Integrating Human-Provided Information Into Belief State Representation Using Dynamic Factorization

Rohan Chitnis[1], Leslie Pack Kaelbling[1], and Tomás Lozano-Pérez[1]

***Abstract*— In partially observed environments, it can be useful for a human to provide the robot with declarative *information* that augments its direct sensory observations. For instance, given a robot on a search-and-rescue mission, a human operator might suggest locations of interest. We provide a representation for the robot's internal knowledge that supports efficient combination of raw sensory information with high-level declarative information presented in a formal language. Computational efficiency is achieved by dynamically selecting an appropriate factoring of the belief state, combining aspects of the belief when they are correlated through information and separating them when they are not. This strategy works in open domains, in which the set of possible objects is not known in advance, and provides significant improvements in inference time, leading to more efficient planning for complex partially observable tasks. We validate our approach experimentally in two open-domain planning problems: a 2D discrete gridworld task and a 3D continuous cooking task.**

I. Introduction

As robots become increasingly adept at understanding and manipulating the world around them, it becomes important to enable humans to interact with them to convey goals, give advice, or ask questions. A typical setting is a partially observed environment in which the robot has uncertainty about the surrounding objects, but a human can give it *information* about some of these objects. The robot would be expected to remember this information and utilize it as needed when given a task or query. This gives rise to an important question: what is the best way to represent the robot's internal knowledge so that this information is correctly processed and combined with the robot's own sensory observations? It is important for the chosen representation to be able to accurately and efficiently answer queries (i.e. do inference) that require it to draw on the given information.

In this work we focus on one specific type of human interaction: *information-giving*. In this setting, we assume that the robot is already aware of its goal, and that the goal is achievable even without external information. However, the given information can help make planning or execution more efficient. For example, consider a search-and-rescue robot tasked with locating two people somewhere within a large, previously unexplored area. A plan P for achieving this goal given no additional information might navigate the area and make observations until both people have been located. Now suppose a human operator gives the robot information that the people are located near each other. Although this information is not necessary for solving the task (P would still suffice), it can help in developing a more efficient plan

[1] {ronuchit, lpk, tlp}@mit.edu

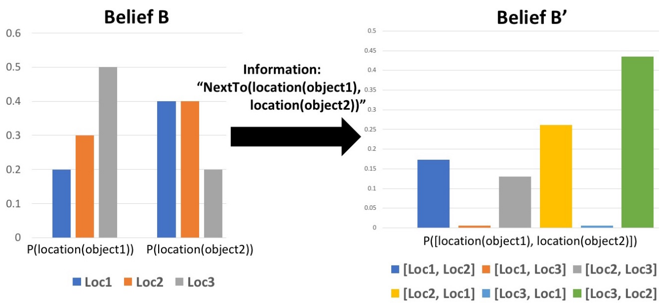


Fig. 1: A schematic illustration of a dynamically factored belief. The initial belief B tracks distributions over possible values for the locations of two objects. There are three locations in the world (not shown): Loc1 on the left, Loc2 in the middle, and Loc3 on the right. When the agent is given information that the objects are next to each other, the belief is updated to produce B'. This is a new factoring in which the two old factors are joined into a single one, corresponding to a distribution over the joint location of both objects. Other factors (not shown) would not be affected.

P' that only navigates until it finds one person, then searches the vicinity for the other. This type of information-giving interaction will become more and more prevalent as robots become integrated with daily human life.

In order to support inference in partially observed environments, one typically maintains a belief state, a probability distribution over the space of possible world states. Unfortunately, the full distribution is generally intractable to work with. A popular alternative approach is to represent a factored belief state (Section III-B), in which the world state is decomposed into a set of features, each with a value. The factored belief then comprises each feature and a probability distribution over its value. This introduces an assumption of independence among the features, so updating the belief with any information that links two or more features would require approximation or a change of representation.

The typical approach to using a factored belief state involves committing to a (possibly domain-specific) representational choice at the very start [1], [2], [3]. In this paper, we argue that in a human-robot interactive setting, we should instead think about a factored belief state as a fluid, dynamic data structure in which the factoring itself is molded by the observations the robot receives, as in Figure 1. We call this a *dynamically factored belief* (Section V).

We consider a specific class of open-domain planning problems in which objects exist in the world, but the agent does not know the universe of objects (Section IV-A). We formalize our setting as a partially observable Markov decision process (Sections III-A, IV-B) in which the robot has two sources of observations: its own perceptual capabilities, and *assertions* about the environment presented in a formal language. These assertions, which simulate human-provided information, are assumed to represent Boolean-valued constraints on the world state and are always accurate. Note that our focus in this work is on the algorithmic incorporation of the assertions into the belief state, hence our simplifying assumption that the assertions are given in a formal language.

For this setting, we show that a dynamically factored belief is a good representational choice for achieving improvements in inference time and planning efficiency, compared to a fixed factoring. We validate our approach experimentally in two open-domain planning problems: a 2D discrete gridworld task and a 3D continuous cooking task.

II. Related Work

We focus on the setting of *information-giving* for open-domain human-robot collaboration. Information-giving was first explored algorithmically by McCarthy in a seminal 1959 paper on advice-takers [4]. Much work in open-domain collaboration focuses on the robot understanding goals given by the human [5], [6], whereas our work is focused on the robot understanding information about its environment.

A. Adaptive Belief Representations

Our work explores belief state representations that adapt to the structure of the observations the robot receives.

