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

by

Rohan Chitnis

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Author .....
Department of Electrical Engineering and Computer Science
May 11, 2018

Certified by .....
Leslie P. Kaelbling
Panasonic Professor of Computer Science and Engineering
Thesis Supervisor

Certified by .....
Tomás Lozano-Pérez
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

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

by

Rohan Chitnis

Submitted to the Department of Electrical Engineering and Computer Science
on May 11, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

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.

Thesis Supervisor: Leslie P. Kaelbling
Title: Panasonic Professor of Computer Science and Engineering

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor of Computer Science and Engineering

Acknowledgments

To my family, who has been there for me since the beginning: thank you for your unconditional love and support.

To my advisers and labmates: thank you for giving me the opportunity to learn from and alongside you every day.

Contents

Chapter 1

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

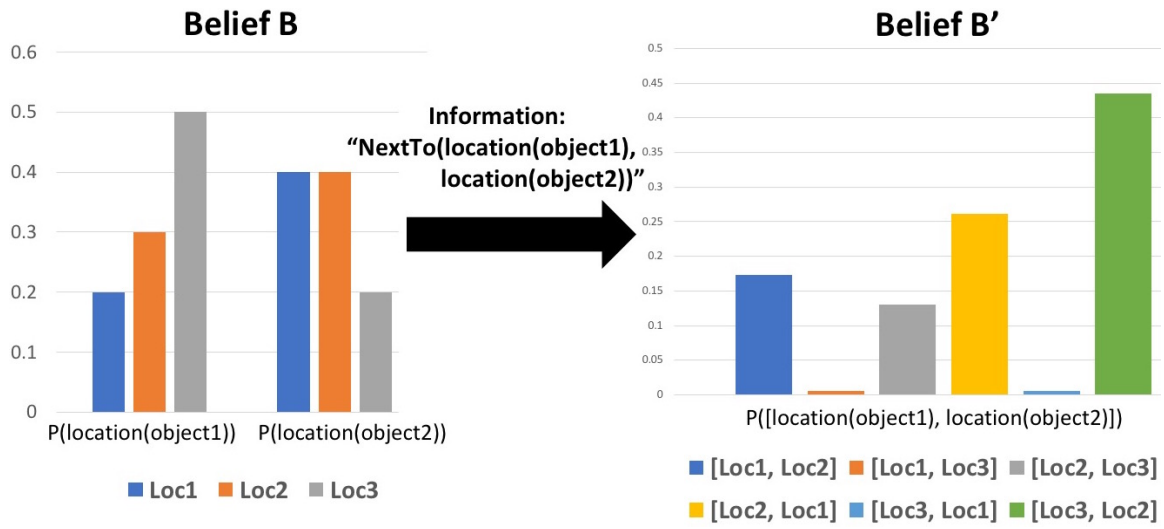


Figure 1-1: A schematic illustration of a dynamically factored belief. The initial belief B tracks distributions over possible values for the locations of two objects. There are three locations in the world (not shown): Loc1 on the left, Loc2 in the middle, and Loc3 on the right. When the agent is given 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.

solving the task (P would still suffice), it can help in developing a more efficient plan 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 2.2), 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 [4, 5, 22]. 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 1-1. We call this a *dynamically factored belief* (Chapter 5).

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 4.1). We formalize our setting as a partially observable Markov decision process (Sections 2.1, 4.2) 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, represent constraints on the world state. To build intuition, our initial presentation of dynamically factored beliefs in Chapter 5 assumes that observations are accurate and noiseless (i.e., they represent Boolean-valued constraints). Chapter 6 presents an alternative view of our work from the perspective of factor graphs (Section 2.4). In Chapter 7, we extend our approach to a setting where observations (both sensory ones and assertions) can be noisy.

Note that our focus in this research 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 the class of open-domain planning problems we consider, 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. In Chapter 8, we validate our approach experimentally in two open-domain planning problems: a 2D discrete gridworld task and a 3D continuous cooking task.

Chapter 2

Background

2.1 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*) [10] 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.

2.2 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.

2.3 Belief Space Planning

Broadly, there are two approaches for generating behavior for a POMDP: online planning [24, 26], or finding a policy offline using a standard POMDP solver. *Belief space planning* [3] 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* [7], in which one seeks an action sequence (with no branches) that solves the POMDP for all possible initial states, despite the partial observability.

The *determinize-and-replan* approach optimistically plans in a determinized version of the environment, brought about by, for instance, assuming that the maximum likelihood observation is always obtained [20, 8]. The agent executes this plan and replans any time it receives an observation contradicting the optimistic assumptions made.

2.4 Factor Graphs

In Chapter 6, we will present an alternative view of our work from the perspective of factor graphs. We give a brief overview of the concept in this section, and refer the reader to Kschischang et al. [13] for a more thorough treatment. A *factor graph* is a bipartite undirected probabilistic graphical model containing two types of

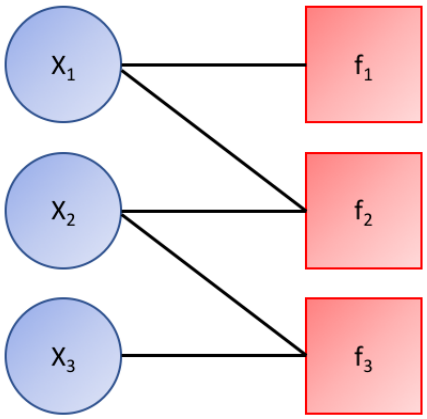


Figure 2-1: An example of a factor graph with 3 variables and 3 factors. This factor graph corresponds to the decomposition $f(X_1, X_2, X_3) = f_1(X_1) \cdot f_2(X_1, X_2) \cdot f_3(X_2, X_3)$.

nodes: *variables* and *factors*. Factor graphs provide a compact representation for the factorization of a function. For instance, suppose a function f on n variables can be decomposed as follows:

$$f(X_1, X_2, ..., X_n) = \prod_{i=1}^{m} f_i(C_i),$$

where each $C_i \subseteq \{X_1, X_2, ..., X_n\}$ is a subset of the variables. This decomposition corresponds to a factor graph in which the variables are the X_j, the factors are the f_i, and there is an edge between any f_i and X_j for which $X_j \in C_i$, i.e. f_i is a function of X_j. See Figure 2-1 for an example.

This representation affords efficient algorithms for marginal inference, which is the process of computing the marginal distribution of a variable, possibly conditioned on observing the values of some other variables. Message-passing algorithms such as the sum-product algorithm typically utilize dynamic programming to recursively send messages between neighboring nodes. This sum-product algorithm is also commonly referred to as *belief propagation*.

Message-passing algorithms compute exact marginals when the underlying factor graph is a tree; however, if the factor graph has cycles, exactness is usually lost.

Typical approaches to approximating the true marginals in the cyclic setting include running several iterations of loopy belief propagation [19] or employing Monte Carlo methods [2]. If exactness is required, the junction tree algorithm [9] does compute exact marginals in the cyclic setting, but it can be expensive because it requires first compiling a junction tree data structure before query time. This preprocessing is especially infeasible in our setting, where we will see that the structure of the factor graph is constantly changing due to the incoming information.

Chapter 3

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 [16]. Much work in open-domain collaboration focuses on the robot understanding goals given by the human [28, 29], whereas our work is focused on the robot understanding information about its environment.

3.1 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. [14], who acknowledge the importance of information fusion and abstraction in environments involving human-robot interaction and uncertainty. Building on Markov Logic Networks [21], 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 CORPP algorithm [32], similar to ours, provides a method for combining external knowledge with sensory observations in partially observed settings. CORPP explicitly reasons about the set of possible worlds (answer sets that map literals to attribute values) using the framework of Answer Set Programming. A probabilistic reasoner module then associates each possible world with a likelihood, from which an initial belief for the POMDP is calculated. The belief is then updated exactly and an optimal policy for the POMDP is calculated using standard methods. By contrast, our work does not rely on explicitly representing the set of possible worlds (which may be prohibitively expensive); we instead work with a factored belief state representation in which the factoring is dynamically chosen based on the external knowledge.

The notion of adaptive belief representations has also been explored in domains outside robotics. For instance, work by Sleep [25] 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.

3.2 Factored Belief Representations for POMDPs

The more general problem of finding efficient belief representations for POMDPs (Section 2.1) is very well-studied.

Boyen and Koller [5] 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 repre-

sentation (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* [15]. 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.

Murphy and Weiss [18] give an alternative algorithm to the one by Boyen and Koller. They also track the belief approximately as a product of independent factors, but instead of doing an exact belief update before re-marginalizing, they compute the approximate marginals directly during the update step. Thus, their algorithm is less accurate in general, but is usable in settings where computing an exact belief update is not feasible.

Albrecht and Ramamoorthy [1] define the notion of *passivity* for efficiently updating factored belief representations. A variable in the state is *passive* if it can only change its value when at least one other variable does. For instance, in a domain with object B enclosed inside object A, the position of object B is a passive variable because it can only change when the position of object A changes. We implicitly make use of passivity when deciding how to dynamically update our belief representation.

Bonet and Geffner [4] 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. The belief update at each timestep is first independently performed for each factor based on the new observations and actions, but then the results are merged so that they are consistent with each other (because some state variables may be present in more than one beam). Due to this expensive merge step, the exact formulation of beam tracking only offers space complexity improvements over traditional factoring. However, the authors also introduce an iterative algorithm

for approximating the merge locally, providing time complexity improvements as well. 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.

3.3 Markov Logic Networks

Markov Logic Networks (MLNs) [21] are a sound and practical mechanism for combining probabilistic inference and first-order logic into a unified representation. An MLN is a template for a ground Markov network, consisting of a knowledge base (KB), a weight for each formula in the KB, and a set of constants. Thus, an MLN can be viewed as a soft version of a KB in which each formula has an associated measure of importance, and each world state has a corresponding likelihood based on the total importance of the formulas it satisfies.

A key difference between MLNs and our work is that the structure of the MLN is fixed and defined by the formulas and constants. On the other hand, our system works online, changing the structure of the underlying representation dynamically based on new formulas and constraints that arise while the robot is acting in the world. This type of dynamic sensitivity is crucial for real-world systems, in which it would be difficult to find a fixed structure that could properly capture the full space of incoming information. Another difference is that our work folds the provided first-order logic observations into the belief representation dynamically rather than storing them explicitly (as is needed for inference with MLNs), making inference easier at the cost of slightly more expensive updates.

3.4 Determinize-and-Replan

One belief space planning technique we make heavy use of is the determinize-and-replan approach. This is the idea of planning in a determinized, optimistic version of

an environment, executing this plan, and replanning if something unexpected occurs. A system that leverages this idea successfully is FF-Replan [30], which synthesizes a classical plan for a version of the input problem that determinizes all probabilistic transitions, then replans if a transition has a differing outcome. Of course, because the planner does not consider all possible results of each action while planning, the plan may be suboptimal. In practice, though, FF-Replan has been shown to perform well on a variety of domains.

These concepts have also been applied to robotic environments, such as in work by Brafman and Shani [23]. They generate and solve a planning problem that samples and plans for only a subset of the full set of initial world states. Hindsight optimization ([12, 31]) works in a similar way: sample several possible worlds that are consistent with the current belief, generate a plan for bringing each world to the goal state, and at each timestep pick the action with the lowest expected cost over this set of samples.

Chapter 4

Formal Problem Setting

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

4.1 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.

4.2 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 2.1, 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. Note that we are assuming these observations to be accurate and noiseless, representing constraints on the state of the world. We begin with this assumption in order to give an intuitive initial presentation of dynamically factored beliefs in the next chapter. Afterward, in Chapter 7, we will extend our approach to a setting where observations (both sensory ones and assertions) can be noisy.

- $T(s, a, s')$ (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 noiseless and that transitions are deterministic. In Chapter 7, we extend our approach to a setting where observations can be noisy.

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.

Chapter 5

Dynamically Factored Belief

5.1 Overview

In trying to find a suitable belief state representation for the POMDP defined in Section 4.2, 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 2.2) 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 5-1 and Section 5.4, 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*

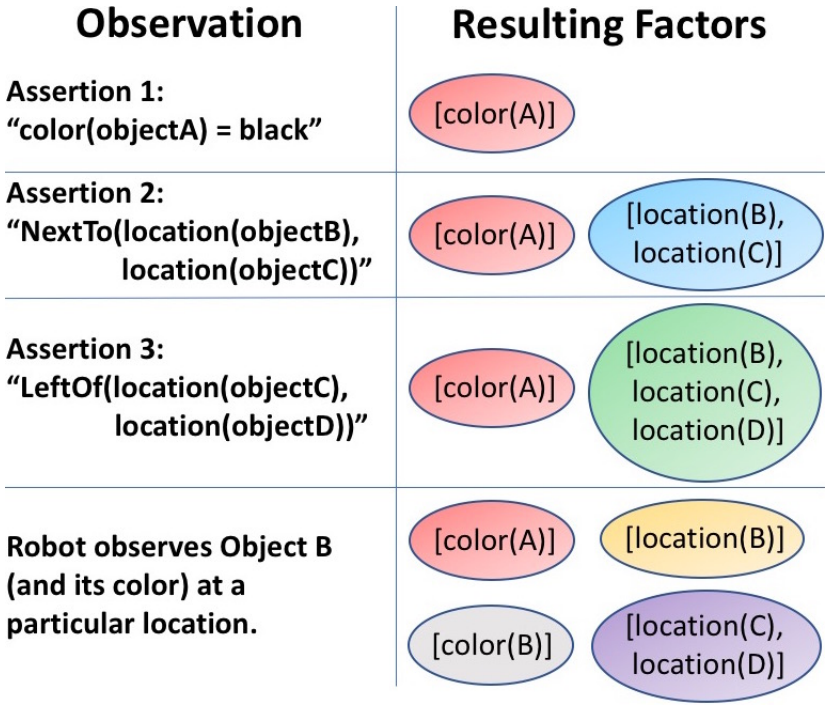


Figure 5-1: Example of dynamic factoring given four sequential observations in a setting with one object type and two properties: *color* and *location*. Each factor maps to a distribution over values (not shown). Initially, there are no factors. *Row 1:* The agent receives a single-fluent assertion 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 color and 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.

probability distribution over possible values for all its variables.

