Leveraging Temporal Structure in Task Specifications for POMDP Planning

Xinyu Liu, Eric Rosen, Suchen Zheng, Tyler Edwards, Ankit Shah, George Konidaris and Stefanie Tellex

***Abstract*— Planning sequential actions in a partially observable environment while satisfying temporal constraints is challenging yet an essential feature of many robotic applications. A constrained natural language command like "Find the new apartment complex while avoiding the park" is difficult for an autonomous delivery drone to understand. Previous planning methods chose to sacrifice generality for optimality and efficiency in large state-action spaces by using domain and task-specific action heuristics or used a full-width backup planner that did not scale well. We represent a set of constrained task specifications as linear temporal logic (LTL) expressions and present a new sampling-based POMDP planner, LTL-POMCP, that leverages structured constraints for efficient planning by constructing a shaping term to bias action selection towards achieving subgoals of an LTL. We augment an environment partially observable Markov decision process (POMDP) with an LTL task specification then use LTL-POMCP to efficiently solve the resultant composite POMDP. Quantitative results show that LTL-POMCP can efficiently solve LTL tasks in various domains, and scale to large environments. We demonstrate the first end-to-end system from temporally-constrained natural language to robot policies in partially observable maps in simulation with up to 50% improvement in wall clock time, and on a mobile manipulator in the real world.**

I. INTRODUCTION

Planning sequential actions in partially observable environments by following natural language commands that specify goals and path constraints is a challenging yet essential feature of robots interacting with humans. We consider a set of natural language commands specifying goals and temporal constraints that can be translated to a linear temporal logic (LTL) expression [1]. For example, the command "Find the new apartment complex while avoiding the park" instructs an autonomous delivery drone to reach the destination and avoid the park due to possible collisions with trees.

Previous work solved this LTL task with a full-width backup planner in small partially observable domains [2]. A sampling-based planner, like Partially Observable Monte-Carlo Planning (POMCP) [3], scales to larger domains but requires domain and task-specific action heuristics to constrain the search space.

Our key insight is to leverage temporal constraints for efficient planning via a shaping term to bias a sampling-based planner to sample trajectories progressing towards automatically extracted subgoals. We use a class of LTLs to represent constrained task specifications because they can be translated to deterministic finite automaton (DFA) [4] which provides structured information about subgoals and final targets. The DFA is used in conjunction with an environment POMDP to automatically construct an LTL-POMDP planning problem. To plan efficiently in large domains, we propose LTL-POMCP, a new sampling-based planner with a shaping term added to its action value estimates to bias the sampling of actions that likely induce DFA transitions leading to subgoals and high returns. During the Monte-Carlo simulation, besides tracking action value estimates, visitation counts of states and actions, LTL-POMCP counts the DFA transitions occurred after taking an action in the current state then uses them to augment action value estimates. Our approach automatically extracts subgoals from constrained task specifications thus does not need domain experts to engineer action heuristics and can scale to large domains.

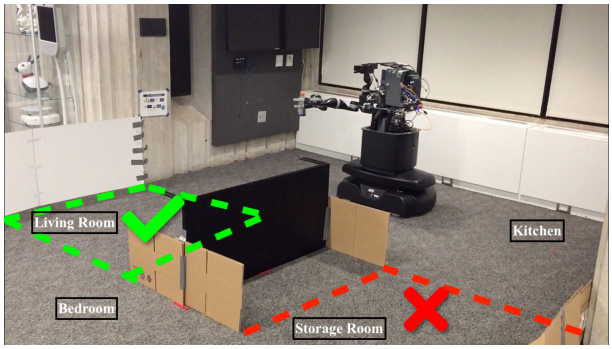


Fig. 1: A mobile manipulator follows the natural language command "Stay in the kitchen until find the water bottle then bring it to the bedroom and always avoid the storage room" in a partially observed domain. The command is translated to LTL formula $\texttt{kitchen}U(\texttt{water} \wedge F(\texttt{bedroom})) \wedge G(\neg\texttt{storage})$. The room locations are known, and the target object is partially observed.

Quantitative results show that LTL-POMCP is more generalizable across LTL task specifications than POMCP with domain and task-specific action heuristics [3] and more scalable than a planner based on value iteration [2]. We demonstrate the first end-to-end system from temporally-constrained natural language to policies in partially observed maps with up to 50% improvement of wall clock time in simulation and on a mobile manipulator in the real world.

The main contributions of this work are as follows,

- A new sampling-based POMDP planner, LTL-POMCP, that leverages temporal constraints from an LTL task specification, generalizes across environments and tasks, and scales to large domains.
- An end-to-end system from temporally-constrained natural language to robot behavior in partially observed robotic domains in simulation and real world, as shown in Fig. 2.