The work perhaps most similar to ours is that of Lison et al. [7], who acknowledge the importance of information fusion and abstraction in environments involving human-robot interaction and uncertainty. Building on Markov Logic Networks [8], they describe a method for belief refinement that 1) groups percepts likely to be referring to the same object, 2) fuses information about objects based on these percept groups, and 3) dynamically evolves the belief over time by combining it with those in the past and future. The obtained belief is grounded spatio-temporally and tracks which knowledge source brought about each of its components. Their focus is on adaptive beliefs about each object in the environment, whereas our work focuses on combining information about multiple objects dynamically, based on the structure of the observations received. Furthermore, we show that our approach is useful for efficient planning.

The notion of adaptive belief representations has also been explored in domains outside robotics. For instance, work by Sleep [9] applies this idea to an acoustic target-tracking setting. The belief representation is initialized as a grid discretization of the environment, with the value at each grid location corresponding to the believed probability of a target being there. The belief is then allowed to expand, storing additional information about the targets (such as their acoustic power) that may be important for locating them. The belief can also contract to remove information that is no longer necessary. However, it would not be easy to update such a factoring to include information linking multiple targets. By contrast, our methods work well in the presence of observations about multiple objects.

B. Factored Belief Representations for POMDPs

The more general problem of finding efficient belief representations for POMDPs (Section III-A) is very well-studied.

Boyen and Koller [2] were the first to provide a tractable method for belief propagation and inference in a hidden Markov model or dynamic Bayesian network. Their basic strategy is to first pick a computationally tractable approximate belief representation (such as a factored one), then after a belief update, fold the newly obtained belief into the chosen approximate representation. This technique is a specific application of the more general principle of *assumed density filtering* [10]. Although it seems that the approximation error will build up and propagate, the authors show that actually, the error remains bounded under reasonable assumptions. In our work, we adopt a more fluid notion of a factored belief representation, not committing to a particular factoring.

Bonet and Geffner [1] introduce the idea of beam tracking for belief tracking in a POMDP. Their method begins by leveraging the fact that a problem can be decomposed into projected subproblems, allowing the joint distribution over state variables to be represented as a product over factors. Then, the authors introduce an alternative decomposition over beams, subsets of variables that are causally relevant to each observable variable. This work shares with ours the idea of having the structure of the belief representation not be fixed ahead of time. A crucial difference, however, is that the decomposition used in beam tracking is informed by the structure of the underlying model, whereas in our work, it is informed by the structure of the observations.

III. Background

A. Partially Observable Markov Decision Processes

We will formalize our domains as POMDPs, which effectively model agent-environment interaction in the presence of uncertainty. At each timestep, the agent selects an action, causing 1) the hidden state to transition according to the transition distribution, 2) the agent to receive a reward according to the reward function, and 3) the agent to receive an observation according to the observation model.

Definition 1: We define an undiscounted *partially observable Markov decision process* (POMDP) [11] as a tuple $\langle \mathcal{S}, \mathcal{A}, \Omega, T, O, R \rangle$:

- $\mathcal{S}$ is the state space.
- $\mathcal{A}$ is the action space.
- Ω is the observation space.
- $T(s, a, s') = P(s' \mid s, a)$ is the transition distribution with $s, s' \in \mathcal{S}, a \in \mathcal{A}$.
- $O(s', a, o) = P(o \mid s', a)$ is the observation model with $s' \in \mathcal{S}, a \in \mathcal{A}, o \in \Omega$.
- $R(s', a)$ is the reward function with $s' \in \mathcal{S}, a \in \mathcal{A}$.
- Some states in $\mathcal{S}$ are *terminal*, ending the episode.

The objective is for the agent, given the history of received observations and taken actions, to select an action at each timestep such that its overall expected reward,

$$\mathbb{E}\left[\sum_t R(s_{t+1}, a_t)\right],$$

is maximized. Thus, a solution to a POMDP is a policy that maps the observation and action history to the next action.

Because the true environment state sequence $s_0, s_1, ...$ is not observed by the agent, it must instead work with a *belief state*, a probability distribution over the space of world states. At each timestep, the agent updates this belief state given the previous belief state, the taken action $a \in \mathcal{A}$, and the received observation $o \in \Omega$. The exact belief update is

$$B'(s') = \frac{1}{Z}\left[O(s', a, o)\sum_{s \in \mathcal{S}} T(s, a, s')B(s)\right],$$

where B and B' are the old and new belief states, $s' \in \mathcal{S}$ is a state, $a \in \mathcal{A}$ is the taken action, $o \in \Omega$ is the received observation, and Z is a normalizing factor. Unfortunately, working with a full probability distribution over the space of world states is intractable for most POMDPs.

B. Belief State Representations

We describe a spectrum of approaches to representing belief. On one end, an *eager* representation incorporates the taken action $a \in \mathcal{A}$ and received observation $o \in \Omega$ into some summarizing data structure at each timestep, without remembering a or o explicitly. The full distribution over the space of world states is an example of an eager representation, though working with it is usually infeasible. On the other end, a *fully lazy* representation just appends a and o to a list at each timestep. Though belief updates are trivial with this representation, inference can be very expensive. In our work, we will give a representation that is sometimes eager and sometimes lazy, based on how difficult it would be to incorporate each observation.

A compact strategy for representing a belief state is *factoring*, in which we assume that the state can be decomposed into a set of features, each of which has a value. The factored belief state then comprises each feature and a probability distribution over its value. Typically, one chooses the features carefully so that observations can be *folded*, i.e. incorporated, into the belief efficiently and without too much loss of information. In other words, the chosen distributions are conjugate to the most frequent kinds of observations. Most factored representations are updated eagerly but approximately.

