Real time planning in structured environments via efficient path compression

Author Names Omitted for Anonymous Review. Paper-ID [add your ID here]

***Abstract*—The abstract goes here.**

I. Introduction

write intro

II. Related work

Include ICAPS paper [**?**]

Relevant papers from our lab: [**?**] [**?**]

III. Problem definition

talk about states vs. configurations

Let G^{full} be a discrete set of configurations corresponding to the set of initial object poses on the conveyor belt that the robot can perceive when the object comes in its field of view. In addition, let s_{home} be a robot configuration corresponding to the robot's initial pose before object grasping is performed. The objective, following the set of assumptions we will shortly state, is to enable planning to any goal pose $g \in G^{\text{full}}$ in bounded time t_{bound} regardless of the current state of the system. More specifically, the perception system is setup such that it sends updated pose estimates on the fly as the object moves along the conveyor belt and as the robot approaches the object.

Before stating our assumptions about the system, let us introduce several definitions and notations:

Definition 1. *A goal state $g \in G^{\text{full}}$ is said to be* reachable *from a state s if there exists a path from s to g and it can be computed in finite time.*

Given a state s we denote the set of all goal states that are reachable from s as $G^{\text{reach}}(s)$ and we say that $G^{\text{reach}}(s)$ is *reachable* from s.

We make the following assumptions about the system.

A1 The goal set G^{full} is reachable from the start state s_{home}. Namely, $G^{\text{reach}}(s_{\text{home}}) = G^{\text{full}}$.

A2 Given a path $\Pi = \{s_0, \ldots, s_k\}$ s.t. $s_0 = s_{\text{home}}$ and $s_k \in G^{\text{full}}$, we have that $G^{\text{reach}}(s_{i+1}) \subset G^{\text{reach}}(s_i)$. Namely, the reachable set of goals for a state on the path is a subset of the reachable set of every other state on that path that exists before it.

A3 G^{full} would accommodate for an error ϵ in the initial pose estimate g_{init} of the perception system. Each subsequent estimate g_{next} will be within the ϵ window around g_{init} (in retrospect because the conveyor is moving). what does this mean? how is this used?

A4 There exists a replan cutoff time t_{rc} from when the robot starts moving, after which the perception system does not update the object's pose.

A5 If the robot starts moving at $t = 0$ then for any time $t < t_{\text{rc}}$, the environment is static. Namely, objects on the conveyor belt cannot collide with the robot during that time.

Assumptions **A1**-**A2** correspond to properties of the environment *without* taking the objects on the conveyor belt into account while Assumptions **A3**-**A5** correspond to properties of the environment that account for the perception system and the objects on the conveyor belt.

IV. Algorithmic framework

Our approach for bounded-time planning relies on a *preprocessing* stage that allows to efficiently compute paths in a *query* stage to any goal state (under Assumptions **A1**-**A5**). Before we describe our approach, we start by describing a naïve method that solves the aforementioned problem but requires a prohibitive amount of memory. This can be seen as a warmup before describing our algorithm which exhibits the same traits but doing so in a memory-efficient manner (Sec. **??**).

A. Straw man approach

We divide the preprocessing stage into two steps. In the first step, we compute from s_{home} a path Π_g to every goal state $g \in G^{\text{full}}$ (such a path exists following **A1**). This allows us to start executing a path once the perception system gives its initial pose estimate. However, we need to allow for updated pose estimations while executing a path Π_g. Following **A4** and **A5**, this only needs to be done up until time t_{rc}. Thus we discretize each path such that consecutive states along the path are no more than δ_t time apart. This will allow the system to start executing a new path within $t_{\text{bound}} + \delta_t$ after a new pose estimation is obtained from the perception system. Finally, in the second step of the preprocessing stage, for each path we compute a new path to each goal state for every state along the path that is less then t_{rc} time from s_{home}. For a visualization, see Fig. 1. The outcome of the preprocessing stage is a set of precomputed collision-free paths starting at states that are at

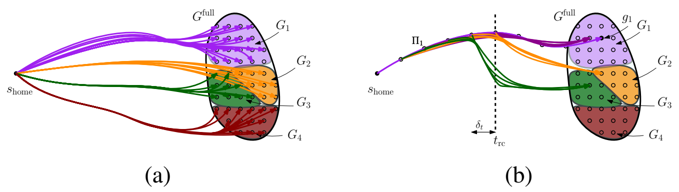

Fig. 1: The two steps of the preprocessing stage of the straw man algorithm. In both figures, paths that are very similar (and their corresponding goal states) are depicted using the same color. (a) First step—the algorithm computes a path from s_{home} to every state in G^{full}. (b) Second step—the algorithm computes for each path a new path to each goal state for every state along the path. Here, one representative path is depicted together with three different goal states.

most δ_t from s_{home} and end at states goal states. The paths are stored in a lookup table that can be queried in $O(1)$ time.

