TAPIR: A Software Toolkit for Approximating and Adapting POMDP Solutions Online

Dimitri Klimenko, Joshua Song, and Hanna Kurniawati
Robotics Design Laboratory
School of Information Technology and Electrical Engineering
University of Queensland, Australia
{dimitri.klimenko@uqconnect, mun.song@uqconnect, hannakur@uq}.edu.au

Abstract

One of the fundamental challenges in the design of autonomous robots is to reliably compute motion strategies in spite of significant uncertainty about sensor reliability, control errors, and unpredicatable events. The Partially Observable Markov Decision Process (POMDP) is a general and mathematically principled framework for this type of problem. Although exact solutions are computationally intractable, modern approximate POMDP solvers have made POMDP-based approaches practical for robotics tasks. However, almost all existing POMDP-based planning software suffers from at least two major issues. Firstly, most POMDP solvers require the model to be known *a priori* and remain constant during runtime, and secondly, quite a lot of the existing software is not very user-friendly. This paper presents the Toolkit for approximating and Adapting POMDP solutions In Real time (TAPIR), which tackles both problems. The need for a constant, fully known POMDP model is averted by implementing the recent Adaptive Belief Tree (ABT) algorithm, while user-friendliness is ensured by a well-documented modular design, which also includes interfaces for the commonly-used Robotics Operating System (ROS) framework, and the high fidelity simulator V-REP. TAPIR can be downloaded from `http://robotics.itee.uq.edu.au/~tapir`. To the best of our knowledge, TAPIR is the first software toolkit that directly addresses the aforementioned problems.

1 Introduction

Uncertainty is ubiquitous. For instance, GPS signals are often blocked, wind and currents accentuate control errors in aerial and marine robots, people may move unpredictably and block a robot's path, etc. Despite such uncertainties, an autonomous robot must be able to compute motion strategies that can consistently and efficiently accomplish the given tasks. Therefore, motion planning under uncertainty is critical for reliable operation of autonomous robots.

The Partially Observable Markov Decision Process (POMDP) is a general and mathematically principled framework for motion planning under uncertainty [Kaelbling *et al.*, 1998]. Due to uncertainty, a robot never knows its exact state, though it can infer a set of possible states of where it might be. POMDP-based approaches represent these sets of possible states as probability distributions over the state space, called *beliefs*, and systematically reason over the belief space (i.e. the set of all possible beliefs); as a consequence, POMDPs naturally combine uncertainty in robot controls, sensor measurements, and limited information and understanding about the environment. Moreover, POMDPs include a well-defined notion of an optimal motion strategy which serves as the overarching goal.

Although solving a POMDP problem exactly is computationally intractable [Papadimitriou and Tsitsiklis, 1987], recent work has shown that by trading optimality with approximate optimality for speed, POMDPs can be made practical [Kurniawati *et al.*, 2008; Silver and Veness, 2010; Smith and Simmons, 2005]. In fact, over the past decade, POMDP algorithms have developed from approaches that take days to solve problems with only a dozen states [Kaelbling *et al.*, 1998] to some that take only a few minutes to generate good solutions for problems with 10-dimensional continuous state space [Bai *et al.*, 2012; Kurniawati *et al.*, 2012]. These improvements have brought POMDPs into the realm of practicality for robot motion planning [Horowitz and Burdick, 2013; Koval *et al.*, 2014; Temizer *et al.*, 2010].

Despite these tremendous advances in POMDP-based motion planners, two major problems have stood as a barrier to their adoption by the wider robotics community. *First* is that most POMDP-based planners require the POMDP model to be completely known *a priori* and to remain the same during runtime. This requirement can often cause significant difficulties when the system has many unknowns, such as when a robot operates in an *a priori* unknown environment. For these kinds of problems, most POMDP-based motion planners are forced to model all unknowns (e.g. all possible environments) as part of the POMDP model, resulting in an *extremely* large POMDP problem that is infeasible to solve by even the best solver today. *The second problem* is the lack of more user-friendly POMDP solvers for robotics problems. In addition to the

modeling complexity, most implementations of POMDP solvers provide only a command-line interface; such interfaces are not very intuitive for robotics problems. Although one can, in principle, extend a command-line interface to a graphical one, this extra work is often undesirable for someone new to POMDPs or those who only want to use POMDPs as a small part of a larger robotics system.

To address both of the aforementioned problems, we introduce a software toolkit, called TAPIR (Toolkit for approximating and Adapting POMDP solutions In Real time). TAPIR is an implementation of our recent algorithm, Adaptive Belief Tree (ABT) [Kurniawati and Yadav, 2013]. ABT can find a good approximate solution and adapt POMDP solutions online in response to changes in the POMDP model. This ability to adapt POMDP solutions online enables us to relax the requirement that the POMDP model must be known *a priori*, and must remain constant at runtime. To handle the second problem, TAPIR includes high-quality documentation, and provides problem templates for common robotics tasks, i.e. target-finding and environmental sampling problems, to ease users through the process of defining a POMDP model. These templates also serve as illustrative examples of how a POMDP can be defined and implemented in TAPIR. Furthermore, to ease roboticists in applying POMDP techniques to their robotics system, TAPIR provides a programming interface with the commonly used Robotics Operating System (ROS) framework [Quigley *et al.*, 2009] and the V-REP robotics simulator [E. Rohmer, 2013].