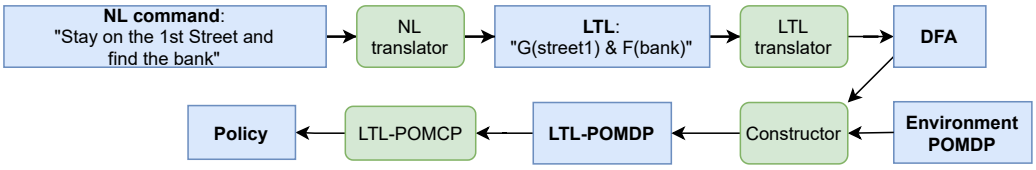


Fig. 2: **End-to-End System for the Robotic Domains.** Natural language is translated to an LTL then a DFA. The DFA and the environment POMDP are composed to construct an LTL-POMDP, which is solved by LTL-POMCP online.

II. Background

This section introduces background knowledge on linear temporal logic, deterministic finite automaton, POMDP, and terminologies used for the rest of the paper.

Linear Temporal Logic (LTL): We use the obligation class of linear temporal logic (LTL) [5] to specify constrained robotic tasks because they can represent both goals and temporal constraints. LTL has the following syntax:

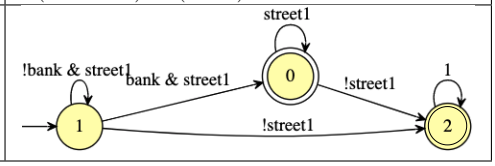
$$\phi := \sigma \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid X\phi \mid F\phi \mid G\phi \mid \phi U \psi, \quad (1)$$

where ϕ and ψ are LTL formulas; $\sigma \in \Sigma$ is an atomic proposition. $\neg, \wedge, \vee$ are logical connectives negation, conjunction and disjunction. LTL extends propositional logic with temporal operators, X (next), F (finally), G (always or globally) and U (until), being applied to future time steps. We evaluate the satisfaction of an LTL formula on an infinite sequence $w = w_0 w_1 \ldots$, where $w_i \in 2^{\Sigma}$. Thus for an LTL formula $X\phi$ to hold true at time i, ϕ must be true at the next time step $i+1$. $F\phi$ is true at time i if ϕ will be true at some future time $j \geq i$. $G\phi$ holds true if ϕ is true for the entire sequence w. $\phi U \psi$ is satisfied by w if ϕ holds true at least until ψ becomes true, which must happen at the current or a future time. The obligation class covers a wide range of realistic robotic tasks with goals and constraints. Table I shows an example of a such LTL task specification. For example, to satisfy $G(street1) \wedge F(bank)$, a robot needs to stay on the First Street and finally reach the bank. Kupferman and Vardi [6] showed that LTLs from the obligation class can be translated to deterministic finite automata (DFA).

Deterministic Finite Automaton (DFA): We use the Spot library [7] to translate an LTL formula to an equivalent DFA [8]. A DFA is a 5-tuple $\mathrm{D} = (\mathbf{Q}, \Sigma, \delta, q_0, \mathbf{F})$, where $\mathbf{Q}$ is a finite set of states; Σ is a finite set of atomic propositions; $\delta : \mathbf{Q} \times 2^{\Sigma} \to \mathbf{Q}$ is a deterministic transition function; $q_0 \in Q$ is the initial state; $\mathbf{F} = \mathbf{F}_{\text{success}} \bigcup \mathbf{F}_{\text{fail}} \subseteq \mathbf{Q}$ is a set of success and failure terminal states. A run on a finite sequence $w = w_0 w_1 \ldots w_n$ with $w_i \in 2^{\Sigma}$ produces a sequence of states $q_0 q_1 \ldots q_n$ with $q_t \in \mathbf{Q}$, where q_0 is the initial state, $q_n \in \mathbf{F}$ is a final state, and $q_{t+1} = \delta(q_t, w_t)$. Table I shows an example of an LTL formula and its corresponding DFA, whose initial state is 1, success state is 0, failure state is 2.