- 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 5.2.
- 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 5.3.
- No state variable can be present in more than one factor. We require this because

it simplifies enforcing consistency within the representation. See Chapter 6 for a more formal treatment of this concept.

- Factors are kept as small as possible: if a joint can be decomposed into a product, 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.

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 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. On the other hand, a fully lazy representation that stores every fluent into a database would also be inefficient, because then inference must be done by brute-force search over the state space, pruned using the constraints in this database.

5.2 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

```
   Algorithm Dynamically Factored Belief Update
1  | B ← InitializeFactoredBeliefMap()
2  | ComplexFluents ← 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 for efficiency.                    */
11 |   for each factor in B.GetFactors() do
12 |   |   B.TrySplit(factor)
```

Algorithm 1: Dynamically factored belief update.

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 (if this factor does not already exist), 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 do not 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 (i.e., during inference time).

Line 10 updates the belief based on the action taken in the environment. Note that this makes the simplifying assumption that belief updates based on environment transitions are always foldable into the current factorization.

Lines 11-12 keep the factors as small as possible, splitting up any whose distribution can be decomposed into a product. In practice, we approximate this by checking for each factor whether any state variable can be marginalized out of the joint.

5.3 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

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

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 use the determinize-and-replan approach utilize this algorithm.

5.4 Illustrative Example

We illustrate updates and inference with a simple example. Consider a planning problem from Π (Section 4.1) 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 (using Algorithm 2) be answered with the assignment {*color(A)*: green}, since the robot does not 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 (using Algorithm 2) 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.

Although this example is simple, it highlights the core idea of our approach: when the robot receives the assertion that *color(B)* = *color(C)*, it updates its belief to contain a joint distribution [*color(B), color(C)*] over the colors of both objects. The assertion is completely folded into this joint, so marginal inference can be done quickly by marginalizing out variables from this joint, and samples can be drawn quickly too. By contrast, a belief representation with a fixed factoring over singleton state variables would not be able to incorporate the assertion that *color(B)* =*color(C)*, so accurate inference would require independently sampling values for *color(B)* and *color(C)* until they agree. This can become prohibitively expensive as the domain size, the number of objects, and the number of assertions increase.

Chapter 6

A Factor Graph Perspective

A dynamically factored belief state representation can be viewed as a simplified factor graph (Section 2.4). In this chapter, we elucidate the connection and discuss some trade-offs of our approach versus the more standard factor graph algorithms. We chose to delay presenting this perspective until now because we preferred to give an intuitive explanation during the initial introduction of our approach.

Let $\langle \mathcal{T}, \mathcal{P}, \mathcal{O}, \mathcal{V}, \mathcal{F}, \mathcal{U}, \mathcal{I}, \mathcal{G} \rangle$ be an open-domain planning problem from Π (Section 4.1) for which we are using a dynamically factored belief. At any time, every state variable in $\mathcal{V}$ that the agent knows about is present in exactly one factor, per our definition. We now construct the corresponding factor graph, which has variable nodes V and factor nodes F.

The V are precisely those state variables from $\mathcal{V}$ that the agent knows about at the current timestep, and the F correspond to the different factors (defined as lists of state variables mapping to joint distributions) in the dynamically factored belief. Each node F connects to all the V that are present in the corresponding list (and thus comprise that factor's joint distribution). Because the factors partition the set of known state variables, this factor graph is actually just a collection of independent subgraphs, where each subgraph contains exactly one node from F connected to one or more nodes from V. See Figure 6-1 for an example.

An important consideration here is that for dynamically factored beliefs, the factors are constantly changing as new information is received: on every timestep, when

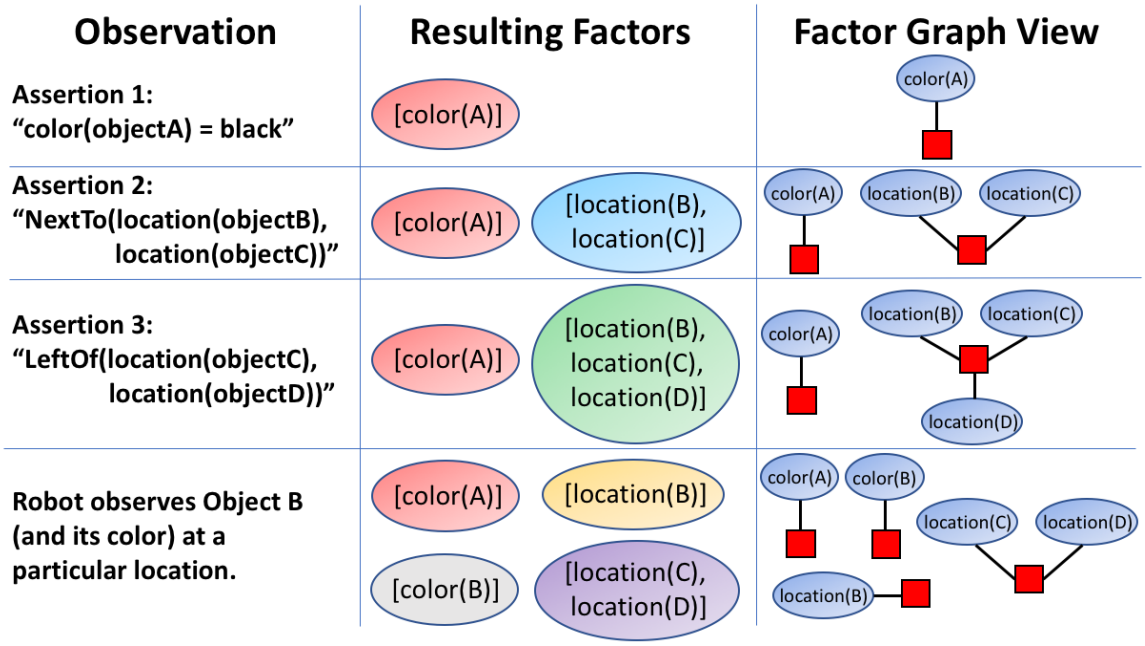


Figure 6-1: The same example as Figure 5-1, but with an extra column illustrating the factor graph representation of our approach. In the factor graph column, the blue ovals are the variable nodes V and the red squares are the factor nodes F. Each factor node (red square) corresponds to one of the ovals from the middle column, and represents a joint distribution over values for all variables connected to it. At any timestep, this factor graph will be a collection of independent subgraphs, where each subgraph contains exactly one node from F connected to one or more nodes from V. Inference is easier in this setting than in a general factor graph.

the robot receives an observation, it may join some factors together and split up other ones. This of course alters the structure of the underlying factor graph: although it is always a collection of independent subgraphs, connections between factor nodes and variable nodes are constantly getting added and deleted. Furthermore, new variable nodes are added to the graph whenever the agent discovers new state variables.

This constantly changing structure has been previously explored in work on *dynamic factor graphs* [17, 11], which extend the classical notion of a fixed factor graph to the sequential setting. Similar to a dynamic Bayesian network, the idea is to augment the nodes with a temporal dimension, so that correlations across one or multiple timesteps can be captured by the model. See Figure 6-2 for an illustration. Of course, this makes inference significantly more costly, as there are now more variables in the factor graph. On the other hand, the factor graph corresponding to a dynamically

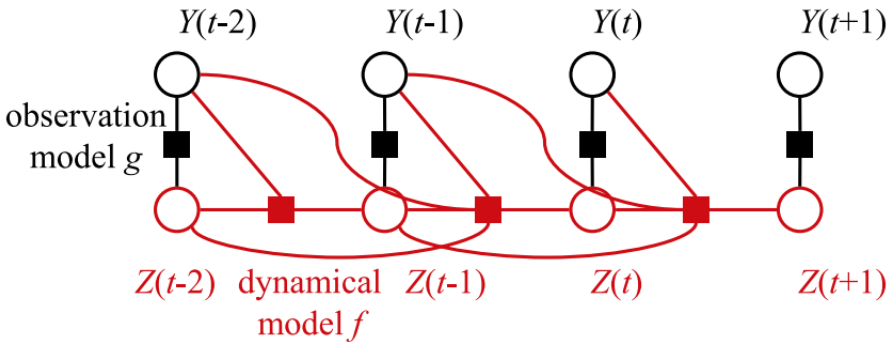


Figure 6-2: An illustration of a dynamic factor graph, taken from Mirowski and LeCun [17]. Here, Y are the observed variables, Z are the latent (unobserved) variables on which we would like to do inference, and the squares are factor nodes. In this model, the dynamics depend on the previous two timesteps. One could also imagine a model obeying the Markov property, which would only require connections between variables for consecutive timesteps.

factored belief does not need to incorporate this temporal aspect, because in Line 10 of Algorithm 1 we have made the simplifying assumption that belief updates based on environment transitions are always foldable into the current factorization, and so our model does not need to contain variables for previous timesteps.

The simplified factor graph representation induced by our approach has certain computational advantages over a more standard factor graph approach. A standard approach would, for every received observation, add a new factor to the graph for each fluent in the observation, and connect this factor to all variables mentioned by the fluent. Other factors would not be affected by this addition, so the factor graph would *not* be merely a collection of independent subgraphs, but rather each variable could be connected to multiple factors. Though it is easy to update such a model with new observations, inference can be expensive: computing marginals requires running a message-passing algorithm and becomes especially difficult if the factor graph becomes cyclic.

By contrast, consider the factor graph induced by our approach. At inference time, we do not need to run any message-passing algorithm: since every variable is connected to at most one factor, the marginal of any variable can be computed by looking at the corresponding factor and directly marginalizing out other connected variables.

Additionally, there is no possibility of ending up with a cyclic factor graph, which would typically necessitate either approximate inference or expensive algorithms for exact inference, as discussed in Section 2.4.

Of course, these benefits come at the cost of needing to do more work at update time: in order to maintain the invariant that each state variable is connected to exactly one factor, potentially larger factors must be computed and stored. To address this drawback, one could imagine splitting factors up more aggressively at the cost of exactness; we explore this idea in the next chapter, where we discuss handling noisy observations.

Chapter 7

Handling Noisy Observations

Thus far in this work, we have assumed that observations received by the agent, whether they be from its own perceptual capabilities or from assertions, are noiseless. This means these observations can be interpreted as Boolean-valued constraints that describe (with certainty) a fact about the world state. The noiseless observation setting simplifies the process of updating the agent's belief given an observation – this process can be thought of as applying a filter on the set of possible world states, removing some from contention because they are inconsistent with the observation. This characterization holds for both discrete and continuous domains.

This noiseless observation setting is, of course, unrealistic. In the real world, sensors are noisy and humans are uncertain about their own remarks; we would expect our inference algorithms to be generally robust to these sources of noise. In a realistic use case of our system, for instance, the human should be able to pair an assertion with a confidence level describing his or her certainty about the veracity of the assertion. In this chapter, we explore what changes we must make to our belief update algorithm, Algorithm 1, in order to support noisy observations. Examples of such observations may include:

- A human-provided assertion that there is a 75% chance that Object A and Object B are the same color.
- An observation that Object A is red, made by the robot's camera which has 95% color detection accuracy.

The new techniques we develop in this chapter will allow dynamically factored beliefs to support these types of noisy observations.

7.1 Problem Setting

In order to incorporate noise into the observations, we must make a small change to the formal problem setting described in Chapter 4. Namely, we change the definition of an observation in the POMDP formulation of Section 4.2. Recall that in the noiseless setting, an observation is a conjunction of fluents from $\mathcal{F}$ that hold in the current state (with certainty). In the noisy setting, we redefine an observation to include associated probabilities that the constituent fluents hold.

Definition 4 *A (potentially noisy) observation in our* POMDP *is a set of pairs* $\langle f, p \rangle$*, where* $f \in \mathcal{F}$ *and* $p \in (0, 1]$*. Intuitively, the interpretation of an observation is that each fluent* f *in this set holds with probability* p *in the current world state.*

Note that when p is always 1, the noiseless setting is recovered.

As a result of this change, we will need to make revisions to Algorithm 1, which updates the dynamically factored belief on each timestep given the taken action and the received observation. Our techniques for inference (Section 5.3, Algorithm 2) are unaffected by this change, since they only depend on the computed belief representation, not on how it is updated.

7.2 Algorithms

To support noisy observations, we must change our implementation of two methods in Algorithm 1: JOINFACTORSANDUPDATE(FLUENT) at Line 9 and TRYSPLIT(FACTOR) at Line 12. The former is called on every constituent fluent of the observation, and the latter is called on every factor as a post-update step that helps maintain a compact representation.

7.2.1 Updating with a Fluent

Recall that in the noiseless setting, JoinFactorsAndUpdate(fluent) creates a new factor containing all state variables mentioned by the fluent (if such a factor is not already present), then maps this factor to a joint distribution in which the fluent holds. For the discrete (resp. continuous) setting, we perform these steps by explicitly (resp. implicitly) building the joint then filtering out values that are inconsistent with the fluent.

Note that in the continuous setting, this filtering occurs implicitly at sample time, since we eschew trying to calculate posteriors exactly in favor of rejection sampling. The algorithmic adaptation that we present applies for discrete distributions but extends naturally to the continuous, implicit setting as well.

