_E$ from a human agent ϕ, for any $s \in \mathcal{S}$ that satisfies $T(s_I) \subseteq T(s)$, we can use follow discussion in Section 4.1 to compute the new belief state. Similarly, for states in $\mathcal{S}$ that do not satisfy $T(s_I) \subseteq T(s)$, we assume that the application of this capability does not change anything. With the new belief state, we can continue the state expansion process to expand the c-plan further.

5.2 Planning Heuristic for MAP-MM

In this section, we discuss a planning heuristic that informs us which state should be chosen to expand at any time. We can adapt the heuristic in Equation (11) to address MAP-MM. Given the current planning state $\hat{b}(\mathcal{S})$, we need to compute $h(\hat{b}(\mathcal{S}))$. For each $s \in \mathcal{S}$, there are three cases:

1) If only capabilities are used afterwards, $h(s)$ can be computed as in Equation (11), except that all capability models must be considered. 2) If only actions are going to be used, $h(s)$ can be computed based on the relaxed plan heuristic (i.e., ignoring all deleting effects), while considering all robot actions. 3) If both capabilities and actions can be used, $h(s)$ can be computed as the minimum cost of an action that achieves any variable in G_s. The final $h(s)$ is chosen as the smallest value among the three cases and $h(\hat{b}(\mathcal{S}))$ can subsequently be computed.

COROLLARY 1. *The heuristic above is admissible for finding a c-plan for MAP-MM that maxmizes ρ, given that $\hat{b}(\mathcal{S})$ accurately represents $b(\mathcal{S})$.*

5.3 Evaluation

In this section, we describe a simple application of MAP-MM involving a human and a robot, in which the capability model of the human is assumed to be learned by the robot. The robot then makes a multi-agent c-plan for both the human and itself.

The setup of this evaluation is identical to that in the evaluation of single agent planning, as we discussed in the previous section.

In this example, we associate the robot actions with a constant cost. After learning the parameters of the human capability model based on the generated partial plan traces, we apply the capability model to solve a problem as shown in Figure 4. Initially, we have $On(B1, B3)$, $OnTable(B3)$, and $OnTable(B2)$ (a complete state), and the goal is to achieve $On(B3, B2)$. The multi-agent c-plan involves the application of two robot actions and one capability of the human. We show three possible complete states (in the belief state) which satisfy the goal variable after applying the capability.

6. RELATED WORK

Most existing approaches for representing the dynamics of agents assume that the models are completely specified. This holds whether the underlying models are based on STRIPS actions (e.g. PDDL [7]) or stochastic action models (such as RDDL [19]). This assumption of complete knowledge is also the default in the existing multi-agent planning systems [3].

Capability models, in contrast, start with the default of incomplete models. They are thus related to the work on planning with incompletely specified actions (c.f. [15, 16, 11]). An important difference is that while this line of work models only incompleteness in the precondition/effect descriptions of the individual actions, capabilities are incomplete in that they completely abstract over actual plans that realize them. In this sense, a capability has some similarities to non-primitive tasks in HTN planning [6, 27]. For example, an abstract HTN plan with a single non-primitive task only posits that there exists some concrete realization of the non-primitive task which will achieve the goal supported by the non-primitive task. However, in practice, all HTN planners use "complete models" in that they provide all the reduction schemas to take the non-primitive task to its concretization. So, the "uncertainty" here is "angelic" [12] – the planner can resolve it by the choice of reduction schemas. In contrast, capability models do not have to (and cannot) provide any specifics about the exact plan with which a capability will be realized.

Capability models also have connections to macro operators [2], as well as options, their MDP counterparts [21, 20, 9], and the BDI models [17]. Capability models are useful when plans must be made with partial knowledge. With complete models, this means that not all actions or macro-actions or options or capabilities in BDI to achieve the goal are provided. None of HTN, SMDP or BDI models can handle the question of what it means to plan when faced with such model incompleteness. Capability models in contrast propose approximate plans (referred to as c-plans) as a useful solution concept in informing the user of how likely there is an actual complete plan.

On the other hand, due to the inherent incompleteness of capability models, they are lossy in the following sense. It is possible to compile a complete model to a capability model (e.g., converting actions in RDDL [19] to a capability model), but new capabilities may also be introduced along with the actions. As a result, the synthesized plans would still be incomplete unless the use of these new capabilities are forcibly restricted. The learning of capability models has connections to learning probabilistic relational models using Bayesian networks [8]. The notion of eventual state captured in capability models is similar to that captured by the F operator (i.e., eventually) in LTL and CTL [5, 23]. Although there are other works that discuss about capability models, e.g., [4], they are still based on action modeling.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced a new representation to model agents based on capabilities. The associated model, called a capability model, is an inherently incomplete model that has several unique advantages compared to traditional complete models (i.e., action models). The underlying structure of a capability model is a generalized 2-TBN, which can encode all the capabilities of an agent. The associated probabilities computed (i.e., via Bayesian inference on the capability model) based on the specifications of capabilities (i.e., a partial initial state coupled with a partial eventual state) determine how likely the capabilities can be fulfilled by an operation (i.e., a complete plan). This information can be used to synthesize incomplete "plans" (referred to as c-plans). One of the unique advantages of capability models is that learning for both model structure and parameters is robust to high degrees of incompleteness in plan execution traces (e.g., with only start and end states). Furthermore, we show that parameter learning for capability models can be performed efficiently online via Bayesian learning.

