# Addressing Uncertainty in Hierarchical User-Centered Planning

Felix Richter and Susanne Biundo

**Abstract** Companion-Systems need to reason about dynamic properties of their users, e.g., their emotional state, and the current state of the environment. The values of these properties are often not directly accessible, hence information on them must be pieced together from indirect, noisy or partial observations. To ensure probability-based treatment of partial observability on the planning level, planning problems can be modeled as Partially Observable Markov Decision Processes (POMDPs).

While POMDPs can model relevant planning problems, it is algorithmically difficult to solve them. A starting point for mitigating this is that many domains exhibit hierarchical structures where plans consist of a number of higher-level activities, each of which can be implemented in different ways that are known a priori. We show how to make use of such structures in POMDPs using the Partially Observable HTN (POHTN) planning approach by developing a Partially Observable HTN (POHTN) action hierarchy for an example domain derived from an existing deterministic demonstration domain.

We then apply Monte-Carlo Tree Search to POHTNs for generating plans and evaluate both the developed domain and the POHTN approach empirically.

## 1 Introduction

*Companion*-Systems offering decision making capabilities need to reason about dynamic properties of their environment. Most notably, this environment consists of the user of the system and its physical surroundings. Aspects of interest include the user's emotional state as well as attributes of relevant physical objects, both of which and may be difficult or costly to measure. It is however often possible to gain some

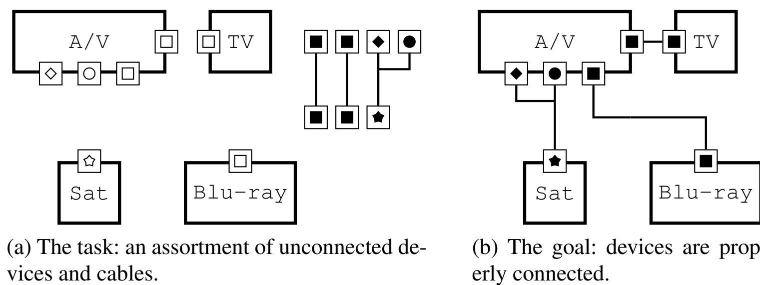
Felix Richter and Susanne Biundo
Institute of Artificial Intelligence, Ulm University e-mail: forename.surname@uni-ulm.de

knowledge about the aspects of interest by observing related, more easily accessible properties, even if these observations offer only noisy or partial information.

A natural planning model that accounts for partial observability within a probability-based framework is given by Partially Observable Markov Decision Processes (POMDPs) [19]. A drawback of POMDPs, however, is that it is difficult to compute policies that prescribe good courses of action. This is especially true for large problems, which often arise when adequately modeling a task at hand requires factoring in many different user and world properties.

At the same time, many problems humans are confronted with typically consist of a number of higher-level activities. These activities can be hierarchically divided into smaller activities and eventually simple actions. Often, there is also a limited number of useful possibilities how a particular activity can be performed. This hierarchical structure opens opportunities for *Companion*-Systems to mitigate the computational burden of computing policies in partially observable environments.

For deterministic planning domains, Hierarchical Task Network (HTN) planning [6, 5] is a practical planning approach that allows modeling and exploiting such hierarchical structures. It allows for efficient plan generation and makes it easy to express expert knowledge about a planning domain. As such, it has been successfully applied to many real-world problems [14]. Typical such hierarchically structured real-world problems that can be tackled with HTN planning are, e.g., given by instances of the home theater setup domain developed for demonstration purposes in the context of intelligent assistance systems [1, 2, 9] and described in Chapter 24. The task in this domain is depicted in Fig. 1a: given an assortment of devices and cables, connect the devices such that in the end, the user has a working home theater, as depicted in Fig. 1b.




(a) The task: an assortment of unconnected devices and cables.

(b) The goal: devices are properly connected.

Fig. 1: Schematic representation of the home theater setup domain. The A/V receiver, TV, satellite receiver, and Blu-ray player each have various female ports, where □, ☆, ◇, and ○ denote HDMI, SCART, cinch video and cinch audio ports, respectively. Black ports on cables ■ denote male ports. There are two HDMI cables and one SCART-to-cinch-AV cable.

A number of approaches exist that make use of hierarchical structure in planning domains that exhibit uncertainty of some kind, in either fully observable MDP or partially observable POMDP settings. Some approaches augment the original set of actions with macro-actions in the spirit of the options framework [20]: approaches in this category either have a one-to-one correspondence between macro-actions and their implementation [21] or try to generate restricted implementations at planning time [8]. Other hierarchical POMDP approaches such as PolCA+ [16] or MAXQ-hierarchical policy iteration [7] define a fixed hierarchical decomposition of a task a priori, similar to the MAXQ decomposition [4] or HAM [15] in the fully observable MDP setting, and individually optimize sub-policies for each abstract action.

An alternative to the above approaches is to directly extend HTN planning to POMDPs, which results in the Partially Observable HTN (POHTN) approach that we describe in our earlier work [12, 13]. In this chapter, we develop a variant of the domain sketched in Fig. 1 that relaxes the full observability assumption and create a suitable POHTN hierarchy. To demonstrate the effectiveness of the POHTN approach, we evaluate our approach empirically on several instances of the home theater setup domain.

The chapter is structured as follows. We first explain POMDP concepts and how the home theater domain is modeled using the Relational Influence Diagram Language (RDDL) [17] in Sect. 2. Next, Sect. 3 reviews an existing popular non-hierarchical approach to POMDP planning, namely Monte-Carlo Tree Search (MCTS) on the basis of observable histories. Section 4 presents the POHTN approach and shows how a POHTN hierarchy can be constructed for the home theater setup domain. Section 4.4 describes the application of MCTS to POHTN planning. Our experiments in Sect. 5 compare MCTS-based POHTN planning and history-based MCTS planning and also show how modeling choices lead to domain variants with different computational difficulty and practical implications. Section 6 concludes with some final remarks.

## 2 The Home Theater Setup POMDP

A POMDP is an 8-tuple $(S, A, O, T, Z, R, b_0, H)$, where $S$, $A$, and $O$ are finite sets of states, actions, and observations, respectively. The effects of executing actions on the system's environment are defined by the transition function $T$, in the sense that for a given state $s \in S$ and action $a \in A$, $T(s,a)$ defines a probability distribution over possible successor world states $s' \in S$. Similarly, the system's sensor model is determined by the observation function $Z$ such that for a given action $a \in A$ and successor state $s' \in S$, $Z(a,s')$ defines a probability distribution over possible observations $o \in O$. The system alternatingly executes actions and receives observations as depicted in Fig. 2. Note that the successor world state $s'$ is not visible to the system, it must infer information on the identity of $s'$ from the observation $o$.

