Distributed Control of Situated Assistance in Large Domains with Many Tasks

Jesse Hoey and **Marek Grześ**
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA N2L 3G1

Abstract

This paper tackles the problem of building situated prompting and assistance systems for guiding a human with a cognitive disability through a large domain containing multiple tasks. This problem is challenging because the target population has difficulty maintaining goals, recalling necessary steps and recognizing objects and potential actions (affordances), and therefore may not appear to be acting rationally. Prompts or cues from an automated system can be very helpful in this regard, but the domain is inherently partially observable due to sensor noise and uncertain human behaviours, making the task of selecting an appropriate prompt very challenging. Prior work has shown how such automated assistance for a single task can be modeled as a partially observable Markov decision process (POMDP). In this paper, we generalise this to multiple tasks, and show how to build a scalable, distributed and hierarchical controller. We demonstrate the algorithm in a set of simulated domains and show it can perform as well as the full model in many cases, and can give solutions to large problems (over 10^{15} states and 10^9 observations) for which the full model fails to find a policy.

1 Introduction

Persons with dementia[1] have difficulty completing activities of daily living (ADL) due to a reduced capacity in working (short-term) memory. They can lose track of what parts of a task have been completed, and have increased difficulty in recognising familiar objects. For example, in the kitchen, a person with moderate dementia who is attempting to make a cup of tea may forget that they have already put sugar in their tea and proceed to put more in. Prompting systems, such as the COACH (Hoey et al. 2010b), can help a person with dementia to complete such tasks autonomously by monitoring their progress, and providing audio or video prompts when necessary. COACH uses a partially observable Markov decision process, or POMDP, to monitor the person's progress, as well as elements of their affective state, and to decide on appropriate prompts. However, the COACH system can only deal with a single task (e.g. handwashing).

Smart home systems (Zhang et al. 2008; Singla, Cook, and Schmitter-Edgecombe 2008; Brdiczka, Crowley, and Reignier 2009) attempt to provide assistance to residents across a wide range of tasks using an array of sensors in the environment, algorithms for the analysis of human behaviour, and controllers for the provision of the assistance. The tasks in which automated assistance are most important are those in the kitchen (preparing meals, washing up), and in the washroom (hygiene, dressing). In order to complete complex tasks in these environments, a person must at least maintain a *goal stack* that provides them with motivation and direction for the task (Duke et al. 1998). For example, in the kitchen, a goal stack might be

make breakfast→make tea→get teabag→open teabox.

Therefore, the person must have the ability to *recall* each of these goals in order, and may need to have some further abilities for *recognizing* objects, or for perceiving *affordances* of the environment (Ryu and Monk 2009). Modelling the person's abilities is critical for building a situated prompting system, as it cannot be assumed that a person with dementia will be able to maintain a particular goal stack for any reasonable period of time.

If we attempt to model an entire smart home system as a POMDP in the style of the COACH assistance system, we will soon run into computational problems. A simple solution to this problem is to distribute the model, and approach each task independently. As in hierarchical approaches for robotics (Theocharous, Murphy, and Kaelbling 2004), each subtask can be solved independently, and the solutions controlled sequentially. This approach does not work for a situated prompting system, however, for three reasons. (1) The client may not be acting rationally, and so may switch tasks suddenly, even to a dependent task that is not possible before the original task is complete, and may leave the original task in a critical state (e.g. the stove left on) (Geib 2002); (2) the control in the assistance domain are suggestions to a client, and may not always be effective; (3) the system must be as *passive* as possible, letting the client accomplish tasks on their own if they can do so in order to maintain feelings of independence and control. These three considerations combine to make the subtasks non-independent, and render approaches to assistance in which the system shares the action space with the client less effective (e.g. (Fern et al. 2007)). In this case, it is not optimal for the system to "take over" and do things for the client if they forget. Instead, the system must remind the person as passively as possible what to do.

[1] In this paper, we refer to our group of potential users as *clients*, or as *persons with dementia*.

This paper describes a method for distribution of control in a smart-home system. The idea is to encode all tasks that require assistance into a tree-shaped hierarchy, such that each path from the root to a leaf describes a particular set of abilities required to complete the task at the leaf. Each node in the tree consists of a stand-alone controller (agent), encoded as a POMDP, that prompts the client for a subset of abilities, but relies on its ancestors to prompt for any abilities higher in the tree. The children of a node can be seen as *macro*-behaviours that require the abilities controlled by the parent. An initial, *off-line* phase computes policies for each node in the tree. Subsequently, during interactions with a client *on-line*, the control algorithm processes each new observation in a two-pass distributed fashion. A bottom-up traversal of the tree allows each node to compute its expected value given each possible constraint placed by its parent and each value reported by its children, and to report this expected value to the parent. Subsequently, a top-down traversal reveals the optimal action to take at each level.

The problem we are approaching in this paper is similar to the weakly coupled MDP approach presented in (Meuleau et al. 1998). Similarly to (Meuleau et al. 1998), we assume *additive utility independence* between subtasks, and we assume the state and action spaces can be decomposed into sets of features such that each feature is relevant only to one subtask.[2] However, our domains are inherently partially observable due to observations from sensors, and we cannot assume that subtasks are independent given only an allocation of resources as in (Meuleau et al. 1998). Instead, as in a cooperative multi-player game, the subtasks are "mixed" by the uncertainty in the actions of the client.

The main contribution of this paper is to demonstrate a simple and scalable method for distributed control of a large-scale, partially observable, system for assistance for persons with cognitive disabilities in a smart-home situation. Our method is developed as a solution to a class of problems for which *a-priori* information exists about the structure of the domains, and we take advantage of this as fully as possible. We implement a simple solution to the distributed planning problem, but mainly focus is on the control problem. We demonstrate performance in simulation using a set of large-scale realistic problems, with state space sizes up to 10^{15} and significant partial observability, well beyond the reach of any solution to the full problem.

