Accepted for publication in IEEE International Conference on Robotics and Automation (ICRA), May 2022

Learning to Retrieve Relevant Experiences for Motion Planning

Constantinos Chamzas, Aedan Cullen, Anshumali Shrivastava, Lydia E. Kavraki

***Abstract*— Recent work has demonstrated that motion planners' performance can be significantly improved by retrieving past experiences from a database. Typically, the experience database is queried for past similar problems using a similarity function defined over the motion planning problems. However, to date, most works rely on simple hand-crafted similarity functions and fail to generalize outside their corresponding training dataset. To address this limitation, we propose (FIRE), a framework that extracts local representations of planning problems and learns a similarity function over them. To generate the training data we introduce a novel self-supervised method that identifies similar and dissimilar pairs of local primitives from past solution paths. With these pairs, a Siamese network is trained with the contrastive loss and the similarity function is realized in the network's latent space. We evaluate FIRE on an 8-DOF manipulator in five categories of motion planning problems with sensed environments. Our experiments show that FIRE retrieves relevant experiences which can informatively guide sampling-based planners even in problems outside its training distribution, outperforming other baselines.**

I. Introduction

Motion planning is used in real-time autonomous vehicles [1], manipulators in dynamic environments [2], and as a subroutine in planners for complex missions (e.g. task and motion planning [3]), all of which rely heavily on efficiency. However, motion planning is still challenging, especially for high-dimensional systems [4]. Sampling-based planners [5]–[7] are a class of motion planning algorithms that have found widespread adoption in the planning community. Although significant progress has been made over the years, planning is still computationally expensive [8], hindering the adoption of robotic solutions. Thus, to endow robots with real-time capabilities, faster motion planning algorithms are necessary.

A promising avenue is to guide planning by leveraging the past experiences of a robot. Several methods have shown that storing and retrieving experiences [9], [10] can significantly improve motion planners' efficiency. These methods have focused on what to store and how to adapt/repair it for the current situation, but not on how to retrieve the most relevant experiences, defaulting to simple similarity functions. In other words, little emphasis has been placed on finding suitable functions that quantify the similarity of motion planning problems, limiting the generalizability of retrieval-based methods outside their training dataset.

In this context, for similar motion planning problems or subproblems, the solution path of one can be used to expedite the search when solving the other. Capturing this notion of similarity is the core investigation of this work. Designing a good similarity function is very challenging for motion planning problems. For example, in Fig. 1 two visually dissimilar workspaces M_0, M_1 have similar solution paths while visually similar workspaces M_1 and M_2 have different solution paths. A good similarity function should capture the commonalities between M_0 and M_1 while still distinguishing between M_1 and M_2. These problems are part of the "Tall-Shelf" dataset described in Sec. V.

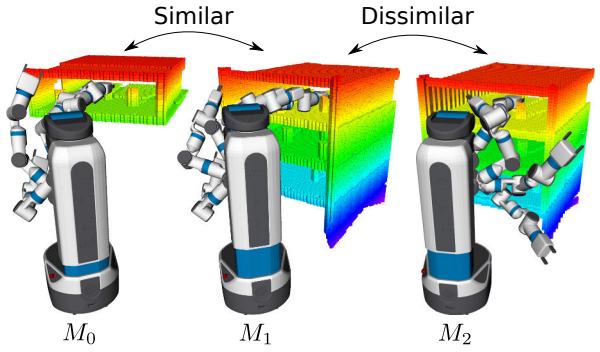

Fig. 1. Three example problems M_0, M_1, M_2 where the robot is tasked with picking one object from a shelf starting from the same tuck (home) configuration (not shown for visual clarity). The motion planning problems M_0 and M_1 have similar solution paths even though their workspaces are visually different. On the other hand, visually similar workspaces such as M_1 and M_2 can have different solution paths for subtle reasons (e.g. slightly different goals, obstacle arrangements, and robot base orientation).