2 Related work

2.1 POMDP Background

A POMDP model is a tuple $\langle S, A, O, T, Z, R, b_0, \gamma \rangle$, where S is the set of states, A is the set of actions, and O is the set of observations. At each step, the agent is in a state $s \in S$, takes an action $a \in A$, and moves from s to an end state $s' \in S$. To represent uncertainty in the effect of performing an action, the system dynamic from s to s' is represented as a conditional probability function $T(s,a,s') = f(s'|s,a)$. To represent sensing uncertainty, the observation that may be perceived by the agent after performing action a and ends at state s', is represented as a conditional probability function $Z(s',a,o) = f(o|s',a)$. At each step, the agent receives a reward $R(s,a)$, if it takes action a from state s. The agent's goal is to choose a suitable sequence of actions that will maximize its expected total reward, while the agent's initial belief is denoted as b_0. When the sequence of actions has infinite length, we specify a discount factor $\gamma \in (0,1)$ so that the total reward is finite and the problem is well defined.

In many problems with large state space, an explicit representation of the conditional probability functions T and Z may not be available. However, one can use a *generative model*, which is a black box simulator that outputs an observation perceived, reward received, and next state visited when the agent performs the input action from the input state.

A POMDP planner computes an *optimal policy* that maximizes the agent's expected total reward. A POMDP policy $\pi\colon B \to A$ assigns an action a to each belief $b \in B$, where B is the belief space. A policy π induces a value function $V(b,\pi)$ which specifies the expected total reward of executing policy π from belief b, and is computed as $V(b,\pi) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t,a_t)|b,\pi]$. A policy can be represented in a variety of ways, e.g. by a policy graph [Bai *et al.*, 2010], or by pairs of belief and action [Thrun, 2000].

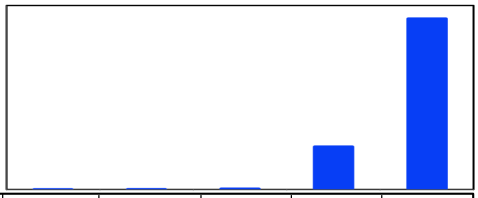
To execute a policy π, an agent executes action selection and belief update repeatedly. For example, if the agent's current belief is b, it selects the action referred to by $a = \pi(b)$. After the agent performs action a and receives an observation o according to the observation function Z, it updates b to a new belief b' given by $b'(s') = \tau(b,a,o) = \eta Z(s',a,o) \int_{s \in S} T(s,a,s') ds$ where η is a normalization constant. When a generative model is used, a belief is represented as a set of particles and the above belief update can be approximated using a particle filter.

2.2 Related POMDP Solvers

The past few years have seen tremendous advances in the capability of both offline and online POMDP solvers [Kurniawati *et al.*, 2008; Silver and Veness, 2010; Smith and Simmons, 2005]. Offline solvers compute the entire policy *a priori* before the policy must be executed, while existing online solvers compute the policy for the current belief during execution of the policy. Advances in both kinds of solvers have enabled POMDP-based approaches to start being practical for robot motion planning problems.

Not long ago, the best solver could take hours to compute exact solutions to POMDPs with only a dozen states. In 2003, Pineau, et al. introduced the first offline POMDP solver capable of computing a good approximate solution for a benchmark problem with 870 states [Pineau *et al.*, 2003], albeit taking 50 hours. Key to this solver is *sampling*—it takes a representative sample of the belief space, and plans with respect to only this set of sampled beliefs rather than the entire belief space. The success of this approach depends heavily on quickly sampling a small but representative set of beliefs. Since then, more suitable sampling strategies [Kurniawati *et al.*, 2008; Smith and Simmons, 2004; Smith and Simmons, 2005; Spaan and Vlassis, 2005] have been proposed, resulting in substantial increases in solving speed (as illustrated in Figure 1) and bringing POMDPs into the realm of the practical for robot motion planning problems.

Many works [Hauser, 2010; He *et al.*, 2010; Silver and Veness, 2010] have proposed advanced online solvers as well. As with offline solvers, the fastest online POMDP solvers [Silver and Veness, 2010] today are based on sampling. Most of these solvers represent the sampled belief in a *belief tree*. A belief tree is a tree where each node represents a belief and each edge represents an action–observation pair. A parent–child relation in a belief tree means that the belief that corresponds with the parent node evolves to the belief that corresponds with the child node whenever the action and observation that correspond to the



	PBVI Pineau, et. al.	**HSVI** Smith & Simmons	**Perseus** Spaan & Vlassis	**HSVI2** Smith & Simmons	**SARSOP** Kurniawati, et. al.
Time (s)	180,880	10,113	1,670	24	6
Reward	−9.18	−6.17	−6.17	−6.36	−6.13

Figure 1: Performance of several offline POMDP solvers on a benchmark problem called Tag, which is a robot target-finding problem with 870 states. The higher the reward, the better the solution.

edge between the parent and child nodes have been applied and perceived. Sampling in a belief tree is performed by sampling which node to expand, and which action and observation to use in expanding that node.

In general, all of the aforementioned offline and online solvers assume that the POMDP model is known *a priori* and does not change during execution of the policy. In order to handle changes in the POMDP model, these solvers would be forced to discard prior computation and recompute the policy from scratch, wasting all of the previosuly expended computation effort. In contrast, ABT can reuse and improve the policy, and is capable of runtime policy updates.

Prior work [Kurniawati and Patrikalakis, 2012] has proposed a point-based method to modify a pre-computed policy. However, the time it needs to update a policy is too long to be practical for a fast-changing environment, and the types of model changes that can be handled are limited to changes in the transition, observation, and reward functions. In contrast, ABT can handle any type of model change except for changes in the number of state variables, and is fast enough to update a policy online.

3 The Algorithm: ABT

Algorithm 1 presents an overview of ABT; the details of this algorithm are presented in [Kurniawati and Yadav, 2013]. For completeness, we present key components of the algorithm in this section.

