HTN-Style Planning in Relational POMDPs Using First-Order FSCs

Felix Müller and Susanne Biundo

Institute of Artificial Intelligence, Ulm University, D-89069 Ulm, Germany

Abstract. In this paper, a novel approach to hierarchical planning under partial observability in relational domains is presented. It combines hierarchical task network planning with the finite state controller (FSC) policy representation for partially observable Markov decision processes. Based on a new first-order generalization of FSCs, action hierarchies are defined as in traditional hierarchical planning, so that planning corresponds to finding the best plan in a given decomposition hierarchy of predefined, partially abstract FSCs. Finally, we propose an algorithm for solving planning problems in this setting. Our approach offers a way of practically dealing with real-world partial observability planning problems: it avoids the complexity originating from the dynamic programming backup operation required in many present-day policy generation algorithms.

1 Introduction

Assisting human users in performing exceptional, complex tasks or even in managing their day-to-day life requires advanced planning technology. The reason is that user-centered planning is challenging in two respects. First, the real-world planning domains underlying these applications are large; they show hierarchical structures and typically there are numerous options to solve a given problem in a user-conform manner. Second, information about the user state is essential as the automatically generated plans should not just solve the problem, but should do so by taking also the user's emotional state into account. Information about the user state, just like information about the world state, depends on sensory input and is thus only partially observable and inherently uncertain.

In order to address partial observability, relational partially observable Markov decision processes (POMDPs) can be employed. They allow to handle complex domain dynamics and are thus appropriate to represent many real-world decision problems that arise in the context of user-centered planning. A POMDP-based system to provide assistance for persons with dementia is described by Boger *et al.* [2], for example. Solving POMDPs has been shown to be expensive, however. It ranges from PSPACE-complete for finite horizons [11] to undecidable for infinite horizons [8] even without factored representations. Therefore, applying POMDPs works well for small to medium sized domains, while scaling beyond this size is problematic.

The challenge to get along with complex domains is addressed by approaches that use (variants of) hierarchical task network (HTN) planning [4]. Among others, HTNs provide the means to encode standard solutions to planning problems and with that enable the exploitation of expert knowledge in solution discovery. Applications in many real-world domains rely on HTN-based systems [9, 1]. While making use of sophisticated representation schemes that allow to compactly describe large domains, HTN-based systems are aimed at planning for fully observable deterministic domains, however.

Planning in POMDPs with first-order abstraction has only recently started to be investigated by Sanner and Kersting [16] as well as Wang and Khardon [17]. Here, state, action, and observation spaces are factored into a first-order representation, as in traditional AI planning. There are approaches to (non-first-order) hierarchical planning in POMDPs, such as the one introduced by Hansen and Zhou [7]. Their approach uses POMDPs with abstract actions, but instead of a set of predefined finite state controllers per abstract action, a sub-POMDP is solved to compute a policy for each abstract action.

HTN planning in partially observable domains is introduced by Bouguerra and Karlsson [3], where HTN planning is extended by probabilistic action effects and observations, and uncertainty about the environment is represented using belief states. The approach does not allow for the specification of cyclic methods and does not cover full POMDP semantics with rewards, however.

In this paper, we present an approach to transfer the benefits of HTN planning to the POMDP setting, thereby allowing for the exploitation of domain expert knowledge to generate good policies in large, compactly represented POMDP domains. We begin by briefly reviewing (PO)MDPs and HTN planning in Section 2. We then enhance the FSC policy representation such that controllers for relational domains can be compactly specified (Section 3): transitions between controller nodes are governed by formulas, which represent conditions that need to be fulfilled by observations instead of annotating transitions with all possible observations directly. After that, Section 4 defines action hierarchies by introducing abstract actions, each of which is associated with one or more implementations that are given as predefined enhanced controllers, analogously to traditional HTN planning. In Section 5, we describe how solutions can be generated in our formalism before we conclude in Section 6.

2 Background

Our work builds on the POMDP model and HTN planning. We will give short introductions to both in the following.