What the system can see about its environment is the actions it executes and the observations it receives. After $t$ steps, the system's entire knowledge about the

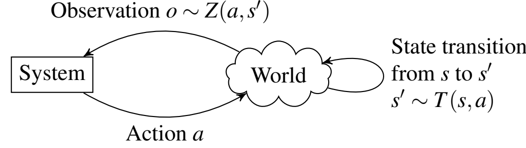



Fig. 2: The POMDP interaction cycle

evolution of its environment is thus the sequence $a^1 o^1 \dots a^t o^t$, called an *observable history*. Because of that, the system's policy can be represented as a function that maps observable histories to actions.

The system's goals are given in terms of a real-valued reward function $R(s,a,s')$ that determines how beneficial it is for the system when the result of executing $a \in A$ in $s \in S$ is $s' \in S$. The system's success is determined by the expected accumulated reward it is able to gather in a given number of time steps $H$, called the horizon, starting in a state that is sampled from the probability distribution $b_0$, the initial belief state. Formally, this can be captured as follows: for a given history $h = a^1 o^1 \dots a^t o^t$, let *hao* denote the history extended by $a$ and $o$, i.e., $a^1 o^1 \dots a^t o^t ao$, and let $\pi$ denote the system's policy. Then the quality of $\pi$ in a given state $s$ is given by

$$V_\pi^h(s) = \begin{cases} \mathbb{E}_{s' \sim T(s,\pi(h))}[R(s,\pi(h),s')] & \text{if } h \text{ has } H \text{ steps} \\ \mathbb{E}_{s' \sim T(s,\pi(h))}[R(s,\pi(h),s') + \mathbb{E}_{o \sim Z(s',\pi(h))}[V_\pi^{h\pi(h)o}(s')]] & \text{else,} \end{cases} \tag{1}$$

where $\mathbb{E}_{x \sim X}[f(x)]$ denotes the expectation of $f(x)$ when $x$ is distributed according to $X$. The quality of $\pi$ in the initial belief state is $V_\pi^h(b_0) = \mathbb{E}_{s \sim b_0}[V_\pi^h(s)]$. The goal in POMDP planning is finding an optimal policy, i.e., a policy $\pi^* = \text{argmax}_\pi V_\pi^h(b_0)$.

In RDDL, POMDPs are defined using typed first-order logic. First, a set of types is defined that determines the relevant objects. For the home theater domain, these include types for devices, ports, and so on:

```
Device: object; // TV, Blu-ray player, etc.
SignalType: object; // audio, video, ...
// ultimate source of signal, e.g., Blu-ray player
SignalSource: object;
Port: object; // a port such as HDMI, cinch, ...
// numbers for counting the number of free ports
count: {@zero,@one,@two,@three};
// how tight a connection is
tightness: {@none,@loose,@tight};
```

For defining the set of states $S$, typed parametrized state fluents are used in RDDL, which are either predicates or functions in the first-order logic sense. The home theater domain uses four state fluents to capture relevant aspects of a given state. The first two, `freeFemalePorts(Device,Port)` and `freeMalePorts(Device,Port)`, determine the number of free female

and male ports on a device, respectively, and can assume values from zero to three. Whether a device has received a signal of a certain source and type is kept track of via `hasSignal(Device,SignalType,SignalSource)`. E.g., `hasSignal(TV,audio,Sat)` means that the audio signal of the satellite receiver has reached the TV. The `connected(Device,Device,Port)` fluent models how tightly two devices are connected—not at all, loosely, or tightly. This represents a deviation from the original domain, where devices are connected tightly or not at all [1]. The difference between a tight and a loose connection is that while the cable is plugged in in both cases, a loose connection does not transport a signal of any kind. This could be interpreted as a halfheartedly plugged in cable, for example. We will later use this to add partial observability to the domain.

The set of states is then given by the set of possible interpretations of state fluents. E.g., suppose the available devices are a TV, a satellite receiver, and an HDMI cable. A state where both the satellite receiver and the TV have one free female HDMI port each, the HDMI cable has two free male HDMI ports, the satellite receiver creates an audio and a video signal, and nothing is connected yet is defined as follows:

```
freeFemalePorts(Sat,HDMI) = @one;
freeFemalePorts(TV,HDMI) = @one;

freeMalePorts(HDMI_Cable,HDMI) = @two;

hasSignal(Sat,audio,Sat);
hasSignal(Sat,video,Sat);
```

Actions and observations are defined in a similar manner, using specific action and observation fluents, respectively. The home theater setup domain features a `connect(Device,Device,Port)` action fluent for instructing the user to connect two devices via some port, a `tighten(Device,Device,Port)` action fluent for instructing the user to tighten loose connections, and a `checkSignals(Device)` action fluent for instructing the user to check the signals on a given device. The only observation fluent of the home theater setup domain is `hasSignalObs(Device,SignalType,SignalSource)`, which signifies whether a given device has a signal of a given type from a given source.

RDDL also has the possibility to define so-called intermediate fluents, whose values are calculated from the current state and which can be used to calculate the successor state. These non-observable fluents are useful when several successor state fluents depend on a single probabilistic outcome. The home theater setup domain uses this feature in `connectSucceeded(Device,Device,Port)` for determining whether a connection between two devices is tight or loose after they are connected. The last type of fluent, called non-fluents, is useful for static properties, such as whether a signal type can be transported through a certain kind of port. E.g., HDMI ports will transport both video and audio signals, but a cinch video port will only transport video.

State transitions are defined in terms of parametrized functions, one for each state fluent and intermediate fluent, called conditional probability functions. For

each instantiation of a state fluent or intermediate fluent, they define a probability distribution over its value in the successor state. E.g., the value of `connectSucceeded(Device,Device,Port)` is defined as follows using RDDL syntax:

```
connectSucceeded(?d1,?d2,?p) =
 if (connect(?d1,?d2,?p) ^
  (freeFemalePorts(?d1,?p) ~= @zero ^
  freeMalePorts(?d2,?p) ~= @zero |
  freeMalePorts(?d1,?p) ~= @zero ^
  freeFemalePorts(?d2,?p) ~= @zero)) then
   Discrete(tightness,@none:0,@loose:0.2,@tight:0.8)
 else if (tighten(?d1,?d2,?p) ^
  connected(?d1,?d2,?p) == @loose) then
   KronDelta(@tight)
 else KronDelta(@none);
```

This means that when `connect` is executed and suitable ports are free on the devices that should be connected, the connect action will succeed. However, the connection will be loose with probability 0.2. This can be fixed by executing `tighten` to make sure the connection is tight.

The value of `connected(Device,Device,Port)` persists unless the connection is tighter than it was before. Similarly, the number of free ports on a device remains unchanged unless `connect` was executed. Signal availability is determined via `hasSignal(Device,SignalType,SignalSource)` by propagating signals over tight connections in every time step. Note that depending on the order in which connect actions are executed, it will take several time steps for a signal to propagate over a chain of devices. It would be preferable to have instantaneous signal propagation, but this would require computing the transitive closure over `hasSignal(Device,SignalType,SignalSource)`, which is not supported in RDDL.