To address this problem we propose **F**ast retr**I**eval of **R**elevant **E**xperiences (FIRE). As detailed in Sec. IV, FIRE extracts suitable local representations, called local primitives, from previous problems. FIRE finds pairs of similar and dissimilar local primitives using a self-supervised method. With these pairs, a similarity function is learned which can be used to retrieve relevant experiences and guide a motion planner. We demonstrate the effectiveness of FIRE with an 8-DOF mobile manipulator in five categories of diverse problems with sensed environments as shown (Fig. 1). Through our experiments (Sec. V) we show that FIRE generalizes better outside its training dataset even with less data, and is faster in terms of planning time than prior work. The implementation of FIRE and the generated datasets are open-source [1].

Overall, the main contributions of this work lie in 1) defining suitable local representations of motion planning problems, 2) learning a similarity function over them, and 3) applying it in the motion planning problem through our new framework. Although FIRE is tailored to retrieval frameworks that use local features and biased sampling distributions [11], [12] we believe it could be easily adapted to work with other retrieval-based methods [13]–[15].

All authors are affiliated with the Department of Computer Science, Rice University, Houston TX, USA {chamzas, aedan, anshumali, kavraki}@rice.edu. This work was supported in part by NSF 1718478, NSF-GRFP 1842494 and Rice University Funds.

[1] https://github.com/KavrakiLab/pyre

II. Problem Description and Notation

Feasible Path Planning: Consider a robot in a workspace $\mathcal{W}$. A configuration of the robot x is a point in the configuration space ($\mathcal{C}$-space), $x \in \mathcal{C}$. Obstacles in the workspace induce $\mathcal{C}$-space obstacles $X_{\text{obs}} \subset \mathcal{C}$. The set of configurations that are not in collision is denoted by $X_{\text{free}} = \mathcal{C} - X_{\text{obs}}$. We are interested in finding a path p, from $x_{\text{START}} \in X_{\text{free}}$ to $x_{\text{GOAL}} \in X_{\text{free}}$, as a continuous map with $p(0) = x_{\text{START}}$, $p(1) = x_{\text{GOAL}}$ such that for all $t \in [0, 1]$, $p(t) \in X_{\text{free}}$. We denote the motion planning problem by $\mathcal{M} = (x_{\text{START}}, x_{\text{GOAL}}, \mathcal{W})$.

"Challenging Regions" and "Critical Samples": In this work, we are concerned with planning for high-dimensional robotic manipulators, and focus on sampling-based motion planners. A common theme in learning-based approaches is to produce configurations in $\mathcal{C}$-space regions with low visibility [16], which are the main bottleneck for sampling-based motion planners [17]. We denote these "challenging regions", and configurations inside them "critical samples."

Retrieval-Based Learning for Motion Planning: Given a dataset $\mathcal{DS} = \{\mathcal{M}^i : p^i\}_{i=1}^N$ of past problems $\mathcal{M}$ and their feasible paths p, retrieval-based methods extract information from $\mathcal{DS}$ and store it in a database denoted $\mathcal{DB}$. In this context, $\mathcal{DB}$ is a structure that contains $\langle key : value \rangle$ entries, with the experiences (values) being "critical samples." The indices (keys) of the database are local primitives denoted by $\ell \in \mathcal{L}$, where $\mathcal{L}$ is the space of local primitives. Each local primitive includes local workspace information [12] along with x_{START}, x_{GOAL} information (as defined in Sec. IV-A). This work aims to learn a suitable similarity function $\text{SIM} : \mathcal{L} \times \mathcal{L} \rightarrow \{0, 1\}$ over the local primitives in order to retrieve relevant "critical samples" for a given problem $\mathcal{M}$.

III. Related work

Over the years many techniques have been proposed to guide sampling-based motion planners. Many examples use heuristics to bias sampling, such as Bridge sampling [17], Gaussian sampling [18], Medial-Axis sampling [19], and workspace-based sampling [20]. However, these predefined heuristics may or may not apply in different situations.

Thus, a growing number of works attempt to learn how to guide planning by utilizing past solutions to motion planning problems. One set of methods learns interesting regions in $\mathcal{W}$ [21], [22] but requires an inverse kinematics solver to infer samples in "challenging regions". A similar class of methods directly computes relevant configurations in $\mathcal{C}$ from a motion planning problem $\mathcal{M}$ using a neural network. For example, some methods train a conditional variational autoencoder to reconstruct samples from previous paths [23] or "challenging regions" [24]–[26]. The authors of [27], [28] use a 3D CNN to sample in "challenging regions", while [29]–[31] use neural networks as motion planners.