2.1 MDPs and POMDPs

MDPs are a framework for planning in fully observable domains with probabilistic actions [14]. An MDP is a tuple (S, A, T, R), where S is the set of states, A is the set of actions, $T(s, a, s') = P(s'|s, a)$ is the transition function, giving the probability for ending up in state s' upon executing action a in state s, and $R(s, a)$ determines the immediate rewards incurred by executing a in s. Additionally, one often specifies a horizon h for the number of time steps after which the process terminates, or a discount factor, $0 \leq \gamma \leq 1$, for discounting rewards earned at later decision stages, or both. POMDPs generalize MDPs by making the world state only indirectly observable. A POMDP introduces a set of observations O and an observation function $Z(s, a, o) =$

$P(o|a, s)$, denoting the probability of receiving observation o when a is executed and the *resulting* state is s. Since the state cannot be directly observed, a belief state, i. e., a probability distribution over states, is usually maintained by the planning agent.

A solution for a POMDP is a policy π that maximizes the accumulated reward, or equivalently minimizes accumulated cost, for an initial belief state b_0. One possibility to represent a policy is a finite state controller, i. e., a directed graph where nodes are labeled with actions and edges are labeled with observations [5]. Policies can be generated using dynamic programming approaches such as policy iteration, where a dynamic programming backup operation is used to iteratively improve an explicitly or implicitly represented policy. Other algorithms include approximative approaches, e. g., point-based value iteration as described by Pineau *et al.* [13].

In this paper, we will use a POMDP variant introduced by Hansen, called indefinite-horizon POMDPs with action-based termination [6]. Such POMDPs include one or more terminal actions, which cause immediate transition to a terminal state from any state. Also, no discounting is used (so that $\gamma = 1$) and action rewards are restricted to be negative. As a result, there is no bound on the number of steps before termination, but the optimal policy (and all policies with finite value) eventually terminate with probability 1. For the sake of readability, we will sometimes speak in terms of positive costs. We will restrict the description of our approach below to a single terminal action, though it can be generalized to more than one terminal action.

To overcome the explicit state enumeration used in the definition above, Sanner and Kersting [16] use a first-order representation based on the situation calculus. We will employ a similarly powerful representation, namely the probabilistic planning domain definition language (PPDDL) [18], which is based on a probabilistic extension of the action description language ADL [12] with MDP semantics. In PPDDL, a domain is given in terms of predicates and typed objects, and states are sets of ground predicates. A ground predicate is true in a state if it is contained in the state and false otherwise (closed world assumption).

A PPDDL action as shown in Figure 1 transforms a state into another state by adding or removing predicates when it makes them true or false, respectively. It also defines the conditions and probabilities that govern their addition and removal, alongside with the condition for receiving rewards. Actions are parametrized, which is why we distinguish between *action schemas*, which are the members of A, and *action instances* $I(A)$, where (some) action parameters are bound to domain objects, i. e., ground. An action schema is instantiated into an action instance by substituting parameters by objects of appropriate type. To allow for partial observability, we augment PPDDL actions with the possibility to include observations by introducing the field `:observation`. Observations are also sets of ground predicates, similar to states. The conditions and probabilities of receiving an observation are specified in the same manner as for an action's effects. As usual for POMDP observations, the conditions for receiving observations are evaluated after the action effects are applied.

As an example from the context of companion systems, we will use the task of renovating an apartment as part of the larger domain of moving: the task is to generate a plan that guides a user in painting each room of an apartment in their preferred color. At the same time, the plan should account for the user's preferences and emotional

```
(:action paint_room
 :parameters (?r - Room ?c - Color)
 :effect
  (and
   (has_color ?r ?c)
   (forall (?c_old - Color) (when (not (?c_old = ?c)) (not (has_color ?r ?c_old))))
   (when (happy) (decrease (reward) 1))
   (when (not (happy)) (decrease (reward) 10))
   (when (happy) (probabilistic 0.3 (not (happy)))))
 :observation
  (and
   (when (happy) (probabilistic 0.9 (happy_O) 0.1 (not (happy_O))))
   (when (not (happy)) (probabilistic 0.8 (not (happy_O)) 0.2 (happy_O)))))
```

Fig. 1: The `paint_room` action of our example domain, formulated in augmented PPDDL, which changes the color of a room and has low cost if the user is happy and high cost if they are not. Painting a room also has a chance to make the user unhappy and gives a noisy observation of the user's mood.