In the query stage we obtain an estimation g of the goal pose by the perception system. The algorithm then retrieves the path $\Pi(s_{\text{home}}, g)$ from s_{home} to g and the robot starts executing $\Pi(s_{\text{home}}, g)$. For every new estimation g' of the goal pose by the perception system, the algorithm retrieves the path $\Pi(s, g')$ from the next state s along $\Pi(s_{\text{home}}, g)$ to g' and the robot will then execute $\Pi(s, g')$ once it reaches s.

discuss guarantees this approach yields

B. Algorithmic approach

While the straw man algorithm presented in Sec. IV allows for planning to any goal pose $g \in G^{\text{full}}$ in bounded time t_{bound}, its memory footprint is prohibitively large. We suggest to reduce the memory footprint by building on the observation that many paths to close-by goals traverse very similar parts of the configurations space (see Fig. 1).

Similar to the straw man algorithm, our preprocessing stage runs in two steps. In the first step we we compute from s_{home} a path Π_g to every goal state $g \in G^{\text{full}}$ (such a path exists following **A1**). However, this is done by computing a set of so-called "root paths" $\{\Pi_1, \ldots, \Pi_k\}$ from s_{home} to a subset of goals in G^{full}. As we will see these paths will allow to efficiently compute paths to every goal state in the query stage and the system only needs to explicitly store these root paths in memory (and not a path to every goal state as in the straw man algorithm). In the second step of the preprocessing stage, the algorithm computes for each root path a new path to each goal state for every state along the path. However, this is done by attempting to re-use previously computed root paths which allows for a very low memory footprint. The remainder of this section formalizes these ideas.

C. Planning using experiences

Before we can detail our algorithm, we show how *experience graphs* [**?**] can be used in our framework.

FAHAD - can you sketch something here?

D. Algorithmic details

Our algorithm starts by sampling a goal state $g_1 \in G^{\text{full}}$ and computing a root path Π_1 from s_{home} to g_1. We then associate with Π_1 all goal states $G_1 \subset G^{\text{full}}$ such that Π_1 can be used as an experience in reaching any state in $g'_1 \in G_1$. We then repeat this process but instead of sampling a goal state from G^{full}, we sample from $G^{\text{full}} \setminus G_1$. At the end of this step, we obtain a set of root paths. Each root path Π_i is associated with a goal set $G_i \subseteq G^{\text{full}}$ such that Π_i can be used as an experience in reaching any state in $g'_i \in G_i$. there is a lot of redundant text here. perhaps this can be compressed by introducing notation in Sec IV-C . For a visualization of this step, see Fig. 3. For pseudocode see Alg. 1. I would opt for putting the pseudo code in the supplementary material if we are tight on space.

Algorithm 1 COMPUTEROOTPATHS($s_{\text{start}}, G^{\text{UNCOV}}$)

```
1:  $\Psi \leftarrow \emptyset$                                    ▷ A list of pairs $(\Pi, G)$)
2:  $G'^{\text{UNCOV}} \leftarrow \emptyset$;   $G'^{\text{COV}} \leftarrow \emptyset$;   $i = 0$
3:  while $G^{\text{UNCOV}} \neq \emptyset$ do   ▷ Runs until all uncovered points
    are considered
4:      $g_i \leftarrow$SAMPLE($G^{\text{UNCOV}}$)
5:      $G^{\text{UNCOV}} \leftarrow G^{\text{UNCOV}} \setminus \{g_i\}$
6:      $\Pi_i \leftarrow$ PLANROOTPATH($s_{\text{start}}, g_i$)
7:      if $\Pi_i = \emptyset$ then                       ▷ path does not exist
8:          $G'^{\text{UNCOV}} \leftarrow G'^{\text{UNCOV}} \cup \{g_i\}$
9:      else
10:         $G_i \leftarrow \emptyset$
11:         for each $g_j \in G^{\text{UNCOV}}$ do
12:             $\pi_j \leftarrow$PLANUSINGROOTPATH($s_{\text{start}}, g_j, \Pi_i$)
13:             if $\pi_j \neq \emptyset$ then              ▷ planner succeeded
14:                 $G_i \leftarrow G_i \cup \{g_j\}$
15:                 $G^{\text{UNCOV}} \leftarrow G^{\text{UNCOV}} \setminus \{g_j\}$
16:         $\Psi \leftarrow \Psi \cup \{(\Pi_i, G_i)\}$        ▷ Insert $(\Pi_i, G_i)$ to $\Psi$
17:         $i \leftarrow i + 1$
18: return $\Psi, G'^{\text{UNCOV}}$
```