Observation probabilities are defined analogously to state transition probabilities. The home theater domain has `hasSignalObs(Device,SignalType,SignalSource)` as its sole observation fluent. When `checkSignals(Device)` was executed on a device and the device can be checked for the type of signal in question (modeled using a non-fluent), it will reveal every signal available on the device to the system. The rationale behind this is that a TV can be checked for both video and audio signals by simply turning it on, as opposed to an HDMI cable. Checking for signals is the only way for the system to determine whether connections are tight.

The reward function in the model is constructed to fulfill several conditions:

1. The system's first priority should be to bring signals from source devices, such as a satellite receiver, to target devices, such as a TV. The target devices are identified using a non-fluent. In its simplest form, the reward for each time step is simply given by summing the number of signals that have already been brought to their target devices:

```
sum_{?d: Device, ?t: SignalType, ?s: SignalSource}
 DEVICE_NEEDS_SIGNAL(?d,?t,?s)*hasSignal'(?d,?t,?s)
```

A system trying to maximize its expected accumulated reward will therefore strive to bring all signals to their respective target devices in as few time steps as possible. It also urges the system to order connect actions such that cables are plugged in from source devices to target devices. This is more an artifact introduced by the signal propagation mechanism than intended. However, one could argue that this is in the interest of the user of the system, since the system's action recommendations can then be understood as "bringing" the signals to the target devices.

2. The system should avoid executing actions when their "preconditions" are not fulfilled (attempting connects when the required ports do not exist or are not free, attempting tighten when there is no connection at all, checking uncheckable devices). While these conditions are not observable in a strict sense, the system can in principle still derive whether the corresponding actions will succeed. E.g., the number of free ports evolves deterministically, therefore the system can always know whether a connect would fail to at least establish a loose connection. Similarly, whether a device can be checked for signals is known in advance, so checking uncheckable devices can be avoided.
3. The system should avoid executing unnecessary actions (tighten when a connection is already tight, checking for signals more than once on a given device). In contrast to the conditions described in (2), these conditions are harder to fulfill, because they depend on partially observable state properties: suppose a satellite receiver and a TV are connected via a cable and the system has determined that the signal does not reach the TV. This means that one or both of the connections is loose. In this case, the system cannot see which connection is loose, so it must risk tightening an already tight connection to make sure both connections are tight. Still, these conditions are useful since, e.g., tighten never needs to be executed more than once for a given connection.
4. Checking signals and then deciding whether tightening is necessary should be more attractive than simply tightening connections without looking. Some care needs to be taken considering the balance between the different rewards mentioned above. As argued, the system sometimes cannot avoid tightening an already tight connection. But doing so should still be unattractive, otherwise the system can simply ignore the possibility for checking signals and just tighten all connections directly after connecting.

We conclude the description of the home theater setup domain by noting that, in the initial state of every instance, all devices are unconnected and the only devices that have any signals are the signal source devices.

## 3 History-based POMDP Planning

Next, we review a non-hierarchical POMDP planning approach, which will serve as baseline for our experiments in Sect. 5. Monte-Carlo Tree Search is a very popular approach to planning in uncertain environments [3], in particular UCT (Upper Confidence Bound applied to Trees [11]) and its variants, such as MaxUCT [10]. MCTS is a round-based anytime algorithm that incrementally constructs an explicit representation of the search space in memory. The tree contains alternating layers of decision nodes $n_d$ and chance nodes $n_c$, where the root node is a decision node. The number of visits and estimated value of a node in the tree after $k$ rounds of search are denoted by $C^k(n)$ and $V^k(n)$, respectively. Chance nodes also have estimates $R^k(n_c)$ for the immediate reward of executing their associated action. Applied to POMDPs, the search space is the space of observable histories [18] as depicted in Fig. 3.

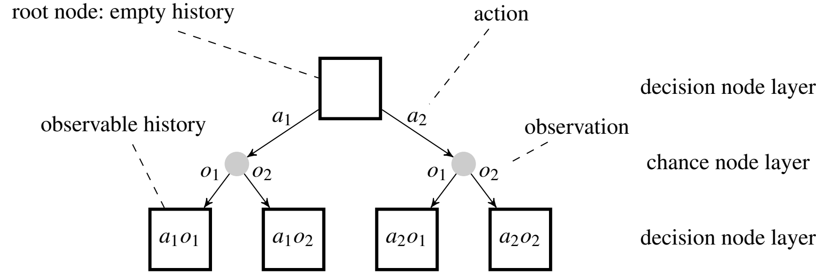



Fig. 3: Search tree for history-based POMDP planning. Rectangular nodes represent observable histories and are called decision nodes. Grey round nodes are called chance nodes.

Each round of search consists of two phases, tree traversal and backup. In the traversal phase, a generative model of the search space is used in conjunction with a tree traversal strategy to explore promising parts of the search space: starting from the root node, the tree is traversed by selecting actions and simulating their outcomes, until a terminal node is reached. In the case of search in the space of observable histories, the root node represents the empty history and terminal nodes are histories of length $H$. A typical tree traversal strategy is the UCT action selection formula [11], which, in a given decision node $n_d$, chooses the action (equivalently chance node $n_c$) that maximizes $V^k(n_c) + B\sqrt{(\log C^k(n_d))/C^k(n_c)}$, where $B$ is a parameter that trades of between exploitative (favor high $V^k(n_c)$) and explorative (favor rarely visited $n_c$) behavior.

The backup phase updates the value estimates by incorporating the information gathered during the traversal, in reverse order of traversal. Chance node values are defined as

$$V^k(n_c) = R^k(n_c) + \frac{\sum_{n_d \in \mathrm{succ}^k(n)} C^k(n_d) V^k(n_d)}{C^k(n_c)}, \tag{2}$$

where the counters $C^k(n)$ are simply incremented in each visit, the immediate reward estimates $R^k(n_c)$ are averages over the experienced immediate rewards, and $\mathrm{succ}^k(n)$ denotes the successor nodes of $n$ in the search tree.

Several useful variants for backing up decision node values exist, most notably Monte-Carlo backup used in UCT [11], i.e.,

$$V^k(n_d) = \begin{cases} 0 & \text{if } n_d \text{ is a terminal node} \\ \frac{\sum_{n_c \in \mathrm{succ}^k(n_d)} C^k(n_c) V^k(n_c)}{C^k(n_d)} & \text{else,} \end{cases} \tag{3}$$

and Max-Monte-Carlo backups used in MaxUCT[10]:

$$V^k(n_d) = \begin{cases} 0 & \text{if } n_d \text{ is a terminal node} \\ \max_{n_c \in \mathrm{succ}^k(n_d)} V^k(n_c) & \text{else.} \end{cases} \tag{4}$$

Both have different strengths and weaknesses, the discussion of which is beyond the scope of this paper. We use a custom combination of both based on the weighted power mean which we call Soft Max-Monte-Carlo backups. For this, let $p = C^k(n_d) / \left|\mathrm{succ}^k(n_d)\right|$ and define

$$V^k(n_d) = \begin{cases} 0 & \text{if } n_d \text{ is a terminal node} \\ \left( \frac{\sum_{n_c \in \mathrm{succ}(n_d)} C^k(n_c) \left(V^k(n_c)\right)^p}{C^k(n_d)} \right)^{1/p} & \text{else.} \end{cases} \tag{5}$$

Until all actions have been tried once, this resembles Monte-Carlo backups when used in conjunction with UCT tree traversal, and converges to Max-Monte-Carlo backups as the number of samples grows.