Environment POMDP: We model the environment as a Partially Observable Markov Decision Process (POMDP). A POMDP is defined by a 7-tuple $(\mathbf{S}, \mathbf{A}, \mathbf{O}, T, O, R, \gamma)$. The dynamics $T(s,a,s') = P(s'|s,a)$ and $R(s,a) = E[r|s,a]$ determine the distribution of the next state $s' \in \mathbf{S}$ and the step reward after taking action $a \in \mathbf{A}$ in state $s \in \mathbf{S}$. In POMDP, states cannot be fully observed. Instead the agent receives an observation $o \in \mathbf{O}$ based on an observation model $O(o|a,s') = P(o|a,s')$. A policy of a POMDP is defined by $\pi(h) = a$, where h is a history of actions and observations. Any POMDP has at least one optimal policy π^* that maximizes a value function $V^{\pi}(h) = E_{\pi}[\sum_{t}^{\infty} \gamma^{t-1} r_t | h]$. A belief state is a probability distribution over possible states given the history, $B(s,h) = P(s|h)$, and it is represented by a set of particles in this work. We define our environment POMDP to be generative such that given a transition (s,a,s'), we can sample from T, R and O to get the next state, an immediate reward and an observation.

TABLE I: An example of natural language, linear temporal logic (LTL) and Deterministic Finite Automaton (DFA).

Language	Stay on the First Street and find a bank.
LTL	$G(\texttt{street1}) \wedge F(\texttt{bank})$
DFA	

III. Related Work

A large body of research has studied navigation in fully-observable environments while satisfying goals and temporal constraints provided by an LTL formula [9, 10, 11, 12, 13]. We consider a more challenging partially observable setting, where an agent must actively plan to gather information. Previous work also proposed models to identify temporal constraints from demonstrations, but focused on specification inference in a fully observable domain, where the truth values of the propositions are unambiguously known [14, 15, 16, 17, 18, 19]. Icarte et al. [20] introduced reward machines to represent non-Markov reward decision processes (NMRDP), and introduced planning and reinforcement learning algorithms to compute policies that optimize the non-Markov reward. Camacho et al. [21] further showed that many formal logic languages, including a fragment of LTL, can be compiled into an equivalent reward machine. However, prior work in planning and reinforcement learning with reward machines has primarily focused on fully observable domains. LTL-POMDP generalizes planning and reinforcement learning with reward machines to partially observable domains.

Bouton et al. [2] solved LTL-POMDP problems with a full-width backup planner and an exact model of environment

in small domains. The planning could not scale because Bellman backups are intractable in large state-action spaces due to the curse of dimensionality and the curse of history. Instead of estimating the value function via iterative applications of the Bellman equation using an exact model, a sampling-based method, like POMCP [3], use Monte-Carlo simulations to estimate the values from interactions with a generative model of the environment. But POMCP requires domain and task-specific action heuristics to constrain the search space. These heuristics use observations received during simulations to help decide the best action to take next. To solve an LTL task in partially observable environment, Bradley et al. [22] assumed perfect local perception to partition the environment into known and unknown areas and did not maintain a probability distribution over states in the unknown area. Our new sampling-based POMDP planner automatically extracts subgoals from an LTL to guide the action selection and is applicable to LTL-POMDP problems with generative models and noisy sensors.

IV. LTL-POMCP Planning

This section provides technical details on how we augment an environment POMDP with a DFA to construct an LTL-POMDP and describes LTL-POMCP, a new sampling-based planner that leverages temporal structure provided by the DFA to solve LTL-POMDPs and how we translate constrained natural language commands to LTL expressions.

LTL-POMDP: We augment an environment POMDP with a DFA, so the resultant LTL-POMDP states are Markovian and encoding high-level subgoals and termination conditions of the task. LTL-POMDP $= (\tilde{\mathbf{S}}, L, \mathbf{A}, \mathbf{O}, \tilde{T}, \tilde{O}, \tilde{R}, \gamma)$, where $\tilde{\mathbf{S}} = \mathbf{S} \times \mathbf{Q}$ is a Cartesian product of environment POMDP and DFA states; $L : \mathbf{S} \to 2^{\Sigma}$ is a labeling function that maps environment POMDP states to atomic propositions. Because the environment POMDP states are partially observable, their corresponding DFA states are also partially observable. During planning, an LTL-POMCP agent maintains a belief over composite states $\tilde{s} = (s, q)$.

The transition probability of entering LTL-POMDP state $\tilde{s}' = (s', q')$ after taking action a from state $\tilde{s} = (s, q)$ is

$$\tilde{T}(\tilde{s}, a, \tilde{s}') = \begin{cases} T(s, a, s'), & \text{if } q' = \delta(q, L(s')) \\ 0, & \text{otherwise} \end{cases}. \quad (2)$$

As the example shown in the Table I, taking an action in the environment to reach a bank while staying on the First Street induces the DFA transition from the state $q = 1$ to the goal $q = 0$.