state and choose actions accordingly. More precisely, states are defined by the predicates `(user_wants_color ?r - Room ?c - Color)`, `(has_color ?r - Room ?c - Color)`, and `(happy)`, which represent the desired and current colors of each room and the user's emotional state, respectively. Note that for the sake of presentational simplicity, the user's emotional state can only be happy or unhappy. A more complex emotional model, such as the three-dimensional pleasure arousal dominance (PAD) model [15], could be modeled easily by introducing predicates `(pleasure ?l - Level)`, `(arousal ?l - Level)`, and `(dominance ?l - Level)`. Both the user's emotional state and the color the user wants to paint a room can only be observed indirectly via the separate observation predicates `(user_wants_color_O ?r - Room ?c - Color)` and `(happy_O)`.

There are five available actions. The first is `(ask_user ?r - Room)` which asks the user to tell the system the preferred color for a room. It generates a perfect observation of the user's desired color for a room at a low cost. The second action is the "easy" way to paint a room: call a painter, i. e., `(call_painter ?r - Room ?c - Color)`. This action changes the color of the room accordingly at a medium cost. The other possibility to paint a room is to let the user paint the room themselves using `(paint_room ?r - Room ?c - Color)` (the action represented by Figure 1), which has a low cost if the user is happy and high cost if they are not. Painting a room by themselves also has a chance to make the user unhappy. To improve the user's mood, it is possible to cheer up the user with the action `(cheer_up)`, which also has a low cost and a good chance to make them happy. Both the `(paint_room)` and `(cheer_up)` actions generate a noisy observation of the user's mood. Finally, there is the terminal action `(finish)`, which ends the process and has a high cost if there exists a room for which the actual color differs from the color the user wants. As a result, there is some choice as to how the goal of having the rooms of the apartment painted according to the user's preferences can be fulfilled using the actions in the domain: when the user is unhappy, there is a trade-off between cheering them up and letting them paint by themselves versus calling a painter.

2.2 HTN Planning

HTN planning [4] is an approach to planning in fully observable deterministic domains. World states are described in a similar manner as shown above, yet actions are deterministic and the (single) initial world state is completely known. Thus, no observations are required, because the evolution of the world can be predicted with certainty. Apart from the domain dynamics, the HTN planning problem definition includes a hierarchy of actions: in addition to normal, directly executable actions, there are also abstract actions. Abstract actions cannot be executed directly. Instead, one or more implementations, called decomposition methods, are provided for each abstract action in form of partial plans: a method is a partially ordered set of partially instantiated primitive or abstract actions, called a task network. Methods are specified by domain experts and represent known possible solutions to subproblems in the form of abstract actions, therefore encoding a domain expert's knowledge about typical problem-solving "recipes". Some HTN planners such as the original SHOP [10] use a more simple kind of method, where the tasks are totally ordered into a sequence of actions.

In contrast to reward-based goal models or goal state specifications, the goal in HTN planning is to *perform* one or more abstract actions specified in the initial task network. At planning time, abstract actions are iteratively eliminated by replacing abstract actions with implementing methods until a plan is found that contains only primitive actions. Because the goal is to perform actions instead of generating a particular state, actions are also often called (primitive or abstract, respectively) tasks in HTN planning. In the course of our paper, we will use both interchangeably.

3 First-Order FSCs

For relational POMDPs such as our example from the previous section, policies can be represented using a first-order version of finite state controllers. The definition of first-order FSCs (FOFSCs) is the first contribution of this paper and is required for the introduction of action hierarchies in Section 4. The rationale behind introducing FOFSCs is that because the number of possible observations is very large in relational POMDPs, including an edge label for every observation as in ordinary FSCs is inconvenient. FOFSCs allow for a more concise representation by using formulas to govern transitions. Note that our definition already includes terminal controller states and nodes associated with partially instantiated actions to allow for the addition of hierarchical elements described below.