2 POMDPs for Assistance

2.1 POMDPs

A discrete-time POMDP consists of: a finite set S of states; a finite set A of actions; a stochastic transition model $\Pr : S \times A \rightarrow \Delta(S)$, with $\Pr(t|s,a)$ denoting the probability of moving from state s to t when action a is taken; a finite observation set O; a stochastic observation model with $\Pr(o|s)$ denoting the probability of making observation o while the system is in state s; and a reward assigning reward $R(s,a,t)$ to state transition s to t induced by action a (Figure 1(a)). Since the system state is not known with certainty, a *policy*, π, maps *belief states* (i.e., distributions over S) to actions. A policy maximizes the expected discounted sum of rewards, $\sum_t \gamma^t r_t$, where r_t is the reward obtained at time t, and $\gamma \in [0,1]$ is a *discount factor*.

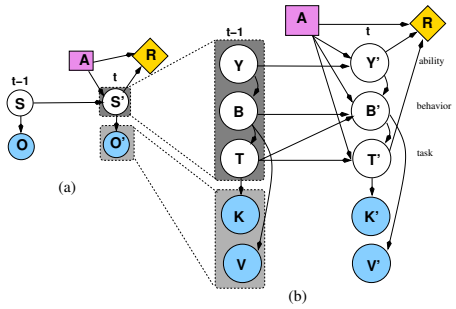


Figure 1: Two time slices of (a) a general POMDP; (b) a factored POMDP for modelling assistive technology;

2.2 Situated Assistance

The general purpose POMDP can be specialised to the assistance domain as shown in Figure 1(b), in which the state, S, has been factored into three sets of variables: *task* (T), *ability* (Y) and *behaviour* (B). The *task* variables are a characterisation of the domain in terms of a set of high-level variables. For example, in the first step of tea making, these include the teabox condition (open, closed) and the cup contents (empty or with teabag). The task states are changed by the client's *behaviour*, B. For the first step in tea making, these include opening/closing the box, moving the teabag to the cup, and doing nothing or something unrelated. The client's *abilities* are their cognitive state, and model, e.g., the ability to recognise the tea box and the ability to perceive the affordance of moving the teabag to the cup. Jointly, $S = \{T, B, Y\}$ is known as the state, and we let $X = \{B, T\}$. Note that the task state, T', is independent of the ability, Y', given the behaviour, B' (Hoey et al. 2010b).

The system actions are prompts that help the client regain a lost ability. We define one system action for each necessary ability in the task. The actions correspond to a prompt or signal that will help the client with this particular ability, if missing. The observations $O = \{K, V\}$ are generated by the *task* and *behaviour* variables. For example, in a kitchen environment there may be sensors in the counter-tops to detect if a cup is placed on them, sensors in the teabags to detect if they are placed in the cup, and sensors in the kettle to detect "pouring" motions (Hoey et al. 2010a).

2.3 Hierarchical Abilities: The Goal Stack

We assume that there are two different types of cognitive abilities: *goal-recall* and *behaviour-recall*, and that a client's goals are organised in a stack. The *goal-recall* abilities are those that affect only the mental state of the client and their goal stack. These abilities allow a person to recall a sub-goal that is necessary to complete during the task. For example, if a person is making a coffee, and has put granules in the

[2] although in the general case this is not strictly true in our domain, we make this assumption to simplify the problem at this stage

cup, then they must recall that the next step is to boil the water. This act of recall pushes a new goal onto their goal stack, and has this effect only. The *behaviour-recall* abilities are then required to accomplish the subtask of boiling water (e.g. recognising the kettle), but these call for specific environmental behaviours (e.g. filling the kettle). However, these abilities will not be relevant if they client does not first have the appropriate *goal-recall* ability. The *goal-recall* abilities define a hierarchical breakdown, whilst the *behaviour-recall* abilities define sequential steps within a level. We assume in this paper that this hierarchy is defined and fixed using a task analysis methods (e.g. (Hoey et al. 2010a)).

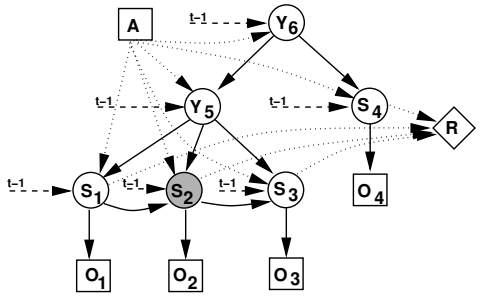


Figure 2: Example hierarchical model for task assistance. The dashed lines represent connections from the previous time step. Dotted and dashed lines are used only for clarity. The shaded node has a 'penalty' state with a large cost.

Figure 2 shows an example for a hierarchical model involving four subtasks with state spaces S_i $i = 1 \ldots 4$ (possibly containing *behaviour-recall* abilities), with observation sets (O_i $i = 1 \ldots 4$), and two sets of *goal-recall* abilities Y_5 and Y_6. This figure is showing the same POMDP model as in Figure 1, except we have factored the *goal-recall* abilities C out of Y, and organised these factors graphically in a tree structure for clarity. The tree structure shows that, to perform subtask S_1, a client will need to recall goals Y_5 and Y_6, and to have abilities $Y_1 \subset S_1$ (in that order). For example, if Y_6 is the goal of making a breakfast of tea and toast, then Y_5 may be making the tea, which involves getting the teabag out of the box and placing it in the cup (S_1) and then boiling water (S_2) and adding it to the cup (S_3), while S_4 may be making the toast. The tree structure will be specific to each individual and each environment, but can be elicited from clients using task analysis methods such as (Hoey et al. 2010a). In the example above, the task of making toast involves no *goal-recall* abilities (other than the recall of the goal of making breakfast). However, another client may forget that the toast is in the toaster and require an additional level in the tree for a *goal-recall* ability for making toast.