V. Evaluation

VI. Conclusion & future work

Acknowledgments

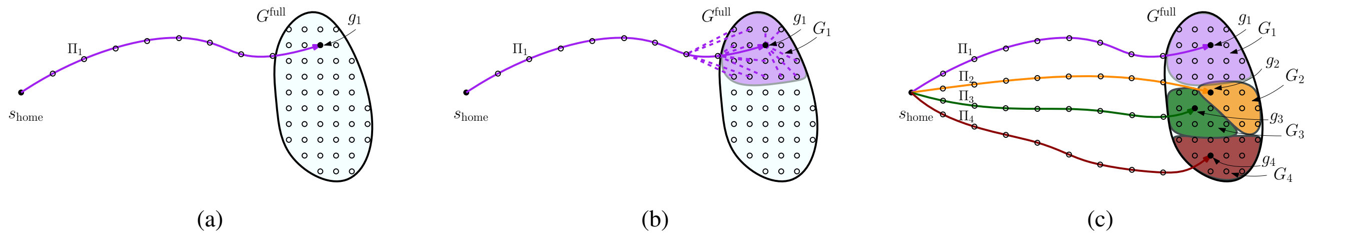
Fig. 2: First step of the preprocessing stage. (a) A goal state g_1 is sampled and the root path Π_1 is computed between s_{home} and g_1. (b) The set $G_1 \subset G^{\text{full}}$ of all states that can use Π_1 as an experience is computed and associated with Π_1. (c) The goal region covered by four root paths from s_{home} after the first step of the preprocessing stage terminates.

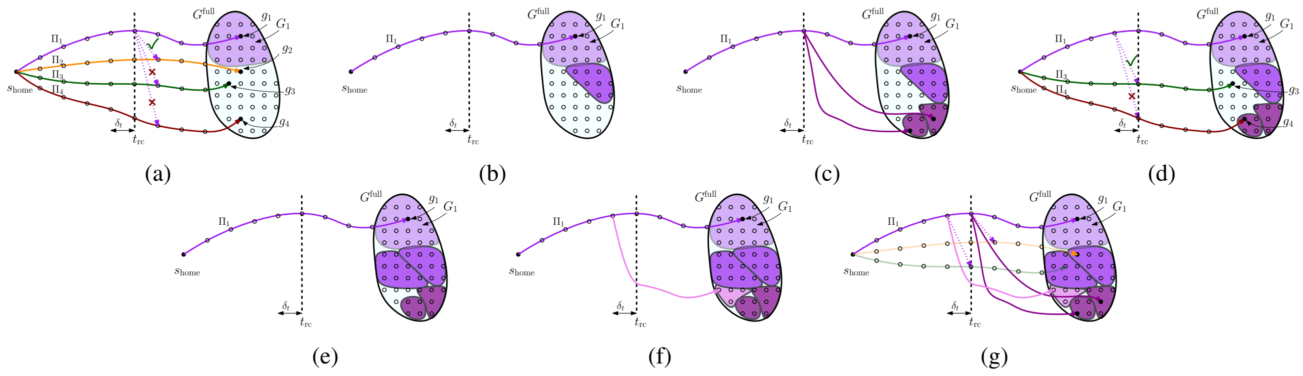
Fig. 3: Preprocess loop (a) text. (b) text. (c) text.

VII. Algorithm

A. Problem Definition and Assumptions

We define a goal region G^{full} as a discretized set of initial object poses on the conveyor belt that the robot can perceive when the object comes in its field of view. Given a home configuration for the robot s_{home}, we want to be able to plan for any goal pose g_{init} in bounded time t_{bound}. The perception system is setup such that it sends updated pose estimates on the fly during execution as it gets better estimates over time. Given any current state of the robot during execution s_{start}, the planner should be able to replan from s_{start} to any subsequent goal g_{next}.

We make the following assumptions about the system.

- G^{full} would accommodate for an error ϵ in the initial pose estimate g_{init} of the perception system. Each subsequent estimate g_{next} will be within the ϵ window around g_{init} (in retrospect because the conveyor is moving).
- We assume that all $\forall g \in G^{\text{full}}$, there exists a path from s_{home} to g and we can find it in finite preprocessing time (not sure about this one)
- We assume that the reachable set of goals for a state on a path is a subset of the reachable set of every other state on that path that exists before it.
- There exists a replan cutoff time t_{rc} from when the robot starts moving, after which the planner does not accept more replanning requests.
- We assume that before t_{rc}, the environment is static. In other words the moving target object cannot collide with the robot during that time.