ABT is an online and anytime POMDP solver capable of handling large and even continuous state spaces. It uses a generative model, which is a black box simulator that enables the solver to generate experiences of the system dynamics and behavior at various states. By using a generative model, ABT does not need an explicit model of the probability distributions for transitions and observations, which can often be difficult to obtain for problems with large state spaces.

Unlike most solvers, ABT can update the policy as necessary during runtime in response to changes in the POMDP model. This update capability is founded on the following two observations. First, a non trivial change in

Algorithm 1 Adaptive Belief Tree (initial POMDP model P_0, initial belief b_0)

PREPROCESS (OFFLINE)
$(H, \mathcal{T}) \leftarrow$ GeneratePolicy(P_0, b_0).
Let S' be the set of all sampled states in H,
  i.e., $S' \leftarrow \{h_i.s \mid i \in [0,|h|], h \in H\}$
Let $\mathcal{R}$ be a spatial index (e.g. a range tree) representing S'.
$b \leftarrow b_0$.

RUNTIME (ONLINE)
while running **do**
  if $P_t \neq P_{t-1}$ {P_i is the POMDP model at time-i.} **then**
    $H' \leftarrow$ IdentifyAffectedEpisodes(P_{t-1}, P_t, H, $\mathcal{R}$, $\mathcal{T}$).
    ReviseEpisodes(P_t, $\mathcal{T}$, b, H').
    UpdateValues($\mathcal{T}$, b, H').
  while there is still time **do**
    ImprovePolicy(P_t, H, $\mathcal{R}$, $\mathcal{T}$, b).
  $a \leftarrow$ Get best action in $\mathcal{T}$ from b.
  Perform action a.
  $o \leftarrow$ Get observation.
  $b \leftarrow \tau(b,a,o)$.
  $t \leftarrow t+1$.

the POMDP model must be directly reflected as a change in the robot's behaviour at a particular set of states. Second, a change in a single optimal mapping $\pi^*(b)$ from a belief $b \in B$, may only affect the optimal policy $\pi^*(\cdot)$ at b or at other beliefs that can reach b. Using the insight from these observations, ABT represents the policy as pairs of belief and action, and explicitly represents the relation between beliefs, states, and their reachability information, so that it can quickly identify the subset of the policy affected by changes in the POMDP model and quickly update the policy whenever necessary.

To explicitly maintain the aforementioned relations, ABT embeds the policy in an *augmented belief tree*—denoted as $\mathcal{T}$. An augmented belief tree is a belief tree where each of its paths is augmented by a set of sampled state trajectories. As with any belief tree, each node in the augmented belief tree $\mathcal{T}$ represents a belief; for the sake of brevity we refer to the node and the belief it represents interchangeably. The root of $\mathcal{T}$ represents the given initial belief b_0. Each edge $\overline{bb'}$ in $\mathcal{T}$ is labelled by a pair of action and observation a–o, where $a \in A$ and $o \in O$. An edge $\overline{bb'}$ with label a–o means that when a robot at belief b performs action a and perceives observation o, its next belief would be b', i.e. $b' = \tau(b,a,o)$ where $b, b' \in B$. Unlike most belief trees, each path in $\mathcal{T}$ is additionally augmented with a set of sampled state trajectories. A sampled state trajectory is often called an *episode*, denoted as h. It is a sequence of quadruples (s,a,o,r) of state $s \in S$, action $a \in A$, observation $o \in O$, and immediate reward $r = R(s,a)$. ABT maintains a set of all sampled episodes, which we denote as H. Figure 2 illustrates an augmented belief tree.

To construct the policy, ABT samples episodes. To sam-

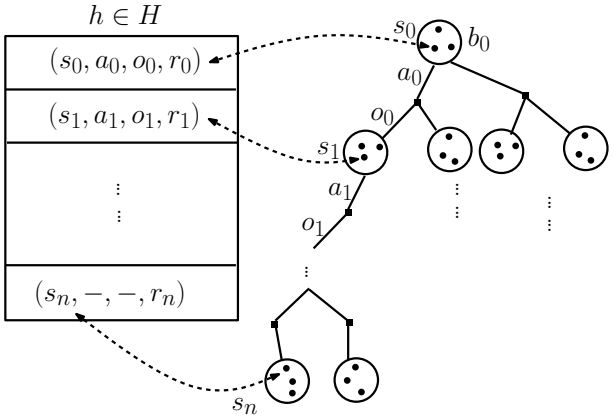


Figure 2: An augmented belief tree $\mathcal{T}$. Illustration of an association between an episode $h \in H$ and a path in $\mathcal{T}$.

ple an episode h, ABT samples an initial state $s_0 \in S$ from a given initial belief b_0 and selects an action $a_0 \in A$. The details of action selection are presented in [Kurniawati and Yadav, 2013]; they are outside of the scope of this paper. After an action a_0 is selected, ABT calls the generative model to sample an observation $o_0 \in O$, an immediate reward r_0, and a next state s_1 when the agent performs a_0 at s_0. ABT inserts the quadruple (s_0, a_0, o_0, r_0) as the first element of h, and iteratively repeats the above steps starting from s_1. The iteration stops after either a terminal state is reached, or some other stopping criterion is met (e.g. the sampling reaches a previously unvisited node in the belief tree, or h has exceeded a certain length). As a last step, ABT inserts $(s,–,–,r)$ as the last element of h, where s is the next state sampled by the last call to the generative model and $r = R(s)$ is the reward of being at state s. For a terminal state this reward value $R(s)$ can be calculated directly from the model, but if the sampled episode stops before a terminal state is reached ABT can also use a *heuristic estimate* in order to estimate the value of future rewards. One simple and general-purpose heuristic is the rollout, which is a widely-used technique in sampling-based algorithms, e.g. POMCP [Silver and Veness, 2010]. In ABT this can be done by continuing to sample as per the above approach while only storing the total discounted reward. This saves a significant amount of memory compared to explicitly storing a longer episode and all of the belief nodes visited during that episode. An improved approach [Kurniawati and Yadav, 2013], which is the one that is typically used in the provided software, is to use a solution to a simplified version of the problem (e.g. a fully observable MDP) to derive a value estimate. Examples of this approach are given in section 5.2.