Although these methods have shown some promising results, mapping $\mathcal{M}$ to paths or "challenging regions" in $\mathcal{C}$ is hard in high-dimensional problems. Motion planning is sensitive to input; small changes in $\mathcal{W}$, x_{START}, or x_{GOAL} can drastically alter the resulting solution [12], [14], [32]. Furthermore, this mapping is usually multi-modal, since a motion planning problem may have multiple solution paths or multiple disjoint "challenging regions" [15], [33].

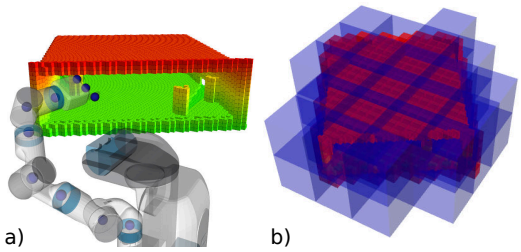

Fig. 2. **a)** The blue dots depict the 10 projections defined on the arm and gripper of the Fetch robot. Each blue dot is one projection point $\pi(x)_p \in \mathbb{R}^3$ of $x \in \mathcal{C}$. Specifically, each robotic link of the arm+gripper was used as a projection, as described in its `urdf`. **b)** Examples of local occupancy grids and their position in space derived from the sensed scene ($lw = (b, v)$). Note that only non-empty local occupancy grids are generated.

For these reasons, some approaches have adopted retrieval-based methods, also known as library- [34] or memory-based [35] methods. Such methods typically store in memory a database $\mathcal{DB}$ and retrieve relevant information in the form of paths [36], [37] or sampling distributions [11], [38] based on a similarity function over $\mathcal{M}$. These methods naturally apply to multi-modal problems, since for similar or identical $\mathcal{M}$ multiple outputs can be retrieved. Another advantage of these methods is that they are incremental since new experiences can simply be added to the database $\mathcal{DB}$. The main challenge lies in constructing a good similarity function over $\mathcal{M}$.

Defining a similarity function is challenging because $\mathcal{M}$ contains heterogeneous parameters; $x_{\text{START}}, x_{\text{GOAL}} \in \mathcal{C}$ while $\mathcal{W}$ is a 3D representation. Some approaches do not use a similarity function but learn problem invariants [39], [40], others construct the similarity only over x_{START} and x_{GOAL} [10], [36], and some construct it only over $\mathcal{W}$ [12], [13]. In [12] a hand-crafted similarity function over local workspaces is defined, while [13] defines workspace similarity based on geometric deformation of obstacles. Most similarly to our work, [9] learned a similarity function over x_{START}, x_{GOAL}, and $\mathcal{W}$ using a weighted combination of global workspace features. In contrast, our work uses local features and leverages latent space representations obtained from neural networks.

Learning similarity functions [41] in the latent space has been successfully employed in computer-vision tasks, such as image classification [42] and 3D object classification [43]. Our work is inspired by these methods, and applies similar metric learning methods to the motion planning problem.

IV. Methodology

We propose FIRE, a framework that learns a similarity function to retrieve relevant experiences from a database in the form of "critical samples". In Sec. IV-A we formulate the local primitives which are the input to the similarity function, and we extract them from past problems in Sec. IV-B. Then, we describe how to generate similar and dissimilar local primitives (Sec. IV-C). In Sec. IV-D, we train a Siamese network by minimizing the contrastive loss of the local primitive pairs and realize the similarity function in the learned latent space. Finally, Sec. IV-E explains how the similarity function can guide a sampling-based planner.

A. Local primitives

First, we define a set of projections $\pi(x) : \mathcal{C} \rightarrow \mathbb{R}^3$ used to extract and compare local primitives. Each configuration x is projected to multiple points in $\mathcal{W}$ and stacked as a vector

$$\Pi(x) = [\pi_1(x), \pi_2(x), \ldots, \pi_P(x)] \in \mathbb{R}^{3\times P}$$

where P is the number of projections. Fig. 2a) shows the 10 projections on the Fetch which we used. Specifically, we used the link frames of the arm+gripper from the Fetch [44] `urdf`. Projections have often been used to guide motion planners [45] and specifying them is often a research problem in itself, albeit outside the scope of this work.