The observation model of LTL-POMDP is the same as the environment POMDP, i.e.

$$\tilde{O}(\tilde{s}, a, \tilde{s}') = O(s, a, s'). \quad (3)$$

The reward function is specified by a DFA transition from state q to q'.

$$\tilde{R}(\tilde{s}, a, \tilde{s}') = \begin{cases} r_{\text{goal}}, & \text{if } q' \in \mathbf{F}_{\text{success}} \\ r_{\text{fail}}, & \text{if } q' \in \mathbf{F}_{\text{fail}} \end{cases}, \quad (4)$$

Algorithm 1 LTL-POMCP Planner

procedure SIMULATE(s, h, $depth$)
  if $\gamma^{depth} < \varepsilon$ **then**
    return 0
  end if
  if $h \notin T$ **then**
    for $a \in A$ **do**
      $T(ha) \leftarrow (N_{init}(ha), V_{init}(ha), \emptyset,$ `e2freq = {}`$)$
    end for
    return ROLLOUT(s, h, $depth$)
  end if
  `e = q2edge[s.q]`
  $N(hbe) =$ `e2freq`$[e]$
  $a \leftarrow \arg\max_b V(hb) + \alpha\sqrt{\frac{N(h)}{N(hb)}} + \beta\frac{N(hbe)}{N(hb)}$
  $(s', o, r) \sim \mathbf{G}(s, a)$
  $R \leftarrow r + \gamma \cdot$ SIMULATE(s', hao, $depth + 1$)
  $B(h) \leftarrow B(h) \bigcup \{s\}$
  $N(h) \leftarrow N(h) + 1$
  $N(ha) \leftarrow N(ha) + 1$
  $e = (s.q, s'.q)$
  `e2freq`$[e] =$ `e2freq`$[e] + 1$
  $V(ha) \leftarrow V(ha) + \frac{R - V(ha)}{N(ha)}$
  return R
end procedure

where $r_{\text{goal}} \gg 0$ and $r_{\text{fail}} \ll 0$. We use $r_{\text{goal}} = 100$ and $r_{\text{fail}} = -100$ for the experiments presented in this paper.

LTL-POMCP: LTL-POMDPs model POMDP planning problems with temporally constrained task specifications. We introduce LTL-POMCP, a new sampling-based planner that leverages subgoals and termination conditions provided by LTLs to efficiently solve LTL-POMDP problems in large domains.

We adopt the POMCP algorithm [3] with two modifications. In addition to the estimated Q-values $\hat{Q}(h, a)$, state and action visitation counts $N(h)$ and $N(h, a)$, LTL-POMCP tracks $N(h, a, e)$, the frequencies of a preferred DFA transition occurred after taking action a in the current history state h during Monte-Carlo simulation. We then augment the original Q-value estimates (the first two terms) with a shaping term (the third term) as follows,

$$Q(h, a) = \hat{Q}(h, a) + \alpha\sqrt{\frac{\log N(h)}{N(h, a)}} + \beta\frac{N(h, a, e)}{N(h, a)}, \quad (5)$$

where $N(h, a, e)$ represents the number of times a preferred DFA transition e occurred after taking action a from the current history state h. In every non-terminal DFA state, there is a preferred DFA transition e leading towards a goal, and we can compute a mapping from states to preferred transitions `q2edge` via graph search in the DFA.

Intuitively, the shaping term approximates the proportion of trajectories that induce a preferred DFA transition after

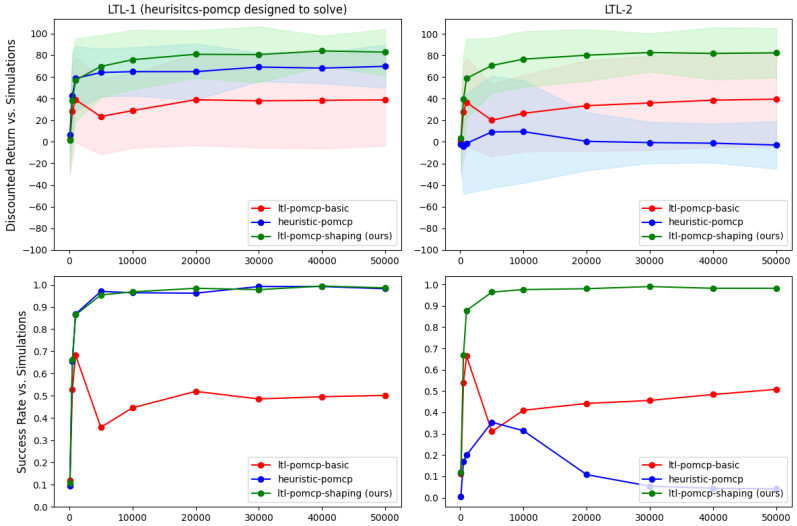


Fig. 3: **Generalization across LTLs.** Each column shows the performance of a different LTL task. The top plot shows the mean discounted return, and the bottom plot shows the success rate vs. simulations. Every data point is an average over 500 runs. *The action heuristics used by Heurisitc-POMCP are designed to specifically solve LTL-1.*

taking action a in the current state h. It encourages the agent to exploit actions that induce a favorable DFA transition leading to high returns, like the DFA transition from 1 to 0 in Table I. With the shaping term, more preferred DFA transitions occur during planning compared with the baseline methods.