For our experiments, we denote MCTS in the space of observable histories in conjunction with Soft Max-Monte-Carlo backups and UCT tree traversal as POUCT.

## 4 Partially Observable HTN Planning

Hierarchical Task Network planning, as originally proposed for deterministic planning domains [5], has been successfully applied to a range of real-world planning problems [14]. HTN planning domains are defined in terms of a hierarchy of actions: *abstract* actions are introduced to represent higher-level activities. Normal actions are then often called primitive to distinguish them from abstract actions. An HTN planning problem is given as a number of activities to be performed in a certain order. Since these activities are higher-level, they cannot be directly executed and act as placeholders. For each abstract action, several implementations, called methods, are given. They represent possible ways in which the activity can be performed in terms of a “sub-plan”, in turn consisting of primitive or abstract actions. Plan

generation in HTN planning means iteratively replacing the abstract actions of the initially specified abstract plan with suitable implementations until a solution plan is found that only contains primitive actions. Next, we review the definition of the POHTN approach given in our earlier work [12, 13].

### 4.1 HTN Planning

As a basis, we start by giving a formal description of a simple deterministic HTN planning framework based on the HTN planning formalization of Geier and Bercher [6]. The formalism only uses parameter-less fluents for representing states and actions to keep its description as simple as possible. However, the description can be easily generalized. Furthermore, we simplify the formalism by requiring that all task networks are totally ordered. We will therefore prefer to speak of action sequences instead of task networks, and denote the set of action sequences over action set $X$ as $\mathrm{TN}_X$. An HTN planning problem is a 6-tuple $(L, C, A, M, c_I, s_I)$, where $L$ is a finite set of state fluents. The primitive and abstract actions are given by the disjoint finite sets $A$ and $C$, respectively. The available methods are given by $M \subseteq C \times \mathrm{TN}_{A \cup C}$, and $c_I \in C$ and $s_I \in 2^L$ denote the initial action and initial state, respectively. The dynamics of a primitive action $a$ is defined in terms of preconditions, add and delete lists $(\mathrm{prec}(a), \mathrm{add}(a), \mathrm{del}(a)) \in 2^L \times 2^L \times 2^L$.

Let $\mathrm{ts}_1 = a_1, \ldots, a_{k-1}, c, a_{k+1}, \ldots, a_n$ be an action sequence, $c$ an abstract action, and $m = (c, \mathrm{ts}_m)$ with $\mathrm{ts}_m = a_1^m, \ldots, a_l^m$ be a method for $c$. Applying $m$ to $\mathrm{ts}_1$ creates a new action sequence $\mathrm{ts}_2 = a_1, \ldots, a_{k-1}, a_1^m, \ldots, a_l^m, a_{k+1}, \ldots, a_n$ and is denoted $\mathrm{ts}_1 \to_m \mathrm{ts}_2$. When $\mathrm{ts}_2$ can be created from $\mathrm{ts}_1$ by applying an arbitrary number of methods from $M$, we write $\mathrm{ts}_1 \to_M^* \mathrm{ts}_2$.

An action sequence $a_1, \ldots, a_n$ is called executable in $s$, if and only if every $a_i$ is primitive and there exists a state sequence $s_0, \ldots, s_n$ such that $s_0 = s$, and $\mathrm{prec}(a_i) \subseteq s_{i-1}$ and $s_i = (s_{i-1} \setminus \mathrm{del}(a_i)) \cup \mathrm{add}(a_i)$ for all $1 \leq i \leq n$. Finally, an action sequence $\mathrm{ts}_S$ is a solution to an HTN problem if and only if it is executable in $s_I$ and can be created from $c_I$, i.e., $c_I \to^* \mathrm{ts}_S$.

### 4.2 POHTN Planning

We will now describe the POHTN formalism in order to apply the HTN planning principles just described to POMDPs. We will again describe the POHTN formalism for parameter-less fluents only for brevity, but will give parametrized examples from the home theater domain.

First, we need an appropriate policy representation. We choose logical finite state controllers [12] for this purpose, since they can compactly represent POMDP policies and are a natural generalization of action sequences. A (logical) finite state controller $\mathrm{fsc} = (N, \alpha, \delta, n_0)$ is a directed graph with node set $N$. Each node $n \in N$

is labeled with an action from action set $X$ via $\alpha(n) \in X$. Edges are labeled with transition conditions $\delta(n,n')$, which are first-order formulas over a set of observation fluents $Y$. We call a controller well-defined if the outgoing transition conditions for every given node are mutually exclusive and exhaustive [13], i.e., if transitions define a distinct successor node for every possible observation. The controller is equipped with an initial node $n_0 \in N$. We denote the set of well-defined finite state controllers over $X$ and $Y$ as $\mathrm{FSC}(X,Y)$. Figure 4 shows a finite state controller for the home theater POMDP.

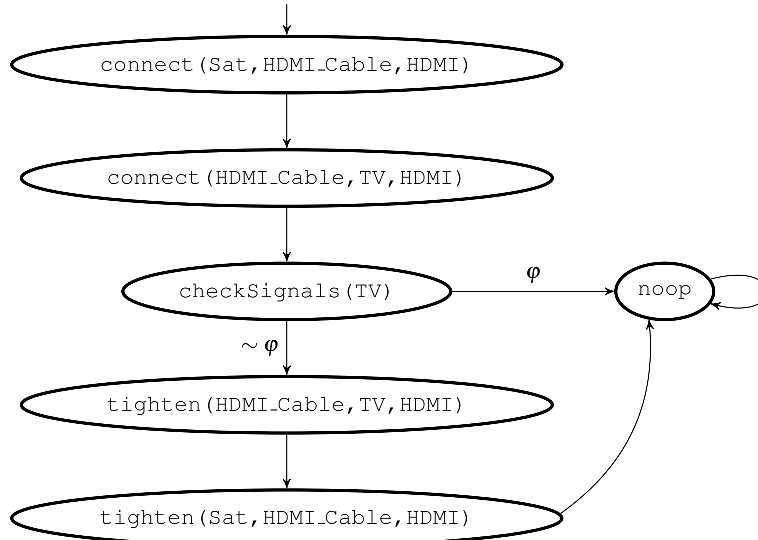



Fig. 4: A finite state controller for the home theater POMDP. The $\varphi$ symbol is short for `exists_{?type,?source} hasSignalObs(TV,?type,?source)`. To reduce clutter, unlabeled edges denote `true` transition conditions say that the transition condition between two unconnected nodes is `false`.

Executing a finite state controller in a given POMDP works by executing the action associated with the current node, starting with the initial node. It is then checked which transition condition $\varphi$ is fulfilled by the received observation $o$, i.e., whether $o \models \varphi$, and the current node is updated to the target node of the transition whose condition is fulfilled. The process is repeated with the new current node until the horizon is reached. Given a history $h$, it is thus simple to determine the prescribed action of a controller by following observation edges.