C. Belief Space Planning

Broadly, there are two approaches for generating behavior for a POMDP: online planning [12], [13], or finding a policy offline using a standard POMDP solver. *Belief space planning* [14] is the strategy of solving a POMDP using online search in the space of beliefs of the agent. *Conditional planning* techniques produce graph-structured plans that contain conditional actions, which branch based on the observations received. A special case of conditional planning is *conformant planning* [15], in which one seeks an action sequence (with no branches) that solves the POMDP for all possible initial states, despite the partial observability.

The *replanning via determinization* approach optimistically plans in a determinized version of the environment, brought about by, for instance, assuming that the maximum likelihood observation is always obtained [16], [17]. The agent executes this plan and replans any time it receives an observation contradicting the optimistic assumptions made.

IV. Formal Problem Setting

In this section, we formalize our problem setting, for which we will show that a dynamically factored belief is a good choice for representing the belief state.

A. Planning Problem Class

We consider planning problems for *open domains*, in which objects exist in the world but the agent does not know the universe of objects. Planning in open domains is significantly more complex than planning in settings where the universe of objects is known in advance.

Definition 2: The class of *open-domain planning problems* (ODPP) Π contains tuples $\langle \mathcal{T}, \mathcal{P}, \mathcal{O}, \mathcal{V}, \mathcal{F}, \mathcal{U}, \mathcal{I}, \mathcal{G} \rangle$:

- $\mathcal{T}$ is a known set of object *types*, such as locations or movables. For some types, the set of objects may be known to the agent in advance; for others, it may not.
- $\mathcal{P}$ is a known set of object *properties* (such as color, size, pose, or contents) for each type from $\mathcal{T}$. Each property has an associated (possibly infinite) domain.
- $\mathcal{O}$ is a possibly unknown, finite set of *objects* in the world, each of a type from $\mathcal{T}$.
- $\mathcal{V}$ is the set of *state variables* resulting from applying each property in $\mathcal{P}$ to every object in $\mathcal{O}$ of the corresponding type. Each variable has a domain based on the property.
- $\mathcal{F}$ is a set of *fluents*, Boolean-valued expressions made up of a predicate applied to state variables and (possibly) values in their domains. Examples: *Equals(size(obj1), 6.5)* and *Different(color(obj2), color(obj3))*.
- $\mathcal{U}$ is a set of object-parametrized *operators* that represent ways the agent can affect its environment. Each has preconditions (partial assignment of values to $\mathcal{V}$ that must hold for it to be legal), effects (partial assignment of values to $\mathcal{V}$ that holds after it is performed), and a cost.
- $\mathcal{I}$ is an assignment of values to $\mathcal{V}$ defining the *initial state*.
- $\mathcal{G}$ is a partial assignment of values to $\mathcal{V}$ defining the *goal*.

A solution to a problem in Π is a minimum-cost sequence of parametrized operators $u_1, ..., u_n \in \mathcal{U}$ (a *plan*) such that starting with $\mathcal{I}$ and applying the u_i sequentially results in the partial assignment $\mathcal{G}$ holding. Variables in $\mathcal{V}$ not mentioned in $\mathcal{G}$ may take on any value.

B. POMDP Formulation

With Π defined, we can formulate as a POMDP our human-robot interactive setting, in which a human observing the robot may provide it with *information* that helps it plan or execute more efficiently. We assume the robot is already aware of its goal, even without any such external information.

Let $\langle \mathcal{T}, \mathcal{P}, \mathcal{O}, \mathcal{V}, \mathcal{F}, \mathcal{U}, \mathcal{I}, \mathcal{G} \rangle$ be a problem from Π. Following the notation from Section III-A, define the POMDP:

- $\mathcal{S}$ (the state space) is the space of all possible assignments of values to $\mathcal{V}$. A state is, thus, an assignment of a value to each variable in $\mathcal{V}$. Note that $\mathcal{I}$ is a state.
- $\mathcal{A}$ (the action space) is $\mathcal{U}$.

- Ω (the observation space) is the space of all conjunctions of fluents in $\mathcal{F}$. An observation is, thus, a conjunction of fluents (could be just a single fluent) that holds true in the current state. We assume each observation comes from either 1) the robot's own perceptual capabilities or 2) an *assertion* about the environment that simulates human-provided information. Because $\mathcal{O}$ is unknown, an observation may mention state variables that refer to objects unknown to the agent. Observations always provide accurate information about the state.
- $T(s, a, s')$ (the transition distribution) is 1 if s satisfies a's preconditions and s' its effects; 0 otherwise.
- $O(s', a, o)$ (the observation model) is, for simplicity, a uniform distribution over all valid observations. (If O were non-uniform, then to properly incorporate any assertion, we would need to explicitly reason about why *it* was given as opposed to any other assertion.)
- $R(s', a)$ (the reward function) is the negative cost of a.
- $s \in \mathcal{S}$ is *terminal*, ending the episode, if $\mathcal{G}$ holds in s.

Note that we have made some simplifying assumptions in order to make progress on this difficult problem: that observations are non-noisy and that transitions are deterministic. We hope to remove these assumptions in future work.

Next, we present the *dynamically factored belief* as a good belief state representation for this POMDP. Though this representation does not depend on the presence of assertions or an open domain, we present it within this context because it best motivates the approach and manifests its strengths.

