Planning in Dynamic Environments using Evolving Time-Indexed Graphs

Vaibhav V. Unhelkar*[a], Rares-Darius Buhai*[a] and Julie A. Shah*
*Computer Science and Artificial Intelligence Lab,
Massachusetts Institute of Technology, Cambridge, Massachusetts 02139
Email: unhelkar@csail.mit.edu, rbuhai@mit.edu, julie_a_shah@csail.mit.edu

***Abstract*—In several scenarios of human-robot interaction, the robot operates in a dynamic environment and has limited knowledge of its environment as well as the surrounding humans. In such cases modeling and reasoning over time can be beneficial, since both the environment and robot's model of the environment changes over time. Here, we discuss planning with explicit consideration of time using "evolving time-indexed graphs," and study it in the context of robot navigation among humans. Following, the safe-interval path planning (SIPP) algorithm, we pose the planning problem as graph search with time included as part of the state space. We propose forward safe-interval planner (FSIP), an incremental extension of SIPP, for applications where the robot's model of the environment evolves during execution. Initial evaluations of the algorithms for path planning are carried out using simulation within Robot Operating System (ROS). We discuss the performance of the incremental approach, its limitations and our on-going work.**

I. Introduction

Human-robot interaction often takes place in dynamic, partially known and uncertain environments. Modeling humans — their beliefs, intentions and plans — from first principles or using data-derived models is no easy task. Further, behavior of the interacting agents is inter-dependent and often changes during the course of the interaction. These aspects make planning for human-robot interaction (HRI) significantly challenging. Here, we consider explicit consideration of time in the planning process to address some of the aforesaid challenges.

Timing has been identified as one of the critical components for designing and affecting human-robot interaction. Effect of timing has been studied in various contexts of HRI, including communication (spoken dialogue, non-verbal gestures) and synchronization in collaborative tasks [2]. Moreover, considerations of timing and anticipatory behavior are essential in time-critical tasks, such as, those encountered in disaster response and collaborative manufacturing.

The concept of timing has also been explored within planning for HRI. Broz et al. [1] include time within the state of partially observable Markov decision processes (POMDP) to model human-robot interaction tasks. Incorporating time allows for finer modeling of the interaction and potentially better plans (higher expected reward); however, it also results in a significantly larger state space and correspondingly higher computational costs. One method to tackle this computational challenge is to use state aggregation, e.g., state aggregation using expected rewards for POMDPs [1]. However, despite the state aggregation - due to the dual challenge of modeling uncertainty as well as time - the computational costs of planning using POMDPs with time may remain prohibitively large.

An alternate approach is to pose planning as an instance of graph search. Phillips and Likhachev [8] model path planning in dynamic environments using graphs that include the time dimension as part of the state space. This allows for a way to utilize any predictive information regarding humans and dynamic obstacles in the environment. The computational challenge due to explosion of state space is resolved by using an approach similar to state aggregation; contiguous states with identical non-temporal components are merged into safe time intervals. By reasoning over these safe time intervals, the algorithm safe-interval path planner (SIPP) can explicitly reason in time [8]. Note that though planning through graph search does not consider uncertainty, it is a computationally attractive alternative for reasoning in time. Further, anytime approaches to graph search have been developed to comply with hard time constraints of robot navigation [7].

Due to its ability to explicitly model time while still being computationally feasible, we further explore use of graph search for planning in HRI applications. We define "time-indexed graphs" as graphs whose nodes are characterized by a non-temporal component and a time index. Any predictive information available regarding the future can thus be encoded as part of the state space. For instance, in the path planning application this corresponds to prediction of trajectory of the dynamic obstacles. As discussed above, SIPP and Anytime-SIPP algorithms provide computationally attractive options for reasoning over such time-indexed graphs.