Just as in HTN planning, we introduce a set of abstract actions $C$ to complement the primitive actions $A$ of the POMDP. For the home theater scenario, we will in-

troduce the abstract action `connect_abstract(Device,Device)`[1]. The intended meaning of the action is that it is an abstraction of bringing the signals of the first device to the second device via any number of intermediate devices, while also making sure that all intermediate connections are tight by checking signals on the target device and tightening connections if necessary.

Additionally, we also model observations on an abstract level and introduce a set of abstract observation fluents $O^C$. The idea is that, just as abstract actions are an abstraction of different courses of action with a common purpose, abstract observations are an abstraction of the observations made while executing such a course of action. The sole abstract observation fluent in the home theater domain is `loose_connection_found`, which is used to represent the fact that a loose connection was found somewhere along the intermediate connections created by `connect_abstract(Device,Device)`.

Methods in POHTN are, analogously to HTN planning, tuples consisting of an abstract action and an implementing controller. The controller representing the implementation part of the method is however a little more complicated due to the need to represent the abstract outcome of the associated abstract action. Such a method controller is a finite state controller which is augmented with a set of terminal nodes $N_t$, $N_t \cap N = \emptyset$. Each terminal node $n_t$ is labeled with an interpretation $L(n_t)$ of the abstract observation fluents and can be the target of transitions. As an example, consider one of the implementations of `connect_abstract(Device,Device)` given in Fig. 5. We denote the set of method controllers over action set $X$, observation fluent set $Y$ and abstract observation fluent set $T$ as $\mathrm{MFSC}(X,Y,T)$.

With the above elements, we can syntactically define the POHTN planning problem as a tuple $(P, C, O^C, M, \mathrm{fsc}_I)$, where

- $P = (S, A, O, T, Z, R, b_0, H)$ is a POMDP,
- $C$ is a finite set of abstract actions with $C \cap A = \emptyset$,
- $O^C$ is a finite set of abstract observations with $O \cap O^C = \emptyset$,
- $M \subseteq C \times \mathrm{MFSC}(A \cup C, O \cup O^C, O^C)$ is the set of methods, and
- $\mathrm{fsc}_I \in FSC(A \cup C, O \cup O^C)$ is the partially abstract initial controller.

In spirit, method application in POHTN planning works very similarly to how it works in HTN planning, albeit a little more involved due to the multiple possible "results" of abstract actions. Let $\mathrm{pred}_{\mathrm{fsc}}(n) = \{n' \in N | \delta(n', n) \neq \texttt{false}\}$ be the set of predecessor nodes of a controller node $n$, i.e., the set of nodes from which $n$ can be reached in one step. Let $\mathrm{fsc}^1 = (N^1, \alpha^1, \delta^1, n_0^1)$ be a partially abstract controller, $c \in C$ an abstract action, and $n_C^1 \in N^1$ with $\alpha^1(n_C^1) = c$ the node to decompose. Let further $m = (c, \mathrm{mfsc})$ be a decomposition method and let $\mathrm{fsc}^2 = (N^2, \alpha^2, \delta^2, n_0^2, N_t^2, L^2)$ be an isomorphic copy of mfsc, for which $N^1 \cap (N^2 \cup N_t^2) = \emptyset$. Applying $m$ to $\mathrm{fsc}^1$ results in a new controller $\mathrm{fsc}^3 = (N^3, \alpha^3, \delta^3, n_0^3)$ which is defined as follows:

- The resulting node set $N^3 = (N^1 \cup N^2) \setminus \{n_C^1\}$ contains all nodes from $N^1$ and $N^2$ except the decomposed node $n_C^1$.

[1] To be precise, this actually introduces $n^2$ parameter-less abstract actions when $n$ is the number of devices.

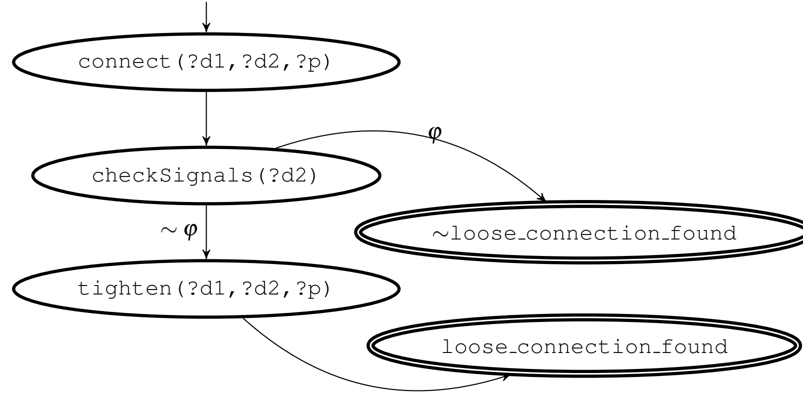



Fig. 5: The method controller of a method for `connect_abstract(?d1,?d2)`. The $\varphi$ symbol is short for `exists_{?type,?source} hasSignalObs(?d2,?type,?source)`. The method implements `connect_abstract(?d1,?d2)` by connecting the devices directly, checking the connection between them, and creating an abstract observation with the result. To represent this method in our parameter-less framework, consider an instantiation of `connect_abstract(?d1,?d2)` with a fixed pair of values for `?d1` and `?d2`. This yields one method for each possible value for `?p`.

- Action labels are kept from the original controllers, i.e.,

$$\alpha^3(n) = \begin{cases} \alpha^1(n), & \text{if } n \in N^1 \\ \alpha^2(n), & \text{if } n \in N^2. \end{cases}$$

- Unless the initial node of the original controller was decomposed, the initial node remains unchanged:

$$n_0^3 = \begin{cases} n_0^1, & \text{if } n_0^1 \neq n_C^1 \\ n_0^2, & \text{if } n_0^1 = n_C^1. \end{cases}$$

- For the node transitions, inner transitions of $\text{fsc}^1$ and $\text{fsc}^2$ are kept. Transitions to the replaced node $n_C^1$ are converted to transitions to the initial node of the method controller $n_0^2$. Transitions to the terminal nodes of $\text{fsc}^2$ are redirected to the successor nodes of $n_C^1$ by checking whether the terminal nodes labels, i.e., interpretations of abstract observation fluents, fulfill the outgoing transition conditions of the decomposed node, which are formulas over abstract observation fluents:

$$\delta^3(n,n') = \begin{cases} \delta^1(n,n'), & n,n' \in N^1 \\ \delta^2(n,n'), & n,n' \in N^2, n' \neq n_0^2 \\ \delta^1(n,n_C^1), & n \in \mathrm{pred}_{\mathrm{fsc}^1}(n_C^1), n' = n_0^2 \\ \bigvee_{t \in N_t^2, L^2(t) \models \delta^1(n_C^1,n')} \delta^2(n,t), & n \in \mathrm{pred}_{\mathrm{fsc}^2}(t), n' \in N^1 \\ \bigvee_{t \in N_t^2, L^2(t) \models \delta^1(n_C^1,n_C^1)} \delta^2(n,t) \vee \delta^2(n,n'), & n \in \mathrm{pred}_{\mathrm{fsc}^2}(t), n' = n_0^2 \\ \texttt{false}, & \text{else} \end{cases}$$

It can be shown that if the transitions in $\mathrm{fsc}^1$ and $\mathrm{fsc}^2$ are well-defined, then so are the transitions in $\mathrm{fsc}^3$ [13].