Let H be the set of sampled episodes. The paths in the augmented belief tree $\mathcal{T}$ are associated with the episodes in H. Suppose ϕ is a path in $\mathcal{T}$ and $\phi = \langle b_0, a_0, o_0, \ldots, a_n, o_n, b_{n+1} \rangle$, where $b_i, b_{n+1} \in B$, $a_i \in A$, and $o_i \in O$ for $i \in [0,n]$. Then, ϕ is associated with the set of episodes $H_\phi \subseteq H$ which consists of all episodes in H that contains $\langle (s_0, a_0, o_0, *), \ldots, (*, a_n, o_n, *), (*,–,–,*) \rangle$, where s_0 is any state sampled from b_0, a_i and o_i ($i \in [0,n]$) are the corresponding actions and observations in ϕ, and $*$ means any relevant value. Figure 2 illustrates the relation between an episode in H and a path in the belief tree $\mathcal{T}$. *Each episode in H corresponds to exactly one path of $\mathcal{T}$, but a path of $\mathcal{T}$ may be associated with many episodes.*

Each belief in $\mathcal{T}$ is represented by a set of particles, which is comprised of the states in the corresponding quadruples of the corresponding episodes. Suppose b is a node at level-l of $\mathcal{T}$ (the root has level 0). Suppose $\Phi(b)$ is the set of all paths in $\mathcal{T}$ that starts from the root and contains the node b, and $H_b = \bigcup_{\phi \in \Phi(b)} H_\phi$. Then, b is approximated by the set of particles $\{h_l.s \mid h \in H_b\}$, where the notation $h_l.s$ refers to state in the l^{th} quadruple of an episode $h \in H_b$ (the quadruples are indexed from 0).

The policy π of ABT is embedded in the augmented belief tree $\mathcal{T}$, with

$$\pi(b) = \arg\max_{a \in A(E,b)} \hat{Q}(b,a) \tag{1}$$

$$\text{and value} \quad V(b,\pi) = \max_{a \in A(E,b)} \hat{Q}(b,a) \tag{2}$$

where $b \in B$, E is the set of edges in $\mathcal{T}$, and $A(E,b) \subseteq A$ is the set of actions that have been used to expand b, i.e. the actions that comprise the labels of the out-edges of b in $\mathcal{T}$. The value $\hat{Q}(b,a)$ denotes the estimated Q-value; the true Q-value $Q(b,a)$ is the value of performing action a from belief b and continuing optimally afterwards, i.e.

$$Q(b,a) = R(b,a) + \gamma \sum_{o \in O} \tau(b,a,o) V^*(\tau(b,a,o)), \tag{3}$$

where $V^*(b) = V(b,\pi^*)$ is the *optimal value function* - the expected value of following an optimal policy from an initial belief b. ABT estimates $Q(b,a)$ as

$$\hat{Q}(b,a) = \frac{1}{|H_{(b,a)}|} \sum_{h \in H_{(b,a)}} V(h,l) \tag{4}$$

where $H_{(b,a)} \subseteq H$ is the set of all episodes associated with all paths in $\mathcal{T}$ that start from b_0 and contain the sequence (b,a), l is the depth level of b in $\mathcal{T}$, and $V(h,l)$ is the value of an episode h starting from the l^{th} element. $V(h,l)$ is computed as

$$V(h,l) = \sum_{i=l}^{|h|} \gamma^{i-l} R(h_i.s, h_i.a) \tag{5}$$

where γ is the discount factor and R is the reward function. Note that each state in the l^{th} quadruple of each episode in $H_{(b,a)}$ is a particle of b and the action in that quadruple is the action a. Therefore, it is clear that $\hat{Q}(b,a)$ approximates the first component of $Q(b,a)$ well. However, it may seem odd that eq. (4)-(5) can approximate the second component of $Q(b,a)$, which is $\sum_{o \in O} \tau(b,a,o) V^*(\tau(b,a,o))$, as $V(h,l+1)$ for different h may correspond to different policies. It turns out by using an appropriate action selection strategy when sampling the episodes, one can ensure that as the number of episodes in $H_{(b,a)}$ increases, $V(h,l+1)$ converges to $\sum_{o \in O} \tau(b,a,o) V^*(\tau(b,a,o))$ in probability, and hence ABT's policy converges to the optimal policy in probability [Kurniawati and Yadav, 2013].

The above policy representation and value calculation enables ABT to quickly identify and update the policy following changes in the POMDP model. To identify which parts of the policy need to be updated, ABT only needs to find the episodes in H that contain states that are affected by the changes in the POMDP model. To update the policy, ABT disconnects the association between each affected episode h and its corresponding nodes in $\mathcal{T}$, revises h according to the new POMDP model, and associates it back with the nodes of $\mathcal{T}$ (which may be different than the previously associated path). Then, ABT updates the values and Q-values of the beliefs that have been newly associated or disassociated with h. Using eq. (2)-(5), the value and Q-value revisions require only simple arithmetic calculation. With proper data structures these values can be updated incrementally, and the processes of finding the affected episodes and of association/disassociation can both be done quickly.