B. Properties

- The planning time for each planning/replanning request is bounded (t_{bound})
- We discretize the trajectories with the resolution δt. The reaction time of the robot to a replanning request is bounded by $t_{\text{bound}} + \delta t$

C. Offline Motion Planner

In this section we will describe the motion planner that is used in the PlanPath() and PlanPathUsingRootPath() functions in Algs. 4 and 5. We use a heuristic search based planning approach with motion primitives (cite papers here). There are a few advantages of this approach for our particular problem.

- It can elegantly handle under-defined goals. We define a goal as a 6DOF grasp pose for the goal object. The goal is underdefined as there is a redundant degree of freedom of the robot and also because the grasping time is not specified.
- We make use of a set of predefined motion primitives and dynamically generated motion primitives which can

be computed such that the kinodynamic constraints of the robot are respected. These primitives are used to generate an implicit graph which is searched by a heuristic search-based planner to compute a plan

- We can design informative heuristic functions to speed up the search
- We use an existing search-based algorithm e-graphs (cite here) that reuses past experiences and significantly reduces planning times for repetitive tasks

1) State Space and Graph Construction:

2) Search and Heuristic function: Need a figure to explain the heuristic function

3) Assumptions: To be able to plan in the joint space of robot configuration space and time, we make an assumption that the robot has no torque limits.

Algorithm 2 PREPROCESSMAIN()

```
Inputs $s_{\text{home}}, G^{\text{full}}$
1: PREPROCESS($s_{\text{home}}, G^{\text{full}}, \emptyset$)
```

Algorithm 3 PREPROCESS($s_{\text{start}}, G^{\text{UNCOV}}, G^{\text{COV}}$)

```
1:  $\Psi_{\text{work}}, G'^{\text{UNCOV}}_{\text{work}} \leftarrow$ COMPUTEROOTPATHS&GOALREGIONS($s_{\text{start}}, G^{\text{UNCOV}}$)
2:  if $s_{\text{start}} = s_{\text{home}}$ then
3:      $\Psi_{\text{home}} = \Psi_{\text{work}}$
4:  $G'^{\text{COV}} \leftarrow G^{\text{COV}} \cup (G^{\text{UNCOV}} - G'^{\text{UNCOV}}_{\text{work}})$
5:  if $t(s_{\text{start}}) \geq t_{\text{rc}}$ then
6:      return $G'^{\text{UNCOV}}_{\text{work}}, G'^{\text{COV}}$
7:  for each $(\Pi_{G_i}, G_i) \in \Psi_{\text{work}}$ do
8:      $t = t_{\text{rc}}$
9:      $G_i^{\text{uncov}} \leftarrow G'^{\text{COV}} - G_i$
10:     $G_i^{\text{cov}} \leftarrow G_i$
11:     while $t \geq t(s_{\text{start}})$ do
12:         $s \leftarrow$ GETSTATE($\Pi_{G_i}, t$)
13:         for each $(\Pi_{G_j}, G_j) \in \Psi_{\text{home}}$ do
14:             if CHECKSNAP($s, \Pi_{G_j}$) then
15:                 $G_i^{\text{uncov}} \leftarrow G_i^{\text{uncov}} - G_j$
16:                 $G_i^{\text{cov}} \leftarrow G_i^{\text{cov}} \cup G_j$
17:         if $G_i^{\text{uncov}} = \emptyset$ then
18:             break
19:         $G_i^{\text{uncov}}, G_i^{\text{cov}} \leftarrow$ PREPROCESS($s, G_i^{\text{uncov}}, G_i^{\text{cov}}$)
20:         if $G_i^{\text{uncov}} = \emptyset$ then
21:             break
22:         $t = t - \delta t$
23: return $G'^{\text{UNCOV}}_{\text{work}}, G'^{\text{COV}}$
```