α and β are coefficients that balance the exploration behavior induced by the second term and the exploitation behavior induced by the shaping term in Equation 5, and can be adjusted based on the complexity of a domain. A large α and a small β encourage the agent to explore actions and have good estimates of their values, but the algorithm takes more simulations to converge. By contrast, if the β value is relatively large, an agent acts quickly by exploiting the trajectories experienced in simulations that have high returns, and it may lead to undesired behaviors that violates an LTL specification. For the experiments in this paper, we use $\alpha = 100$ and $\beta \in [10, 100]$. The β value is chosen to balance the success rate and planning time.

Thus Equation 5 balances exploring less taken actions and exploiting actions that have led to a DFA transitions with high returns. The LTL-POMCP algorithm leverages high-level subgoals encoded in the DFA and automatically favors the transitions leading to the DFA goal state without explicitly constructing preferred action heuristics [3]. The difference between LTL-POMCP and POMCP are boxed is in Algorithm 1. We refer readers to the POCMP paper [3] for the complete algorithm.

Translating Natural Language to LTL: The language model first uses a pretrained name entity recognizer (NER) [23] to replace all landmark names from a natural language command by place holders, then feeds the masked language into a sequence-to-sequence (Seq2Seq) model with LSTM cells, and finally substitute the landmark names back in the output LTL expression. With the help of a pretrained NER, we only need to train the Seq2Seq model to memorize LTL templates, not landmark names.

TABLE II: Comparison of success rates in RockSample domains of increasing complexity.

	RS(3,3)	**RS(5,5)**	**RS(7,7)**	**RS(9,9)**
Ours	100%	100%	99%	96% ($\sim$ 2hrs)
LTL-SARSOP [2]	100%	100%	100%	0% ($>$ 24hrs)

V. Experiments

The aim of our experiments is to test the hypotheses that the LTL-POMCP planner is more general than POMCP used with domain and task-specific action heuristics [3] and more scalable than using a full-width backup planner to solve LTL-POMDP problems [2] in the RockSample domain, and demonstrate the end-to-end system from temporally-constrained natural language to policies in a partially observed robotic domains in simulation and real world. The implementation uses the pomdp_py framework [24].

RockSample: A RockSample problem models a rover exploring and sampling two types of rocks [25]. $RS(n,k)$ has k rocks randomly placed in an $n \times n$ grid with an exit area on the right. The agent and rock locations are known; the rock types are partially observable. An agent in $RS(n,k)$ has $k+5$ actions (i.e. 4 move, 1 pick up and k sensing actions, one for each rock), 2 observations of rock types (i.e. good, bad) and deterministic transitions. The sensing accuracy decreases exponentially as the distance to a rock increases. We use an uniform initial belief over the rock types.

To show LTL-POMCP is more generalizable across LTL task specifications than POMCP, which uses domain and

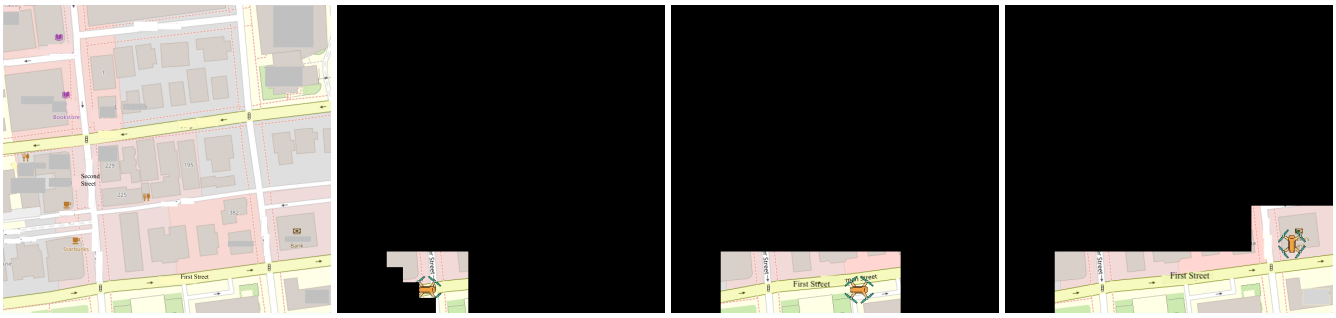


Fig. 4: Image 1 shows the fully observed map. Images 2-4 show points on a trajectory that follows the natural language command "Always stay on the First Street and find the bank." The fog of war effect shows agent's partial observability of the bank. The agent has perfect knowledge of where streets are.