V. Dynamically Factored Belief

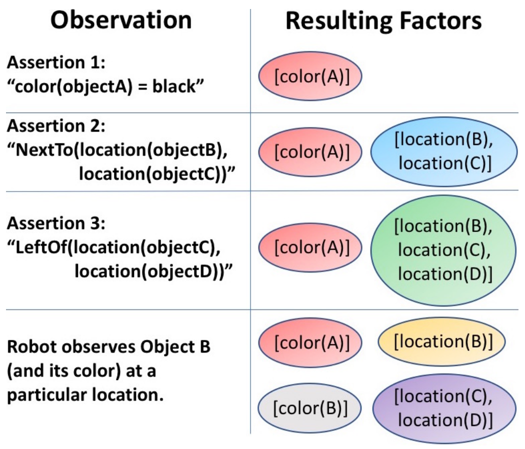
A. *Overview*

In trying to find a suitable belief state representation for the POMDP defined in Section IV-B, we must be cognizant of the fact that the agent may not know $\mathcal{O}$, the complete set of objects in the world. A natural representation would be a factored one (Section III-B) over the state variables $\mathcal{V}$, but unfortunately, if $\mathcal{O}$ is unknown then so is $\mathcal{V}$. Furthermore, the assertions could comprise complicated fluents involving many state variables; we would like our belief state representation to be able to incorporate such information.

We note the following. First, we do not need to know the full set of state variables in order to maintain a (partial) factored belief representation. Second, the structure of the factoring should be influenced by the observations, allowing us to cope with assertions more gracefully than if we had tried to fix a particular factoring. Building on these ideas, illustrated with explanatory examples in Figure 2 and Section V-D, we present the following definition.

Definition 3: A *dynamically factored belief state representation* is a factored representation for which each factor is a *list* of one or more state variables from $\mathcal{V}$. The factors partition the set of state variables that the agent knows about so far (either from prior knowledge or from a received observation). Each factor maps to a joint probability distribution over possible values for all its variables. Also:

- To update with an observation $o \in \Omega$ (a conjunction of fluents): for each fluent f in the conjunction, all factors containing state variables referenced by f are introduced (if they are not represented yet) and joined, so long as the resulting joint would not be too large. We describe computation of the joint in Section V-B.
- If the joint resulting from incorporating a fluent would be too large, the fluent is lazily placed into a database ComplexFluents and considered only at inference time. We describe inference with this database in Section V-C.
- No state variable can be present in more than one factor. We require this because it simplifies enforcing consistency within the representation.
- Factors are kept as small as possible: if a joint can be decomposed into a product of marginals, the factor is split up. This implies that factors do not necessarily get bigger each time a new observation arrives.
- The factoring is initialized with a singleton factor for each state variable in $\mathcal{V}$ that is associated with an object in $\mathcal{O}$ known a priori to exist.
- Thus at any timestep, the distribution implicitly encoded by all the factors together is a joint over assignments to all state variables that the agent knows about so far.



Observation
Resulting Factors
Assertion 1:
"color(objectA) = black"
[color(A)]
Assertion 2:
"NextTo(location(objectB), location(objectC))"
[color(A)]
[location(B), location(C)]
Assertion 3:
"LeftOf(location(objectC), location(objectD))"
[color(A)]
[location(B), location(C), location(D)]
Robot observes Object B (and its color) at a particular location.
[color(A)]
[location(B)]
[color(B)]
[location(C), location(D)]

Fig. 2: Example of dynamic factoring given four sequential observations in a setting with one object type and two properties: *color* and *location*. Each factor maps to a distribution over values (not shown). Initially, there are no factors. *Row 1:* The agent receives a single-fluent assertion containing one state variable, *color(A)*; a singleton factor for this variable is introduced. *Row 2:* The assertion contains two state variables, so a joint factor is introduced. *Row 3:* The assertion contains a new state variable, *location(D)*, so a factor is introduced for it, then joined with the [*location(B), location(C)*] factor. *Row 4:* The agent observes the location of Object B; the joint distribution over the locations of B, C, and D is now uniform across the first dimension, so the factor gets split into two. This type of splitting implies that factors do not necessarily get bigger each time a new observation arrives.

Intuition. In open domains with potentially complicated assertions, we should not commit to one factoring, because we may receive observations that are difficult to fold into it. In a setting where observations only come from the robot's own perception (no assertions), this might not be such an issue: we could fix a factoring based on the semantic structure of the observations. For instance, if the only observation source were the robot's own camera, which at each

```
Algorithm Dynamically Factored Belief Update
1   $B \leftarrow$ InitializeFactoredBeliefMap()
2   ComplexFluents $\leftarrow$ set({})
    Subroutine BeliefUpdate(observation, takenAction)
3       for each fluent within observation do
4           for each stateVar mentioned in fluent do
5               if !B.Contains(stateVar) then
6                   $B$.Add([stateVar], defaultDist())
7           if joint would be too big then
8               ComplexFluents.Add(fluent)
            else
9               $B$.JoinFactorsAndUpdate(fluent)
10      $B$.UpdateWithAction(takenAction)
        /* Keep factors small: efficiency.     */
11      for each factor in B.GetFactors() do
12          $B$.TrySplit(factor)
```

Algorithm 1: Dynamically factored belief update.

timestep can observe what is at a location, then a factored belief state that tracks location contents might be suitable. Unfortunately, this kind of static factoring can become inefficient when human-provided assertions are involved; instead, observation-influenced dynamic factoring can be beneficial.

B. Belief Update

We now describe in detail the dynamically factored belief update algorithm. It is given in pseudocode as Algorithm 1.