4 The Software Toolkit: TAPIR

The TAPIR software package is provided as a C++ source code package, licensed under the GNU General Public License v2.0; it can be downloaded from the TAPIR website, `http://robotics.itee.uq.edu.au/~tapir/`. This source code package contains a full, modular, and customizable implementation of the ABT algorithm. Also included (in the directory `src/problems`) are implementations for two example problems - the aforementioned Tag problem, as well as another common benchmark problem called RockSample [Smith and Simmons, 2004]. All of the TAPIR source code comes with documentation comments, which are used to generate detailed HTML documentation (via Doxygen). This documentation and a couple of quick start guides are available on the TAPIR website. In this section we will give a brief overview of the structure of TAPIR, while the next section will provide a quick guide to its usage.

The core of TAPIR is the implementation of ABT, which is located in the directory `src/solver` within the source package. Directly in this directory are source files containing concrete classes, which are implementations of essential data structures used by ABT, most notably the belief tree and state trajectories. On the other hand, the subdirectories of `src/solver` can be considered as "modules" allowing customization of the ABT algorithm. This modularity is achieved by using abstract base classes to define the core functionality; this means that default implementations can be provided while also allowing for the possibility of custom problem-specific or application-specific implementations where needed. The key modules in TAPIR are:

- `abstract-problem` - abstract base classes to define a generative POMDP model. Further details are given in Section 5.1.
- `value-estimators` - estimation of the values of future beliefs based on the sampled values of the available actions, as discussed in the previous section. The default approach, as described by eq. (4)-(5), is simply to average over all of the episodes passing through that belief.
- `changes` - modification of trajectories in response to changes in the model. The standard provided approach simply revises an episode by resampling the affected parts of the episode using the same actions as before, but with the newly-modified generative POMDP model instead of the old one.
- `indexing` - indexing of the states, allowing for efficient lookup of which states (and hence which trajectories) are affected by a change in the environment. The default implementation uses an R*-tree, which is optimized for efficient spatial queries - e.g. all of the sampled states within a specified geometric region.
- `mappings` - handles the mapping of actions and observations to child nodes in the belief tree. Using abstract base classes allows customization of the data stored at each node and the data structures used to handle the mappings (e.g. a map for sparsely sampled children vs. a vector for densely sampled children); this can also allow for more advanced approaches like dynamically aggregating similar actions.
- `search` - handles the sampling policy, which controls how trajectories are sampled inside and outside the belief tree. The default behaviour is to use UCT [Kocsis and Szepesvri, 2006] to select actions within the belief tree and then to estimate new, previously unreached belief nodes by using a heuristic estimate (typically based on a solution to the underlying MDP).
- `serialization` - serialization of the various ABT classes, so that the state of the algorithm can be saved and loaded. This can, for example, allows TAPIR to run ABT offline and later use this precalculated solution to improve its online performance.

The modular design of TAPIR will help users interested in developing better POMDP solvers to quickly test their ideas. For instance, users interested in developing better sampling strategies only need to modify the `search` module of TAPIR; those interested in developing better policy adaptation strategies only need to modify the `changes` module; etc. TAPIR also provides interfaces to assist users who only want to use POMDPs to solve their problems, as detailed in the next section.

5 Using TAPIR

5.1 Defining a Problem

As stated in the previous section, the `abstract-problem` module defines the basic interface for implementing a new POMDP problem. The core of this module is the abstract base class `Model`, which represents a black box POMDP model in its entirety. In this section we will give a quick overview of how a new problem can be defined via the TAPIR interface; a more comprehensive guide is available via the TAPIR website.

Since ABT is based on a generative POMDP model, most of the structures in the POMDP tuple $\langle S, A, O, T, Z, R, b_0, \gamma \rangle$ are defined *implicitly*, rather than explicitly.

- S - states should be instances of a problem-specific subclass of the abstract class `State`. Since ABT is based on sampling, there is no need to explicitly define which states are in the space; ABT will only ever consider states that are reachable from the initial belief. This sampling-based approach is also the reason that TAPIR can easily deal with continuous state spaces. Additionally, the `Model` should implement an `isTerminal` method in order to specify whether a state is considered to be terminal.
- A - actions should be instances of a problem-specific subclass of the abstract class `Action`. Since ABT must be aware of the possible actions from a given belief state, the action space cannot be entirely implicit. This is handled by the `ActionPool` abstract base class, which allows control over which actions will be tried, and in which order. A useful implementation, `EnumeratedActionPool`, is provided, which takes a vector of all of the actions in the POMDP as an argument, and allows the actions to be tried in random order, with independent randomization at every belief node.
- O - observations should be instances of a problem-specific subclass of the abstract class `Observation`. As with states, since ABT receives observations via sampling from the generative model, there is no need to explicitly define which observations are in the space.
- T, Z, R - the conditional probabilities for state transitions and observations, as well as the reward function, are all implicitly defined via the generative model. In particular, the method `Model::generateStep` takes a state and an action as an argument, and must return sampled values for the subsequent next state and observation, and the associated reward.
- b_0 - the initial belief is defined implicitly via the method `Model::sampleAnInitState`, which should return a sampled state from that distribution.
- γ - the discount factor is defined in the `Options` instance associated with an instance of `Model`; this options class unifies a number of ABT- and TAPIR-related configuration settings, and allows problem-specific options to be added as well. TAPIR also comes with the functionality to read options from the command line and/or from a text file in INI format, with the possibility to add problem-specific options for any individual problem.

To deal with model changes an additional method, `Model::applyChanges`, must be implemented in order to inform the ABT algorithm of which states are affected by a change in the model.