task-specific action heuristics [3] (Heurisitc-POMCP), we compare the discounted return and success rate of the output policies satisfying given LTL expressions. LTL-POMCP are given,

- LTL-1: $F(good \wedge F(exit)) \wedge G(\neg bad) \wedge (\neg exit U good)$,
- LTL-2: $F(bad \wedge F(exit)) \wedge G(\neg good) \wedge (\neg exit U bad)$.

The first task requires the robot to pick up a good rock then go to exit area while always avoiding bad rocks. The second task requires the robot to pick up a bad rock then exit while avoiding good rocks. The until clauses in both LTLs specify constraints that prevent the behavior of exiting without a desired rock. Using LTLs to specify tasks this way is different from using numerical rewards. The agent only needs to pick up 1 desired rock then exit to satisfy the 2 LTL formulas above.

The performance of LTL-POMCP with the shaping term (LTL-POMCP-shaping) are compared with Heurisitc-POMCP and regular POMCP provided with an LTL reward (LTL-POMCP-basic) in Fig. 3. LTL-POMCP-shaping performs better in terms of discounted returns and success rate. During planning the shaping term encourages exploiting the trajectories that have led to subgoals in the DFA and helps LTL-POMCP-shaping converge to success rate of nearly 1 with 10,000 simulations for both tasks.

Heuristic-POMCP uses action heuristics defined to solve LTL-1, and its performance is comparable with LTL-POMCP-shaping. But because the action heuristics that are hard-coded to solve LTL-1 prefer the undesired behaviors that violate LTL-2, Heurisitc-POMCP performs worse than LTL-POMCP-basic. More specifically, the action heuristics encourage the robot to pick up good rocks which conflicts with the LTL-2 constraint $G(\neg good)$. As the number of simulations increases, the output policy of Heurisitc-POMCP converges to only moving and checking without every picking up a rock to alleviate the conflict between the action heuristics and LTL-2 constraints. LTL-POMCP-basic does not use any structure information from task specifications, thus it has the largest variances among the three methods. The automatic extraction of preferred transitions from DFA by LTL-POMCP-shaping and the hand-crafted action heuristics used by Heurisitc-POMCP reduce variances of Monte-Carlo simulations.

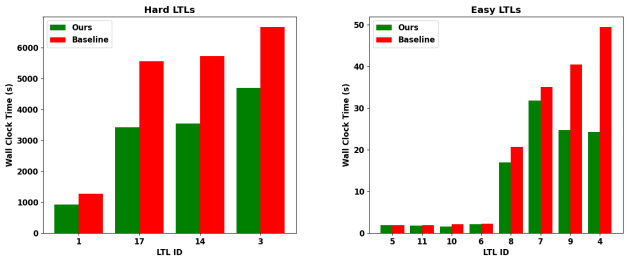


Fig. 5: Wall clock time comparison of LTL-POMCP-shaping (ours in green) and LTL-POMCP-basic (baseline in red) in PO-Map domain for solving LTL tasks of various difficulties.

TABLE III: Examples of LTL task specifications and the corresponding natural language commands in PO-MAP domain.

Natural Language Command	**LTL Expression**
Find cafe.	$F(\texttt{cafe})$
Always stay on 1st Street and find bank.	$G(\texttt{st1}) \wedge F(\texttt{bank})$
Always stay on 2nd Street and first find cafe then find store.	$G(\texttt{st2}) \wedge F(\texttt{cafe} \wedge F(\texttt{store}))$
Find store and always avoid 1st Street.	$G(\neg\texttt{st1}) \wedge F(\texttt{store})$
Stay on 1st Street until find bank.	$\texttt{st1} U \texttt{bank}$
Stay away from 1st Street until find store.	$(\neg\texttt{st1}) U \texttt{store}$

LTL-POMCP is also more generalizable than Heurisitc-POMCP across various domains as we also apply the same planning algorithm to solve a partially observable robotic navigation problem while the action heuristics defined for a RockSample problem do not work for other domains.

To show that LTL-POMCP is more scalable than using a full-width backup planner to solve LTL-POMDP problems [2], we compare the success rate as the domain becomes more complex, i.e. larger grids with more rocks. An full-width planner SARSOP [26] provided with sparse LTL rewards (LTL-SARSOP) cannot produce a policy within a 24 hour time limit for $RS(9,9)$. LTL-POMCP-shaping can constantly solve the problem over 96% success rates within 2 hours, and its speed can be further improved by parallelizing the Monte-Carlo simulations.