As an example, consider the partially abstract controller in Fig. 6. Applying the method shown in Fig. 5 to the node labeled with `connect_abstract(HDMI_Cable,TV)` yields the primitive policy shown in Fig. 4.

Again, we write $\mathrm{fsc}^1 \rightarrow_M^* \mathrm{fsc}^2$ when $\mathrm{fsc}^2$ can be created from $\mathrm{fsc}^1$ by applying an arbitrary number of methods from $M$. Let $L(\mathrm{fsc}_I, M) = \{\mathrm{fsc} | \mathrm{fsc}_I \rightarrow_M^* \mathrm{fsc}, \mathrm{fsc} \in \mathrm{FSC}(A,O)\}$ be the set of primitive controllers that can be constructed from initial controller $\mathrm{fsc}_I$ by applying methods from $M$. The solution to a POHTN planning problem $\mathrm{fsc}^* = \mathrm{argmax}_{\mathrm{fsc} \in L(\mathrm{fsc}_I,M)} V_{\mathrm{fsc}}^H(b_0)$ is then defined as the best primitive controller in $L(\mathrm{fsc}_I, M)$.

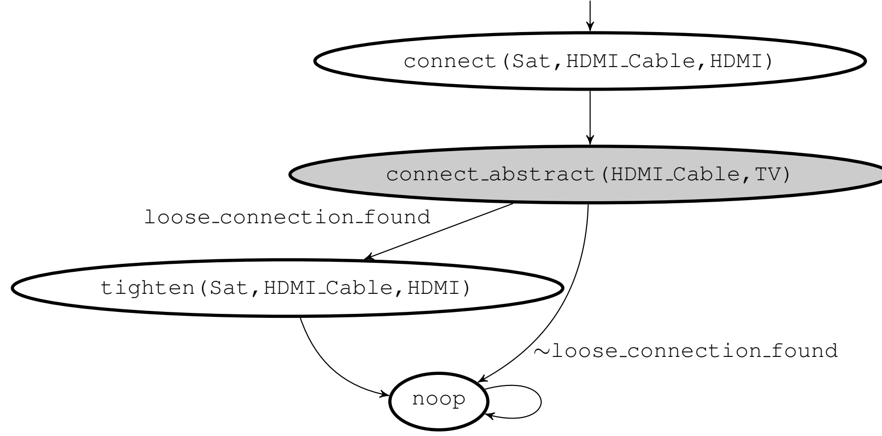



Fig. 6: A partially abstract policy. Grey nodes denote abstract actions.

For deterministic problems, the POHTN approach very closely resembles total-order HTN planning in the formalism described above. Intuitively, a total-order HTN problem can be converted into a POHTN problem by using a sufficiently large planning horizon and a reward function that is defined as minus 1 for violated preconditions and 0 else. If a policy with an accumulated reward of 0 exists, then it also

represents an HTN solution. The task sequences in methods are converted to controllers where each element of the sequence is a node and there are `true` transition conditions between nodes.

## 4.3 A Hierarchy for Home Theater Setup

We are now ready to define a POHTN hierarchy for the home theater setup domain, which boils down to defining methods for `connect_abstract(?d1,?d2)`. The key assumption is that `connect_abstract(?d1,?d2)` is always used in such a way that `?d1` has a signal that needs to be transported to `?d2`, and that `?d2` is checkable for that signal. We also assume that `?d2` is checked for signals and that `loose_connection_found` is true whenever there is no signal at `?d2`.

Given that this is the case, we distinguish four cases. The simplest case is that the two devices can be connected directly, which leads to the method controller depicted in Fig. 5, i.e., the devices are connected, `?d2` is checked, and the connection is tightened if necessary.

When the devices cannot be connected directly, e.g., a `Sat` and `TV`, we can attempt connecting via a third device, e.g., an `HDMI_Cable`. This is realized by first executing `connect(?d1,?d3,?p)` for some port type `?p`, and afterwards using `connect_abstract(?d3,?d2)`. When the result of `connect_abstract(?d3,?d2)` is `loose_connection_found`, we execute `tighten(?d1,?d3,?p)` to make sure the connection between `?d1` and `?d3` is tight.

We also address the case that a connection between `?d1` and `?d2` was already established in the course of connecting two other devices. In this case, we only check `?d2` for signals and report the result in `loose_connection_found`.

Sometimes, it is necessary to create more than one connection between `?d1` and `?d2`, for example when the video and audio signals need to be transported via dedicated video and audio cables. Therefore, the last method contains `connect_abstract(?d1,?d2)` twice.

An initial controller for a given problem instance is always easily created: it simply consists of a sequence of `connect_abstract(?d1,?d2)` nodes for each pair of signal source device `?d1` and target device `?d2`.

## 4.4 Monte-Carlo Tree Search for POHTNs

MCTS can also be applied to POHTNs [13]. The search space in this case is $L(\mathrm{fsc}_I, M)$, i.e., the set of controllers that can be generated by applying methods to the initial controller. The root node of the tree is consequently labeled with $\mathrm{fsc}_I$, and the available "actions" are the methods applicable to a given controller, as illustrated in Fig. 7. Note that method application is deterministic, therefore each chance node

only has a single successor. Also, it does not incur an immediate reward. Terminal nodes correspond to primitive controllers, where both uncertainty and non-zero rewards occur: Once a primitive controller is reached, its execution is simulated once and the resulting accumulated reward is propagated up the tree.

If the hierarchy is chosen suitably, the number of decision that need to be made during planning can be much smaller than in the case of MCTS in the space of histories. An important commonality of MCTS in the space of controllers and MCTS in the space of histories is that they do the same number of simulator calls in each round. This means that we can compare their performance on the basis of the number of iterations, i.e., their sample complexity.

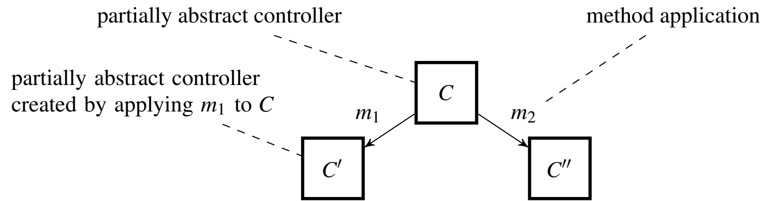



Fig. 7: A visualization of the MCTS search space when applied to POHTNs. The redundant chance node layer is omitted.