The core idea is to build the joint then rescale the probabilities such that p mass goes to the joint values which are consistent with the fluent, and the remaining $1-p$ mass goes to those which are inconsistent. Some algebra shows that this can be achieved by rescaling the probability of all values inconsistent with the fluent by $\frac{(1-p)(1-m)}{pm}$, where m is the total probability of these inconsistent values, then normalizing. Observe that when $p = 1$, this just boils down to filtering out joint values inconsistent with the fluent, as expected.

Algorithm 3 shows JoinFactorsAndUpdate for the noiseless setting where $p = 1$, while Algorithm 4 shows it for the noisy setting where $0 < p < 1$. In both cases, Line 1 first builds a full joint distribution that does not incorporate the fluent. Afterward, the filtering or rescaling occurs. See Figure 7-1 for a simple example of how different settings for p affect the posterior that results from running this algorithm.

7.2.2 Maintaining a Compact Representation

We now explore how to adapt TrySplit(factor) to the noisy observation setting. This method is an important part of our algorithm, as it helps the belief representation remain compact. Recall that in the noiseless setting, our approach checks whether any state variable mentioned in the factor can be marginalized out of

Algorithm *Updating with a fluent for* $p = 1$

Subroutine JoinFactorsAndUpdate*(B, fluent)*

```
1  joint ← (join factors F containing any state variable present in fluent)
2  for each value in joint do
3      if value is inconsistent with fluent then
4          joint.SetProbability(value, 0)
5  joint.Normalize()
6  Add joint to dynamically factored belief B and remove all F from B.
```

Algorithm 3: JoinFactorsAndUpdate algorithm for $p = 1$. Joint values that are inconsistent with the fluent are simply filtered out. This was our previous approach, but it does not work in the noisy observation setting. See Algorithm 4 for an adaptation to the noisy observation setting.

Algorithm *Updating with a fluent for* $0 < p < 1$

Subroutine JoinFactorsAndUpdate*(B, fluent, p)*

```
1  joint ← (join factors F containing any state variable present in fluent)
2  m ← (total probability of values in joint inconsistent with fluent)
3  for each value in joint do
4      if value is inconsistent with fluent then
5          joint.RescaleProbabilityBy(value, (1−p)(1−m) / pm)
6  joint.Normalize()
7  Add joint to dynamically factored belief B and remove all F from B.
```

Algorithm 4: JoinFactorsAndUpdate algorithm for $0 < p < 1$. Probabilities for the inconsistent values are rescaled so that after normalization, joint values consistent with the fluent have total probability p. This is an adaptation of Algorithm 3 to the noisy setting.

the joint.

In the noisy observation setting, however, it is unlikely that such marginalization can ever be done in a lossless manner, as this would require the joint to be exactly decomposable into a product involving this marginal. Instead, we allow ourselves to have some amount of approximation error and perform this marginalization whenever the reconstruction error is not too high. We measure this reconstruction error as the Jensen-Shannon divergence between the true joint and the approximate joint which is reconstructed from the attempted decomposition.

Let P and Q be arbitrary probability distributions. The Jensen-Shannon divergence is a smooth, symmetric, bounded measure of similarity between P and Q. It

Algorithm *Maintaining a compact representation for* $p = 1$

```
    Subroutine TrySplit(B, factor)
1       for each state variable V in factor do
2           if B[factor] decomposes into B[V] and B[factor \ {V}] then
3               Marginalize out V from B[factor].
```

Algorithm 5: TrySplit algorithm for $p = 1$. Exact marginalization is performed when possible. This was our previous approach, but it does not work in the noisy observation setting where exact decomposition is generally unlikely. See Algorithm 6 for an adaptation to the noisy observation setting.

Algorithm *Maintaining a compact representation for* $0 < p < 1$

```
    Subroutine TrySplit(B, factor, error_threshold)
1       for each state variable V in factor do
2           reconstructed ← Join(B[V], B[factor \ {V}])
3           if D_JS(B[factor] ‖ reconstructed) < error_threshold then
4               Marginalize out V from B[factor].
```

Algorithm 6: TrySplit algorithm for $0 < p < 1$. Marginalization is performed when possible, so long as the reconstruction error (measured by the Jensen-Shannon divergence) does not exceed the given threshold. Varying this threshold allows the designer to trade off between compactness of the belief representation and accuracy of inference. This is an adaptation of Algorithm 5 to the noisy setting.

is based on the Kullback-Leibler (KL) divergence, defined as

$$D_{KL}(P \parallel Q) = -\sum_i P(i) \log \frac{Q(i)}{P(i)},$$

where the summation can be replaced by integration for continuous distributions. Then letting $A = \frac{1}{2}(P + Q)$, the Jensen-Shannon divergence is defined as

$$D_{JS}(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel A) + \frac{1}{2} D_{KL}(Q \parallel A).$$

Assuming the natural logarithm is used, the bound $0 \leq D_{JS}(P \parallel Q) \leq \log 2$ always holds. Because our reconstruction error is bounded, it is reasonable to use a simple threshold on the Jensen-Shannon divergence to decide whether or not a marginalization operation would be too lossy to perform. Varying this threshold allows the designer to trade off between compactness of the belief representation and accuracy

of inference. We leave further exploration of a better decision boundary to future work.

Algorithm 5 shows TrySplit for the noiseless setting where $p = 1$ (only exact marginalization is performed), while Algorithm 6 shows it for the noisy setting where $0 < p < 1$. The latter algorithm (for the noisy setting) is based on the preceding discussion. See Figure 7-2 for an illustration of the idea behind this algorithm.

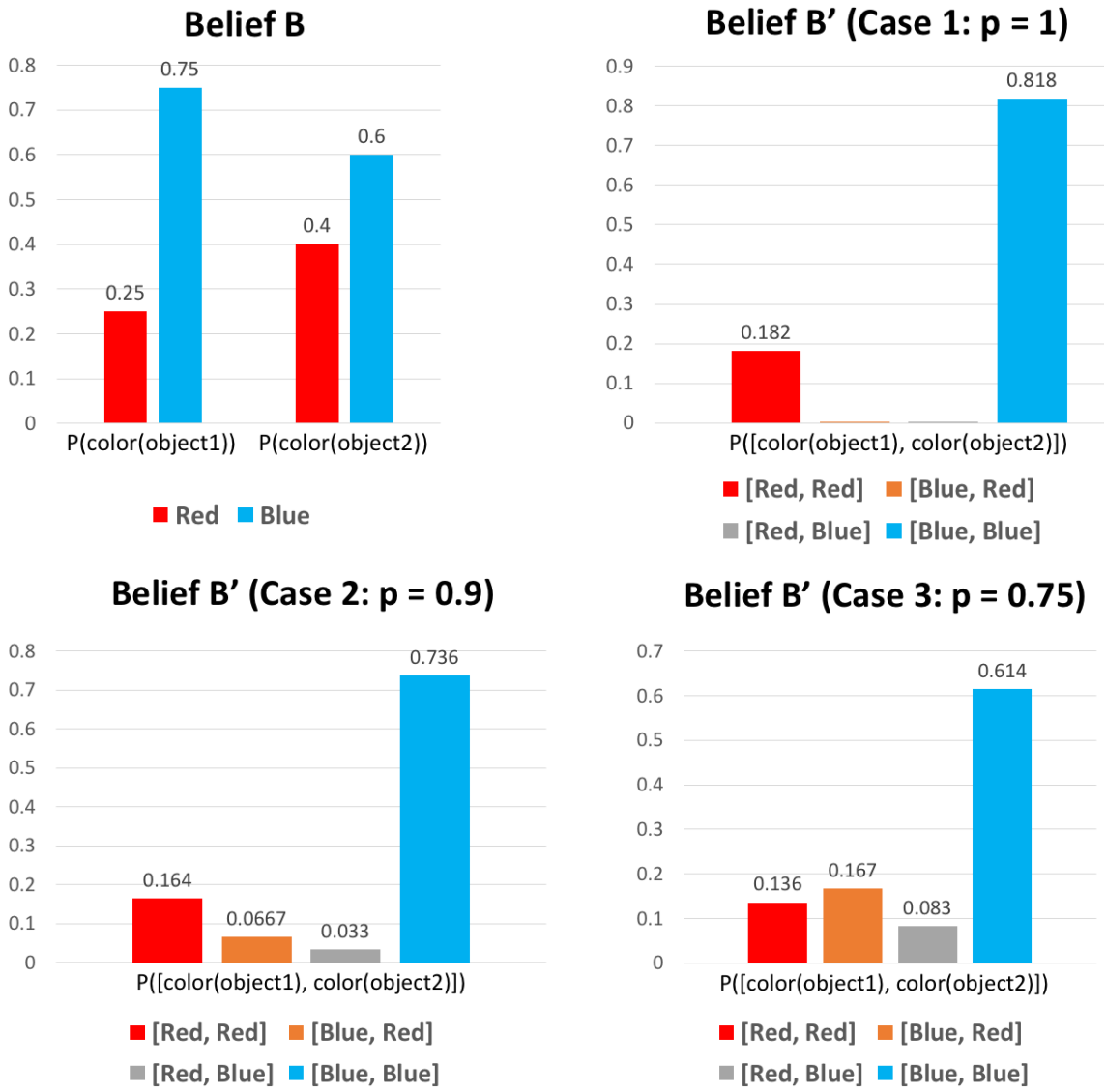


Figure 7-1: Various posteriors resulting from running JoinFactorsAndUpdate with the fluent *Same(color(object1), color(object2))* and different values of p. The top-left image shows the initial belief, while the other images show the updated belief for three different values of p: 1, 0.9, and 0.75. Observe that the values of the distribution always get scaled so that the total probability of the joint values which are consistent with the fluent, namely [*Red, Red*] and [*Blue, Blue*], is p.

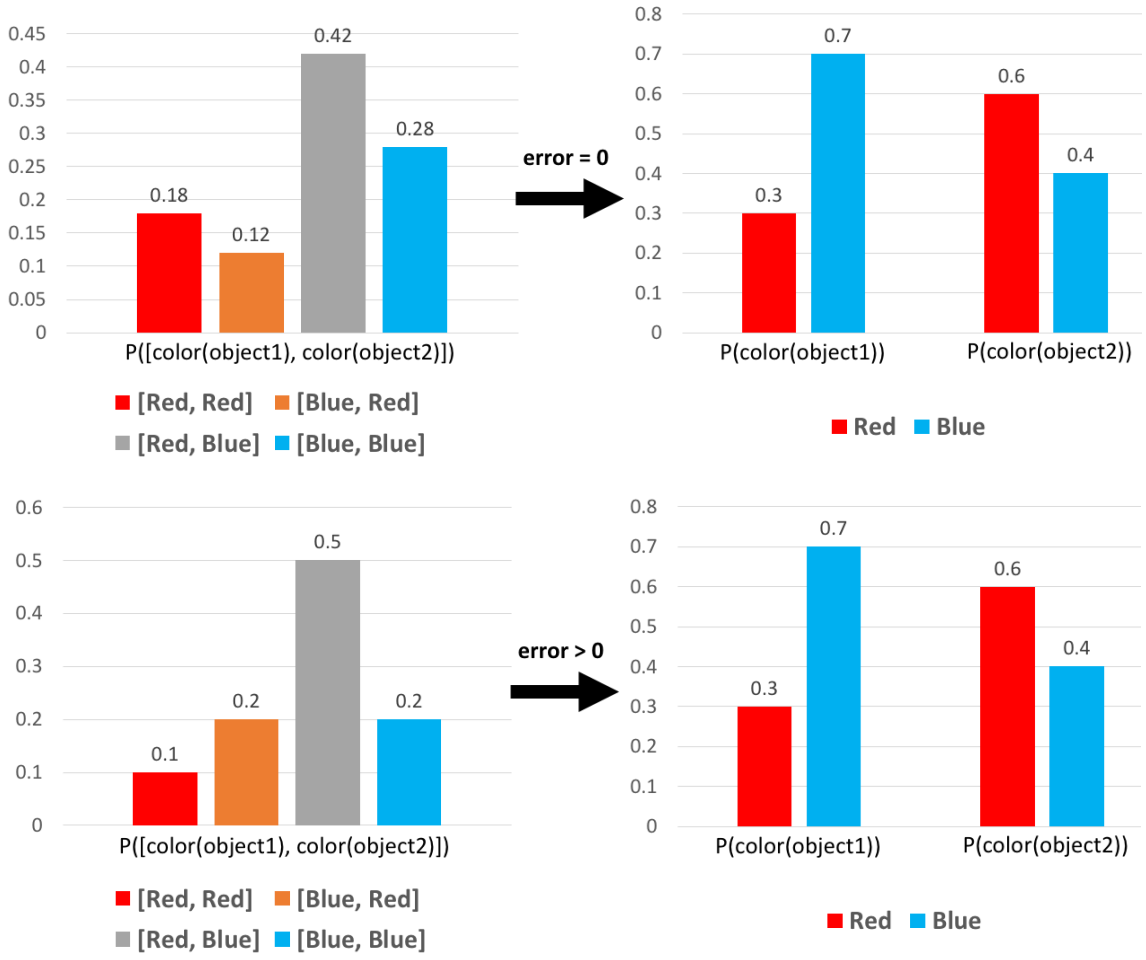


Figure 7-2: Two examples of trying to split up the factor [*color(object1)*, *color(object2)*]. *Top:* The two random variables *color(object1)* and *color(object2)* are independent, so the joint can be split into two marginals without incurring any error. *Bottom:* The random variables are not independent, so splitting into the same two marginals incurs an error based on the Jensen-Shannon divergence between the joint and the product of the marginals (about 0.01 for this example). Algorithm 6 decides whether to perform such marginalization based on the given error threshold. In the noisy observation setting, lossless marginalization will almost never be possible.