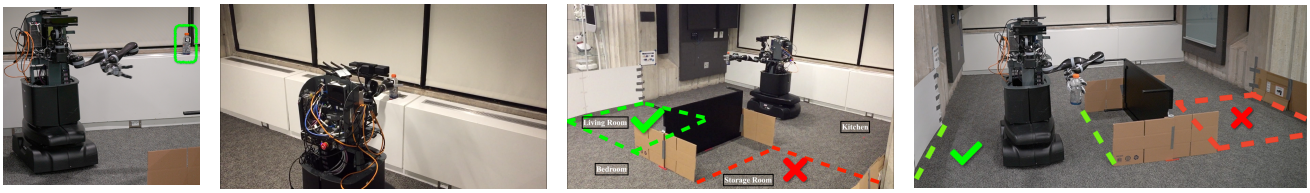


Fig. 6: The images show a mobile manipulator searching for the target object, grasping, finding paths after retrieving the object, and arriving at the destination while satisfying constraints by the natural language command "Stay in the kitchen until find the water bottle then bring it to the bedroom and always avoid the storage room".

We have shown the generalizability and scalability of LTL-POMCP compared to two prior approaches [3][2]. Now we demonstrate an application of LTL-POMCP planner in an end-to-end system from temporally-constrained language to policies in a partially observable robot navigation domain.

Partially Observable Robotic Navigation Domain: In the partially observable map (PO-Map), the locations of major landmarks, e.g. streets, are known, and the locations of small landmarks, e.g. cafes, stores and banks, are partially observed. It mimics the real-world scenarios where some places, like a new apartment complex, are not stored in a map database, and an agent, like an autonomous drone, needs to find them by following natural language commands with temporal constraints from humans. In this experiment, we consider 5 landmarks in a 20×20 grid. The goal is to compute a policy to reach the destination while satisfying a temporally-constrained natural language command that can be translated to an LTL from the obligation class. Table III shows examples of natural language commands and their corresponding LTLs. The state contains the agent's pose and the location of each target landmark. The initial belief is uniformly distributed over possible target locations, which can be constrained by an LTL. The agent can rotate in 4 cardinal directions or move forward. After taking an action, the agent receives a noisy observation of whether the target landmark is in its fan-shaped sensor range. The sensor has a depth of 2 and a 95% accuracy. We use a deterministic transition function and a non-deterministic observation model for computational reasons. They are also realistic assumptions of the drones because existing drones can reliably move around the environment but have less reliable sensors. There is a $+100$ reward for completing a given task while satisfying all its constraints and a -100 reward for violating any constraints. The executions terminates if the agent violates a constraint. The discount factor $\gamma = 0.98$. If the agent does not reach the destination within 50 steps, it receives no rewards, and this trial is counted as a failure.

Based on the planning time observed in the RockSample domain, LTL-SARSOP cannot solve $RS(9,9)$ in a timely fashion. We therefore only tested sampling-based planners in PO-Map due to limited computing resources. We measured the wall clock time for both LTL-POMCP and the baseline when they reach a success rate of 1. As shown in Fig. 5, LTL-POMCP-shaping runs faster than the basic POMCP without the shaping term in Equation 5 (LTL-POMCP-basic) with an average improvement of 25.77% over 17 LTL task specifications. The planning times vary for different tasks because some LTLs restrict the search space and make the problem easier to solve. For example, LTLs 5, 6, 10, 11 take less than 2 seconds to solve because their constraints restrict the search space to the Second Street, which occupies only 1/4 of the entire grid.

Mobile Manipulation: To demonstrate the ability of LTL-POMCP planning in the real world, we deployed the planner on a Kinova Movo robot, a mobile manipulator with a 7-DoF JACO arm. In a household environment, the robot must follow natural language commands to find objects and bring them to certain rooms while satisfying path constraints. The robot has perfect knowledge of the room locations, and a partial observability of whether a target object is in its sensor range with 95% accuracy. It can execute 6 discrete actions, i.e. rotate in 4 cardinal directions, move forward and grasp. When it decides to grasp an object, the robot executes a predefined manipulation skills. We tested 5 natural language commands in a 6×6 environment with 1536 states and varying initial poses of the robots. Fig. 6 shows the execution of one command "Stay in the kitchen until find the water bottle then bring it to the bedroom and always avoid the storage room.". The video recording of the robot demonstration can be found online [1].