The dynamics of the leaves are such that progress toward the goal is only made if the entire path of *goal-recall* abilities leading to the root of the tree are true (the client has these goals on their stack), otherwise, progress will stall (as the client will have forgotten what they are doing). The dynamics can be further complicated by the fact that some subtasks rely on other subtasks to be complete before they can begin (e.g. the arrows $S_1 \rightarrow S_2 \rightarrow S_3$ in Figure 2).

The action A is the controller's prompt, and conditions the dynamics throughout the tree. The controller has one prompt for each *goal-recall* ability, and one prompt for each *behaviour-recall* ability. Prompts affect the dynamics of the abilities by making it more likely the relevant ability will be gained. The rewards, R, are task dependent (the person completes the task). Prompts are costly, as we are building a *passive* system that allows a client to do things independently whenever possible.

3 Distributed Monitoring and Control

The full model defined in the previous section will become intractable for even a moderate number of subtasks. To handle this complexity, we exploit hierarchical probabilistic independencies (Figure 2) in order to distribute the model into a set of individual controllers. There is one controller for each node in the tree, as shown in Figure 3(a) for a single internal node (e.g. for node Y_5 in Figure 2, with $N_c = 3$ children corresponding to the three tea-making steps). The separation is defined by factoring the action space into actions relevant to each node, and adding two new sets of variables to each node. The first, C, is deterministically set by the parent node, and represents some function of the set of *goal-recall* abilities relevant to the child. The simplest such function is a binary indicator for whether the complete stack of *goal-recall* abilities along the path to that child are present or not. However, this can be generalised to a complete specification of the abilities, and can accommodate multiple clients. In fact, the problem we are analysing can be viewed as a type of resource allocation problem (Meuleau et al. 1998) in which the resource is not fully under system control.

We therefore refer to C as a *resource* allocated to a node from its parent. We assume in this paper that C is binary (so resource or control is either present or not). For example, in Figure 2, the resource may be allocated by the root node by giving a prompt to the user to recall they are making breakfast. The sub-task of making tea (Y_5) then has $C = 1$, and so assumes the client knows they are trying to make breakfast. The second variable added, X, describes the state of activity at each child subtask (X is not added to leaf nodes, but instead is replaced with the usual states of $X = \{B, T\}$ in the full model). A subtask is defined as (i) *active* if it has not reached its goal state and there has been client activity in one of its child subtasks; (ii) *complete* if the goal state is reached; and (iii) *inactive* otherwise. This is reported to the parent through the additional observation O_x with the same values (*active, complete, inactive*), and an observation function $P(O_x|X)$ encapsulating the sensor noise at the leaves. For example, in Figure 2, the internal node Y_5 receives indications from each of its three children, and uses this to determine which node to allocate control to (or whether the entire tea-making task is done, in which case this is reported to the parent). Finally, we add additional rewards at each internal node, rewarding the activity states being *complete*.

These additional variables at each node (C and X) also have some dynamics associated with them (e.g. the probability that X is true at time t given its state at time $t-1$).

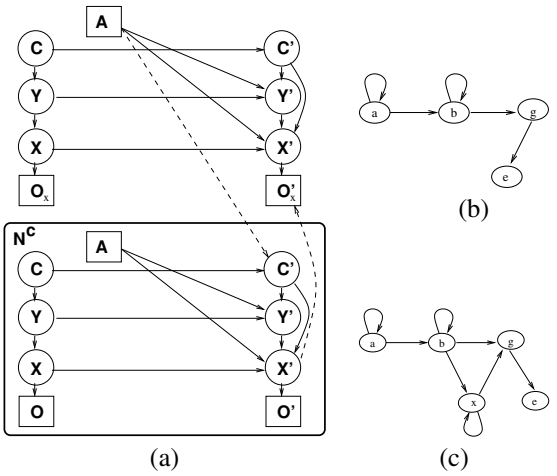


Figure 3: (a) Parent node (top) with N^c child nodes (in a plate) as a distributed model. The dashed line shows a deterministic setting of o' in the parent node from the child node's state of x, and of c' in the child node from the parent's action. (b) Type-I subtask shown as a state-space model. (c) Type-II subtask with penalty state x.

These dynamics are important as they allow a parent node to estimate how quickly a child will complete its task, and allow a child node to determine how likely it is to get a resource at each time step. The dynamics can be computed from the full model through simulation, but are set to fixed *a-priori* values for simplicity in this paper. The action space of each non-leaf node is augmented by the set of control allocations to its children, with $c(a)$ denoting the control allocation of action a. The addition of C and X turns each node in the tree into a POMDP model as shown in Figure 1(b) if we make the association of X with a macro-behaviour/task (indicating which subtask is currently being pursued by the client - behaviour - and which have been completed - task) and of C with a macro-ability (indicating that the client has all abilities higher up the tree to complete the subtask). This elegant decomposition means that a single class of POMDP model can be used at each node.

3.1 Control Algorithm - Fully Observable

We have a tree-structured hierarchy with each node corresponding to a POMDP controller having a unique index j, with $children(j)$ being the child nodes of node j, and N_j^c is the number of children of node j. We let $c_i \in \mathcal{C}_i$ be a resource (ability) allocation to child i. We will write a vector $\vec{x}$ at node j to be a N_j^c component vector $\{x_1, \ldots, x_{N^c}\}$, one for each child of j. A vector with a resource superscript c, $\vec{x}^c = \{x_1^{c_1}, x_2^{c_2}, \ldots, x_{N^c}^{c_{N^c}}\}$ denotes the quantity x in child i if given resources c_i. For notational convenience, we will write the state space for node i as a product of c_i (the resource allocation from the parent), s_i (the node's state space) and $\vec{x}$ (the activity of all the children). To simplify notation, we include $\vec{x}$ and c_i in s_i if not explicitly shown otherwise. We write the full state space for all nodes as $\mathbf{s}$.