Lines 1-2 initialize the full belief state, composed of 1) the factoring map and 2) the database ComplexFluents of fluents for which joining all relevant factors would be too expensive. The factoring is initialized with a singleton factor for each state variable in $\mathcal{V}$ that is associated with an object in $\mathcal{O}$ known a priori to exist. These factors initially map to whatever prior distribution the agent has over their values.

The BeliefUpdate subroutine is called at each timestep, after the agent takes an action and receives an observation. Recall that an observation is a conjunction of fluents, so to model the agent receiving assertions and doing sensing on the same timestep, we simply pass in a conjunction of all constituent fluents to BeliefUpdate as the observation.

Line 3 iterates through every fluent in the conjunction.

Lines 7-8 determine whether joining all factors containing a state variable mentioned by the fluent would be too expensive. If so, the fluent is lazily stored in ComplexFluents and considered only at inference time. An implementation of this test could, for instance, take the product of the relevant factor sizes and check whether it is above a certain threshold.

Line 9 dynamically incorporates the fluent by 1) concatenating all factors containing a state variable mentioned by the fluent into a single new factor, and 2) mapping this factor to a joint distribution in which the fluent holds. We describe how to perform step 2). *Case 1:* If the distributions are discrete, take a Cartesian product and filter out joint values that don't satisfy the fluent. *Case 2:* If the distributions are continuous, the task is more challenging. In our implementation, we eschew calculating the posterior explicitly, and instead use rejection sampling to implicitly draw from it when required.

Lines 11-12 keep the factors as small as possible, splitting up any whose distribution can be decomposed into a product.

```
Algorithm Incrementally Sample World State
    Subroutine SampleState(B, ComplexFluents)
1       state $\leftarrow$ map(each state variable in $B \rightarrow$ NULL)
2       factors $\leftarrow$ $B$.GetFactors()
3       curIndex $\leftarrow$ 0
4       while curIndex $<$ factors.Size() do
5           factor $\leftarrow$ factors[curIndex]
6           if sampling limit reached then
7               for each stateVar in factor do
8                   state[stateVar] $\leftarrow$ NULL
9               curIndex $\leftarrow$ curIndex $-$ 1
                continue
10          values $\leftarrow$ $B$[factor].Sample()
11          for stateVar, value in zip(factor, values) do
12              state[stateVar] $\leftarrow$ value
13          if any fluent in ComplexFluents cannot hold then
                continue
14          curIndex $\leftarrow$ curIndex + 1
        return state
```

Algorithm 2: An incremental algorithm for sampling a world state consistent with all observations, using a dynamically factored belief B. The returned state is an assignment of currently known state variables to values.

C. Inference

Inference is a generic term for answering queries about the world state using the belief. Dynamically factored beliefs can handle two types of queries: 1) a marginal on any state variable or set of state variables that is a subset of some factor, and 2) a sample from the full joint of current factors.

To answer 1), we observe that if a state variable or set of state variables is a subset of some factor, then our representation already stores a joint distribution over values for those variables. Therefore, any query about them can be answered straightforwardly using this joint, e.g. by sampling. Note that we cannot answer queries about a state variable *not* present in any factor, which makes sense because the lack of presence in any factor means the agent has not received any observations indicating the existence of this variable. Also, note that we do not have to worry about violating the constraints in ComplexFluents, because those are guaranteed to be constraining the joint values of state variables across multiple factors, but we are only considering one factor here.

To answer 2), we must draw from the distribution implicitly encoded by all factors together, which produces an assignment to every state variable that the agent knows about so far (either from prior knowledge or from an earlier observation referencing that variable). Algorithm 2 shows one approach for answering this query; the constraints in ComplexFluents are considered in Line 13. Our experiments that do replanning via determinization utilize this algorithm.

D. Illustrative Example

We illustrate updates and inference with a simple example. Consider a planning problem from Π (Section IV-A) with one object type and one property, *color*, whose domain is {red, green}. The robot is tasked with determining the color of every object in the world. Unbeknownst to the robot, there are 3 objects: A, B, and C. Initially, there are no factors.

Suppose the robot takes an action resulting in an observation that Object A is green. The belief is updated with a new factor [*color(A)*] mapping to a distribution over values; thus, the new belief is {[*color(A)*]: {[red]: 0, [green]: 1}}. At this point, a query asking for a sample from the full joint would be answered with the assignment {*color(A)*: green}, since the robot doesn't know about Objects B or C.

Next, suppose the robot receives an assertion that Objects B and C are the same color. The new belief is {[*color(A)*]: {[red]: 0, [green]: 1}, [*color(B), color(C)*]: {[red, red]: 0.5, [red, green]: 0, [green, red]: 0, [green, green]: 0.5}}. Now, a query asking for a sample from the full joint would be answered with either {*color(A)*: green, *color(B)*: red, *color(C)*: red} or {*color(A)*: green, *color(B)*: green, *color(C)*: green}, each with equal probability.

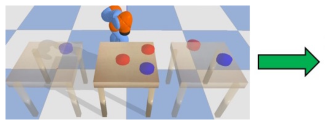
VI. Experiments

We evaluate the performance of our approach experimentally using the cooking task, a planning problem from Π (Section IV-A). The robot is tasked with gathering ingredients and using them to cook a meal. There are three object types: $\mathcal{T} = \{$*locations*, *vegetables*, *seasonings*$\}$. We will use the term *ingredients* to refer to vegetables and seasonings together. Each object type has one property. Locations have a *contents* property, which is either 1) "vegetable" or "seasoning" based on the type of the ingredient located closer to it than to any other location (we set up the environment so there can only be one), or 2) "empty" if there is no such ingredient. Ingredients have a *position* property, which could be continuous- or discrete-valued. Initially, the robot knows the set of locations but not the set of ingredients. Thus, $\mathcal{O}$ (the set of locations and ingredients together) is partially known at the start. There is also a pot at a fixed, known position.