Now we define the local primitives ℓ, which include a local 3D occupancy grid and its position lw [12] along with some auxiliary $\mathcal{C}$-space information x_{TARGET} and x_{PROJ}:

$$\ell = [lw, x_{\text{TARGET}}, x_{\text{PROJ}}]$$

More specifically, $lw = (b, v)$ where $b \in \{0, 1\}^{64}$ is a 64-bit binary vector that represents a (4x4x4) local occupancy grid and $v \in \mathbb{R}^3$ is the center position of the grid. Examples of lw are shown in Fig. 2b. The variable $x_{\text{TARGET}} \in \mathcal{C}$ is either x_{START} or x_{GOAL}, depending on the situation as explained in Alg. 1 and Sec. IV-E. Finally, we calculate x_{PROJ} from x_{TARGET} and the center position v of lw. We project x_{TARGET} to P points in the workspace $\Pi(x_{\text{TARGET}}) \in \mathbb{R}^{3\times P}$ and then aggregate all the distances between the P points and v to calculate x_{PROJ}:

$$x_{\text{PROJ}} = [\|v - \pi_1(x_{\text{TARGET}})\|, \ldots, \|v - \pi_P(x_{\text{TARGET}})\|] \in \mathbb{R}^P$$

The variable x_{PROJ} serves as an interleaved representation of x_{TARGET} and lw and was empirically validated to improve the latent space structure.

B. Creating the experience database

Alg. 1 describes how to create the experience database $\mathcal{DB}$ from $\mathcal{DS} = \{(x_{\text{START}}, x_{\text{GOAL}}, W)^i : p^i\}_{i=1}^N$ by associating each local primitive with a configuration from a solution path.

First, the paths are shortcutted [46] to remove redundant nodes not in "challenging regions" (line 1 in Alg. 1) and keep only "critical samples". Finding "critical samples" is still an open research problem [12], [22], [24] but this simple shortcutting heuristic has been used previously in [11], [39].

Next, `TARGET` (line 2 in Alg. 1) samples near x_{START} and x_{GOAL} and chooses the one which yielded the most in-collision samples with the workspace. This aims to create the same local representation for motion plans with the same solution path but swapped x_{START} and x_{GOAL}. Consider for example the task in Fig. 1, where the robot plans from the home (x_{START}) to a grasp configuration (x_{GOAL}). The same solution path applies for planning between the grasp configuration (x_{START}) back to the tuck configuration (x_{GOAL}). Thus, to ensure that both plans have the same local representations `TARGET` should choose the same configuration as x_{TARGET} (e.g. the grasp configuration). We then decompose the workspace to local occupancy grids (line 3 in Alg. 1) by traversing the octomap tree similarly to [12].

Algorithm 1: Creating the experience database

Input : MP problem $\mathcal{W}, x_{\text{START}}, x_{\text{GOAL}}$ Path p
Output : Database $\mathcal{DB}$

1 Shortcut $p' =$ `SHORTCUT`(p)
2 Find target $x_{\text{TARGET}} \leftarrow$ `TARGET` $(x_{\text{GOAL}}, x_{\text{START}})$
3 Decompose $\mathcal{W}$ to $\mathcal{LW} \leftarrow \{lw_1, \ldots, lw_M\}$
4 **foreach** $lw \in \mathcal{LW}$ **do**
5 **foreach** $x \in p'$ **do**
6 **foreach** $\pi \in \Pi$ **do**
7 **if** `CONTAINS` $(lw, \pi(x))$ **then**
8 $x_{\text{PROJ}} \leftarrow |\tilde{v} - \Pi(x_{\text{TARGET}})|$
9 $\ell \leftarrow [lw, x_{\text{TARGET}}, x_{\text{PROJ}}]$
10 $x^n \leftarrow$ `NEXT`(x, p)
11 $x^p \leftarrow$ `PREV`(x, p)
12 Insert $\langle \ell : x^p, x, x^n \rangle$ in $\mathcal{DB}$
13 **return** $\mathcal{DB}$

Afterward, we iterate over the configurations in each path, the local occupancy grids, and the projections. The subroutine `CONTAINS` associates each configuration with its relevant regions in the workspace. `CONTAINS` checks for every projection $\pi(x)_p \in \mathbb{R}^3$ of the configuration x if it is contained in the bounding box of an occupancy grid; if so we store the local primitive ℓ along with the critical x, the previous waypoint configuration x^p, and the next waypoint configuration x^n in $\mathcal{DB}$. The previous and next configurations are only used to help us create similar pairs as described in Alg. 2 and are not part of the retrieved experience.