Language Model: The NER we used to recognize landmarks and objects in natural language commands was pre-trained on a very large dataset. Thus the language model can recognize places and objects unseen in the training set. It took 54 seconds, 421 data points and 5 epochs to train the Seq2Seq model to achieve 100% accuracy on the joint test set of PO-Map and the mobile manipulation domain. The average inference time for the natural language commands is 0.2528 ± 0.0098 seconds.

VI. CONCLUSIONS

We present a natural way to leverage structure from challenging task specifications with temporal constraints via automatic extraction of subgoals from an LTL and using a shaping term to bias action selection in a new sampling-based planner. This powerful idea of leveraging structure from constrained natural language commands is not limited by specific logic forms or planning algorithms. We will investigate using different semantic representations and planners.

[1] https://youtu.be/pakb8BvNGfo

References

[1] A. Pnueli, "The temporal logic of programs," in *IEEE Symposium on Foundations of Computer Science*, 1977.

[2] M. Bouton, J. Tumova, and M. J. Kochenderfer, "Point-based methods for model checking in partially observable markov decision processes," in *AAAI*, 2020.

[3] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Advances in neural information processing systems*, 2010.

[4] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM journal of research and development*, vol. 3, no. 2, pp. 114–125, 1959.

[5] Z. Manna and A. Pneuli, "A hierarchy of temporal properties," in *ACM symposium on Principles of distributed computing*, 1990.

[6] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[7] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2.0 — a framework for LTL and ω-automata manipulation," in *Automated Technology for Verification and Analysis*, 2016.

[8] J. R. Büchi, "On a decision method in restricted second order arithmetic," in *The collected works of J. Richard Büchi*. Springer, 1990, pp. 425–435.

[9] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *International Conference on Robotics and Automation*, 2005.

[10] M. L. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan, "Environment-independent task specifications via GLTL," *arXiv preprint arXiv:1704.04341*, 2017.

[11] Y. Oh, R. Patel, T. Nguyen, B. Huang, E. Pavlick, and S. Tellex, "Planning with state abstractions for non-markovian task specifications," in *Robotics: Science and Systems*, 2019.

[12] M. Berg, D. Bayazit, R. Mathew, A. Rotter-Aboyoun, E. Pavlick, and S. Tellex, "Grounding language to landmarks in arbitrary outdoor environments," in *International Conference on Robotics and Automation*, 2020.

[13] R. Patel, E. Pavlick, and S. Tellex, "Grounding language to non-markovian tasks with no supervision of task specifications," in *Robotics: Science and Systems*, 2020.

[14] M. Vazquez-Chanlatte, S. Jha, A. Tiwari, M. K. Ho, and S. Seshia, "Learning task specifications from demonstrations," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[15] A. Shah, P. Kamath, J. A. Shah, and S. Li, "Bayesian inference of temporal task specifications from demonstrations," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[16] Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta, "Temporal logic inference for classification and prediction from data," in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*. ACM, 2014, pp. 273–282.

[17] Z. Kong, A. Jones, and C. Belta, "Temporal logics for learning and detection of anomalous behavior," *IEEE Transactions on Automatic Control*, vol. 62, no. 3, pp. 1210–1222, 2017.

[18] R. Toro Icarte, E. Waldie, T. Klassen, R. Valenzano, M. Castro, and S. McIlraith, "Learning reward machines for partially observable reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[19] G. Chou, N. Ozay, and D. Berenson, "Explaining multi-stage tasks by learning temporal logic formulas from suboptimal demonstrations," in *Robotics: Science and Systems*, 2020.

[20] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *ICML*, 2018.

[21] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "LTL and beyond: Formal languages for reward function specification in reinforcement learning." in *the Twenty-Eighth International Joint Conference on Artificial Intelligence*, vol. 19, 2019, pp. 6065–6073.

[22] C. Bradley, A. Pacheck, G. Stein, S. Castro, H. Kress-Gazit, and N. Roy, "Learning and planning for temporally extended tasks in unknown environment," in *International Conference on Robotics and Automation*, 2021.

[23] M. Honnibal, I. Montani, S. V. Landeghem, and A. Boyd. (2020) spaCy: Industrial-strength natural language processing in python. [Online]. Available: https://doi.org/10.5281/zenodo.1212303

[24] K. Zheng and S. Tellex, "pomdp_py: A framework to build and solve pomdp problems," in *ICAPS 2020 Workshop on Planning and Robotics (PlanRob)*.

[25] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *The Conference on Uncertainty in Artificial Intelligence*, 2004.

[26] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based POMDP planning by approximating optimally reachable belief spaces." in *Robotics: Science and systems*, 2008.