When any vegetable is placed into the pot, it transitions to a *cooking* state; 5 timesteps later, it transitions to a *cooked* state. The robot's goal is to reach a world state in which all ingredients are in the pot and all vegetables are cooked. Importantly, the robot is penalized heavily for placing any seasoning into the pot too early, before all vegetables have been cooked. Note that to achieve the goal, the robot must figure out the positions of all ingredients, either by doing observations or by learning about them from assertions.

The state variables $\mathcal{V}$ comprise each location's contents and each ingredient's position. The world state contains an assignment of these variables to values. It also tracks which ingredients are held by the robot and the pot, and the current robot pose; these are all assumed to be known and thus do not need to be tracked by the belief state. Note that this world state implies a discrete set of locations, a limitation arising from our simplifying assumption that $\mathcal{O}$ is finite.

The robot is capable of only one type of observation in this domain: looking at a location and learning the set of ingredients that is there. Figure 3 shows how such an observation is converted into a conjunction of fluents in the continuous 3D cooking task. Each fluent corresponds to a viewed ingredient and constrains the space of its possible positions to be on the middle table.



Fluents: At(table_mid, position(broccoli)) ∧
At(table_mid, position(potato)) ∧
At(table_mid, position(salt))

Fig. 3: A visualization of how an observation by the robot is converted into a conjunction of fluents in the continuous 3D cooking task. The robot observes the middle table and learns the set of ingredients located on it. For assertions, we assume they are already represented as fluents when given to the robot.

Observation (Fluent)	Dynamic Factors	Foldable for dynamic (ours)?	Foldable for static (baseline)?
(Initial factors)	[c(L1)] [c(L2)] [c(L3)] [c(L4)]	—	—
Assertion 1: NextTo(position(carrot), position(salt))	[c(L1)] [c(L2)] [c(L3)] [c(L4)] [p(carrot), p(salt)]	✓	
Assertion 2: Equal(contents(L2), contents(L3))	[c(L1)] [c(L2), c(L3)] [c(L4)] [p(carrot), p(salt)]	✓	
Assertion 3: NextTo(position(carrot), position(potato))	[c(L1)] [c(L2), c(L3)] [c(L4)] [p(carrot), p(salt), p(potato)]	✓	
Assertion 4: NotEqual(contents(L3), contents(L4))	[c(L1)] [c(L2), c(L3), c(L4)] [p(carrot), p(salt), p(potato)]	✓	
Assertion 5: AtMost2SeasoningsExist()	[c(L1)] [c(L2), c(L3), c(L4)] [p(carrot), p(salt), p(potato)]		
Observation by robot: At(position(carrot), L2)	[c(L1)] [c(L2)] [c(L3), c(L4)] [p(carrot)] [p(salt), p(potato)]	✓	✓
Assertion 6: NotEqual(contents(L1), "vegetable")	[c(L1)] [c(L2)] [c(L3), c(L4)] [p(carrot)] [p(salt), p(potato)]	✓	✓

Fig. 4: An illustration of the types of assertions we use in our experiments, with a simplified execution of the gridworld cooking task. There are four locations in a 2x2 grid: L1, L2, L3, L4. Factors are color-coded based on the number of state variables they represent: blue for one, red for two, and green for three. c($\cdot$) means *contents($\cdot$)*, with domain {vegetable, seasoning, empty}. p($\cdot$) means *position($\cdot$)*, with domain {L1, L2, L3, L4}. Initially, there are 4 factors: singletons tracking each location's potential contents. Distributions over values are not shown. In our actual experiments, the robot takes an action after receiving each assertion; we omit this here for clarity. The last two columns tell whether the fluent is foldable (able to be incorporated) into a dynamic factoring (our approach), and into a static, fixed factoring that tracks the potential contents of each location (baseline B2). Unfoldable fluents get placed into ComplexFluents, which slows down inference. Observe that singleton factors go in and out of joints, and that our experiments use AtMost/AtLeast fluents (as in Assertion 5) which are not foldable into a dynamically factored belief.

We assume that assertions are already represented in formal language. Figure 4 walks through a simplified execution that shows the types of assertions we use in our experiments. At each timestep, we sample an assertion uniformly at random from all the valid ones (following our observation model) and give it to the robot; the information could be redundant. Assertions help the agent perform the task because they are constraints that must hold true in the world; the more knowledge the agent has about the world, the more correctly it can answer queries. The assertions were chosen to reflect the kind of information a robot might receive from a human when hunting for ingredients in a real kitchen.

We now describe $\mathcal{U}$, the set of operators (actions):

- Observe(location): Moves and observes the ingredient(s) at a location. Cost: 5.
- Pick(position): Moves and picks at a continuous- or

discrete-valued position (domain-dependent). The robot can hold up to 10 ingredients at once. Cost: 20.

- PLACEINPOT(): Places all n held ingredients into the pot. Vegetables in the pot are either *cooking* or, 5 timesteps later, *cooked*. Cost: 100 + 50n, plus an additional 1000 if a seasoning is placed in before all vegetables are cooked.
- NO-OP(): Takes no action. Cost: 0.