Chapter 8

Experiments

We evaluate the performance of our approach experimentally using the cooking task, a planning problem from Π (Section 4.1). The robot is tasked with gathering ingredients and using them to cook a meal. There are three object types: $\mathcal{T} = \{\textit{locations}, \textit{vegetables}, \textit{seasonings}\}$. 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

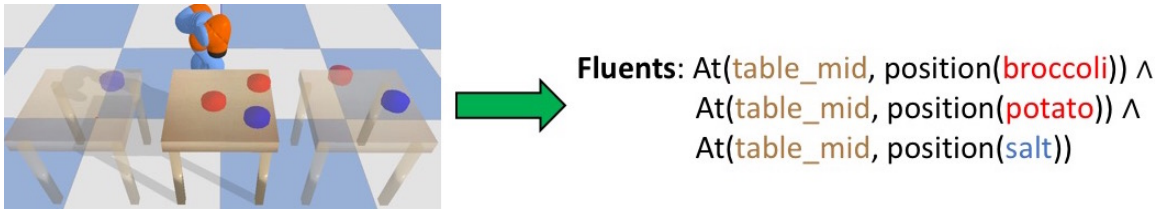


Figure 8-1: A visualization of how an observation by the robot is converted into a conjunction of fluents in the continuous 3D cooking task. The robot observes the middle table and learns the set of ingredients located on it. For assertions, we assume they are already represented as fluents when given to the robot.

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 8-1 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.

Assertions. We assume that assertions are already represented in formal language. Figure 8-2 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 on the world state; the more knowledge the agent has about the world, the more accurately 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. Note that in this experiment, we assume all observations (including assertions) are accurate and noiseless. In Section 8.4, we show results for our experiments with noisy observations.

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.

8.1 Baseline Approach

We test against a baseline belief representation that is meant to simulate prior work on factored representations, which typically commit to a representational choice at the start. This baseline, which we refer to henceforth as a *statically factored belief* or *static factoring*, represents the agent's belief as a distribution over the potential contents of each location. This factoring does not change based on observations, or as ingredients are discovered. We choose this particular factoring (over each location's contents) as our baseline because initially, the robot only knows the set of locations, not the set of ingredients. Thus, this baseline is the most reasonable choice for a static belief representation that can be chosen and held fixed at the start of execution. Any fluent that cannot fold into this static factoring is placed into ComplexFluents and considered only at inference time. Recall that Section 5.3 describes how ComplexFluents is used during inference.

One perspective on the difference between statically factored beliefs and our method, dynamically factored beliefs, is that they represent two different degrees of laziness during belief updates, with respect to the proportion of fluents stored lazily in

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)]	✓	✓

Figure 8-2: 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(·) means *contents(·)*, with domain {vegetable, seasoning, empty}. p(·) means *position(·)*, 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 observation; 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 factoring that tracks the potential contents of each location (baseline approach). 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 either a dynamically factored belief or a statically factored belief.

ComplexFluents versus eagerly incorporated into the probability distributions. The statically factored belief only tracks a distribution over the potential contents of each location, so information that (for instance) links the contents of two locations would be placed into ComplexFluents under this representation. On the other hand, our

approach would adapt the structure of the factoring so that the new information can be folded in without needing to be remembered.

One could also imagine belief state representations lying at the two extremes of this "laziness" spectrum. At one end, a fully lazy representation would, on each belief update with an observation o, simply store every fluent in o into the ComplexFluents database. Belief updates with this representation are trivial, but inference must be done by brute-force search over the state space, pruned using the constraints in ComplexFluents. At the other end of the spectrum would be a representation that stores a single full joint over values for all currently known state variables. This eliminates the need for ComplexFluents, since every observation can be folded into such a distribution, but it is prohibitively expensive to store and update.

8.2 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 the *position* property of each ingredient is a discrete value, corresponding to some 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 8-2. Factors for the ingredient positions are added in only as the agent learns about the ingredients in the world, either through its own observations or through assertions.

8.2.1 Preliminary Experiment

We first demonstrate the strength of our system at accurately answering a series of factor-specific queries which require it to draw on the information given to it through assertions. To show this, we disable ComplexFluents for this preliminary experiment only, meaning any assertion that cannot be folded into the belief representation eagerly (whether dynamically or statically factored) is ignored. So, this experiment investigates the usefulness of the information contained in assertions toward producing lower-cost execution traces.

We ask the agent to follow a conditional plan that solves the gridworld cooking task. At branches in the plan, the belief state is queried for information about the contents of 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. Since a dynamically factored belief is able to fold in a higher proportion of assertions eagerly than the baseline, we would expect inference accuracy to be higher, meaning that the system is able to answer more of these queries accurately.

Tables 8.1 and 8.2 show our quantitative results for this task, with each row averaging over 100 randomly generated independent problem instances. The results are also visualized in Figure 8-3.

Discussion. Using a dynamically factored belief state representation produces consistently lower-cost execution traces than using a statically factored one, when ComplexFluents is disabled. This difference stems from the ability of our method to dynamically adapt to the logical form of assertions, meaning it can fold in a higher proportion eagerly. This does, of course, come with a trade-off of more expensive belief updates, which is expected because our approach runs Algorithm 1 in the update phase. However, the data suggests that this trade-off is net-positive. Though our approach can never produce higher-cost execution traces than the baseline (since it can never incorporate *fewer* assertions than the baseline does), it could perform

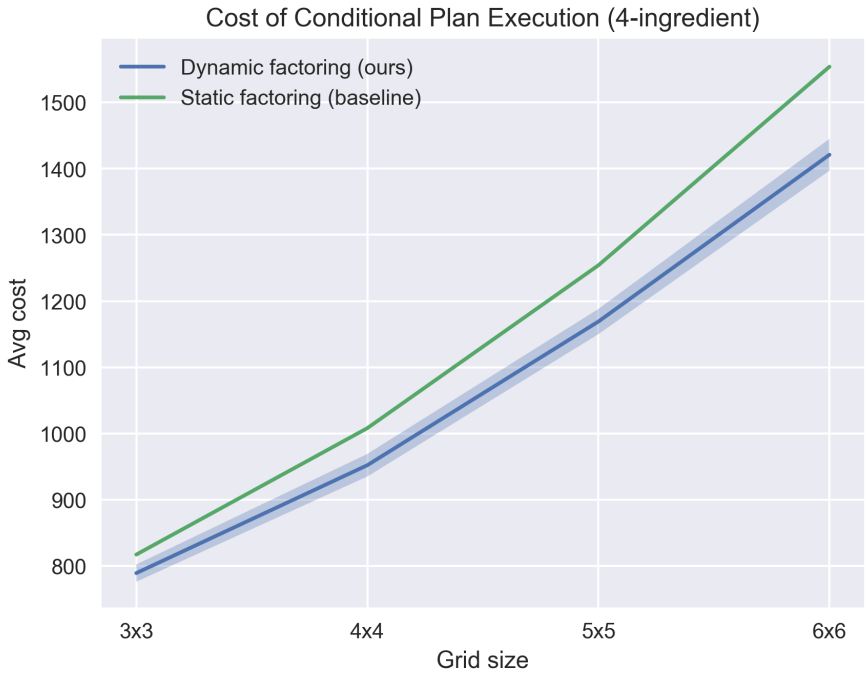
Setting	System	Bel. Upd. Time	Avg. Cost (SD)
3x3, 4ing	S (baseline)	0.01	817 (2)
3x3, 4ing	D (ours)	0.01	**789** (13)
3x3, 6ing	S (baseline)	0.01	949 (2)
3x3, 6ing	D (ours)	0.02	**920** (13)
3x3, 8ing	S (baseline)	0.01	1080 (5)
3x3, 8ing	D (ours)	0.02	**1052** (10)
4x4, 4ing	S (baseline)	0.01	1008 (2)
4x4, 4ing	D (ours)	0.02	**952** (17)
4x4, 6ing	S (baseline)	0.01	1140 (2)
4x4, 6ing	D (ours)	0.03	**1082** (17)
4x4, 8ing	S (baseline)	0.01	1272 (10)
4x4, 8ing	D (ours)	0.03	**1216** (19)
4x4, 10ing	S (baseline)	0.01	1389 (24)
4x4, 10ing	D (ours)	0.04	**1344** (21)

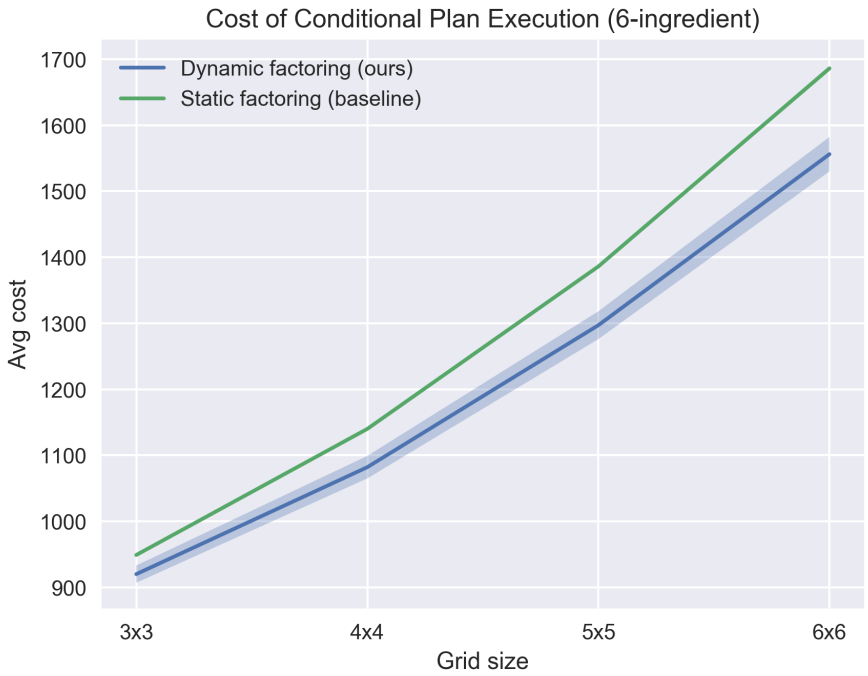
Table 8.1: Conditional plan execution costs for the 2D gridworld cooking task, for grid sizes 3x3 and 4x4. Belief update time (seconds) and execution cost are shown. Each row reports averages over 100 independent random trials. S: Statically factored belief (baseline), D: Dynamically factored belief (our method). *Setting* column gives (grid size, number of ingredients). See Figure 8-3 for a visualization of this data. Note that grid size 3x3 cannot fit 10 ingredients, so those rows are omitted. Our approach leads to lower-cost execution traces than a static factoring does, when ComplexFluents is disabled.

worse if the increase in belief update time or storage requirements dominates the improvement in inference time. This would happen if, for instance, an assertion were given that links several state variables having large domains, so that the resulting joint is extremely expensive to both compute and store in memory.

Setting	System	Bel. Upd. Time	Avg. Cost (SD)
5x5, 4ing	S (baseline)	0.01	1254 (3)
5x5, 4ing	D (ours)	0.04	**1169** (19)
5x5, 6ing	S (baseline)	0.01	1386 (3)
5x5, 6ing	D (ours)	0.1	**1297** (21)
5x5, 8ing	S (baseline)	0.01	1518 (20)
5x5, 8ing	D (ours)	0.1	**1426** (23)
5x5, 10ing	S (baseline)	0.02	1616 (48)
5x5, 10ing	D (ours)	0.12	**1553** (28)
6x6, 4ing	S (baseline)	0.01	1554 (3)
6x6, 4ing	D (ours)	0.07	**1421** (24)
6x6, 6ing	S (baseline)	0.01	1686 (4)
6x6, 6ing	D (ours)	0.15	**1556** (24)
6x6, 8ing	S (baseline)	0.03	1819 (30)
6x6, 8ing	D (ours)	0.24	**1684** (23)
6x6, 10ing	S (baseline)	0.03	1891 (52)
6x6, 10ing	D (ours)	0.26	**1788** (54)

Table 8.2: Conditional plan execution costs for the 2D gridworld cooking task, for grid sizes 5x5 and 6x6. Belief update time (seconds) and execution cost are shown. Each row reports averages over 100 independent random trials. S: Statically factored belief (baseline), D: Dynamically factored belief (our method). *Setting* column gives (grid size, number of ingredients). See Figure 8-3 for a visualization of this data. Our approach leads to lower-cost execution traces than a static factoring does, when ComplexFluents is disabled.