The aforementioned aspects cover the core functionality of `Model`, although there is some additional functionality that is important. Many POMDP problems can be solved much more efficiently by using domain knowledge that is specific to the problem. TAPIR provides programming interfaces to incorporate such knowledge in some of the critical components of ABT.

One aspect that is of particular importance is the choice of a heuristic to estimate the value of sampled episodes that are cut off before a terminal state is reached; this is controlled via the method `Model::getHeuristicFunction`. TAPIR offers a general-purpose rollout-based heuristic approach which can be easily used for any problem by implementing `Model::getRolloutAction`. Custom problem-specific heuristics can be seen in the implementations for RockSample and Tag, including approaches based on solving fully observable versions of the problem.

Additionally, in order to deal with particle depletion efficiently, a `Model` should also implement a more sophisticated particle filtering method, which is used to generate new particles for the current belief whenever there are not enough particles from the trajectory-based sampling. A standard implementation based on rejection sampling is provided, but can be overridden in order to improve efficiency. Examples of custom particle filtering approaches can be seen in the provided code for both Tag and RockSample.

The provided example code for Tag and RockSample illustrates how to implement all of the above aspects for a specific POMDP; further detail on these example problems is provided in the following section.

5.2 Problem Templates

The source code for the Tag and RockSample problems in TAPIR serves two purposes. First, it demonstrates how individual POMDP problems can be implemented in the TAPIR framework. Second, these two implementations also serve as templates for broad categories of problems.

Tag – A Template for Target Finding/Tracking

The Tag problem [Pineau *et al.*, 2003] serves as a template for *target finding* and *target tracking* problems, in which the goal is to locate (and/or follow) a target in spite of limited sensing capabilities and uncertainty about the target's motion. Problems of this kind occur in various real-world scenarios, including emergency rescue, security/surveillance, and in-home care.

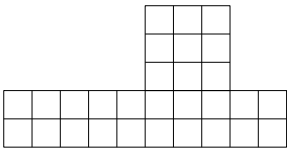


Figure 3: The original environment map of the Tag problem.

In the original Tag problem [Pineau *et al.*, 2003], the environment map is discretized into 29 cells (Figure 3). Initially, the robot and the target may be in any cell with equal probability. The robot's control and localization are perfect, but it can only see the target if the robot is in the same cell as

the target. The target has full observability of the robot's position and always moves away from the robot.

In TAPIR's implementation of the Tag problem, the user can change the environment map without additional programming and change the behavior of the target with relatively little modification to the problem. It also provides the functionalities to visualise the problem in V-REP. These features enable users to use TAPIR for robot target/tracking problems with minimum programming effort.

The implementation of Tag also shows how to implement the core functionality for target finding/tracking type of problem, as well as demonstrating some additional customization for better performance. This includes the implementations of `TagModel::generateParticles`, which demonstrate problem-specific particle filtering methods that are far more efficient than the default general-purpose rejection sampling approach. Also provided is the class `TagMdpSolver`, which demonstrates the use of a general-purpose MDP solver (in this case policy iteration) to generate a solution to a fully observable version of the problem. This solution can then be used as a heuristic function which will guide the sampling of the ABT algorithm, which can improve performance compared to a simple heuristic or to a sampling-based (rollout) heuristic. The MDP-solving code is also included in TAPIR, in `src/problems/shared/policy_iteration.cpp`.

RockSample – A Template for Environmental Sampling

The RockSample problem [Smith and Simmons, 2004] serves as a template for *environmental sampling* problems. It models the problem of a Mars rover seeking to collect samples from valuable rocks. The rover is equipped with a noisy sensor which it can use to gather information about the rocks.

RockSample is a scalable benchmark problem in POMDP. It is parameterized by two numbers, n and k. Rocksample$[n,k]$ means the robot is operating in an environment discretized into $n \times n$ cells and there are k rocks in the environment. Among these rocks, some of them are good rocks that the robot needs to sample. However, the robot does not know whether a rock is good or not. It can perform scanning to check if a rock is good or not, but the result of this check is not perfect. The robot has perfect control and localization capabilities. Details of Rocksample is in [**?**]

TAPIR's implementation of RockSample allows users to change the environment map and the sensing error of the robot without additional programming. This functionality enables users to use TAPIR for autonomous environmental sensing problems with minimum programming effort.

In addition to the basic implementation, the provided code also illustrates some advanced functionality that can be used. As with Tag, the RockSample implementation includes improved particle filtering methods and a `value-estimator` heuristic based on full observability. However, in the case of RockSample, a significantly more efficient approach using backward induction is used to solve the MDP.

The RockSample code also shows how TAPIR can use history-based sampling strategies and heuristic information to improve the sampling strategy of ABT. For instance, `PreferredActionsPool` implements heuristics similar to those used in POMCP [Silver and Veness, 2010] - the idea there is to bias the initial Q-value for certain "preferred" actions on the basis of domain-specific knowledge and inference from the history of actions and observations.

5.3 Interface for ROS with V-REP Simulator

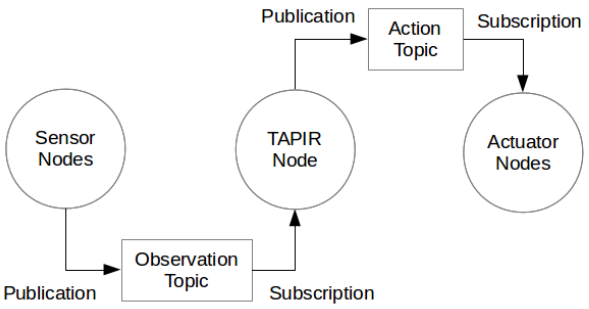


Figure 4: An illustration of how TAPIR interacts with sensing and actuation modules via ROS.