Definition 1 (First-Order Finite State Controller) *A first-order finite state controller (FOFSC) is a tuple $(Q, q_0, q_t, A, \alpha, O, \delta)$. A and O are sets of actions and observations, respectively. Q is the set of controller nodes. The node $q_0 \in Q$ is the start node, and the node $q_t \in Q$, $q_t \neq q_0$, is the terminal node. Each node $q \in Q \setminus \{q_t\}$ is associated with a (partially) instantiated non-terminal action $a_i \in I(A)$ via the action association function $\alpha : Q \setminus \{q_t\} \to I(A)$. Transitions are defined via the transition function $\delta : Q \setminus \{q_t\} \times Q \to \mathcal{F}(O)$, where $\mathcal{F}(O)$ is the set of well-formed formulas over the observation space. Transitions need to be mutually exclusive and exhaustive, i. e., for all obtainable observations o of $\alpha(p)$, (1) whenever $o \models \delta(p, q_i)$ it holds that $o \not\models \delta(p, q_j)$ for all $q_j \neq q_i$ and (2) $o \models \bigvee_{q_i \in Q} \delta(p, q_i)$.*

For nodes p and q and observation o received from executing $\alpha(p)$, $\delta(p, q) = \varphi$ means that the controller state changes from p to q if $o \models \varphi$. Making the formulas mutually exclusive guarantees that there is no ambiguity when choosing the successor state. Exhaustiveness guarantees that every possible observation of an action is covered. However, this requirement can be dropped by assuming transition to an "error node" (associated with an action that incurs strongly negative reward) when no condition is fulfilled.

Executing an FOFSC works by iteratively executing the action associated with a node (starting with q_0), receiving an observation, and following the link whose formula is fulfilled by the observation. This is done until the terminal node is reached, at which point the terminal action is executed. Figure 2 shows an example FOFSC. For the sake of clarity, we omit edges between two nodes p and q when $\delta(p, q) \equiv \bot$. Action parameters do not need to be ground according to the FOFSC definition, allowing for multiple usage of the same variable in several nodes of the FOFSC. However, non-ground parameters introduce a source of nondeterminism into the FOFSC that might prohibit the unique assignment of an accumulated reward value to the policy the FOFSC represents. Therefore, we define that an FOFSC is *executable* if and only if it is fully ground. For an executable FOFSC, the accumulated reward earned by executing it in a given initial belief state can for example be calculated by solving the induced system of linear equations as described by Hansen [5], where the accumulated reward of the terminal state q_t is defined by the cost of the terminal action. When the planning horizon is finite, cycles in the FOFSC can be eliminated for evaluation purposes: since execution is assumed to stop when either the terminal node or the horizon is reached, at the latest, they can be unrolled. This allows for a fast forward-evaluation of FOFSCs.

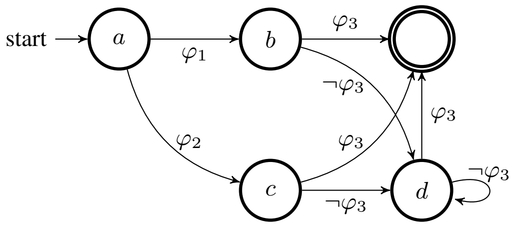


Fig. 2: An example FOFSC, where a = `(ask_user ?r)`, b = `(paint_room ?r WHITE)`, c = `(paint_room ?r LIGHTBLUE)`, d = `(cheer_up)`, φ_1 = `(user_wants_color_O ?r WHITE)`, φ_2 = `(user_wants_color_O ?r LIGHTBLUE)`, and φ_3 = `(happy_O)`. In words, the described course of action proposes to ask the user for the desired room color, paint the room accordingly, and cheer up the user until their mood seems better.

Note that FOFSCs are a natural generalization of the sequences of ground actions generated by many planners for fully observable and deterministic planning domains, such as SHOP.

4 Hierarchical Relational POMDPs

Now, we can begin adding hierarchical information to create action hierarchies by introducing abstract actions and the means to carry them out. Abstract actions are specified in terms of a name and a parameter list just like ordinary actions of the underlying POMDP, but as they are not part of the underlying POMDP, no transition probabilities, observation probabilities or rewards are associated with them. However, as only name and parameter list of an action are needed for the construction of FOFSCs, it is possible to construct FOFSCs containing abstract actions. Obviously, such abstract actions cannot be executed directly and neither can FOFSCs that contain abstract actions.

Intuitively, hierarchical planning as we will define it in this section starts with an initial FOFSC containing abstract actions that have to be iteratively refined into more concrete implementations. This is done by applying *decomposition methods*, each mapping an abstract action to a possible implementation, until all actions are primitive as is done in HTN planning. However, in contrast to normal HTN planning, where the goal is to find an executable primitive plan, the goal here is to find the optimal-reward primitive FOFSC in the induced hierarchy according to evaluation with an initial belief state and a planning horizon.