Cost of Conditional Plan Execution (4-ingredient)
Dynamic factoring (ours)
Static factoring (baseline)
Avg cost
1500
1400
1300
1200
1100
1000
900
800
3x3
4x4
5x5
6x6
Grid size


Cost of Conditional Plan Execution (6-ingredient)
Dynamic factoring (ours)
Static factoring (baseline)
Avg cost
1700
1600
1500
1400
1300
1200
1100
1000
900
3x3
4x4
5x5
6x6
Grid size

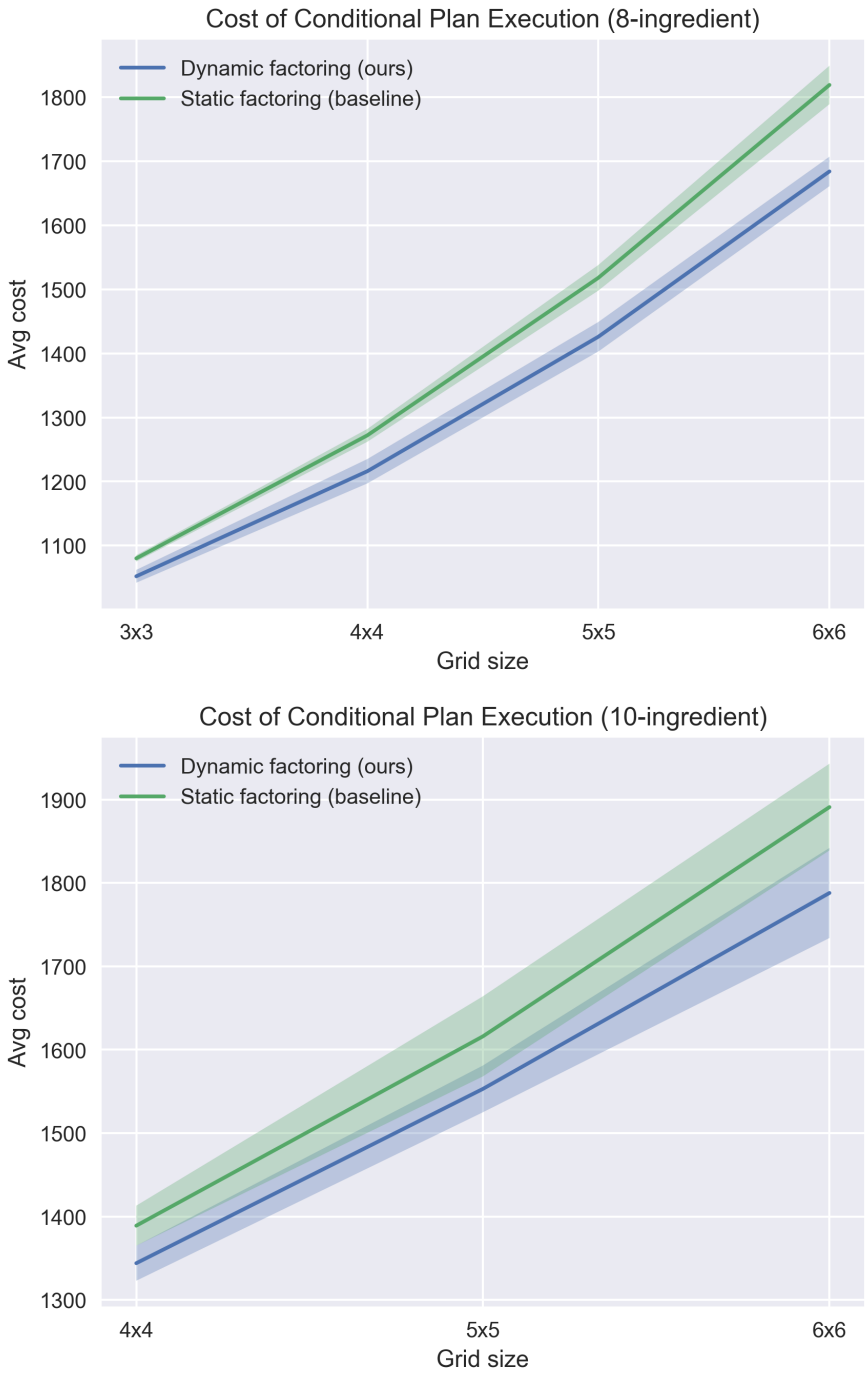


Figure 8-3: Visualizations of the results in Tables 8.1 and 8.2. Our approach leads to lower-cost execution traces than a static factoring does, when ComplexFluents is disabled. This is because it is able to answer queries more accurately, at the cost of slightly more expensive belief updates (see tables).

8.2.2 Main Results

We now demonstrate our main result: the ability of dynamically factored beliefs to quickly and accurately answer queries about global state. For this experiment, we solve the gridworld cooking task using the determinize-and-replan approach to belief space planning (Section 2.3). The determinization is produced using Algorithm 2, which draws a sample from the full joint of current factors: this is a determinization of the world state per the agent's current knowledge. Note that state variables not yet discovered by the agent will not be present in this determinization, as expected. Planning is done using A*. Because this is a computationally challenging task, we impose a 60-second timeout on every call to Algorithm 2, and if the timer expires, we consider that problem instance to be unsolved.

Tables 8.3 and 8.4 show our quantitative results for this task, with each row averaging over 100 randomly generated independent problem instances. The results are also visualized in Figure 8-4, Figure 8-5, and Figure 8-6.

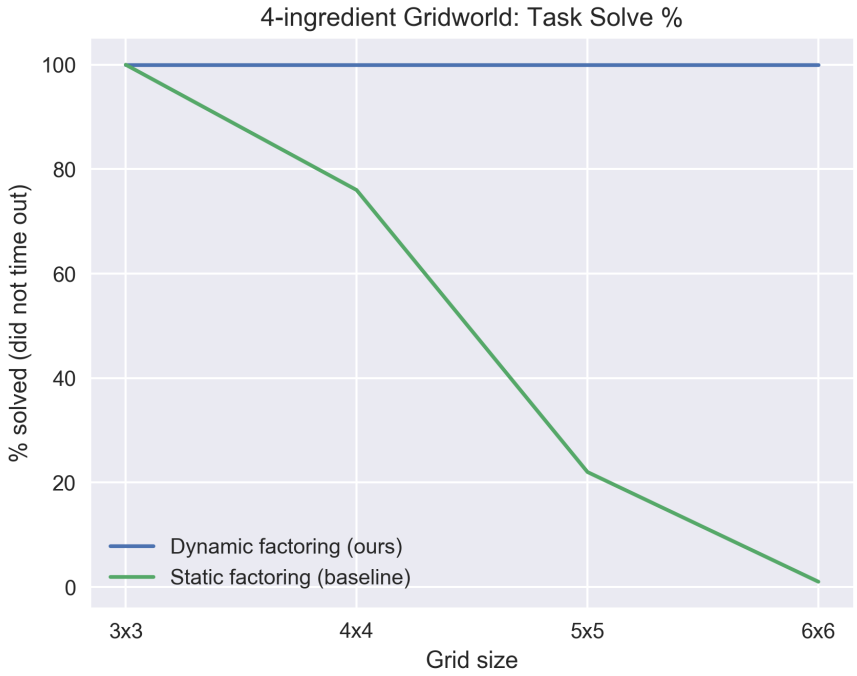
Discussion. Our approach provides significant improvements over the baseline in the percentage of problems that get solved, and also does inference an order of magnitude faster. This improvement stems from the fact that we eagerly incorporate assertions into the belief state more frequently, rather than storing them into ComplexFluents. Thus, our approach leads to fewer timeouts when running Algorithm 2 than the baseline does, and queries are answered more quickly on average as well. As in the previous experiment, this improved performance comes at the cost of more expensive belief updates due to running Algorithm 1, but the data suggests that the trade-off is net-positive. Our approach could perform worse, however, if the increase in belief update time or storage requirements dominates the improvement in inference time. This would happen if, for instance, an assertion were given that links several state variables having large domains, so that the resulting joint is extremely expensive to both compute and store in memory. Typically, though, the designer would choose the joint size cutoff for storing fluents into ComplexFluents appropriately, so that these large distributions never need to be represented.

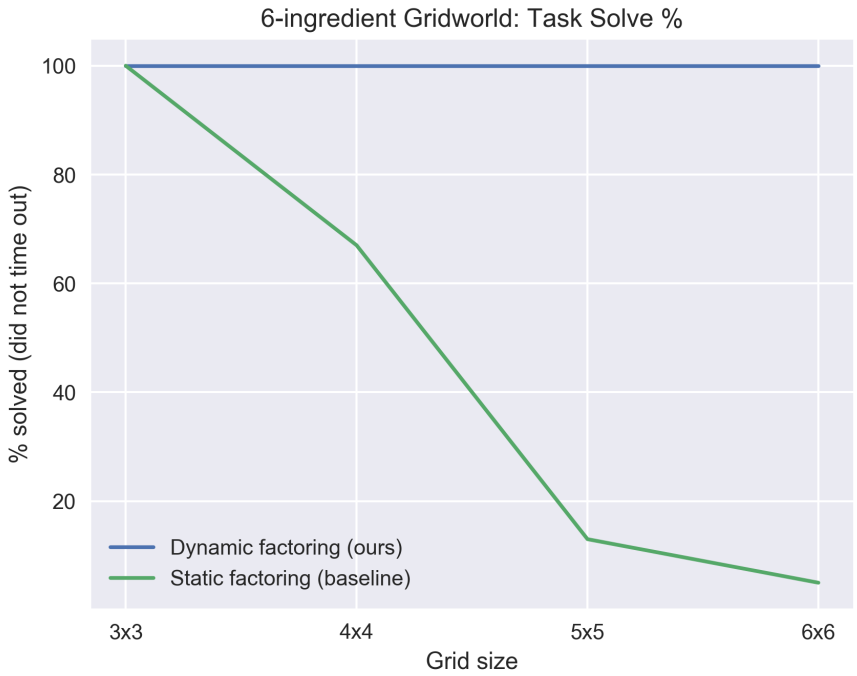
Setting	System	% Solved	Bel. Upd. Time	Queries/Second
3x3, 4ing	S (baseline)	100	0.01	3.13
3x3, 4ing	D (ours)	100	0.01	**50**
3x3, 6ing	S (baseline)	100	0.01	2.78
3x3, 6ing	D (ours)	100	0.02	**50**
3x3, 8ing	S (baseline)	100	0.01	2.04
3x3, 8ing	D (ours)	100	0.02	**50**
4x4, 4ing	S (baseline)	76	0.01	0.138
4x4, 4ing	D (ours)	**100**	0.03	**20**
4x4, 6ing	S (baseline)	67	0.01	0.11
4x4, 6ing	D (ours)	**100**	0.03	**20**
4x4, 8ing	S (baseline)	50	0.01	0.049
4x4, 8ing	D (ours)	**100**	0.04	**16.7**
4x4, 10ing	S (baseline)	50	0.01	0.068
4x4, 10ing	D (ours)	**100**	0.04	**16.7**

Table 8.3: Main results for belief space planning in the 2D gridworld cooking task, for grid sizes 3x3 and 4x4. Percentage of tasks solved, belief update time (seconds), and average number of queries answered per second (across only the solved problem instances) are shown. Each row reports averages over 100 independent random trials. S: Statically factored belief (baseline), D: Dynamically factored belief (our method). *Setting* column gives (grid size, number of ingredients). See Figure 8-4 and Figure 8-5 for visualizations of this data. Note that grid size 3x3 cannot fit 10 ingredients, so those rows are omitted. Our approach leads to more tasks being solved than a static factoring does, and performs inference an order of magnitude faster.

Setting	System	% Solved	Bel. Upd. Time	Queries/Second
5x5, 4ing	S (baseline)	22	0.01	0.044
5x5, 4ing	D (ours)	**100**	0.05	**8.33**
5x5, 6ing	S (baseline)	13	0.01	0.039
5x5, 6ing	D (ours)	**100**	0.06	**2.27**
5x5, 8ing	S (baseline)	9	0.01	0.025
5x5, 8ing	D (ours)	**100**	0.08	**2.33**
5x5, 10ing	S (baseline)	5	0.01	0.031
5x5, 10ing	D (ours)	**99**	0.08	**2.3**
6x6, 4ing	S (baseline)	1	0.01	0.02
6x6, 4ing	D (ours)	**100**	0.08	**3.03**
6x6, 6ing	S (baseline)	5	0.01	0.019
6x6, 6ing	D (ours)	**100**	0.13	**1.15**
6x6, 8ing	S (baseline)	5	0.01	0.0238
6x6, 8ing	D (ours)	**100**	0.19	**0.585**
6x6, 10ing	S (baseline)	5	0.01	0.028
6x6, 10ing	D (ours)	**97**	0.23	**0.338**

Table 8.4: Main results for belief space planning in the 2D gridworld cooking task, for grid sizes 5x5 and 6x6. Percentage of tasks solved, belief update time (seconds), and average number of queries answered per second (across only the solved problem instances) are shown. Each row reports averages over 100 independent random trials. S: Statically factored belief (baseline), D: Dynamically factored belief (our method). *Setting* column gives (grid size, number of ingredients). See Figure 8-4 and Figure 8-5 for visualizations of this data. Our approach leads to more tasks being solved than a static factoring does, and performs inference an order of magnitude faster.