C. Creating a dataset of similar pairs

Alg. 2 describes a novel method to create a dataset of similar pairs of local primitives over which to learn the similarity function. This is the key problem investigated in this paper.

Given a database $\mathcal{DB}$, we iterate over all pairs of local primitives and perform the following checks. First, the subroutine `SAME_PROJ` checks if the two local primitives were generated by the same projection (line 3 in Alg. 2). Then we check whether the centers v of the local occupancy grids are close enough in $\mathcal{W}$ (line 4 in Alg. 2) and whether the stored configurations are also close enough in $\mathcal{C}$-space (line 5 in Alg. 2). The variable lw_{side} is the length of the side of the local occupancy bounding box lw.

Finally (line 5 in Alg. 2) we sample up to N times $x_j^{near} \sim \mathcal{N}(x_j, \sigma^2)$ until a configuration x_j^{near} is found which passes the `VALID` check. The `VALID` subroutine checks if x_j^{near} can connect through a collision-free edge (in the full workspace $\mathcal{W}$ of ℓ_i) with the next x_i^n and previous x_i^p configuration of the local primitive ℓ_i. If such a configuration is found then we consider $\langle \ell_i, \ell_j \rangle$ similar and add them to $\mathcal{S}$. This procedure aims to discover local primitives whose "critical samples" are good substitutes for one another by emulating how "critical samples" are used to bias sampling during planning (Sec. IV-E). To generate dissimilar pairs we randomly choose local primitives from $\mathcal{DB}$ and generate an equal number of dissimilar pairs. We denote the set that includes these dissimilar pairs $\mathcal{NS}$.

Algorithm 2: Creating a dataset of similar pairs

Input : Database $\mathcal{DB}$
Output : Pairs of similar local primitives $\mathcal{S}$

foreach $\langle \ell_i : x_i^p, x^i, x_i^n \rangle \in \mathcal{DB}$ **do**
  foreach $\langle \ell_j : x_j^p, x_j, x_j^n \rangle \in \mathcal{DB}$ **do**
    if SAME_PROJ(ℓ_i, ℓ_j) **then**
      if $\|v_j - v_j\|_1 <= size_{lw}$ **then**
        if $||x_i - x_j|| < 10\sigma^2$ **then**
          repeat N **times**
            $x_j^{near} \sim \mathcal{N}(x_j, \sigma^2)$
            if VALID$(x_i^p, x_j^{near}, x_i^n)$ **then**
              $\mathcal{S} \leftarrow \langle \ell_j, \ell_i \rangle$
              break
return $\mathcal{S}$

Note that Alg. 2 needs the "critical samples" extracted from solution paths to find similar local primitives, and cannot be used as a similarity function when solving a new motion planning problem where only $W, x_{\text{GOAL}}, x_{\text{START}}$ is available.

D. Learning the similarity function

The learned similarity function is realized in the latent space of a Siamese network. A Siamese network [47] is comprised of two identical encoder networks as shown in Fig. 3. Each encoder maps ℓ to a latent variable $z \in \mathbb{R}^8$. The overall network is relatively small with around 3500 parameters, and was trained with the contrastive loss [48]:

$$\mathcal{L}(\ell_i, \ell_j) = \begin{cases} \max(0, d_m - \|z_i - z_j\|^2) & \text{if } \langle \ell_j, \ell_i \rangle \in \mathcal{NS} \\ ||z_i - z_j||^2 & \text{if } \langle \ell_j, \ell_i \rangle \in \mathcal{S} \end{cases}$$