For a formal definition of hierarchical relational POMDPs using FOFSCs with abstract actions, let $\mathrm{FSCs}(X)$ denote the set of FSCs over a set of actions X.

Definition 2 (Hierarchical Relational POMDP) *A* hierarchical relational POMDP *(HPOMDP) is a tuple* $(P, A_a, M, C_{\mathrm{init}}, b_0, h)$*, where*

- $P = (S, A, T, R, O, Z)$ *is the underlying relational indefinite-horizon POMDP with action-based termination,*
- A_a *is a finite set of* abstract actions *with* $A_a \cap A = \emptyset$*,*
- $M \subseteq A_a \times \mathrm{FSCs}(A \cup A_a)$ *is a finite set of* decomposition methods *mapping abstract actions to their possible implementations,*
- $C_{\mathrm{init}} \in \mathrm{FSCs}(A \cup A_a)$ *is the initial FOFSC containing abstract actions,*
- b_0 *is the initial belief state, and*
- $h \in \mathbb{N} \cup \{\infty\}$ *is the planning horizon.*

A possible abstract action for the example POMDP introduced in the previous section is `(handle_room ?r)`, which completely handles one room. Expert knowledge is then used to define useful courses of action as implementation for abstract actions. Consequently, one possible method is to ask the user for the desired room color, painting the room accordingly and cheer them up afterwards, if needed. Hence, the FOFSC given in Figure 2 is a method for `(handle_room ?r)`. Other implementations include asking for the desired color and calling a painter, and so on.

Applying a method transforms a partially abstract FOFSC into a more concrete version by replacing the abstract action with the implementation specified by the method. For a formal definition of method application, let $\mathrm{pd}_{\mathrm{C}}(q)$ denote the set of predecessors of a node q in an FSC C with node set Q and transition function δ, i. e., $\mathrm{pd}_{\mathrm{C}}(q) = \{p \in Q | \delta(p, q) \not\equiv \bot\}$ and $\mathrm{sc}_{\mathrm{C}}(q)$ the set of successors of q, i. e., $\mathrm{sc}_{\mathrm{C}}(q) = \{p \in Q | \delta(q, p) \not\equiv \bot\}$. We use superscripts to disambiguate the members of the different FSCs involved, e. g., we write Q^1 for the node set of the FSC C^1.

Definition 3 (Method Application) *Let C^1 be an FOFSC containing a node q_a^1 with $a_i = \alpha^1(q_a^1)$ being an abstract action instance, $m = (a, C)$ a method for the corresponding action schema a of a_i, and C^2 an isomorphic copy of C with $Q^1 \cap Q^2 = \emptyset$. Let σ be a parameter substitution such that $\sigma(a) = a_i$. Applying m to q_a^1 results in a new FOFSC $C^3 = (Q^3, q_0^1, q_t^1, A^1, \alpha^3, O^1, \delta^3)$ with:*

– $Q^3 := (Q^1 \cup Q^2) \setminus \{q_a^1, q_t^2\})$

– $\alpha^3(q) := \begin{cases} \alpha^1(q), & q \in Q^1 \\ \sigma(\alpha^2(q)), & q \in Q^2 \end{cases}$

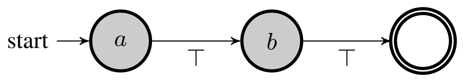
– $\delta^3(p,q) := \begin{cases} \delta^1(p,q), & p,q \in Q^1 \\ \sigma(\delta^2(p,q)), & p,q \in Q^2 \wedge \neg(p \in \mathrm{pd}_{C^2}(q_t^2) \wedge q = q_0^2) \\ \sigma(\delta^2(p,q)) \vee \delta^1(q_a^1, q_a^1), & p \in \mathrm{pd}_{C^2}(q_t^2) \wedge q = q_0^2 \\ \delta^1(p, q_a^1), & p \in \mathrm{pd}_{C^1}(q_a^1) \wedge q = q_0^2 \\ \sigma(\delta^2(p, q_t^2)) \wedge \delta^1(q_a^1, q), & p \in \mathrm{pd}_{C^2}(q_t^2) \wedge q \in \mathrm{sc}_{C^1}(q_a^1) \\ \perp, & \textit{else} \end{cases}$