4-ingredient Gridworld: Task Solve %
% solved (did not time out)
100
80
60
40
20
0
Dynamic factoring (ours)
Static factoring (baseline)
3x3
4x4
5x5
6x6
Grid size


6-ingredient Gridworld: Task Solve %
% solved (did not time out)
100
80
60
40
20
Dynamic factoring (ours)
Static factoring (baseline)
3x3
4x4
5x5
6x6
Grid size

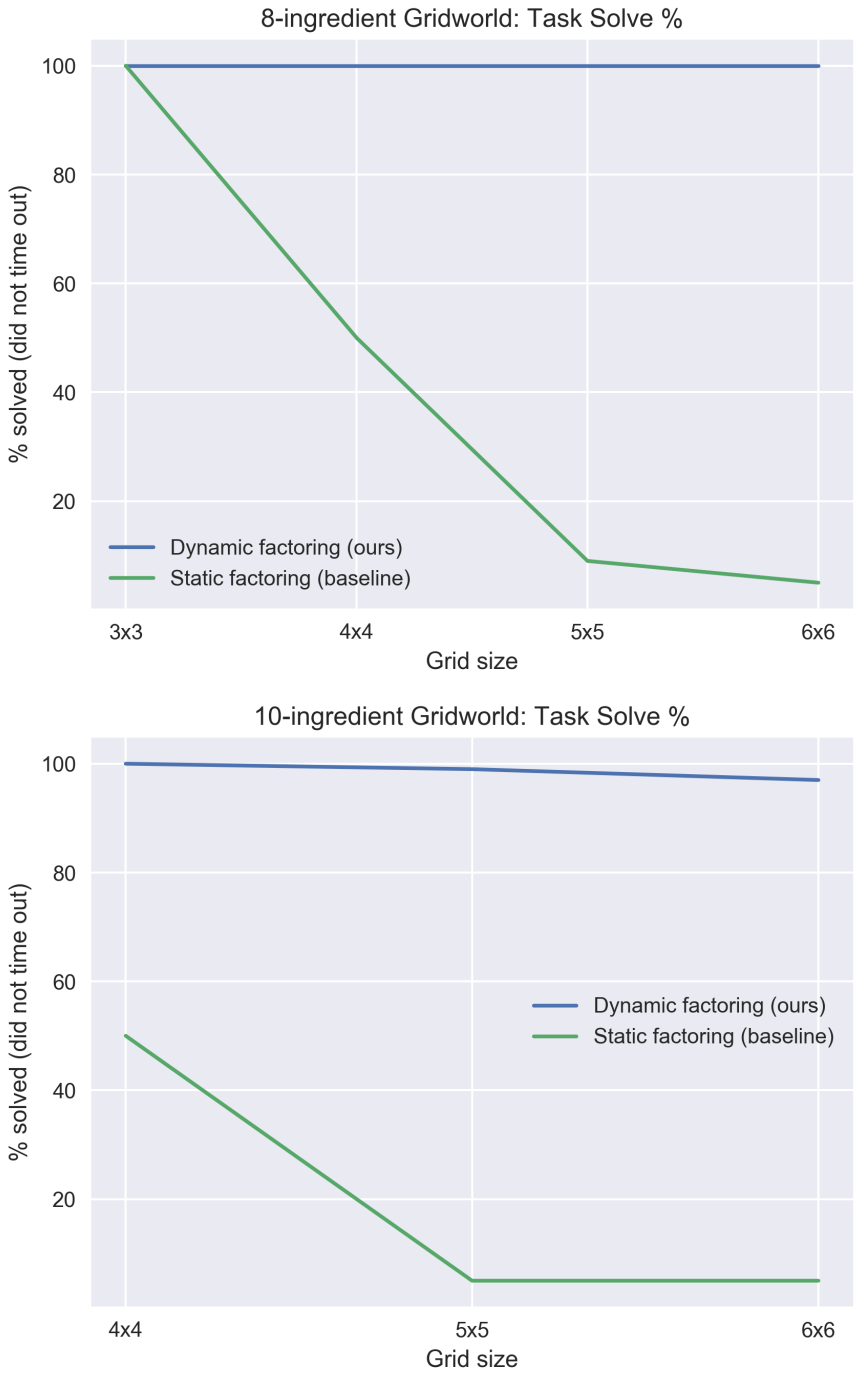
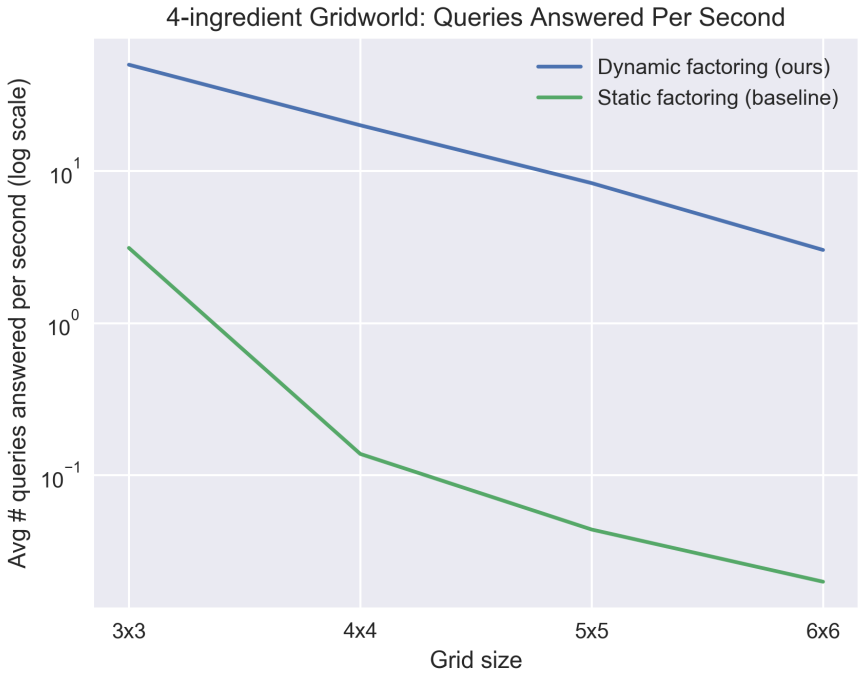
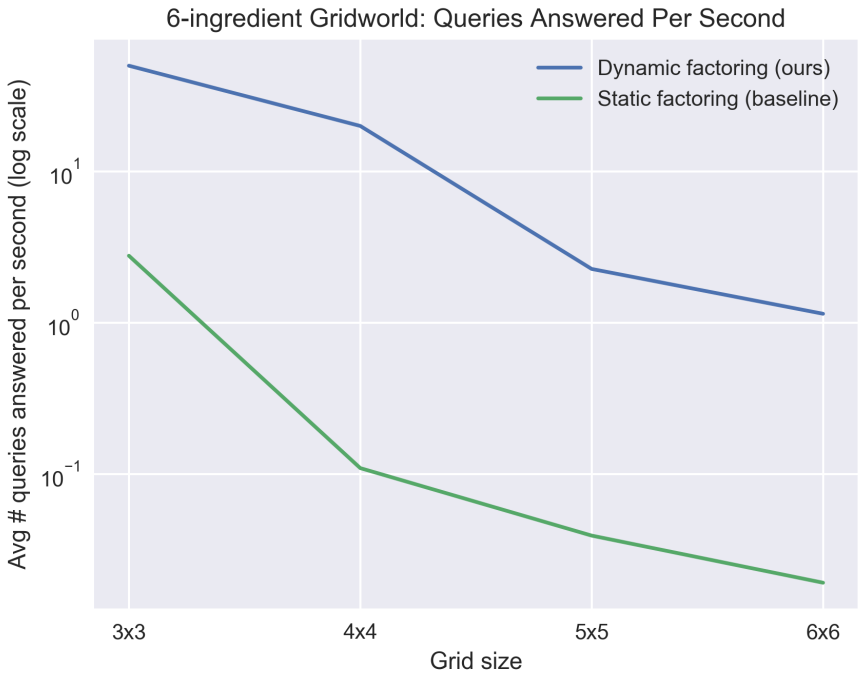


Figure 8-4: Visualizations of the percent solved results in Tables 8.3 and 8.4.


4-ingredient Gridworld: Queries Answered Per Second
Dynamic factoring (ours)
Static factoring (baseline)
Avg # queries answered per second (log scale)
10^1
10^0
10^{-1}
3x3
4x4
5x5
6x6
Grid size


6-ingredient Gridworld: Queries Answered Per Second
Dynamic factoring (ours)
Static factoring (baseline)
Avg # queries answered per second (log scale)
10^1
10^0
10^{-1}
3x3
4x4
5x5
6x6
Grid size

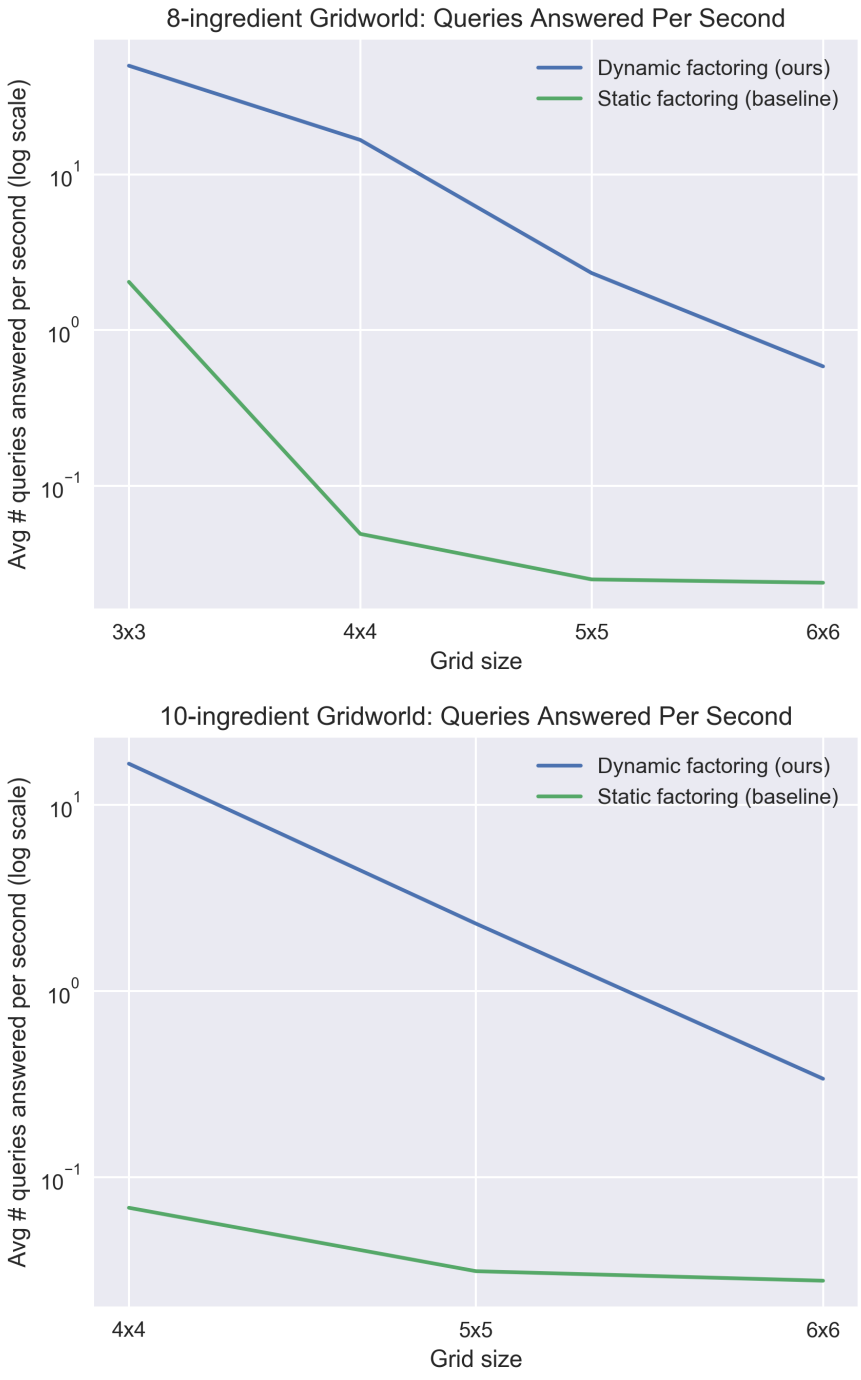


Figure 8-5: Visualizations of the average number of queries answered per second in Tables 8.3 and 8.4. *Note:* the y-axis is on a log scale.

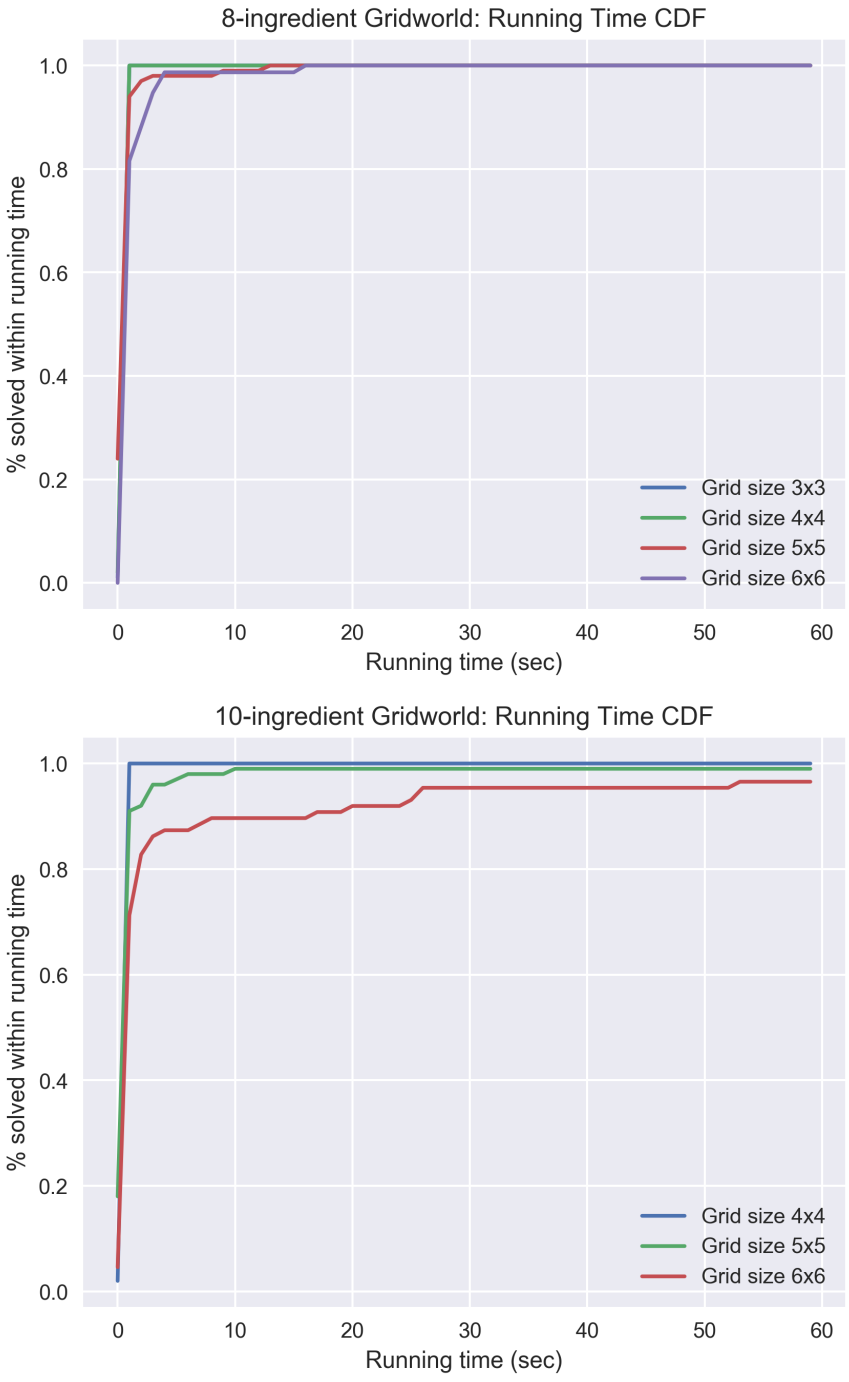


Figure 8-6: Cumulative distribution functions showing the percentage of the 100 experimental trials of our method that got solved within different running times, for the 8-ingredient gridworld (top) and 10-ingredient gridworld (bottom). Although our timeout was set to 60 seconds, most tasks were solved within 10 seconds, whereas the baseline (not shown) timed out frequently, as seen in Tables 8.3 and 8.4.