We first derive the algorithm assuming a fully observable state space, and then generalise to the partially observable case. This computation makes use of the subtask's average model of how resources will be allocated over time, but does not take into account any impact of a subtask's future actions on any other subtask (other than through the subtask partial orderings). Therefore, we start by solving the POMDP model at each node to find a value function $V(s)$.

The computation must take into account that (1) some allocations, of resources are invalid (due to violations of the partial order), (2) that the client may not, actually, have the necessary resource after allocation and (3) that the client may decide to switch subtasks on their own (and this is something we do not want controllers to discourage, if the client is not violating any validity constraints). We therefore define a binary function $\xi_j(c, s_j)$ that is 1 whenever the resource allocation c to subtask i will be successful when $j = parent(i)$ is in state s_j, and is 0 otherwise. In the assistance domain, a resource allocation will be successfully if the subtask is valid (according to the ordering constraints), and if either the client is doing an on-task behaviour, or is doing nothing but has the *goal-recall* ability that is relevant (in which case, the subtask should have the necessary resources to get the client to complete the task). The function ξ allows subtask values to be "mixed" given the resources of the parent. A resource allocation of $c = 0$ is always successful.

We can now write the expected return at state $\mathbf{s}$ to a node j at time t from one of its children i when it allocates c_i' to node i and ends up in state s_j' (at $t+1$) as:

$$\psi_j^i(s_j', \mathbf{s}, c_i') = \xi_j(c_i', s_j')\tilde{Q}_i(\mathbf{s}, a_i^{*c_i'}, c_i') + \left[1 - \xi_j(c_i', s_j')\right] \tilde{Q}_i(\mathbf{s}, a_i^{*(1-c_i')}, (1 - c_i')). \quad (1)$$

Where $a_i^{*c_i'}$ is the best action to take at node i if given resource c_i' in state s_i: $a_i^{*c_i'} = \arg\max_{a_i} \tilde{Q}_i(\mathbf{s}, a_i, c_i')$. Equation 1 says that if the allocation of resources c_i to subtask i is successful, then the expected return is simply the value expected at the child node from the current state when resources c_i' are to be allocated and the optimal (greedy) policy is followed (given by $\tilde{Q}_i(\mathbf{s}, a_i, c_i')$). If, on the other hand, the allocation is unsuccessful, then the expected return is that without the resource.[3]

The expected value at node i of taking a_i in state s_i and being allocated c_i' is given by

$$\tilde{Q}_i(\mathbf{s}, a_i, c_i') = Q_i(s_i, a_i, c_i') + \gamma \sum_{k=1}^{N_i^c} \sum_{s_i'} \left[\psi_i^k(s_i', \mathbf{s}, c_k'(a_i)) P(s_i'|s_i, a_i, c_i')\right]. \quad (2)$$

The second term is the value expected from the N_i^c children of node i, and the first term is the expected return from i itself if given resources c_i' in the next time step:

$$Q_i(s_i, a_i, c_i') = r(s_i, a_i) + \gamma \sum_{s_i'} P(s_i'|s_i, a_i, c_i') V(s_i') \quad (3)$$

and $V(s)$ is the expected value of state s, computed offline.

[3] assuming a binary resource. Multi-valued resources require an expectation over all values.

3.2 Control Algorithm - Partially Observable

In the partially observable case, we replace states s in Equations 1–3 above with belief states $b(s)$, and replace the sums over states with sums of state-observation pairs, and write:

$$Q_i(b(s_i), a_i, c_i') = \sum_{s_i} r(s_i, a_i) b(s_i) + \gamma \sum_{o_i', s_i'} b^{c_i' o_i' a_i}(s_i') V(s_i') p^{c_i' a_i}(o_i') \quad (4)$$

where $b^{c'o'a}(s')$ is the belief state that results from observing o' after action a was taken and resource c' was assigned, $b^{c'o'a}(s') \propto \sum_s P(s'|s,a,c')P(o'|s')b(s)$, and $p^{c'a}(o')$ is the associated probability of observing o', $p^{c'a}(o') = p(o'|c',a) = p(o',c'|a)/p(c'|a)$ and $p(o',c'|a) = \sum_{s,s'} p(c'|s')p(o'|s')p(s'|s,a)b(s)$, and $p(c'|a) = \sum_{s,s'} p(c'|s')p(s'|s,a)b(s)$, where $b(s)$ is the belief at the previous time step.

We can then express $\tilde{Q}$ in a similar way as

$$\tilde{Q}_i(\mathbf{b}(\mathbf{s}), a_i, c_i') = Q_i(b(s_i), a_i, c_i') + \gamma \sum_{k=1}^{N_i^c} \sum_{o_i', s_i'} \left[b^{c_i' o_i' a_i}(s_i') \psi_i^k(s_i', \mathbf{b}(\mathbf{s}), c_k'(a_i)) \right] p^{c_i' a_i}(o_i') \quad (5)$$

where $\psi_i^k(s_i', \mathbf{b}(\mathbf{s}), c_k'(a_i))$ is given by:

$$\psi_i^k(s_i', \mathbf{b}(\mathbf{s}), c_k') = \xi_i(c_k', s_i') \tilde{Q}_k(\mathbf{b}(\mathbf{s}), a_k^{*c_k'}, c_k') + [1 - \xi_i(c_k', s_i')] \tilde{Q}_k(\mathbf{b}(\mathbf{s}), a_k^{*(1-c_k')}, (1 - c_k')) \quad (6)$$

We combine Equations 4–6 in an online algorithm that starts by computing the Q functions for each node in the hierarchy (the *off-line* phase), then iterates between bottom-up computation of values (function computeValue), top-down action selection (function policyQuery), and belief updates (function updateBelief) in the *on-line* phase.