However, one key aspect of various robotic applications is the limited and evolving knowledge of the future. For instance, in path planning applications a robot's knowledge of the trajectories of dynamic obstacles is limited. In practice, this anticipatory information may be obtained through a motion prediction algorithm, such as [5, 6, 9]. Further, this predictive knowledge to be encoded as part of a time-indexed graph changes during execution as more information becomes available to the robot. This motivates the need of "evolving time-indexed graphs" — graphs that not only represent time but also allow for a changing topology — as well as novel incremental algorithms that reason over them. Here, we formally define evolving time-indexed graph and discuss our work towards developing incremental planning algorithms for graph search over them. Following SIPP we explore the domain of path planning in dynamic environments - an application of interest

[a]These authors contributed equally to this work.

for physical HRI. Results from simulation of the incremental approach for path planning in dynamic environments are presented, and its limitation and our on-going work is discussed.

II. Evolving Time-Indexed Graphs

We define "time-indexed graph" as a directed graph $G = (S, E)$ that satisfies the following properties

- states $s \in S$ include a time-independent component x and a time index t, i.e., $s \equiv (x, t)$, and
- edges $e \in E$ only exist between the nodes of the type $s = (x, t)$ and $s' = (x, t+1)$, and have weight as one.

These properties allow the time-indexed graph to explicitly encode any available temporal information. The edges correspond to traversing the graph in time, and the shortest path on this graph corresponds to minimizing travel time between the start and goal configurations. Potential actions a at a state s are denoted by the set $A(x)$, which depends only on the time-independent component. Feasibility of an action $a \in A(x)$ depends on the graph topology and the duration of the action. For an action to be feasible, all states along the action path must be present in the graph.

However, this additional expressive capability to represent temporal information comes at a computational cost. To see this note that a directed graph $G_x = (X, E_x)$ used for non-temporal planning can be converted to a time-indexed graph by augmenting the state $x \in G_x$ with a time-index, i.e, $s = (x, t)$. The state space of the corresponding time-indexed graph will be $|X|N_t$, where N_t denotes the time horizon of the problem. Thus, a planner that reasons over time-indexed graphs has to typically search over a significantly larger state space.

Typically the temporal information to specify such a time-indexed graph will be made available using a prediction algorithm. For instance, a motion prediction algorithm that predicts the trajectory of surrounding humans. These predictions evolve as more information arrives and robot updates its models. This requires us to consider a graph with not only a temporal component but also evolving topology. We denote time-indexed graph with changing topology as "evolving time-indexed graph." The change in topology may reflect in the form of addition and deletion of states and/or edges, and can be used to reflect changes in predictive information. Further extensions of this formalism, to incorporate uncertain execution, can be obtained by modeling actions with bounds on their execution time.

III. Planning over Evolving Time-Indexed Graphs

In theory existing graph-search algorithms, such as A* and its variants, can be used for planning over (evolving) time-indexed graphs. However, as discussed above, inclusion of the time index results in an increase in the size of the state space by a factor of the time-horizon of the problem. This may render applicability of classical graph search approaches in several applications computationally expensive. To alleviate this computational challenge, Phillips and Likhachev [8] have developed safe-interval path planner (SIPP), an algorithm for search over time-indexed graphs. This algorithm and its anytime variant [7] exploit the existence of safe intervals in time-indexed graph. This is possible due to the existence of several nodes with identical time-independent component x but different time indices. Merging such nodes into intervals results in reduction of state space, but requires specialized algorithms (such as, SIPP and anytime-SIPP) to reason over them.