To ease the use of POMDP-based approaches in robotics systems, TAPIR provides an interface such that it can act as a ROS node. Figure 4 illustrates a typical communication diagram between TAPIR and sensing and actuation modules via ROS. The sensor and actuator nodes can be linked to V-REP (for simulation purposes) or to real devices.

TAPIR has been tested with ROS Hydro on Ubuntu 12.04 and ROS Indigo on Ubuntu 14.04, both with V-REP PRO EDU V3.1.2. When used with ROS, TAPIR requires Boost version 1.48 or above, which may conflict with the version used by ROS Hydro by default on Ubuntu 12.04. Instructions on resolving this issue can be found in the readme included in the software package. For convenience, the TAPIR package also provides a script that automatically resolves this problem.

To further enhance accessibility, TAPIR can automatically setup a build system for its use within the ROS framework. ROS uses a build system called catkin, which expects the source files to be organised within a catkin workspace. For users unfamiliar with ROS, tutorials on setting up a workspace are available on the ROS website (`http://wiki.ros.org/ROS/Tutorials`). Experienced ROS users may compile the TAPIR node manually using `catkin_make` and launch nodes with `roslaunch`. For expedient creation of new projects, TAPIR also provides a command (`make ros`) to automatically set up a workspace, add a symbolic link to the TAPIR source directory, and create a script, called `simulate-ros`, which when called will launch ROS, V-REP and TAPIR.

6 Benchmarking TAPIR

6.1 Setup

To test the performance of TAPIR, we used three well-known POMDP benchmark problems, i.e.

Tag [Pineau *et al.*, 2003], and two different instances of RockSample[n,k] [Smith and Simmons, 2004]—RockSample[7,8] and RockSample[11,11], where the map has size $n \times n$ and k is the number of rocks. Due to the larger parameter values, RockSample[11,11] has 247,808 states, as compared to the 12,544 states in RockSample[7,8].

In their default forms, neither Tag nor RockSample involve model changes during runtime. To test the performance of TAPIR when the model changes, we modified the original Tag, RockSample[7,8], and RockSample[11,11] by adding and removing obstacles every 2–4 seconds. When an obstacle is added, the size of the state space decreases; when an obstacle is removed, the size of the state space increases.

To empirically evaluate the overall performance of TAPIR, we chose to compare the performance of TAPIR against POMCP, a state-of-the-art online POMDP solver. For POMCP, we used the software provided by the authors at `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Applications_files/pomcp-1.0.tar.gz`.

To calculate the quality of the motion strategies generated by ABT, we estimated the expected total reward of using ABT to solve the benchmark problems. To this end, for each problem, we first ran a few trial runs to determine the best parameters for ABT to use. We then used the best parameters for each problem and ran 2,000 simulation runs, recording the total discounted reward for every run. In each run, we allowed 1 second of computation time per step. The estimated expected total reward was then calculated as the average discounted total reward over all 2,000 runs. We also used the same procedure to calculate the quality of the motion strategies generated by POMCP.

All experiments were conducted on a PC with an Intel Xeon E5-1620 3.6GHz processor and 16GB RAM.

6.2 Results

Table 1 shows the benchmark results on the original problems. The results indicate that even when the POMDP model does not change, ABT performs better than POMCP. The primary reason for this is that POMCP discards prior results and recomputes the policy from scratch at every step, while ABT reuses prior results and improves them at every step.

Table 2 shows results on the modified benchmark problems where addition and removal of obstacles, which leads to changes in the size of the state space, happens unpredictably every 2–4 seconds. The results indicate that although TAPIR requires additional overhead to update the policy, spending time for this overhead is still beneficial.

Overall, TAPIR has a distinct advantage over POMCP whenever a significant portion of the policy computed so far can be preserved from one step to the next. This advantage is clearly visible in both of the provided tests, with and without any changes in the underlying model. POMCP would have an advantage in extreme cases where the model changes so much that essentially the entire policy would have to be recomputed, since POMCP always recomputes from scratch regardless. However, for such extreme cases TAPIR also offers the option of replanning from scratch, while still providing the option of keeping its policy for all of the intermediate steps between such extreme changes.

Problem	POMCP		TAPIR	
	Average	95% conf.	Average	95% conf.
Tag	-7.33	0.24	-6.72	0.24
RockSample [7,8]	16.92 20.71*	0.48 0.21*	21.23	0.19
RockSample [11,11]	18.94 20.01*	0.94 0.23*	21.36	0.21

Table 1: Average total discounted reward when the POMDP model does not change. The computation time per step is 1 second. The results of POMCP that are marked with * are the results as presented in [Silver and Veness, 2010]. We were not able to generate equivalent results even after extensive parameter adjustments. The POMCP results presented without * are the results we gathered using the best parameters for POMCP.

Problem	POMCP		TAPIR	
	Average	95% conf.	Average	95% conf.
Tag	-7.75	0.30	-7.5	0.23
RockSample [7,8]	17.07	0.21	21.22	0.28
RockSample [11,11]	17.49	0.24	20.73	0.29

Table 2: Average total discounted reward when the POMDP model changes every 2–4 seconds. The computation time per step is 1 second.

6.3 Video Results

We have also provided an example of how TAPIR can interface with V-REP for visualisation purposes.

The first segment of our video submission uses a V-REP simulation to demonstrate how TAPIR can be integrated into a larger ROS-based project. It is also an example where the state space is continuous. A robot needs to monitor if a worker reaches certain workstations (called zones and marked with circles) safely, while minimizing movement cost and avoiding collisions with obstacles and the worker. The robot does not have a prior map of the environment, does not know the exact initial position of the worker, and does not know the worker's walking speed nor the time the worker spends at certain stations. The robot scans the environment using a laser sensor and builds the map incrementally using OctoMap [Hornung *et al.*, 2013]. When more information becomes available and the map is updated, the POMDP model is also updated to reflect the robot's improved understanding of where the obstacles are. When the POMDP model changes, TAPIR updates the solution online. This cycle repeats until the worker no longer needs

the robot's assistance.