The algorithm works by allowing a parent controller to assess the expected values of each of its actions (and hence each resource allocation) after an observation is received. It does so in function computeValue by first requesting that each subgoal compute an expected value ($\tilde{Q}$) for each possible resource allocation (line 3). The parent controller then evaluates each of its possible actions, using the reported values from its children. It does this by first looking up the allocation given by the action (line 5). It evaluates an action by combining four elements: (i) the value it expects at the next time step if a is taken, (ii) the value it expects from its children at the next time step, (iii) the immediate reward it will gather, and (iv) the immediate rewards reported by the children. Note that each belief update $b^{c^p oa}$ used to weight the child contributions in the calculation of W on line 7 are only *test* updates, used to evaluate the relative value from the children if a particular resource allocation is made.

A second function policyQuery then does a top-down traversal of the tree, picking the best action to take (according to a^{*c_p} computed at line 9 of computeValue) and recursively choosing actions in the children according to the resources allocated by a^{*c_p}. Finally, the function updateBelief does a second top-down traversal, and updates the beliefs of each node given the observations from the children, the resources from the parent, and the selected actions. Observations are passed up through the tree by computing the activity of the resulting belief states at each node.

Function $\tilde{Q}$ = `computeValue(`$pomdp, c^p$`)`

Input: POMDPNode $pomdp$, resource allocation from parent c^p

Output: $\tilde{Q}$ expected values at current belief given c^p

1 **foreach** subgoal $i \in pomdp.children$ **do** // none if a leaf
2 **foreach** $c_i \in \mathcal{C}_i$ **do** // valid resources for subgoal i
3 $\tilde{Q}_i^{c_i} \leftarrow$ computeValue$(pomdp.child[i], c_i)$
4 **foreach** action $a \in pomdp.\mathcal{A}$ **do**
5 $\vec{c} \leftarrow \mathsf{c}(a)$ // the resource allocation of a to children
6 $\psi(s', \vec{c}) \leftarrow \sum_{i=1}^{N^c} \left[\tilde{Q}_i^{c_i} \xi_i(c_i, s') + \tilde{Q}_i^{1-c_i} (1 - \xi_i(c_i, s')) \right]$
7 $W^a \leftarrow \sum_o \left[\sum_{s'} \psi(s', \vec{c}) b^{c^p oa}(s') + V(b^{c^p oa}) \right] p^{c^p a}(o)$
8 $\tilde{Q}^a \leftarrow \gamma W^a + \sum_s r(s,a) b(s)$
9 $a^{*c^p} \leftarrow \mathsf{argmax}_a \tilde{Q}^a$ // save best action for c^p
10 **return** $\tilde{Q}^{a^{*c^p}}$

Note that the algorithm combines top-down and bottom-up control, allowing the hierarchy to be responsive to the person switching sub-goals: if the person is busy making coffee and the TV remote is picked up, it means they've switched to a different task (possibly due to losing track of the coffee-making task). If the coffee-making task is not in a critical state, then redirecting resources will be most valuable at the parent level, as the sum of ψ values for the new subtask resource allocation will be greater. On the other hand, if the coffee making task is left in a critical state (e.g. the stove was left on), then this value will be negated by the ψ reported from the neglected subtask (if given no resource), and the parent controller may veto the TV task and prompt the person to return to complete the coffee making task.

4 Experiments

We present results of the distributed controller and the full POMDP model on a set of assistance tasks generated using the task analysis method presented in (Hoey et al. 2010a). These domains are designed to capture some of the key issues with assistance for persons with dementia. We used four domains with a hierarchical structure of varying complexity. The leaf nodes in all four domains are of two types, that we have found across a wide range of assistance tasks. Type-I leaves are simple $3 - state$ chain models, as shown in Figure 3(b), with two *behaviour-recall* abilities corresponding to the state transitions $a \rightarrow b$ and $b \rightarrow g$. The initial state is a, and a reward of $+25$ is given at g, which then transi-

tions to an absorbing end state (e). An example Type-I task is getting a teabag into the cup, where $a \rightarrow b$ = *removing teabag from box*, and $b \rightarrow g$ = *putting the teabag in the cup*. Type-II leaves (shown shaded in Figure 2) are $4-state$ models, as shown in Figure 3(c), also with two *behaviour-recall* abilities, but this time with a penalty state (x) that is entered into with probability 0.1 from the second state in the chain (b), and carries a penalty of -5. The same *behaviour-recall* ability is necessary to transition from $x \rightarrow g$ as for $b \rightarrow g$. In tea-making, a Type-II task is one where the stove gets turned on ($a \rightarrow b$) and then turned off ($b \rightarrow g$), when e.g. boiling the water. The transition from $b \rightarrow x$ happens if the water boils out. Transitions between the states in both types of model occur with probability 0.01 when the corresponding ability is lacking, and with probability 0.9 when it is present. Observations have 1% noise for $d = \{0, 1, 2, 4\}$ and 5% noise for $d = \{5, 7\}$. Prompts cost 1.0.

Domain $d = 0$ is the simplest possible domain, consisting of only two type-I leaf nodes with a common parent (3072 states, 64 observations). Domain $d = 5$ is as shown in Figure 2 with one type-II leaf (shaded) and three type-I leaves (3.2 million states, 8192 observations). Domain $d = 4$ is just the left sub-tree of the root node of $d = 5$ (81920 states, 1024 observations) and domain $d = 2$ is like $d = 5$ but without the type-I leaf S_3 and with no constraint between S_1 and S_2 (163840 states, 1024 observations). Domain $d = 7$ is a realistic recipe/packaging task in which the assistant helps a person with a cognitive disability prepare a simple cake, including mixing, cooking, icing, packaging and sending. This domain has three levels in the hierarchy, 9 leaf nodes (4 of them type-II) and 5 internal nodes, and has 3.4×10^{15} states and 2.2×10^{9} observations. The discount factor is $\gamma = 0.95$ in the full model. However, in the distributed version, we do not want to penalise subtask controllers for doing nothing if they are not in control. We therefore use a discount factor of $\gamma = 1.0$ in these nodes, but doing nothing costs 1.0 if the node has control, and 0.0 otherwise.