Algorithm 1 FSIP

```
1: procedure MAIN( )
2:     g(s_goal) = v(s_goal) = ∞; bp(s_goal) = null
3:     g(s_start) = 0; v(s_start) = ∞; bp(s_start) = null
4:     OPEN = ∅
5:     Insert s_start into OPEN with key(s_start)
6:     COMPUTESHORTESTPATH()
7:     while s_goal ≠ s_start do
8:         s_start = current state of the robot
9:         if changes in edge costs detected then
10:            Mark all nodes that are not descendent of s_start as unexplored
11:            for all states s with changes do
12:                Update cost of all edges connected to s
13:                UPDATESTATE(s)
14:            COMPUTESHORTESTPATH()
15: procedure KEY(s)
16:     return [min(g(s), v(s)) + h(s); min(g(s), v(s))]
17: procedure UPDATESTATE(s)
18:     if s was not visited before then
19:         g(s) = v(s) = ∞; bp(s) = null
20:     if s ≠ s_start then
21:         bp(s) = arg min_{s' ∈ GETPREDECESSORS(s)} v(s') + c(s', s)
22:         g(s) = v(bp(s)) + c(bp(s), s)
23:     if s ∈ OPEN then
24:         Remove s from OPEN
25:     if v(s) ≠ g(s) then
26:         Insert s in OPEN with key(s)
27: procedure COMPUTESHORTESTPATH( )
28:     while key(s_goal) < min_{s ∈ OPEN} key(s) OR v(s_goal) ≠ g(s_goal) do
29:         Remove s with the smallest key(s) from OPEN;
30:         if v(s) > g(s) then
31:             v(s) = g(s);
32:             for each s' ∈ GETSUCCESSORS(s) do
33:                 UPDATESTATE(s')
34:         else
35:             v(s) = ∞
36:             for each s' ∈ GETSUCCESSORS(s) ⋃ {s} do
37:                 UPDATESTATE(s')
```

The existing interval-based algorithms for time-indexed graphs provide a computationally attractive approach to reason over them; however, they need to search from scratch once the graph topology changes. Since, changes may often occur in the predictions available to a robot and correspondingly in the topology of the time-indexed graph, re-planning from scratch might be computationally limiting during execution. This calls for development of incremental approaches for evolving time-indexed graphs, analogous to incremental approaches such as LPA* and D* [3, 4] for classical dynamic graphs. Here, we propose one such incremental approach forward, safe-interval planner (FSIP). FSIP is developed as a hybrid of the LPA* and SIPP graph search algorithms with specialized features for evolving time-indexed graphs, and it can re-use its prior planning process when predictive information changes.

Intuition: Algorithms 1-2 describe the algorithm and the required functions. Similarly to SIPP, FSIP first converts the underlying time-indexed graph to a time-interval based representation. Having generated the graph over safe time intervals, we perform a search on the time-interval graph using a modified LPA* algorithm to allow for efficient replanning. Note that the LPA* algorithm is designed for graphs with changing topology but fixed start state; hence, we modify the

Algorithm 2 FSIP : Computing Successors and Predecessors

```
36: procedure GetPredecessors(s)
37:     predecessors= ∅
38:     for all actions a ∈ A(s) do
39:         x' ← action a reversibly applied to x(s)
40:         t_a ← time to execute a
41:         for all safe interval i ∈ x' do
42:             s' ← state with configuration x', interval i
43:             t_start ← v(s') + t_a
44:             t_end ← endTime(s') + t_a
45:             if startTime(s) > t_end OR endTime(s) < t_start then
46:                 continue
47:             t ← earliest feasible arrival time at s from s'
48:             if t does not exist then
49:                 continue
50:             Insert s' into predecessors
```

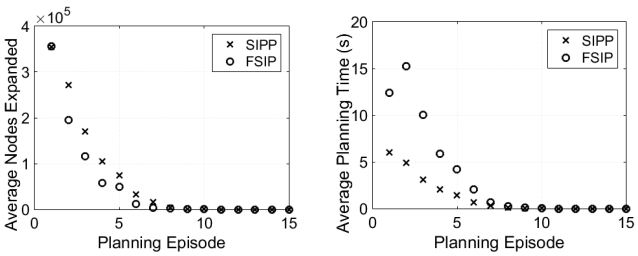
search to allow for changes in start state of the robot.

We use SIPP's GetSuccessors(s) procedure and an analogous GetPredecessors(s) procedure to determine neighbors of each state. The first search by the algorithm is identical to SIPP, except for the computation of key and additional maintenance of the one-step look-ahead cost for each explored node. The underlying time-indexed graph evolves as the available predictive information changes, resulting in the addition and removal of states. The algorithm mimics these changes in the time interval-based graph representation, and adds and removes time intervals as appropriate. The use of modified LPA* allows FSIP to explore only the subset of states that change due to novel predictive information, in order to efficiently generate the new plan.