Applying a method m to a node q in an FOFSC C is denoted by $C[q \leftarrow m]$. Intuitively, when an abstract action is replaced by an implementation, all incoming edges of the abstract action are "redrawn" to point to the initial node of the implementing FOFSC, and all outgoing edges of the abstract action are combined with the edges pointing to the terminal node of the implementing FOFSC. Additionally, the substitution is required to pass the already bound action parameters of the replaced abstract action to its implementation. Figure 3 shows how the abstract action `(handle_room KITCHEN)` is replaced by one of the introduced methods in our example.

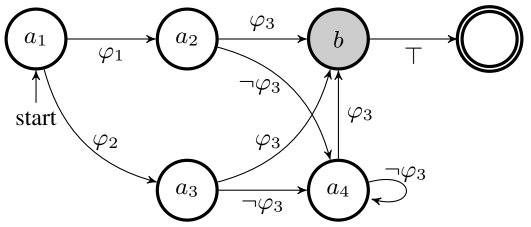
Precisely, the new transition function δ^3 is constructed to capture the intended meaning of method application while fulfilling the "mutually exclusive and exhaustive" requirement of Definition 1: the first two cases preserve the internal transitions of C^1 and C^2. The third case accounts for the special case when there is a sling at q_a^1: a sling means the abstract action can be repeated immediately after its completion. This translates to a transition from a last action to the first action in its implementation. As the implementation may already contain such a transition, a disjunction of both transition conditions is introduced. The fourth case handles transitions to q_a^1, which result in transitions to the initial node of C^2. The fifth case connects the predecessor nodes of q_t^2 to the successor nodes of q_a^1. As both the transitions to q_t^2 and from q_a^1 are labeled with transition conditions in C^1 and C^2, respectively, each possible combination in $\mathrm{pd}(q_t^2) \times \mathrm{sc}(q_a^1)$ is labeled with the conjunction of the respective transition conditions. Finally, the last case states that no other transitions are introduced.

By iteratively replacing all abstract actions and binding action parameters to constants until all actions are primitive and ground, we can generate executable FSCs. For a primitive FSC C, we can evaluate its accumulated cost $V(C, b_0, h)$ with respect to the initial belief state b_0 and horizon h. This leads us to the definition of a solution for a hierarchical relational POMDP:

Definition 4 (Solution) *A solution of a hierarchical relational POMDP is a fully primitive and executable decomposition C^* (i. e., fully ground and without abstract actions)*



(a) A partially abstract FOFSC in our example, where the kitchen and the living room are to be painted, so a = `(handle_room KITCHEN)` and b = `(handle_room LIVING_ROOM)`.



(b) The resulting FOFSC after applying a method (a_1 = `(ask_user KITCHEN)`, a_2 = `(paint_room KITCHEN WHITE)`, a_3 = `(paint_room KITCHEN LIGHTBLUE)`, a_4 = `(cheer_up)`, φ_1 = `(user_wants_color_O KITCHEN WHITE)`, φ_2 = `(user_wants_color_O KITCHEN LIGHTBLUE)`, and φ_3 = `(happy_O)`).

Fig. 3: The initial FOFSC C_{init} for our example domain (Figure (a)) and the resulting FOFSC after the method shown in Figure 2 has been applied to the node associated with action a (shown in Figure (b)). Note how the parameter `?r` is substituted by `KITCHEN`.

of C_{init}, such that there exists no other fully primitive and executable decomposition C with smaller accumulated cost, i. e., $\forall C, V(C, b_0, h) \geq V(C^*, b_0, h)$, with respect to evaluation using b_0 and h.

Note that a solution is required to be a decomposition of C_{init}. This means that it is in general not an optimal policy for the underlying POMDP. The relative solution quality of the HPOMDP solution compared to the optimal solution of the underlying POMDP directly depends on the modeled methods: methods can both be chosen so that the HPOMDP solution approximates the optimal solution arbitrarily closely and so that the HPOMDP solution is arbitrarily bad. Since the methods are modeled by a domain expert, relative HPOMDP solution quality is determined by the quality of the expert knowledge.