This loss tries to bring local primitives that belong in $\mathcal{S}$ (similar) as close as possible in the latent space Z, while local primitives that belong in $\mathcal{NS}$ (dissimilar) must have at least a margin distance $d_m = 0.5$. After having structured the latent space Z the similarity function is defined as follows:

$$\text{SIM}(\ell_i, \ell_j) = \begin{cases} 1 & \text{if } \|z_i - z_j\|^2 < R \\ 0 & \text{otherwise} \end{cases}$$

where $R = 0.2 d_m$ is the retrieval radius. A lower retrieval radius than the margin distance d_m must be used to avoid retrieving dissimilar pairs. After structuring the latent space Z all the local primitives in $\mathcal{DB}$ are projected to Z and added in a K-D tree [49] structure for fast retrieval. Finding similar local primitives with SIM is equivalent [50] to retrieving all the neighbors within radius R in the latent space Z.

E. Retrieving relevant experiences

When solving a new problem $\mathcal{M}$ =(x_{START},x_{GOAL}, $\mathcal{W}$) the new local primitives are created with the following procedure. First, we extract the local occupancy grids from $\mathcal{W}$. Then, for each local occupancy grid lw we generate two local primitives: one with $x_{\text{TARGET}} = x_{\text{START}}$ and one with $x_{\text{TARGET}} = x_{\text{GOAL}}$. The value of x_{PROJ} is calculated from x_{TARGET} and ℓ as explained in Alg. 1. Each created local primitive is projected to Z and its neighbors within radius R are retrieved, effectively obtaining their associated "critical samples" from $\mathcal{DB}$. Finally, similarly

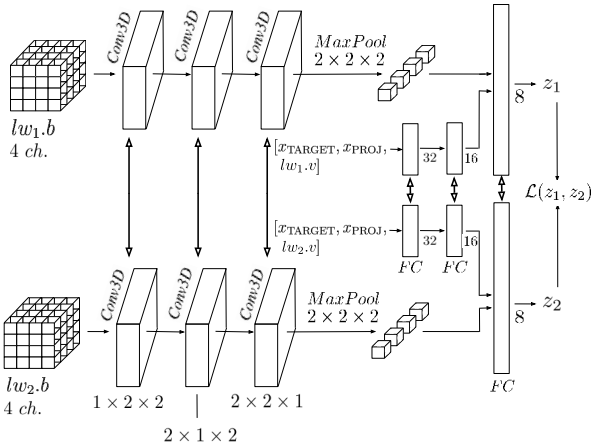

Fig. 3. The Siamese network architecture used. The activation function for all the layers was ReLU. *Conv3D* denotes a 3D convolutional layer, *MaxPool* takes the maximum value out of every subgrid, and *FC* denotes a fully connected layer. The parameters of each layer are shown in the figure.

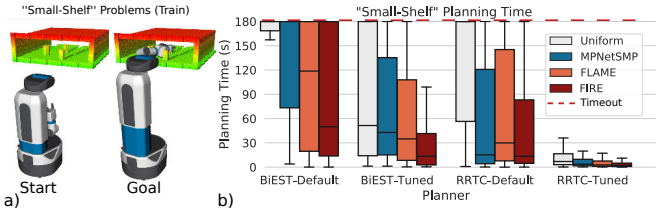

Fig. 4. **a)** An example problem from the "Small-Shelf" dataset. We generate different problems by uniformly sampling the robot pose, the position of the obstacles, and the height of the shelf. This is similar to the "Small-Shelf" used in [12] but the shelf is shorter, making it more challenging due to the narrow area the robot has to traverse. **b)** Planning time (including retrieval) with different underlying planners for 100 test examples from the "Small-Shelf" dataset. The timeout was set to 180 seconds.

to [12], we aggregate all the K "critical samples" and convert them to a Gaussian Mixture Model (GMM):

$$P(x|\mathcal{M}) = \frac{1}{K} \sum_{i=0}^{K} \mathcal{N}(x_i, \sigma^2)$$

The GMM can be used to bias the sampling of any sampling-based planner. To keep the probabilistic completeness guarantees of sampling-based planners we sample from $P(x|\mathcal{M})$ with probability $0 < \lambda < 1$ and from a standard uniform distribution with probability $(1 - \lambda)$. If the planner uses a local expansion strategy like EST [6] we simply sample from the mixtures that are within the local sampling radius.