The second segment of the submitted video shows how a user can interactively modify the environment in V-REP, leading to changes in the POMDP model at runtime. Following these changes to the model, TAPIR updates its policy online, in real time.

7 Summary

The past several years have seen tremendous advances in the capability of POMDP solvers. These solvers have moved the practicality of POMDP-based approaches far beyond robot navigation in a small 2D grid world, into mid-air collision avoidance of commercial aircraft (TCAS) [Temizer *et al.*, 2010], grasping [Koval *et al.*, 2014], and non-prehensile manipulation [Horowitz and Burdick, 2013], to name but a few examples. Despite these tremendous advances, two barriers to adoption have hindered the spread of POMDPs to the wider robotics community. First, most POMDP-based planners require the POMDP model to be known *a priori* and to remain unchanged while a robot is active. Second, there is a lack of user-friendly software for POMDP-based motion planning.

This paper presents a software toolkit, called TAPIR, that tackles both problems. TAPIR implements our recent algorithm, Adaptive Belief Tree (ABT). ABT reuses and improves existing techniques to quickly find a good approximate solution, and also introduces a novel capability to adapt the solution online in response to changes in the POMDP model. This feature enables us to relax the requirement that POMDP model must be known *a priori* and must remain the same while the robot executes its task. Furthermore, TAPIR provides an interface to the commonly-used Robotics Operating System (ROS) framework and V-REP simulator. To the best of our knowledge, TAPIR is the first software toolkit that directly addresses the aforementioned two problems. TAPIR can be downloaded from `http://robotics.itee.uq.edu.au/~tapir`. We hope this software will ease roboticists in using POMDPs—a mathematically principled approach for planning under uncertainty—and hasten the development of reliable and robust autonomous robots.

TAPIR also opens many avenues for future research and applications. For instance, can we speed up POMDP solving further, and generate solutions of equivalent quality but with an update rate above 100Hz? This level of performance can be essential in many field robotics systems.

References

[Bai *et al.*, 2010] H. Bai, D. Hsu, W.S. Lee, and A.V. Ngo. Monte Carlo value iteration for continuous-state POMDPs. In *WAFR*, 2010.

[Bai *et al.*, 2012] H. Bai, D. Hsu, M.J. Kochenderfer, and W.S. Lee. Unmanned aircraft collision avoidance using continuous-state POMDPs. In *RSS*, 2012.

[E. Rohmer, 2013] M. Freese E. Rohmer, S. P. N. Singh. V-REP: a versatile and scalable robot simulation framework. In *IROS*, 2013.

[Hauser, 2010] K. Hauser. Randomized belief-space replanning in partially-observable continuous spaces. In *WAFR*, 2010.

[He *et al.*, 2010] R. He, E. Brunskill, and N. Roy. PUMA: planning under uncertainty with macro-actions. In *AAAI*, 2010.

[Hornung *et al.*, 2013] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[Horowitz and Burdick, 2013] M. Horowitz and J. Burdick. Interactive non-prehensile manipulation for grasping via POMDPs. In *ICRA*, 2013.

[Kaelbling *et al.*, 1998] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *AI*, 101:99–134, 1998.

[Kocsis and Szepesvri, 2006] L. Kocsis and C. Szepesvri. Bandit based Monte-Carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[Koval *et al.*, 2014] Michael Koval, Nancy Pollard, and Siddhartha Srinivasa. Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty. In *RSS*, Berkeley, USA, July 2014.

[Kurniawati and Patrikalakis, 2012] H. Kurniawati and N.M. Patrikalakis. Point-based policy transformation: adapting policy to changing POMDP models. In *WAFR*, 2012.

[Kurniawati and Yadav, 2013] H. Kurniawati and V. Yadav. An online POMDP solver for uncertainty planning in dynamic environment. In *ISRR*, 2013.

[Kurniawati *et al.*, 2008] H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *RSS*, 2008.

[Kurniawati *et al.*, 2012] H. Kurniawati, T. Bandyopadhyay, and N.M. Patrikalakis. Global motion planning under uncertain motion, sensing, and environment map. *Autonomous Robots: Special issue on RSS 2011*, 30(3), 2012.

[Papadimitriou and Tsitsiklis, 1987] C.H. Papadimitriou and J.N. Tsitsiklis. The complexity of Markov decision processes. *Math. of Operation Research*, 12(3):441–450, 1987.

[Pineau *et al.*, 2003] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: an anytime algorithm for POMDPs. In *IJCAI*, pages 1025–1032, 2003.

[Quigley *et al.*, 2009] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[Silver and Veness, 2010] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *NIPS*, 2010.

[Smith and Simmons, 2004] T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *UAI*, 2004.

[Smith and Simmons, 2005] T. Smith and R. Simmons. Point-based POMDP algorithms: improved analysis and implementation. In *UAI*, July 2005.

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

[Temizer *et al.*, 2010] Selim Temizer, Mykel J Kochenderfer, Leslie P Kaelbling, Tomás Lozano-Pérez, and James K Kuchar. Collision avoidance for unmanned aircraft using Markov decision processes. In *Proc. AIAA Guidance, Navigation, and Control Conference*, 2010.

[Thrun, 2000] S. Thrun. Monte Carlo POMDPs. In *NIPS*, pages 1064–1070, 2000.