## 5 Experiments

We conducted several kinds of experiments to (1) check whether our model satisfies our design goals stated at the end of Sect. 2, (2) to compare the performance of the POUCT algorithm against MCTS applied to POHTNs planning (henceforth just POHTN), and (3) to measure the influence of different modeling choices for the home theater domain.

### *5.1 Quality of Hand-Crafted Policy*

In the first experiment, we checked whether our intuition of appropriate courses of action is in agreement with the results of automated planning methods. To this end, we considered a very simple instance of the domain, where there is only a satellite receiver with an HDMI port, a TV with an HDMI port, and an HDMI cable. The satellite receiver has a video and an audio signal, which the TV needs. The TV is also checkable for both signals. The planning horizon is set to 10. We expect that the policy shown in Fig. 4 is an optimal, or at least near-optimal policy for this instance. We therefore compare the expected accumulated reward of this policy with

the result of running POUCT for a very long time. Figure 8 shows the result. It can be seen that POUCT eventually finds a policy of similar quality as the hand-crafted policy in some runs. Also, there does not seem to be a policy of significantly higher accumulated reward than the hand-crafted policy. This indicates that our intuition of what the optimal policy should look like is correct.

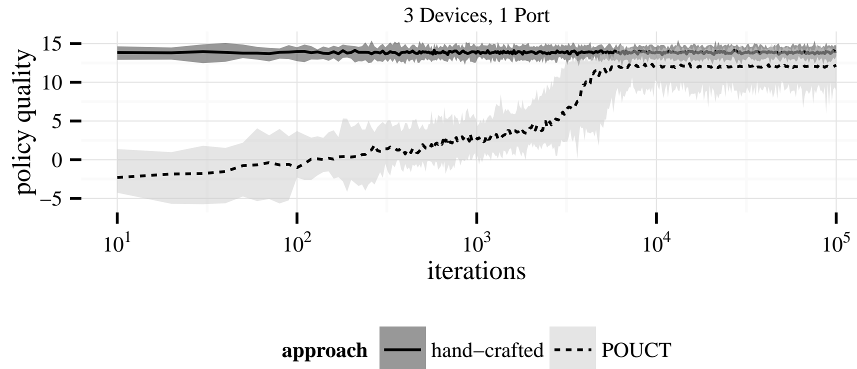



Fig. 8: A plot comparing the policy depicted in Fig. 4 and policies generated using POUCT on a small problem instance. It shows the median (curve) and the max./min. (ribbon) policy quality from thirty runs over the number of iterations. The value of a policy is estimated by simulating its execution forty times and averaging over the accumulated rewards.

## 5.2 POHTN vs. POUCT

For the second experiment, we created several instances of the home theater domain with an increasing number of devices and port types (i.e., difficulty), again with the horizon set to 10. We run both POUCT and POHTN on these instances. We expected the POHTN approach to scale better due to the smaller search space. Indeed, it can be seen in Fig. 9 that the POHTN approach outperforms POUCT on all instances and that the difference in performance grows for larger instances.

## 5.3 Alternative Reward Structure

In the third experiment, we consider a variant of the reward function: the system receives its reward for bringing signals to target devices only when *all* signals are

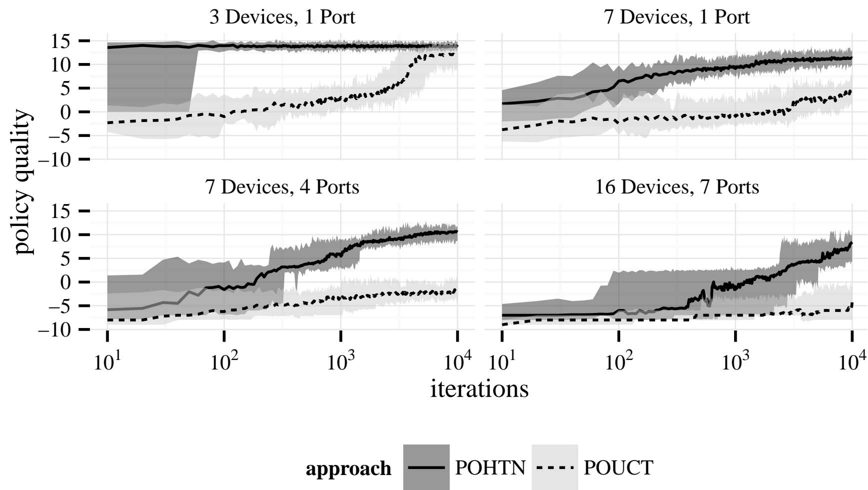



Fig. 9: A plot comparing POUCT and POHTN policy quality on different instances of the home theater setup domain. It shows the median (curve) and max./min. (ribbon) policy quality from thirty runs over the number of iterations.

at their respective target devices. This is closer to how the goal is defined in the original, deterministic domain.

It is not immediately apparent whether this improves or degrades planning performance. On the one hand, this makes it harder for the system to initially identify useful actions. On the other hand, it does not reward "dead-end" cable configurations that cannot be completed to solutions. We present a comparison of the two variants in Fig. 10 on the same instances as in Fig. 9. The policy qualities achieved by the planners are not directly comparable to the results in Fig. 9 due to the different reward structure. However, it can be seen that policy quality improves only very slowly over time for both approaches, except for very small instances. This indicates that rewarding the system for achieving intermediate goals is necessary for generating high-quality policies.

## 5.4 Eliminating Uncertainty

Lastly, we wanted to analyze the effect of eliminating uncertainty in the home theater setup task, i.e., when connecting devices always results in tight connections. The results are shown in Fig. 11. This plot shows that, at least for the smallest instance, the best hierarchical controller is not necessarily the best possible policy for the POMDP. In this case, we can even explain where the hierarchical policy is suboptimal: A controller generated with our hierarchy will always check whether

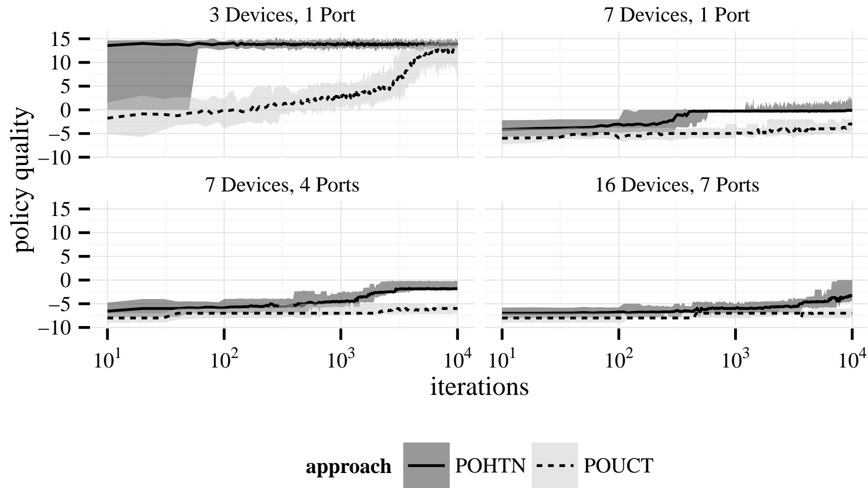



Fig. 10: A plot comparing POUCT and POHTN policy quality when the system receives a positive reward only when all devices are connected correctly. It shows the median (curve) and max./min. (ribbon) policy quality from thirty runs over the number of iterations.

the signals have reached the target devices. However, when connect always results in tight connections, this is unnecessary, and therefore punished. Apart from this, it can be seen from the collapsing ribbon that all POHTN runs on the first three instances eventually generated a policy of the same quality. While this does not guarantee that the best hierarchical controller has been found, it certainly is suggestive.