There is additionally a living cost of 10 per timestep, which incentivizes shorter plans.

All experiments were conducted on a 2015 MacBook Pro with a 2.8 GHz Intel Core i7 processor.

A. Domain 1: Discrete 2D Gridworld Cooking Task

Our first experimental domain is the cooking task applied to a 2D gridworld. Here, the locations are organized in a 2D grid, and the robot is in exactly one of them at any time. Both OBSERVE and PICK actions are performed on single grid locations. As a relaxation from the standard gridworld setup, the robot can move to any location in a single timestep. Each location is initialized to contain either 1) a single ingredient or 2) nothing; note that this means ingredient positions are discrete, and that each is a location. Our experiments measure performance as we vary the grid size and number of ingredients.

Recall that the robot knows the set of locations, but not the set of ingredients, at the start. Thus, the dynamically factored belief is initialized as a map from each location's contents to a distribution over {vegetable, seasoning, empty}, as shown in Figure 4. Factors for the ingredient positions are added in as the agent learns about the ingredients in the world.

We test against two baseline belief representations:

- Baseline B1 is *fully lazy* (Section III-B). This means that on each belief update with an observation o, it just stores every fluent in o into the ComplexFluents database. Updates are trivial, but inference is done by brute-force search over the state space, pruned using the constraints (fluents that must hold) in ComplexFluents.
- Baseline B2 is a static factored representation over each location's contents. This factoring does not change based on observations, or as ingredients are discovered. B2 simulates prior work on factored representations, which typically commit to a representational choice at the start; we choose a factoring over each location's contents because initially, the robot only knows the set of locations. Any fluent (part of a received observation) that cannot fold into this representation is placed into ComplexFluents and considered only at inference time. Section V-C describes how ComplexFluents is used during inference.

B1, B2, and dynamically factored beliefs represent three degrees of laziness during belief updates, with respect to the proportion of fluents stored lazily in ComplexFluents versus eagerly incorporated. B1 is the most lazy (storing all of them), then B2, then our approach. One could also imagine a representation that stores a single full joint over values for all currently known state variables; this eliminates the need for ComplexFluents but is prohibitively expensive.

Setting	System	Bel. Upd. Time	Inf. Time	Ep. Cost (SD)
E1, 4x4, 6ing	B1	0.01	1.1	1097 (47)
E1, 4x4, 6ing	B2	0.01	0.01	1102 (48)
E1, 4x4, 6ing	D (ours)	0.05	**0.01**	**1029** (30)
E1, 7x7, 10ing	B1	0.02	7.8	2200 (104)
E1, 7x7, 10ing	B2	0.05	0.01	2198 (100)
E1, 7x7, 10ing	D (ours)	1.1	**0.01**	**1981** (48)
E2, 3x3, 6ing	B1	0.01	0.26	885 (21)
E2, 3x3, 6ing	B2	0.01	0.10	881 (23)
E2, 3x3, 6ing	D (ours)	0.02	**0.05**	**872** (23)
E2, 4x4, 6ing	B1	0.01	6.5	976 (46)
E2, 4x4, 6ing	B2	0.01	3.0	973 (41)
E2, 4x4, 6ing	D (ours)	0.05	**0.46**	**947** (49)

TABLE I: 2D gridworld cooking task results. Belief update time (seconds), inference time (seconds), and episode cost. Each row reports averages over 100 independent random trials. E1: Experiment 1, E2: Experiment 2. B1: Lazy baseline, B2: Static factoring baseline, D: Dynamically factored belief (our method). *Setting* column gives (experiment, grid size, number of ingredients).

We run two experiments. Experiment 1 tests inference performance on factor-specific queries. It executes a hand-coded *conditional plan* that queries the belief state for information about particular locations, asking questions such as "Is location (0, 0) non-empty?" The plan 1) looks for and picks up vegetables, 2) places them in the pot, 3) looks for and picks up seasonings, 4) waits for the vegetables to be cooked, and 5) places the seasonings in the pot. Experiment 2 tests inference performance on queries about global state. It does belief space planning using the replanning via determinization approach (Section III-C). The determinization is produced using Algorithm 2, and planning is done using A*.

Table I summarizes our quantitative results; each result averages over 100 randomly generated independent problems.

Discussion. Our approach performs the best for all experiments in both average inference time and average episode cost. This suggests that our approach is more efficient at inference than the baselines are. B1 has the lowest average belief update time across the board, as expected because it does almost no work in the update phase, whereas our approach runs Algorithm 1 in the update phase. Our results show a net-positive trade-off of slightly expensive updates in return for much cheaper inference and planning.

B. Domain 2: Continuous 3D Simulated Cooking Task

We now demonstrate the strength of our system in a more realistic robotic domain: the cooking task in a continuous 3D world. See Figure 5 (left) for a pybullet [18] visualization. The main difference is that the ingredient positions are now continuous values: each can be placed anywhere on the surface of one of four tables. The robot can OBSERVE any table and PICK at any position along a table surface. The initial dynamically factored belief state tracks location contents in a grid discretization of each table surface; we

Setting	System	Bel. Upd. Time	Inf. Time	Ep. Cost (SD)
4ing	B2	0.07	0.67	665 (26)
4ing	D (ours)	0.08	**0.06**	**663** (17)
6ing	B2	0.09	5.6	836 (63)
6ing	D (ours)	0.10	**0.08**	**814** (11)
8ing	B2	0.10	19.0	1052 (126)
8ing	D (ours)	0.13	**0.10**	**972** (6)
10ing	B2	0.10	26.4	1268 (161)
10ing	D (ours)	0.16	**0.75**	**1133** (20)

TABLE II: Continuous 3D cooking task results. Belief update time (seconds), inference time (seconds), and episode cost. Each row reports averages over 100 independent random trials. B2: Static factoring baseline, D: Dynamically factored belief (our method). *Setting* column gives number of ingredients, always distributed across 4 tables. Timeout: 60 seconds (included in averages).