The default dynamics of the *goal-recall* abilities (without a prompt) are governed by three key parameters that were varied in the experiments. The probability that a client will gain an ability with/without any prompting are denoted g_p and g_r, respectively, while the probability the client will lose an ability if not prompted is denoted l_r. Thus, large values of l_r indicate a person who requires more prompting, while large values of g_r indicate a person who is more likely to start a new goal, and g_p indicates the responsiveness of the client. The easiest clients to prompt are those with small l_r and large g_r (they tend to keep abilities once gained and will spontaneously start new subtasks once complete), while intermediate clients have both small, and the more challenging clients to prompt have both large (they tend to lose goals quickly and switch goals often). Clients with large g_p are easier as well. The most extreme example is $l_r = 0.0$ and $g_r = 1.0$, representing a client who can always perfectly complete the task. The mean reward for this case gives an upper bound for all other settings and policies. Initial states for both simulator and models have all mass on the initial plansteps, but were distributed across the abilities (with p(ability=yes)=0.8), *except* for the upper bound simulators that started with all abilities set to 'yes' (indicating the client knew what to do to begin with).

Domains were simulated for 20 runs in which the full POMDP is used as a simulator (map from actions to observations). The simulations are run until the simulator enters the terminal state or until a maximum of 60 steps. We tested our distributed control algorithm versus the full POMDP policy and versus five heuristic policies: (i) null (always does nothing) (ii) $\xi = 1.0$ so parents assume children will always be successful (iii) $\psi = 0$ so each node operates independently and ignores its children and (iv) without any additional rewards at the internal nodes.

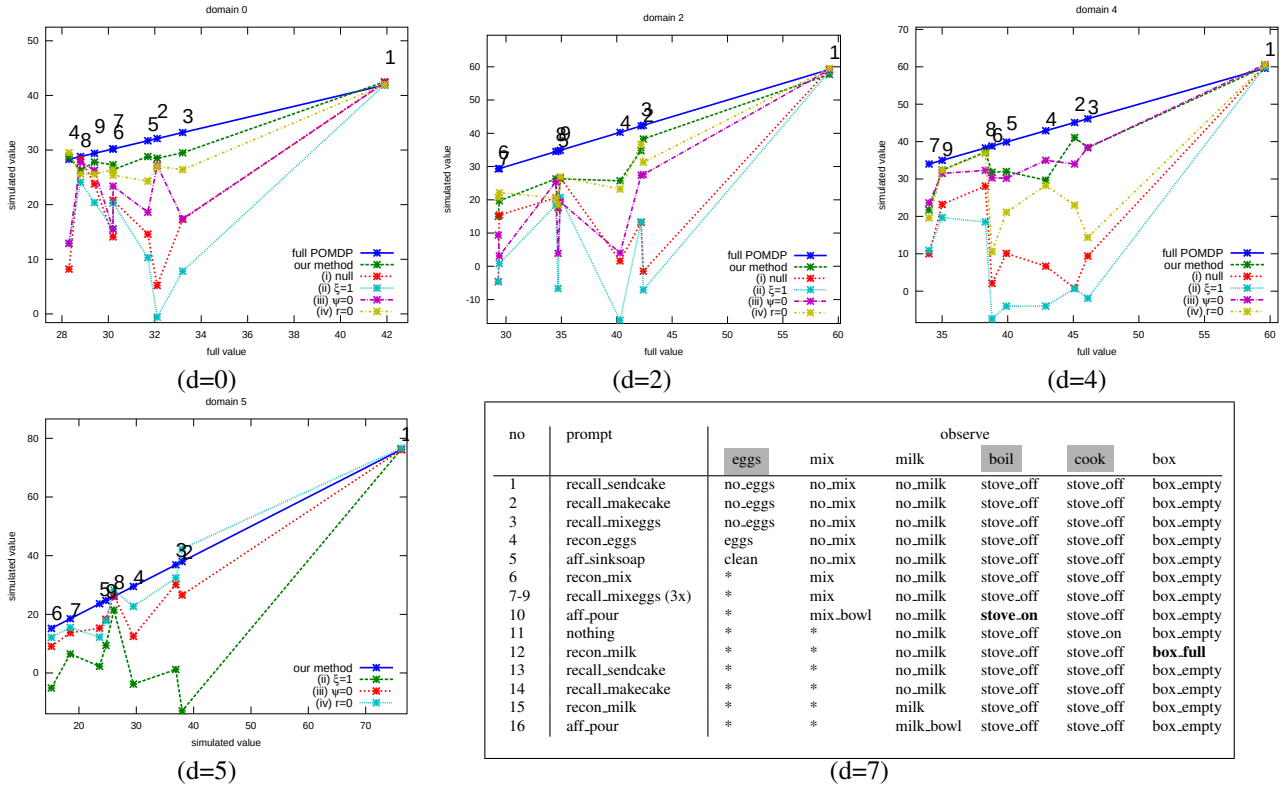
All POMDP policies were generated with symbolic Perseus.[4] It implements a factored, structured point-based approximate solution technique based on the Perseus algorithm (Spaan and Vlassis 2005). We used 100 α vectors and two rounds, with belief states generated using the QMDP policy in the first round, followed by the first round POMDP policy in the second round, mixed with random actions. The distributed nodes were solved using 1500 belief states, while the full POMDP models were solved using 3000 belief states. Policy generation for the full model for $d > 3$ were not possible within 30Gb of memory[5] except for the simplest case with $l_r = 0.0$ and $g_r = 1.0$ for $d = 5$. Domain $d = 7$ is too large to even run the full model in simulation, and so we demonstrate with a simple manual simulation.