## 6 Conclusion

We described an approach for exploiting hierarchical structures in partially observable planning problems. To demonstrate its effectiveness, we developed a partially observable variant of the home theater setup domain and constructed a suitable action hierarchy. Our empirical evaluation pursued two goals. First, we tested our modeling decisions by examining the influence of various parameters on solution quality. Second, we compared the performance of the POHTN approach with non-hierarchical planning. The former experiments indicate that our model of the home theater setup domain is adequate. From the latter experiment, we can conclude that our approach is indeed able to exploit existing hierarchical structures in partially observable planning domains. Also note that the same POHTN hierarchy is used in all experiments, regardless of the number devices in the problem instance, which

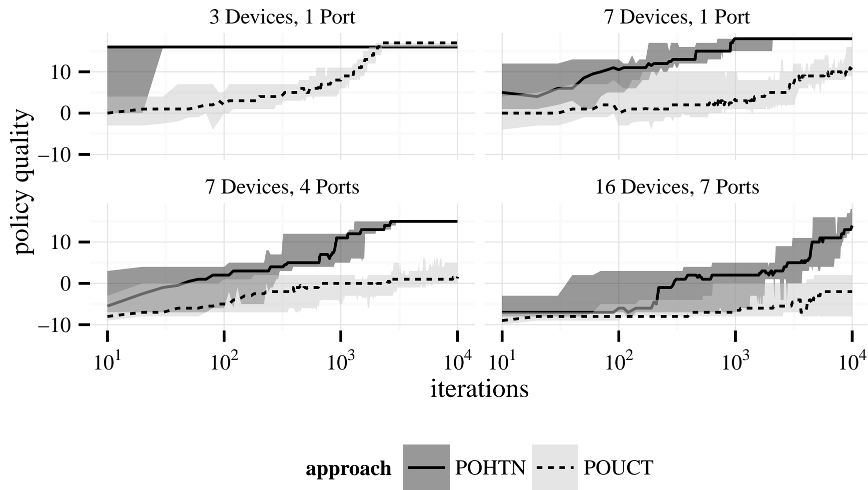



Fig. 11: A plot comparing POUCT and POHTN policy quality when connecting devices always results in tight connections. It shows the median (curve) and max./min. (ribbon) policy quality from thirty runs over the number of iterations.

means that the POHTN approach scales to POMDPs with large state, action, and observation spaces.

**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. Bercher, P., Biundo, S., Geier, T., Hoernle, T., Nothdurft, F., Richter, F., Schattenberg, B.: Plan, repair, execute, explain - how planning helps to assemble your home theater. In: Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014), pp. 386–394. AAAI Press (2014)
2. Bercher, P., Richter, F., Hörnle, T., Geier, T., Höller, D., Behnke, G., Nothdurft, F., Honold, F., Minker, W., Weber, M., Biundo, S.: A planning-based assistance system for setting up a home theater. In: Proceedings of the 29th National Conference on Artificial Intelligence (AAAI 2015). AAAI Press (2015)
3. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. Computational Intelligence and AI in Games, IEEE Transactions on **4**(1), 1–43 (2012)
4. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res. (JAIR) **13**, 227–303 (2000)

5. Erol, K., Hendler, J., Nau, D.: UMCP: A sound and complete procedure for hierarchical task-network planning. In: Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994), pp. 249–254 (1994)
6. Geier, T., Bercher, P.: On the decidability of HTN planning with task insertion. In: Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 1955–1961 (2011)
7. Hansen, E.A., Zhou, R.: Synthesis of hierarchical finite-state controllers for pomdps. In: Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), pp. 113–122 (2003)
8. He, R., Brunskill, E., Roy, N.: PUMA: planning under uncertainty with macro-actions. In: Proc. of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010 (2010)
9. Honold, F., Bercher, P., Richter, F., Nothdurft, F., Geier, T., Barth, R., Hörnle, T., Schüssel, F., Reuter, S., Rau, M., Bertrand, G., Seegebarth, B., Kurzok, P., Schattenberg, B., Minker, W., Weber, M., Biundo, S.: Companion-technology: Towards user- and situation-adaptive functionality of technical systems. In: Proceedings of the 10th International Conference on Intelligent Environments (IE 2014), pp. 378–381. IEEE (2014). DOI 10.1109/IE.2014.60
10. Keller, T., Helmert, M.: Trial-based heuristic tree search for finite horizon mdps. In: Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013), pp. 135–143. AAAI Press (2013)
11. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Proceedings of the 17th European Conference on Machine Learning(ECML 2006), pp. 282–293 (2006)
12. Müller, F., Biundo, S.: HTN-style planning in relational POMDPs using first-order FSCs. In: J. Bach, S. Edelkamp (eds.) Proceedings of the 34th Annual German Conference on Artificial Intelligence (KI 2011), pp. 216–227. Springer (2011)
13. Müller, F., Späth, C., Geier, T., Biundo, S.: Exploiting expert knowledge in factored POMDPs. In: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012), pp. 606–611. IOS Press (2012)
14. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Muñoz-Avila, H., Murdock, J.W., Wu, D., Yaman, F.: Applications of SHOP and SHOP2. IEEE Intelligent Systems (2004)
15. Parr, R., Russell, S.J.: Reinforcement learning with hierarchies of machines. In: Advances in Neural Information Processing Systems 10 (NIPS 1997), pp. 1043–1049 (1997)
16. Pineau, J., Gordon, G., Thrun, S.: Policy-contingent abstraction for robust robot control. In: Proceedings of the 19th conference on Uncertainty in Artificial Intelligence (UAI 2003), pp. 477–484. Morgan Kaufmann Publishers Inc. (2003)
17. Sanner, S.: Relational dynamic influence diagram language (rddl): Language description (2010). Http://users.cecs.anu.edu.au/ ssanner/IPPC_2011/RDDL.pdf
18. Silver, D., Veness, J.: Monte-carlo planning in large POMDPs. In: J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, A. Culotta (eds.) Advances in Neural Information Processing Systems 23, pp. 2164–2172. Curran Associates, Inc. (2010)
19. Sondik, E.: The optimal control of partially observable markov decision processes. Ph.D. thesis, Stanford University (1971)
20. Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence **112**(1), 181–211 (1999)
21. Theocharous, G., Kaelbling, L.P.: Approximate planning in pomdps with macro-actions. In: S. Thrun, L. Saul, B. Schölkopf (eds.) Advances in Neural Information Processing Systems 16 (NIPS 2004), pp. 775–782. MIT Press (2004)