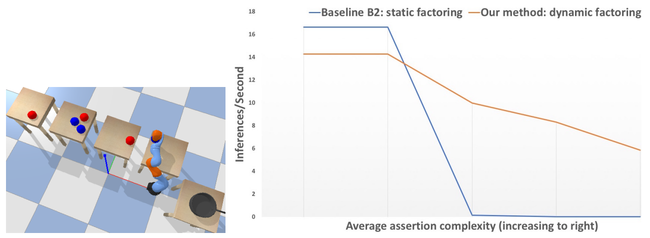


Fig. 5: *Left*: A visualization of the continuous 3D cooking task. The robot is a blue-and-orange arm. Vegetables (red) and seasonings (blue) are placed across 4 tables. The pot is at a fixed, known position. *Right*: Performance of our approach (inferences per second) versus average assertion complexity, measured by the number of state variables referenced. Our method, the dynamically factored belief, continues to answer queries as assertion complexity increases, while using a static factoring becomes prohibitively slow.

do this discretization because we require a discrete set of locations. As ingredients get discovered, the belief maps their positions to continuous distributions over potential values.

We only run Experiment 2 (belief space planning), and we only compare against B2 (static factoring) because B1 (lazy) was prohibitively slow, taking over 10 minutes to answer a single query. In our experiments, we impose a timeout of 60 seconds for answering each query. Table II and Figure 5 (right) summarize our quantitative results.

Discussion. We again see a trade-off: our approach requires slightly more time to do belief updates, but average inference time and episode cost are improved. B2 timed out often while answering queries, whereas our approach never did. Our approach also scales much better than B2 as the average number of state variables per assertion increases. We conclude that dynamically factored beliefs are a good representation for efficient inference in these types of domains.

VII. Conclusion and Future Work

We have considered the problem of belief state representation in a human-robot information-giving setting, and formulated it as a POMDP for a particular class of open-domain planning problems. We showed that a dynamically factored belief is a good representational choice for efficient inference and planning in this setting. We hope that this work provides a step toward the greater goal of fully interpretable communication between humans and robots.

One future direction for this work is to represent the assertions using natural language. Ideally, we should have a system that learns from data to automatically map language to a formal representation, i.e. a fluent. Another interesting direction to explore is a non-uniform observation model: if the agent is given information I, it can learn something not only from I itself but also from the fact that it was told I as opposed to anything else. We also hope to explore environments with stochastic transitions and noisy assertions; making our methods work in these types of settings would require new techniques for dealing with uncertainty within the human-provided information itself.

References

[1] B. Bonet and H. Geffner, "Belief tracking for planning with sensing: Width, complexity and approximations," *Journal of Artificial Intelligence Research*, 2014.

[2] X. Boyen and D. Koller, "Tractable inference for complex stochastic processes," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 1998.

[3] B. Sallans, "Learning factored representations for partially observable markov decision processes," in *Advances in neural information processing systems*, 2000, pp. 1050–1056.

[4] J. McCarthy, *Programs with common sense*. RLE and MIT Computation Center, 1959.

[5] K. Talamadupula, J. Benton, S. Kambhampati, P. Schermerhorn, and M. Scheutz, "Planning for human-robot teaming in open worlds," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 1, no. 2, p. 14, 2010.

[6] K. Talamadupula, G. Briggs, M. Scheutz, and S. Kambhampati, "Architectural mechanisms for handling human instructions in open-world mixed-initiative team tasks," *Advances in Cognitive Systems (ACS)*, vol. 6, 2013.

[7] P. Lison, C. Ehrler, and G. J. M. Kruijff, "Belief modelling for situation awareness in human-robot interaction," in *19th International Symposium in Robot and Human Interactive Communication*, 2010.

[8] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1, pp. 107–136, 2006.

[9] S. R. Sleep, "An adaptive belief representation for target tracking using disparate sensors in wireless sensor networks," in *Proceedings of the 16th International Conference on Information Fusion*, 2013.

[10] P. S. Maybeck, *Stochastic models, estimation, and control*. Academic press, 1982, vol. 3.

[11] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, pp. 99–134, 1998.

[12] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," in *Advances in neural information processing systems*, 2010, pp. 2164–2172.

[13] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "Despot: Online pomdp planning with regularization," in *Advances in neural information processing systems*, 2013, pp. 1772–1780.

[14] B. Bonet and H. Geffner, "Planning with incomplete information as heuristic search in belief space," in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, 2000, pp. 52–61.

[15] R. P. Goldman and M. S. Boddy, "Expressive planning and explicit knowledge." in *AIPS*, vol. 96, 1996, pp. 110–117.

[16] R. Platt Jr, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, "Belief space planning assuming maximum likelihood observations," 2010.

[17] D. Hadfield-Menell, E. Groshev, R. Chitnis, and P. Abbeel, "Modular task and motion planning in belief space," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 2015, pp. 4991–4998.

[18] E. Coumans, Y. Bai, and J. Hsu, "Pybullet physics engine." [Online]. Available: http://pybullet.org/