5 Algorithm

The algorithm we propose for hierarchical relational POMDP planning is outlined in Algorithm 1 and employs an A* search in the space of FOFSCs induced by the action hierarchy: it iteratively decides whether an FOFSC is a solution (line 4), generates successor FOFSCs if it is no solution (line 6-9), ranks them according to their cost-plus-heuristic value (line 10), and proceeds by considering the lowest-cost candidate next. A closed-set D that contains already visited FOFSCs is used to prevent the algorithm from considering the same FOFSC twice (lines 5 and 10). Note that the method application in line 8 generates only ground instances, so that the cost and heuristic functions defined below can be evaluated unambiguously.

```
Input  : A HPOMDP $(P, A_a, M, b_0, C_{\mathrm{init}}, h)$.
Output : A solution or fail.
1  $F := \langle C_{\mathrm{init}} \rangle$, $D := \emptyset$
2  while $F \neq \varepsilon$ do
3      $C_{\mathrm{cur}} := \mathrm{head}(F)$, $F := \mathrm{tail}(F)$
4      if $C_{\mathrm{cur}}$ is executable then return $C_{\mathrm{cur}}$
5      $D := D \cup C_{\mathrm{cur}}$
6      $Q :=$ the nodes $q$ in $C_{\mathrm{cur}}$ with $\alpha(q)$ abstract
7      foreach $q \in Q$ do
8          $\langle m_1, \cdots, m_n \rangle :=$ the methods for $\alpha(q)$
9          $F_q := \langle C_{\mathrm{cur}}[q \leftarrow m_1], \cdots, C_{\mathrm{cur}}[q \leftarrow m_n] \rangle$
10         $F := \mathrm{merge}(F, F_q \setminus D)$
11 return fail
```

Algorithm 1: The HPOMDP Planning Algorithm, which employs A* search in the space of FOFSCs. The call to merge() sorts all plans C in F and F_q in ascending order according to their cost-plus-heuristic value $f(C) = g(C) + h(C)$.

5.1 Costs and Heuristics

For the cost and heuristic definitions, we distinguish between the part of an FSC that can be assessed exactly cost-wise, and the part for which cost must be estimated. Cost can be calculated exactly when there is no abstract action involved, i. e., for the part between q_0 and the first nodes associated with an abstract action. For an abstract action, neither immediate cost nor transition probabilities are known, therefore it is impossible to calculate costs correctly beyond the first abstract action encountered. Let $Q_f(C)$ denote the set of those "first abstract nodes", i. e., nodes q for which $\alpha(q)$ is abstract and there is a fully primitive path with probability greater than zero leading from q_0 to q. The cost function $g(C)$ of an FSC C is then defined as the accumulated cost V of an FSC C' where every edge pointing to a node in $Q_f(C)$ is redrawn to the terminal node. Note that for a fully primitive FSC C, $C = C'$ and hence, $g(C) = V(C)$, i. e., $g(C)$ is the true cost of C.

The heuristic estimate $h(C)$ tries to assess the cost induced by the remaining part of C. We will present admissible estimates h for both the finite and the indefinite horizon cases. For the indefinite horizon case, we will calculate a lower bound for the cost incurred by reaching q_t from each node $q \in Q_f(C)$. The value of $h(C)$ is then calculated as the weighted sum of the cost bounds for each q according to the probabilities of reaching q through a fully primitive path, which are in turn lower bounds for the true probabilities of reaching each q. To calculate the estimate for each $q \in Q_f(C)$, we assume that we can choose transitions at each node in C instead of determining the successor node according to received observations. Additionally, we assign to each edge the minimal cost of the action of its predecessor node if it is primitive and 0 if it is abstract. The cost bound for a node $q \in Q_f(C)$ is then given by the cost of the shortest path from q to q_t in the resulting graph.

In the finite horizon case, it is possible that an expensive action "close" to the end of the planning horizon is "pushed beyond" the horizon when an abstract action is replaced by an implementation and is replaced with a cheaper action for evaluation purposes. Therefore, for each time step remaining after reaching a node $q \in Q_f(C)$, we simply allocate the minimum cost of *any* action and 0 when the terminal node is reached. This way, the heuristic is guaranteed to underestimate the remaining costs. Note that FOFSCs are unrolled for evaluation and comparison for finite horizons.