V. Experiments

We demonstrate the effectiveness of the learned similarity function on five generated datasets with MotionBench-Maker [51]. Each dataset contains an 8-DOF (arm+torso) Fetch robot [44] with a workspace represented by an octomap [52], performing a pick task as shown in Fig. 4a. We consider this a realistic representation since point clouds can easily be obtained from a simple depth camera. The five datasets generated were "Small-Shelf" (Fig. 4a), "Tall-Shelf" (Fig. 5a), "Thin-Shelf" (Fig. 5b), "Table" (Fig. 5c), and "Cage" (Fig. 7a). As shown in the figures, the starting configuration x_{START} for all datasets was a home (tuck) position, except

for "Table" where x_{START} is a random configuration under the table. The goal configuration x_{GOAL} is an inverse kinematics (IK) solution placing the end-effector in a grasping pose relative to an object. For the "Shelf" datasets, one object per shelf is grasped and it is always the one furthest back. For "Table" and "Cage" the grasped object is shown in the figures. We generate different motion planning problems similarly to [12] by uniformly sampling poses for the robot base and scene objects. Note that such variation generates highly diverse planning problems since even small changes in the positions of the obstacles relative to the robot drastically affect X_{obs} and the resulting x_{GOAL}.

All evaluated methods produce biased samples in $\mathcal{C}$ which can guide any sampling-based motion planner. We evaluated these methods within RRT-connect (RRTC) [53] and bidirectional EST (BIEST) [6], implemented in the Open Motion Planning Library (OMPL) [54]. Additionally, we considered two versions of each planner: one with default OMPL parameters (RRTC-DEFAULT and BIEST-DEFAULT) and one with a tuned range parameter (RRTC-TUNED and BIEST-TUNED) found by a parameter sweep over a diverse set of problems. In our experiments we compare FIRE with the following methods:

- UNIFORM: Default uniform sampling of the $\mathcal{C}$-space.
- MPNET-SMP [29]: This is the sampling-biasing version of Motion Planning Networks. Given a training dataset of 3D point cloud workspaces, x_{START}, x_{GOAL}, and solution paths, MPNET-SMP learns to iteratively produce samples that mimic the solution paths. We adapted the provided implementation and tuned its hyperparameters to achieve the best performance for the given problems.
- FLAME [12]: This framework is similar to FIRE and also retrieves "critical samples" from a $\mathcal{DB}$. However, the local primitives are simpler, including only workspace information (lw) and not considering x_{GOAL} or x_{START}. The similarity function considers lw_i similar to lw_j if they have the same position and binary representation.
- STATIC [39], [40]: These methods generate a static sampling distribution by extracting key configurations from past trajectories. They do not rely on a similarity function but instead attempt to capture the problem's invariants. We emulate the static sampling idea of these methods by retrieving all the $\mathcal{C}$-space samples we have stored in $\mathcal{DB}$.

We consider these methods representative of the works discussed in Sec. III, with MPNET-SMP being a non-retrieval method that directly maps $\mathcal{M}$ to $\mathcal{C}$-space samples using a neural network, FLAME a retrieval-based method with a hand-crafted similarity function, and STATIC a method that learns problem invariants.

We evaluate the performance of FIRE and the generalization of the learned similarity function when both the training and testing examples come from the same dataset (Sec. V-A), and also when the testing dataset is increasingly different from the training dataset (Sec. V-B). Finally, we evaluate FIRE when retrieving experiences it was not trained on, and while the $\mathcal{DB}$ includes unrelated experiences (Sec. V-C). For our experiments we used Robowflex with MoveIt [55], [56] and the OMPL benchmarking tools [57]. The sampling parameters for FIRE were the same as [12] ($\sigma^2 = 0.2, \lambda = 0.5$).