Figure 4 shows the results for a range of user models (settings of l_r and g_r), plotted as the mean value returned by each method as a function of the mean value returned by the full POMDP policy (the best we could expect for the given setting of l_r, g_r). $g_p = 0.80$ for all experiments. Our distributed method consistently performs better than any of the heuristic methods, across a large range of user models. Most of the policies perform fairly well on the user models where l_r and g_r are both large. In these cases, the client easily loses and gains abilities, making any prompting strategy ineffectual since the necessary sequences of prompts has little chance of maintaining the person's abilities for long enough to effectuate an appropriate behaviour. In these cases, it is just as valuable to do nothing as it is to prompt according to any other policy. Method (ii) with $\xi = 1.0$ works poorly in most cases, as its assumption that the resource allocation will always work is not valid in many situations. Method (iv) with no value at internal nodes performs well on any domain in which there are no temporal constraints between nodes, as there is no incentive for the parent to do any proper sequencing of the children. In domain $d = 4$ and $d = 5$, this method performs more poorly due to its inability to sequence.

Perhaps the most interesting method is (iii) in which $\psi = 0$. In this case, each node uses its own average (generic) model of the dynamics of its children (of X) and of its parent (of C). This works well in many situations, but fails when something goes wrong (e.g. a person leaves a subtask in a critical state). For example, in domain $d = 2$ and $d = 5$ this method results in a number of trials with very negative re-

[4] www.cs.uwaterloo.ca/∼ppoupart/software. Results generated on a 3.2GHz $AMD\ PhenomIIX61090T$ CPU 8Gb of RAM.

[5] a 64 CPU Intel Itanium with 32Gb of RAM for each process.



no	prompt	observe					
		eggs	mix	milk	boil	cook	box
1	recall_sendcake	no_eggs	no_mix	no_milk	stove_off	stove_off	box_empty
2	recall_makecake	no_eggs	no_mix	no_milk	stove_off	stove_off	box_empty
3	recall_mixeggs	no_eggs	no_mix	no_milk	stove_off	stove_off	box_empty
4	recon_eggs	eggs	no_mix	no_milk	stove_off	stove_off	box_empty
5	aff_sinksoap	clean	no_mix	no_milk	stove_off	stove_off	box_empty
6	recon_mix	*	mix	no_milk	stove_off	stove_off	box_empty
7-9	recall_mixeggs (3x)	*	mix	no_milk	stove_off	stove_off	box_empty
10	aff_pour	*	mix_bowl	no_milk	**stove_on**	stove_off	box_empty
11	nothing	*	*	no_milk	stove_off	stove_on	box_empty
12	recon_milk	*	*	no_milk	stove_off	stove_off	**box_full**
13	recall_sendcake	*	*	no_milk	stove_off	stove_off	box_empty
14	recall_makecake	*	*	no_milk	stove_off	stove_off	box_empty
15	recon_milk	*	*	milk	stove_off	stove_off	box_empty
16	aff_pour	*	*	milk_bowl	stove_off	stove_off	box_empty

(d=7)

Figure 4: Results for different domains. Indices $1-9$ along the top line refer to the user model settings in order as follows: $l_r/g_r = \mathbf{1}: 0/1; \mathbf{2}: 0.05/0.005; \mathbf{3}: 0.1/0.05; \mathbf{4}: 0.2/0.05; \mathbf{5}: 0.3/0.1; \mathbf{6}: 0.4/0.1; \mathbf{7}: 0.5/0.2; \mathbf{8}: 0.4/0.3; \mathbf{9}: 0.5/0.3$.

turn (indicating the client remaining in a critical state for a significant period of time), whereas our method avoids this by focusing effort on the critical subtask.

The distributed algorithm requires about 8 seconds per time step for domain $d = 5$ to compute values, select actions and update beliefs. However, the recursive computation is implemented sequentially, and a parallel version would make the algorithm scale with the depth of the tree.

Figure 4 shows the actions taken by the system for domain $d = 7$ in a simulated run, and the observations entered manually for each of 9 subtasks, shaded meaning type-II subtasks, unshaded are type-I. In the simulation, the client is unresponsive for the first three time steps, and is prompted down a branch of the tree from node recall_send_cake at the root, through node recall_make_cake and recall_mix_eggs, and then prompted at the leaf to perform the first action (recon_eggs - the client needs to recognise the eggs). The client follows this prompt and the next (aff_sink_soap - the client needs to clean hands after breaking eggs, a critical step) and finishes the first subtask. The controller then starts prompting for the second subtask (recon_mix - the client needs to find the cake mix) which the client follows, but then gets stuck from steps 7-9. The controller tries some higher level prompts (rl_mix_eggs), and then again tries to get the client to complete the second subtask (using recon_mix), which works (step 10). There are two sensor mis-fires (in **bold**): stove_on/box_full in subtask boil/box at time 10/12 that are ignored by the controller. The controller then waits, allowing the client to act, but the client starts the cook task out of sequence, and is prompted return to the milk subtask.

5 Discussion and Related Work

Our work is related to four major strands of research: POMDP-based assistance systems (Hoey et al. 2010b; 2010a), plan recognition (Geib 2002), decision-theoretic assistance (Fern et al. 2007), and hierarchical reinforcement learning (Dietterich 2000) and MDP decomposition (Meuleau et al. 1998). The domain we are investigating is that of building assistance systems for smart homes. The problem of assistance is discussed in (Fern et al. 2007), and a relational context is built for a range of assistance tasks, including those in the home, allowing different tasks to share computational resources. However, their model assumes (i) full observability, (ii) a rational client and (iii) a system that shares the action space with the client (is not *passive*). Our model is partially observable, explicitly models the dynamics of a person behaving irrationally, and only has actions that advise, rather than actively interfere with, the client. Although it is possible to include shared actions in our model, these are not desirable for clients with dementia, where independence is a key factor in quality of life.

POMDP-based approaches for assistance of cognitively disabled persons (Hoey et al. 2010b; 2010a) only handle a single task. Multiple, interleaved and abandoned tasks are

discussed by Geib (Geib 2002). While our method deals with action recognition at a basic level, we also aim for control (system actions), and deal with partial observability.