5.2 Properties

The algorithm is correct for both the finite and indefinite horizon cases, because the requirements for the usage of A* search are met. Firstly, since all action costs are non-negative, applying a method never decreases the cost g of an FOFSC. Therefore, there are no negative path costs as required for A* search. Secondly, the heuristics are constructed to underestimate the real costs. Thirdly, because both the cost and the heuristic estimate only depend on the FOFSC at hand, not on the path that generated it, the usage of a closed-set is also allowed.

In the finite horizon case, it is also complete and guaranteed to terminate. This is because there are only finitely many unrolled FOFSCs whose length is limited by the horizon h. Since every possible FOFSC in the hierarchy is generated and no FOFSC is considered twice through the use of the closed-set D, the execution terminates.

Because of the infinite plan space in the indefinite horizon case, completeness cannot be guaranteed in this setting. For some special cases though, the algorithm is also complete and guaranteed to terminate, for example when the action hierarchy is non-recursive, i. e., an abstract action is never reintroduced after it has been decomposed.

6 Conclusion and Future Work

We have presented an approach that allows for exploitation of domain knowledge in relational POMDP planning. While the FOFSCs generated by our approach do not represent optimal policies for the underlying POMDP, good policies can be generated through the use of suitable expert knowledge in the form of methods. If an optimal policy for the underlying POMDP is required, the FOFSC generated by our approach can be used as an input for traditional, optimal POMDP solution algorithms such as value or policy iteration.

Our approach covers both finite and indefinite horizons. Especially in the context of user-centered planning, finite horizons seem appropriate to represent the limited patience of a human user. As a prerequisite for our approach, we introduced a first-order generalization of FSCs that also corresponds to a generalization of plans generated by traditional AI planning systems for fully observable deterministic domains.

As the next step, we plan to implement our formalism and evaluate its effectiveness in terms of plan generation speed and ease of modeling, using domains from the 2011 International Probabilistic Planning Competition IPPC-2011. We expect that expert knowledge required for our approach is available in domains like elevator control or traffic light control, where present-day solutions are hand-crafted by experts. Another important part in our research is the development of suitable heuristics to improve plan generation speed.

Acknowledgements

This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

1. S. Biundo, P. Bercher, T. Geier, F. Müller, and B. Schattenberg. Advanced user assistance based on AI planning. *Cognitive Systems Research, Special Issue on “Complex Cognition”*, pages 219–236, 2011.
2. J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *IJCAI*, pages 1293–1299, 2005.
3. A. Bouguerra and L. Karlsson. Hierarchical task planning under uncertainty. In *3rd Italian Workshop on Planning and Scheduling*, 2004.
4. K. Erol, J. Hendler, and D. S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, pages 249–254, 1994.
5. E. A. Hansen. Solving POMDPs by searching in policy space. In *UAI*, pages 211–219, 1998.
6. E. A. Hansen. Indefinite-horizon POMDPs with action-based termination. In *AAAI*, pages 1237–1242, 2007.
7. E. A. Hansen and R. Zhou. Synthesis of hierarchical finite-state controllers for POMDPs. In *ICAPS*, pages 113–122, 2003.
8. O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5 – 34, 2003.
9. D. Nau, T.-Ch. Au, O. Ilghami, U. Kuter, H. Muñoz-Avila, J. W. Murdock, D. Wu, and F. Yaman. Applications of SHOP and SHOP2. In *IEEE Intelligent Systems*, 2004.
10. D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: simple hierarchical ordered planner. In *IJCAI*, pages 968–973, 1999.
11. Ch. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.
12. E. P. D. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. In *KR*, pages 324–332, 1989.
13. Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *JAIR*, 27:335–380, 2006.
14. M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.
15. J. A. Russell and A. Mehrabian. Evidence for a three-factor theory of emotions. *Journal of Research in Personality*, 11(3):273 – 294, 1977.
16. S. Sanner and K. Kersting. Symbolic dynamic programming for first-order POMDPs. In *AAAI*, pages 1140–1146, 2010.
17. Ch. Wang and R. Khardon. Relational partially observable MDPs. In *AAAI*, pages 1153–1158, 2010.
18. H. L. S. Younes and M. L. Littman. PPDDL 1.0: an extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University, Pittsburgh, 2004.