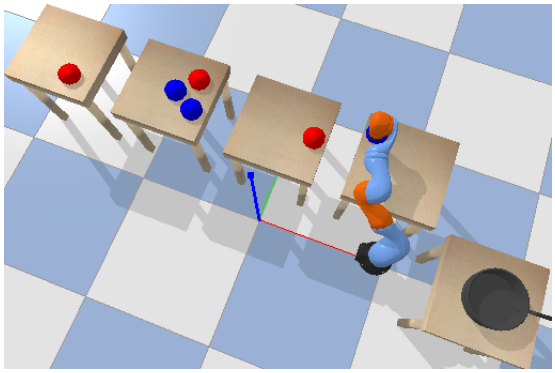


Figure 8-7: A visualization of the continuous 3D cooking task. The robot is a light-blue-and-orange arm. Vegetables (red) and seasonings (blue) are placed across four tables. The pot (black) is at a fixed, known position.

8.3 Domain 2: Continuous 3D 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 8-7 for a pybullet [6] visualization. The core difference is that the *position* property of each ingredient is now a continuous value: each ingredient 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 – thus, the action space is continuous. The initial dynamically factored belief state tracks location contents in a grid discretization of each table surface; we do this discretization because we require a discrete set of locations. Note that ingredient positions are continuous values, and thus will not be exactly equal to any of these locations, whereas in the gridworld setting, the ingredient positions did correspond to locations. As ingredients get discovered, the dynamically factored belief will adapt to track continuous distributions over potential positions. Recall that Section 5.2 describes how we do belief updates with these continuous distributions. Our experiments measure performance as we vary the number of ingredients distributed across the four tables.

To do planning despite the continuous state and action space, we introduce abstractions as described in Srivastava et al. [27]. We replace continuous variables, such as object grasping poses, by discrete references to useful continuous values, a process known as *skolemization*. We then plan using these discrete variables and rely on

Setting	System	% Solved	Bel. Upd. Time	Queries/Second
4ing	S (baseline)	84	0.05	3.23
4ing	D (ours)	**100**	0.07	**8.33**
6ing	S (baseline)	58	0.05	0.509
6ing	D (ours)	**100**	0.09	**6.25**
8ing	S (baseline)	55	0.07	0.495
8ing	D (ours)	**100**	0.11	**5.56**
10ing	S (baseline)	48	0.07	0.287
10ing	D (ours)	**100**	0.14	**4.76**

Table 8.5: Results for belief space planning in the 3D continuous cooking task. Percentage of tasks solved, belief update time (seconds), and average number of queries answered per second (across only the solved problem instances) are shown. Each row reports averages over 100 independent random trials. S: Statically factored belief (baseline), D: Dynamically factored belief (our method). *Setting* column gives number of ingredients, always distributed across four tables. See Figure 8-8 and Figure 8-9 for visualizations of this data. Our approach leads to more tasks being solved than a static factoring does.

samplers, also called generators in their work, to propose good continuous values for them that satisfy real-world constraints.

As in the gridworld setting, we solve this task using the determinize-and-replan approach to belief space planning (Section 2.3). The determinization is again produced using Algorithm 2, which works for continuous as well as discrete distributions. Note that state variables not yet discovered by the agent will not be present in this determinization, as expected. Planning is done using A* over discrete variables. Because this is a computationally challenging task, we impose a 60-second timeout on every call to Algorithm 2, and if the timer expires, we consider that problem instance to be unsolved.

Table 8.5 shows our quantitative results for this task, with each row averaging over 100 randomly generated independent problem instances. The results are also visualized in Figure 8-8 and Figure 8-9. Additionally, Figure 8-10 shows that our method continues to outperform the baseline even as the average assertion complexity

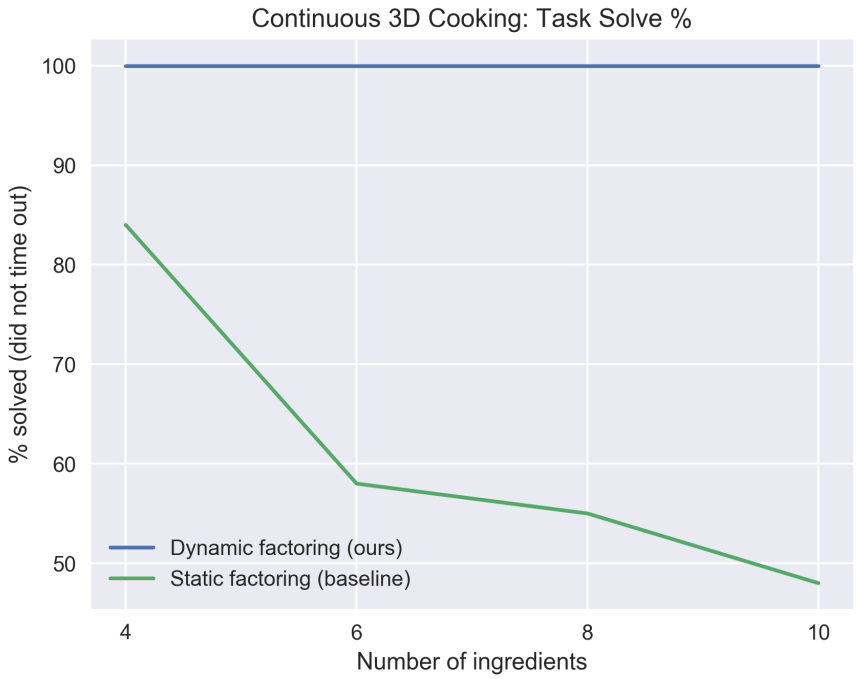


Figure 8-8: Visualization of the percent solved results in Table 8.5.

increases.

Discussion. Our approach performs extremely well in this challenging continuous domain. Whereas a static factoring is unable to efficiently answer queries without timing out, a dynamically factored belief continues to perform well as the number of objects scales up. For intuition on why our approach works much better, consider an assertion that two ingredients are within, say, 1 meter of each other. A static factoring would add this assertion to ComplexFluents, and to find a determinized world state, Algorithm 2 would keep sampling positions for the two ingredients independently until by chance it draws samples that are within 1 meter of each other. With continuous distributions in particular, this requires massive computational effort. Since assertions are coming in on every timestep, a static factoring quickly fails to perform well. On the other hand, a dynamic factoring would incorporate this assertion by introducing a new joint over the positions of the two ingredients; every sample from this joint satisfies the constraint by construction. As before, this improved performance comes at the cost of more expensive belief updates, but overall the trade-off is beneficial.

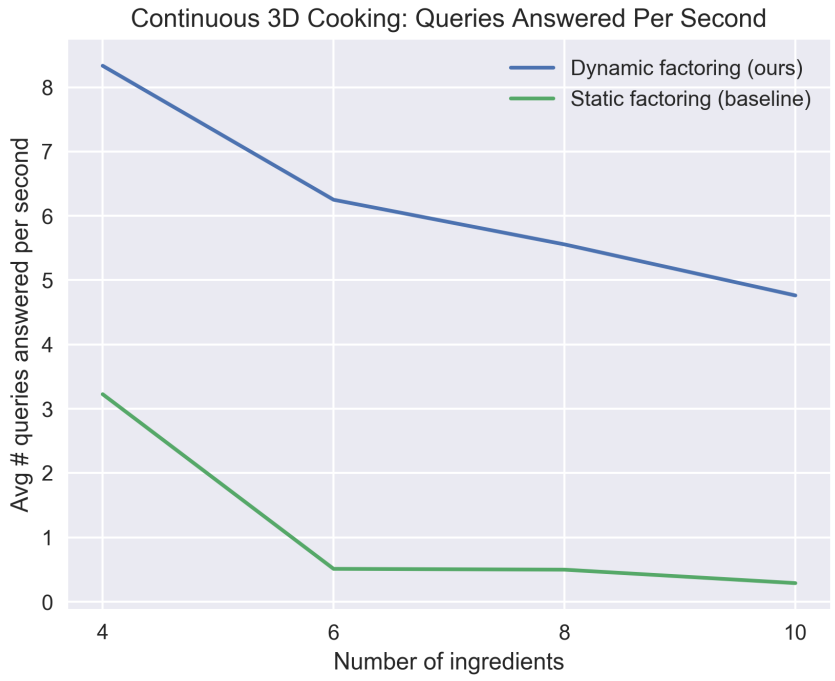


Figure 8-9: Visualization of the average number of queries answered per second in Table 8.5.

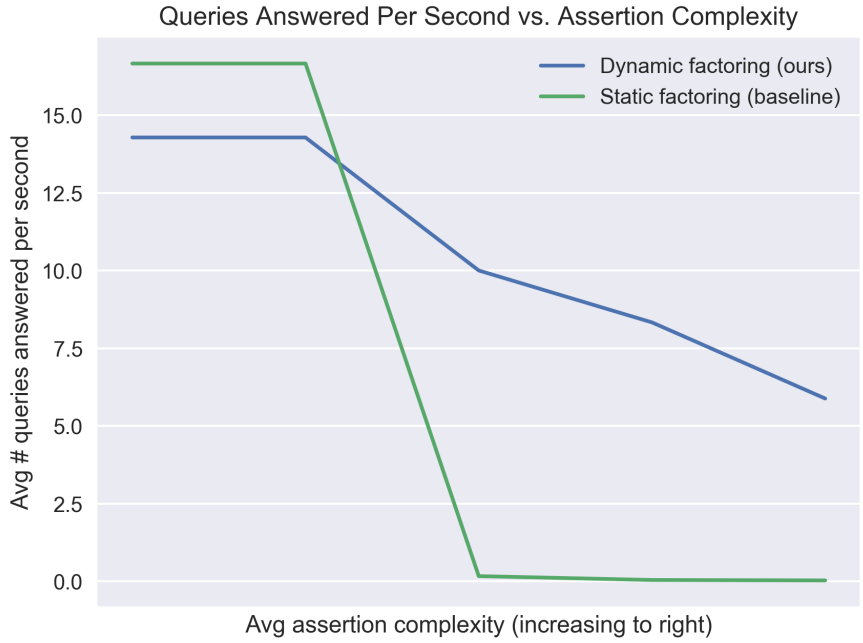


Figure 8-10: Performance of our approach (inferences per second) versus average assertion complexity, measured by the number of state variables referenced. Our method, the dynamically factored belief, continues to answer queries as assertion complexity increases, while using a static factoring becomes prohibitively slow.

8.4 Noisy Observations

In this section, we show results for our experiments with noisy observations, which we introduced and gave algorithms for in Chapter 7. We rerun the previous set of experiments while varying two parameters: p, which is the probability that each fluent holds in the current world state as defined in Section 7.1, and the marginalization error threshold used in Algorithm 6. p controls the degree of noise in the observations, with $p = 1$ corresponding to the noiseless setting; for simplicity of presentation, these experiments use a single value of p across all observations. The marginalization error threshold controls how aggressively we split up factors to maintain a compact belief representation; as the threshold increases, we are willing to marginalize state variables out of factors more frequently (even if this operation is lossy), providing a more compact representation at the cost of accuracy. Recall that the measure of error we use, the Jensen-Shannon divergence between the true joint and the reconstructed approximate joint, is bounded between 0 and $\log 2$.

Table 8.6 shows our quantitative results for this noisy observation setting, with each row averaging over 100 randomly generated independent problem instances. In the table, the size of a factor is the number of state variables involved in the joint distribution to which the factor maps. The results are also visualized in Figure 8-11 and Figure 8-12.