Meuleau et al. (1998) showed that weakly coupled resource allocation tasks (in which subtasks are independent given resources) can be successfully tackled with an approximate, greedy algorithm for solving MDPs. In a similar approach, Dolgov (2006) focuses on global resources, and assumes one-shot allocations. We have an unlimited global resource and an instantaneous, dynamic allocation problem.

There are many examples of approaches to distributed control in which the problem is inherently *serial*, such as in robot navigation (Theocharous, Murphy, and Kaelbling 2004; Pineau, Roy, and Thrun 2001), and hierarchical RL (Sutton, Precup, and Singh 1999; Dietterich 2000). Guestrin and Gordon propose a method for distributed control in factored MDPs that deals with *parallel* and *coupled* tasks by incrementally solving a set of approximate linear programs for each node in a hierarchy (Guestrin and Gordon 2002). Their method relies on full observability, but their reward shaping ideas could be used to find better approximate Q-values in our method, effectively learning the *a-priori* success function ξ in our method, along with the necessary dynamics of c and X. Generalising our method to learn value functions for a more globally optimal solution is subject of our future work, but partial observability poses a significant unsolved hurdle in this area.

The literature on Dec-POMDPs (Seuken and Zilberstein 2007) investigates theoretical properties (e.g., due to coordination) of standard Dec-POMDPs(Boutilier 1999), whereas we use fairly detailed a-priori domain knowledge to construct specific solution methods that work for the large domains we are working on. For example, the JESP algorithm (Nair et al. 2003) seeks a Nash Equilibrium (NE), but in our case the NE is enforced by the hierarchical structure. Specific classes of DEC-POMDPs have been developed (Varakantham et al. 2009), but rely on having "highly" autonomous agents that interact only in certain locales. Our method, on the other hand, has very dependent agents (they share resources), but imposes a structure that can be exploited for policy construction using a greedy approach.

6 Conclusions

We have presented a hierarchical and distributed algorithm for control in very large assistance domains, with a specific focus on assistance for persons with a cognitive disability. We argued that this problem can be viewed as a resource allocation problem in which the person in need of assistance (the client) is the resource, and the allocation is a collaborative task between the client and the controller. We developed a POMDP formulation of this problem, and showed how it can be neatly decomposed into a distributed POMDP model. We developed a distributed control algorithm for this model, and showed results on a challenging set of problems containing the major difficulties in the assistance domain. We have also generated controllers for three real assistance tasks using the methodology in (Hoey et al. 2010a), and we plan to implement these controllers in real systems in the near future. **Acknowledgement:** This research was sponsored by American Alzheimers Association grant numbers ETAC-08-89008 and ETAC-07-58793.

References

Boutilier, C. 1999. Sequential optimality and coordination in multiagent systems. In *Proc. IJCAI*, 478–485.

Brdiczka, O.; Crowley, J. L.; and Reignier, P. 2009. Learning situation models in a smart home. *IEEE Trans. on Systems, Man and Cybernetics - Part B: Cybernetics* 39(1).

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *JAIR* 13:227–303.

Dolgov, D. A., and Durfee, E. H. 2006. Resource allocation among agents with MDP-induced preferences. *JAIR* 27:505–549.

Duke, D.; Barnard, P.; Duce, D.; and May, J. 1998. Syndetic modelling. *Human-Computer Interaction* 13(4):337.

Fern, A.; Natarajan, S.; Judah, K.; and Tadepalli, P. 2007. A decision-theoretic model of assistance. In *Proc. IJCAI*.

Geib, C. W. 2002. Problems with intent recognition for elder care. In *Proc. of AAAI Wrkshp. on Automation as Caregiver*.

Guestrin, C., and Gordon, G. 2002. Distributed planning in hierarchical factored MDPs. In *Proc. UAI*, 197–206.

Hoey, J.; Plötz, T.; Jackson, D.; Monk, A.; Pham, C.; and Olivier, P. 2010a. Rapid specification and automated generation of prompting systems to assist people with dementia. to appear in *Pervasive and Mobile Computing*. doi:10.1016/j.pmcj.2010.11.007.

Hoey, J.; Poupart, P.; von Bertoldi, A.; Craig, T.; Boutilier, C.; and Mihailidis, A. 2010b. Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. *CVIU* 114(5):503–519.

Meuleau, N.; Hauskrecht, M.; Kim, K.-E.; Peshkin, L.; Kaelbling, L. P.; Dean, T.; and Boutilier, C. 1998. Solving very large weakly coupled Markov decision processes. In *Proc. AAAI*, 165–172.

Nair, R.; Tambe, M.; Yokoo, M.; Pynadath, D.; and Marsella, S. 2003. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proc. IJCAI*.

Pineau, J.; Roy, N.; and Thrun, S. 2001. A hierarchical approach to POMDP planning and execution. In *ICML Workshop on Hierarchy and Memory in Reinforcement Learning*.

Ryu, H., and Monk, A. F. 2009. Interaction unit analysis: A new interaction design framework. *Human-Computer Interaction* 24(4):367–407.

Seuken, S., and Zilberstein, S. 2007. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proc. UAI*.

Singla, G.; Cook, D. J.; and Schmitter-Edgecombe, M. 2008. Incorporating temporal reasoning into activity recognition for smart home residents. In *AAAI Wrkshp. Spatial and Temporal Reasoning*.

Spaan, M. T. J., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *JAIR* 24:195–220.

Sutton, R.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112:181–211.

Theocharous, G.; Murphy, K.; and Kaelbling, L. 2004. Representing hierarchical POMDPs as DBNs for multi-scale robot localization. In *Proc. Intl. Conf. on Robotics and Automation*.

Varakantham, P.; young Kwak, J.; Taylor, M.; Marecki, J.; Scerri, P.; and Tambe, M. 2009. Exploiting coordination locales in distributed POMDPs via social model shaping. In *Proc. ICAPS*.

Zhang, S.; McClean, S.; Scotney, B.; and Nugent, C. 2008. Learning under uncertainty in smart home environments. In *Proc. IEEE EMBS Conference*.