Algorithm 4 QUERY($g, \pi_{\text{curr}}, s_{\text{start}}$)

```
1:  $\Pi_{\text{curr}} \leftarrow$ LOOKUPROOTPATH($s_{\text{start}}, g$)
2:  if $\Pi_{\text{curr}} \neq \emptyset$ then
3:      $\pi \leftarrow$ PLANPATHUSINGROOTPATH($s_{\text{start}}, g, \Pi_{\text{curr}}$)
4:      return $\pi$
5:  $t = t_{\text{rc}}$
6:  while $t \geq t_{\text{curr}}$ do
7:      $s \leftarrow$ GETSTATE($\pi_{\text{curr}}, t$)
8:      $\Pi_{\text{next}} \leftarrow$ LOOKUPROOTPATH($s, g$)
9:      if $\Pi_{\text{next}} \neq \emptyset$ then
10:         $\pi_{\text{next}} \leftarrow$PLANPATHUSINGROOTPATH($s_{\text{start}}, g, \Pi_{\text{next}}$)
11:         $\pi \leftarrow$ MERGEPATHS($\pi_{\text{curr}}, \pi_{\text{next}}, t$)
12:         return $\pi$
13:     $\Pi_{\text{home}} \leftarrow$ LOOKUPROOTPATH($s_{\text{home}}, g$)
14:     if $\Pi_{\text{home}} \neq \emptyset$ then
15:         if CHECKSNAP($s, \Pi_{\text{home}}$) then
16:             $\pi_{\text{home}} \leftarrow$PLANPATHUSINGROOTPATH($s_{\text{start}}, g, \Pi_{\text{home}}$)
17:             $\pi \leftarrow$ MERGEPATHSWITHSNAP($\pi_{\text{curr}}, \pi_{\text{home}}, t$)
18:             return $\pi$
19:     $t = t - \delta t$
20: return failure
```

Algorithm 5 COMPUTEROOTPATHS&GOALREGIONS($s_{\text{start}}, G^{\text{UNCOV}}$)

```
1:  $\Psi \leftarrow \emptyset$                          ▷ A list of pairs $(\Pi, G)$)
2:  $G'^{\text{UNCOV}} \leftarrow \emptyset$
3:  $i = 0$
4:  while true do  ▷ Runs until all $g_i \in G^{\text{UNCOV}}$ are sampled and/or covered
5:      $g_i \leftarrow$SAMPLERANDOMUNCOVEREDGOALINREGION($G^{\text{UNCOV}}$)
6:      if $g_i =$ NULL then
7:          break
8:      $\Pi_i \leftarrow$ PLANROOTPATH($s_{\text{start}}, g_i$)
9:      if $\Pi_i = \emptyset$ then                 ▷ i.e. path does not exist
10:         $G'^{\text{UNCOV}} \leftarrow g_j$
11:     $G_i \leftarrow \emptyset$
12:     for each $g_j \in G^{\text{UNCOV}}$ do
13:         if $g_j$ is covered then
14:             continue
15:         $\pi_j \leftarrow$PLANPATHUSINGROOTPATH($s_{\text{start}}, g_j, \Pi_i$)
16:         if $\pi_j \neq \emptyset$ then          ▷ i.e. planner succeeded
17:             Insert $g_j$ in $G_i$
18:             Mark $g_j$ as covered
19:     Insert pair $(\Pi_i, G_i)$ in $\Psi$
20:     $i = i + 1$
21: return $\Psi, G'^{\text{UNCOV}}$
```

Undefined Functions

- GETSTATE(π, t) returns the state on path π with timestamp t.
- CHECKSNAP(s, Π) checks if the robot can snap from state s onto the the path Π at the next time stamp of what s is at. It checks if the motion is valid w.r.t. kinematic, dynamic and collision constraints.
- LOOKUPROOTPATH(s, g) queries a stored lookup table that maps a start and goal pair to a root path (that could be used to plan path in bounded time).
- SAMPLERANDOMUNCOVEREDGOALINREGION(G) returns a goal in G which is never sampled before and which is still uncovered. If no such goal exists it returns NULL.
- PLANROOTPATH(s, g) uses a search-based planner with motion primitives to plan a path from s to g (details will be in the text).
- PLANUSINGROOTPATH(s, g, Π) uses the e-graph planner to plan a path from s to g using the root path Π (details will be in the text).
- MERGEPATHS(π_i, π_j, t) constructs a new path π by joining two path segments, first being π_i starting from its first state up until the state at time stamp t, and the second being π_j.
- MERGEPATHSWITHSNAP(π_i, π_j, t) constructs a new path π by joining two path segments with a snapping edge in between, first path being π_i starting from its first state up until the state at time stamp t, then the edge connecting state on π_i at time t to the state on π_j at time $t + \delta t$, followed by π_j starting from the state at $t + \delta t$ and up until the end of it.

VIII. EXPERIMENTS

Our Algorithm

- Preprocessing: total preprocessing time, memory consumption, total trajectories stored, total states for which Alg. 3 was called, mean time for PLANPATH() function
- Query (Planning): mean planning time, mean execution time, success rate
- Query (Replanning): success rate for experiments with varied number of replanning requests