Additionally, we provide the details of using capability models in planning. Compared to traditional models for planning, in a planning state, capability models only suggest transitions between partial states (i.e., specified by the capabilities), along with the probabilities that specify how likely such transitions can be fulfilled by an operation. The limitation is that the synthesized c-plans are incomplete. However, we realize that this is unavoidable for planning with incomplete knowledge (i.e., incomplete models). In such cases, the synthesized c-plans can inform the user how likely complete plans exist when following the "guidelines" of the c-plans. In general, a c-plan with a higher probability of success should imply that a complete plan is more likely to exist. We discuss using capability models for single agent planning first, and then extend it to multi-agent planning (with each agent modeled separately by a capability model), in which the capability models of agents are used by a centralized planner. We also discuss how capability models can be mixed with complete models.

In future work, we plan to further investigate the relationships between capability models and traditional models. We also plan to explore applications of capability models in our ongoing work on human-robot teaming [22, 13]. For example, we plan to investigate how to enable robots to learn capability models of humans and plan to coordinate with the consideration of these models.

Acknowledgments

This research is supported in part by the ARO grant W911NF-13-1-0023, and the ONR grants N00014-13-1-0176, N00014-13-1-0519 and N00014-15-1-2027.

REFERENCES

[1] C. Backstrom and B. Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11:625–655, 1996.

[2] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24:581–621, 2005.

[3] R. I. Brafman and C. Domshlak. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *ICAPS*, pages 28–35. AAAI Press, 2008.

[4] J. Buehler and M. Pagnucco. A framework for task planning in heterogeneous multi robot systems based on robot capabilities. In *AAAI Conference on Artificial Intelligence*, 2014.

[5] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer Berlin Heidelberg, 1982.

[6] K. Erol, J. Hendler, and D. S. Nau. Htn planning: Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1123–1128. AAAI Press, 1994.

[7] M. Fox and D. Long. PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:2003, 2003.

[8] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *In IJCAI*, pages 1300–1309. Springer-Verlag, 1999.

[9] P. J. Gmytrasiewicz and P. Doshi. Interactive pomdps: Properties and preliminary results. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '04, pages 1374–1375, Washington, DC, USA, 2004. IEEE Computer Society.

[10] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285, May 1996.

[11] S. Kambhampati. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models, 2007.

[12] B. Marthi, S. J. Russell, and J. Wolfe. Angelic semantics for high-level actions. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS)*, 2007.

[13] V. Narayanan, Y. Zhang, N. Mendoza, and S. Kambhampati. Automated planning for peer-to-peer teaming and its evaluation in remote human-robot interaction. In *ACM/IEEE International Conference on Human Robot Interaction (HRI)*, 2015.

[14] R. E. Neapolitan. *Learning Bayesian networks*. Prentice Hall, 2004.

[15] T. Nguyen and S. Kambhampati. A heuristic approach to planning with incomplete strips action models. In *International Conference on Automated Planning and Scheduling*, 2014.

[16] T. A. Nguyen, S. Kambhampati, and M. Do. Synthesizing robust plans under incomplete domain models. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2472–2480, 2013.

[17] L. Padgham and P. Lambrix. Formalisations of capabilities for bdi-agents. *Autonomous Agents and Multi-Agent Systems*, 10(3):249–271, May 2005.

[18] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[19] S. Sanner. Relational dynamic influence diagram language (rddl): Language description, 2011.

[20] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for dec-pomdps. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2009–2015, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[21] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, Aug. 1999.

[22] K. Talamadupula, G. Briggs, T. Chakraborti, M. Scheutz, and S. Kambhampati. Coordination in human-robot teams using mental modeling and plan recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2957–2962, Sept 2014.

[23] M. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer Berlin Heidelberg, 1996.

[24] B. Y. White and J. R. Frederiksen. Causal model progressions as a foundation for intelligent learning environments. *Artificial Intelligence*, 42(1):99 – 157, 1990.

[25] C. Yuan and B. Malone. Learning optimal bayesian networks: A shortest path perspective. *J. Artif. Int. Res.*, 48(1):23–65, Oct. 2013.

[26] H. H. Zhuo and S. Kambhampati. Action-model acquisition from noisy plan traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 2444–2450. AAAI Press, 2013.

[27] H. H. Zhuo, H. Muñoz Avila, and Q. Yang. Learning hierarchical task network domains from partially observed plan traces. *Artificial Intelligence*, 212:134–157, July 2014.