Discussion. A clear trend in the data is that the average execution cost tends to increase as either p goes down or the marginalization error threshold goes up. The former occurs because as observations become more noisy, the agent must do more observations in order to learn accurate information about its environment, increasing cost. The latter occurs because as the belief representation becomes less accurate, Algorithm 2 (which produces world state determinizations based on the belief) tends to be less accurate, increasing the amount of replanning necessary and thus the execution cost as well. Another trend is that the average factor size decreases as the marginalization error threshold goes up. This is sensible because state variables are more frequently marginalized out of factors, meaning the average factor size decreases.

p	**error_threshold**	**Avg. Cost (SD)**	**Avg. Factor Size**
1	N/A	971 (26)	1.52
0.9	0	1024 (74)	1.9
0.9	0.1	1069 (81)	1.75
0.9	0.2	1092 (116)	1.65
0.9	$\log 2$	1233 (65)	1.54
0.75	0	1179 (144)	2.38
0.75	0.1	1277 (86)	2.14
0.75	0.2	1338 (180)	2.03
0.75	$\log 2$	1637 (109)	1.78
0.5	0	2017 (133)	2.44
0.5	0.1	2298 (190)	2.36
0.5	0.2	2367 (159)	2.36
0.5	$\log 2$	2468 (158)	1.64

Table 8.6: Results for belief space planning with noisy observations, for different settings of p (the probability that each fluent holds in the world) and the marginalization error threshold (recall that the error is bounded between 0 and $\log 2$). See the algorithms in Section 7.2 for more information. Average execution cost (which reflects inference accuracy) and average factor size (which reflects compactness of the representation) are shown. Note that the size of a factor is the number of state variables involved in the joint distribution to which the factor maps. Each row reports averages over 100 independent random trials. See Figure 8-11 and Figure 8-12 for visualizations of this data. Note that when $p = 1$, we follow Algorithm 5, which is exact and does not use an error threshold; otherwise, we follow Algorithm 6.

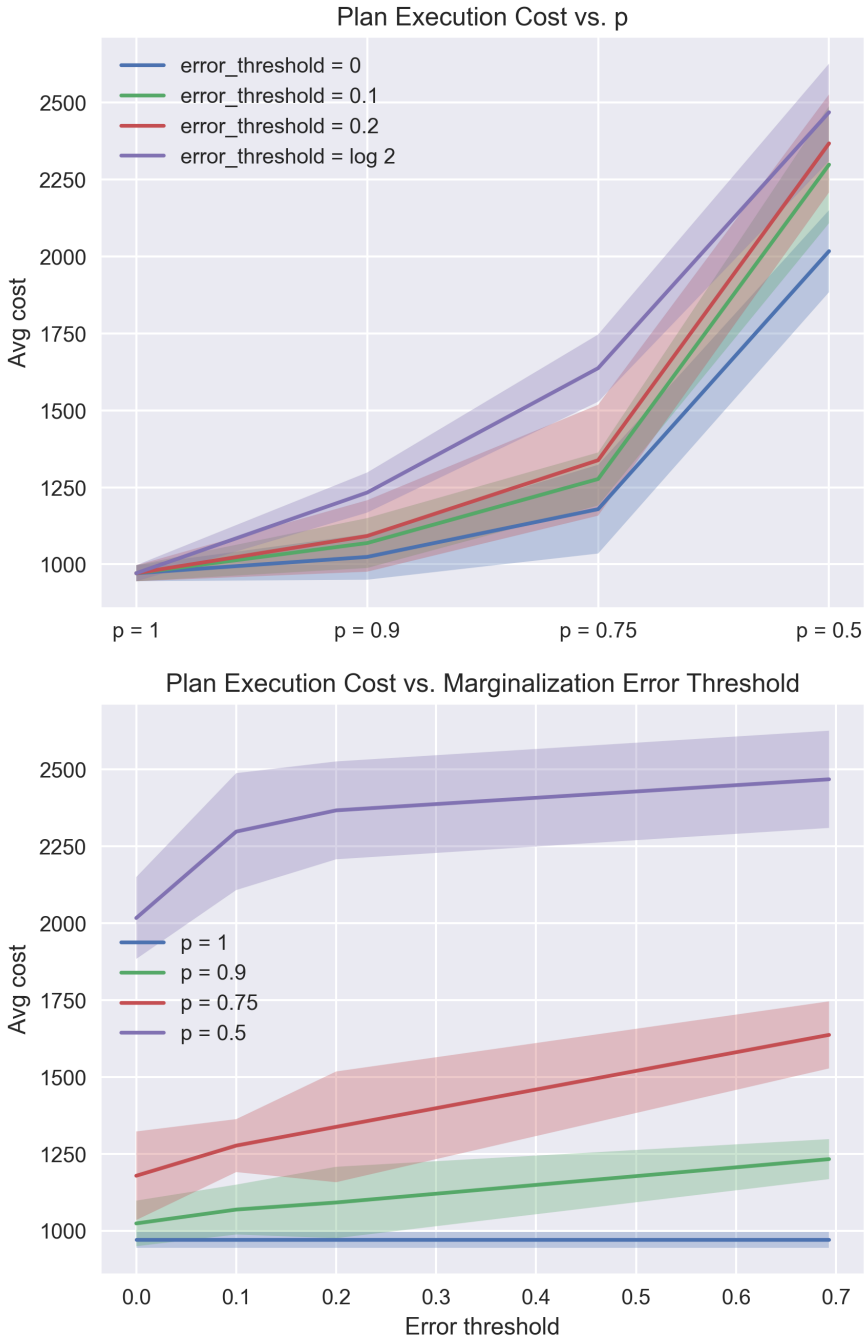


Figure 8-11: Visualizations of the average execution cost results in Table 8.6. *Top:* As p goes down, observations become more noisy. This in turn means the agent gains less information from each observation, and so average execution cost goes up because the agent must do more observations in order to learn accurate information about its environment. *Bottom:* As the error threshold increases, state variables are more frequently marginalized out of factors, meaning the belief representation gets more compact but less accurate. This causes execution cost to increase because the world determinizations produced using Algorithm 2, which are based on the belief representation, tend to be less accurate.

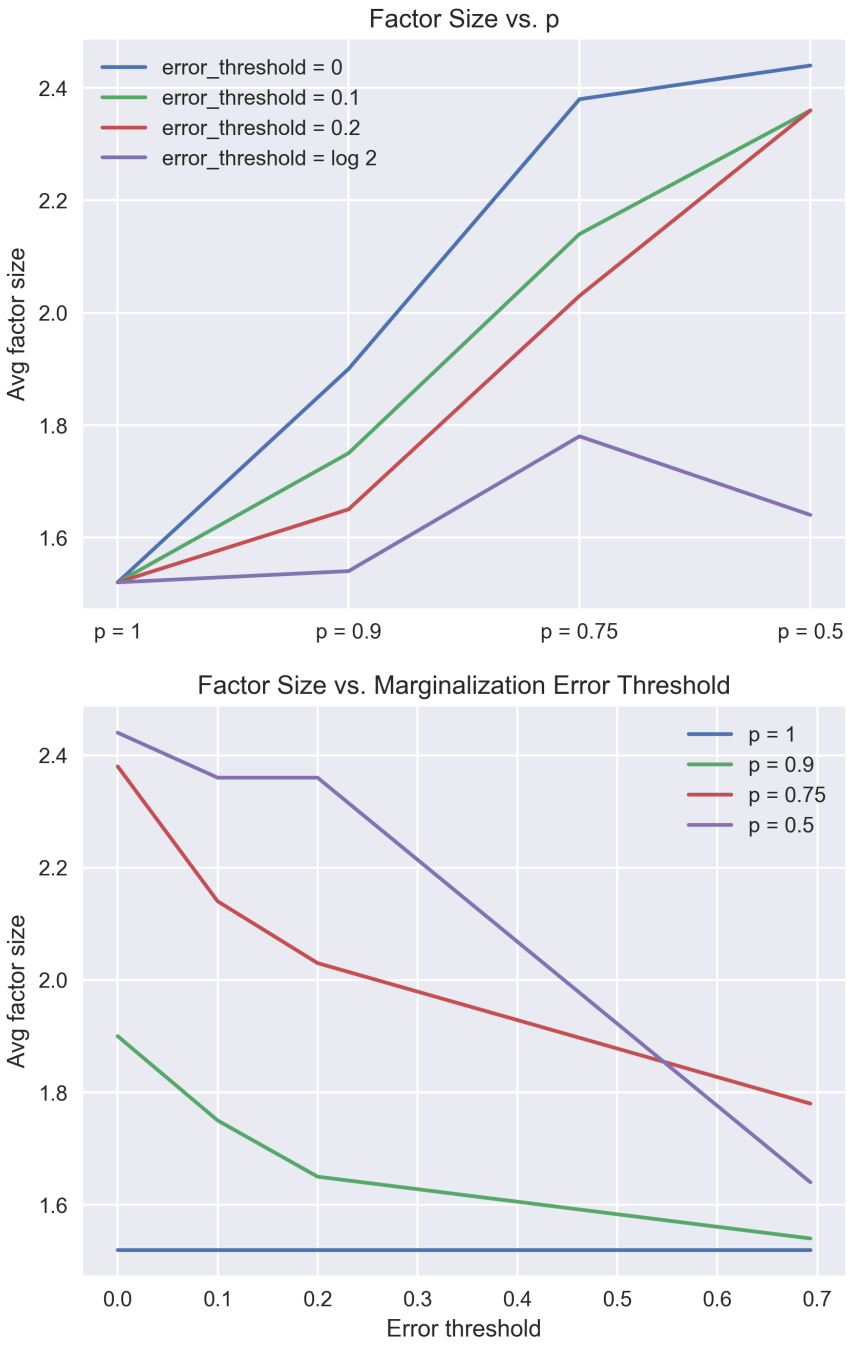


Figure 8-12: Visualizations of the average factor size results in Table 8.6. As the error threshold increases, state variables are more frequently marginalized out of factors, meaning the average factor size decreases. Note that the size of a factor is the number of state variables involved in the joint distribution to which the factor maps.

Chapter 9

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 follows elegantly from the factor graph perspective described in Chapter 6. In our work, the factor graph is always a set of independent cliques, but this may be very inefficient to maintain in practice. Instead, when the robot is given information linking some state variables, we may want to approximately fold this information into the belief representation and eschew calculating a full joint. We should seek a mechanism that allows designers to control the trade-offs between compactness of the representation and accuracy of inference.

Another future direction is to represent the assertions using natural language. In this setting, we would want a system that learns from data to automatically map a natural language assertion to a formal representation, i.e. a fluent. Though this general idea has been previously explored, one could imagine doing learning that is informed by the structure of the factored belief, in order to more quickly learn to produce context-relevant formal representations.

A third direction is to explore is a non-uniform observation model: if a robot 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. For instance, if a human states that there is danger ahead, a robot with an appropriate non-uniform observation model (on the space of information that could be given by the human) would be able to correctly infer the urgency of this remark.

Bibliography

[1] Stefano V Albrecht and Subramanian Ramamoorthy. Exploiting causality for selective belief filtering in dynamic bayesian networks. *Journal of Artificial Intelligence Research*, 55:1135–1178, 2016.

[2] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[3] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 52–61, 2000.

[4] Blai Bonet and Hector Geffner. Belief tracking for planning with sensing: Width, complexity and approximations. *Journal of Artificial Intelligence Research*, 2014.

[5] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 1998.

[6] Erwin Coumans, Yunfei Bai, and Jasmine Hsu. Pybullet physics engine.

[7] Robert P Goldman and Mark S Boddy. Expressive planning and explicit knowledge. In *AIPS*, volume 96, pages 110–117, 1996.

[8] Dylan Hadfield-Menell, Edward Groshev, Rohan Chitnis, and Pieter Abbeel. Modular task and motion planning in belief space. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4991–4998, 2015.

[9] Michael Irwin Jordan. *Learning in graphical models*, volume 89. Springer Science & Business Media, 1998.

[10] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101:99–134, 1998.

[11] Kittipat Kampa, Jose C Principe, and K Clint Slatton. Dynamic factor graphs: A novel framework for multiple features data fusion. In *Acoustics Speech and Signal*

Processing (ICASSP), 2010 IEEE International Conference on, pages 2106–2109. IEEE, 2010.

[12] Scott Kiesel, Ethan Burns, Wheeler Ruml, J Benton, and Frank Kreimendahl. Open world planning for robots via hindsight optimization. In *ICAPS Planning and Robotics Workshop*, volume 2, 2013.

[13] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[14] 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.

[15] Peter S Maybeck. *Stochastic models, estimation, and control*, volume 3. Academic press, 1982.

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

[17] Piotr Mirowski and Yann LeCun. Dynamic factor graphs for time series modeling. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 128–143. Springer, 2009.

[18] Kevin Murphy and Yair Weiss. The factored frontier algorithm for approximate inference in dbns. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, 2001.

[19] Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

[20] Robert Platt Jr, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. 2010.

[21] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1):107–136, 2006.

[22] Brian Sallans. Learning factored representations for partially observable markov decision processes. In *Advances in neural information processing systems*, pages 1050–1056, 2000.

[23] Guy Shani and Ronen I Brafman. Replanning in domains with partial information and sensing actions. In *IJCAI*, volume 2011, pages 2021–2026, 2011.

[24] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[25] 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.

[26] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In *Advances in neural information processing systems*, pages 1772–1780, 2013.

[27] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. *IEEE Conference on Robotics and Automation*, 2014.

[28] Kartik Talamadupula, J Benton, Subbarao Kambhampati, Paul Schermerhorn, and Matthias Scheutz. Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(2):14, 2010.

[29] Kartik Talamadupula, Gordon Briggs, Matthias Scheutz, and Subbarao Kambhampati. Architectural mechanisms for handling human instructions in open-world mixed-initiative team tasks. *Advances in Cognitive Systems (ACS)*, 6, 2013.

[30] Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, volume 7, pages 352–359, 2007.

[31] Sungwook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. Probabilistic planning via determinization in hindsight. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, pages 1010–1016, 2008.

[32] Shiqi Zhang and Peter Stone. Corpp: Commonsense reasoning and probabilistic planning, as applied to dialog with a mobile robot. In *AAAI*, pages 1394–1400, 2015.