Once the start state changes, we prune the existing search tree and only maintain the sub-tree originating from the current start state. This allows FSIP to maintain the previous search process, while searching from a modified start state. However, this also renders the algorithm incomplete. Further, if the start state does not change the algorithm reduces to the LPA* algorithm and regains the associated properties - completeness and optimality. An alternate approach is to use D* lite algorithm for incremental search with changing start state; however, this requires specification not only the goal configuration x_g but also goal time t_g. We are concurrently developing an incremental approach using D* and evolving time-indexed graphs designed for applications where goal time specification is available.

IV. Preliminary Evaluation and Discussion

To evaluate the planning performance of FSIP, we carried out simulation of path planning scenarios in dynamic environments using Robot Operating System (ROS). The performance of FSIP was compared against that of SIPP, which replanned from scratch once novel predictions were made available to the algorithm. Prediction of motion of dynamic obstacles in the environment was made available to the algorithms at the start of the problem. The ground truth motion of the dynamic obstacles changed every 500 time-steps, and correspondingly the predictions were updated. Motion primitives of the robot (used to represent action space of the time-indexed graph) and the static map of the environment were pre-specified. Simulation were carried out for twenty such randomly generated scenarios with one dynamic obstacle in the environment, out of which seventeen scenarios resulted in a feasible solution.



(a) Nodes Expanded (b) Planning Time

Fig. 1: Comparison of FSIP and SIPP for path planning in dynamic environments. Planning episode 1 denotes the planning at the start of the problem, remaining planning episodes correspond to the case when predictions of the dynamic obstacle are updated.

Figure 1 summarizes the results of simulations for which a feasible solution was available. In these simulations, both the algorithms found a solution with comparable path cost. By maintaining the sub-tree originating from the current start node, FSIP needs to expand a smaller number of nodes during the subsequent replans (planning episode 2 onwards). However, the incremental approach requires higher planning time than planning from scratch. This happens since the incremental approach incurs an additional computational cost of updating its graph representation when new predictions are available.

In our on-going work, we are exploring modifications to our incremental planner such that performance benefits observed in the number of nodes expanded are also translated to planning time. The utility of an incremental planning approach depends on the amount of changes made to the graph. Further, online prediction algorithms used for motion prediction typically make frequent but incremental updates to their predictions. Hence, we posit that a robot can realize benefits in computational time by using a mixed approach which involves incremental planning if the changes to the graph are less and replanning from scratch otherwise. Lastly, we aim to evaluate the incremental approach in closed loop with an online prediction algorithm.

References

[1] Frank Broz, Illah R Nourbakhsh, and Reid G Simmons. Planning for Human-Robot Interaction Using Time-State Aggregated POMDPs. In *AAAI*, volume 8, pages 1339–1344, 2008.

[2] Guy Hoffman, Maya Cakmak, and Crystal Chao. Workshop on timing in human-robot interaction. In *HRI*. ACM, 2014.

[3] Sven Koenig and Maxim Likhachev. D* lite. In *AAAI*, 2002.

[4] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning A*. *Artificial Intelligence*, 155(1):93–146, 2004.

[5] Thibault Kruse, Amit Kumar Pandey, Rachid Alami, and Alexandra Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013.

[6] Markus Kuderer, Henrik Kretzschmar, Christoph Sprunk, and Wolfram Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *R:SS*, 2012.

[7] Venkatraman Narayanan, Mike Phillips, and Maxim Likhachev. Anytime safe interval path planning for dynamic environments. In *IROS*, pages 4708–4715. IEEE, 2012.

[8] Mike Phillips and Maxim Likhachev. SIPP: Safe interval path planning for dynamic environments. In *ICRA*. IEEE, 2011.

[9] Brian D Ziebart et al. Planning-based prediction for pedestrians. In *IROS*, pages 3931–3936. IEEE, 2009.