A. Generalizing in similar problems

1) Learning (Training): In this experiment, MPNET-SMP, FLAME, and FIRE were trained in problems that come from the "Small-Shelf" dataset. FIRE and FLAME were given enough training examples for their performance to converge in the "Small-Shelf" dataset. By convergence, we mean that the average planning time did not improve after doubling the number of experiences in $\mathcal{DB}$. Specifically, FIRE was trained with a total of 500 training examples. From these 500 examples, 200 were used to learn the similarity function and all of the 500 examples were added to $\mathcal{DB}$. Training the Siamese network of FIRE took around 1 hour for 200 epochs. FLAME was trained with 1000 examples which were added to $\mathcal{DB}$ as described in [12]. Since it was difficult to profile the convergence of MPNET-SMP ($\approx$1 day of training time) we provided it 5000 training examples to ensure that it has enough data. This is of a similar order to [29] (10000).

2) Evaluation (Testing): The methods were tested in a different set of 100 problems that also come from "Small-Shelf". As seen in Fig. 4b, FIRE outperformed all other methods in all four different settings in terms of planning time. We do include the retrieval time in the total planning time for FLAME and FIRE but it was negligible in all cases ($0.01 - 0.1$ seconds). We also notice that the tuning of the underlying planner and the use of experiences interact synergistically, with the best performance being achieved by FIRE with RRTC-TUNED.

B. Generalizing in increasingly different problems

1) Learning (Training): We do not perform any additional training in these experiments and simply use the methods trained on "Small-Shelf" from Sec. V-A.

2) Evaluation (Testing): In these experiments, the methods were tested on three datasets that are increasingly different from "Small-Shelf" as shown in Fig. 5. The "Tall-Shelf" is created by stacking the "Small-Shelf" three times. The "Thin-Shelf" is also a bookcase but is different from "Tall-Shelf" and "Small-Shelf" because there is a divider and the distance between the shelves has changed. Finally, "Table" is significantly different from "Small-Shelf" regarding $\mathcal{W}$. We used 100 testing examples for each of these three datasets. As shown in Fig. 6, MPNET-SMP could not outperform UNIFORM in "Tall-Shelf" and "Table" except for RRTC-DEFAULT, while in "Thin-Shelf" it was not able to improve upon UNIFORM given the time limits. In some cases MPNET-SMP performed worse than UNIFORM; we attribute this behavior to the testing examples being outside the training dataset of MPNET-SMP. FLAME did offer some improvement for the "Tall-Shelf" environment but could not transfer to "Thin-Shelf" or "Table". Also, in some cases FLAME performed worse than UNIFORM; this is attributed to the retrieval of very few critical samples leading to

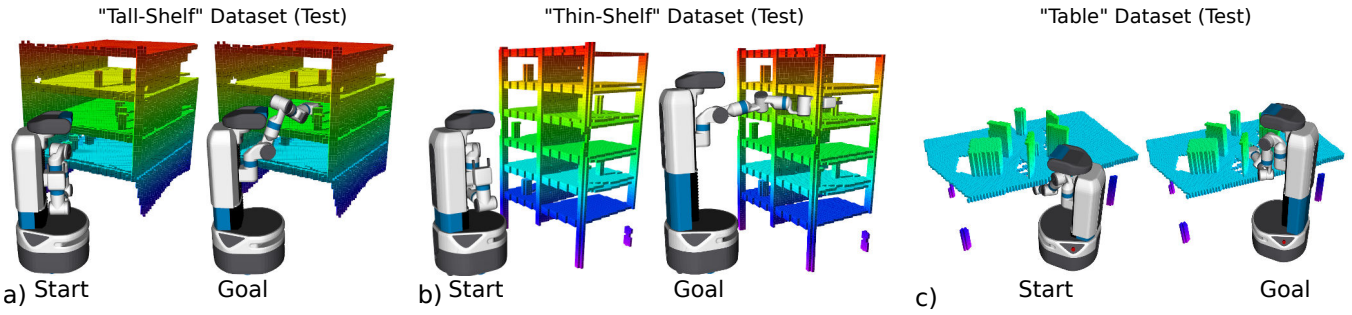

Fig. 5. The three datasets used to test the evaluated methods. Different problems are generated similarly to Fig. 4. **a)** An example environment from the "Tall-Shelf" dataset. The "Tall-Shelf" is created by stacking the "Small-Shelf" three times. **b)** An example environment from the "Thin-Shelf" dataset. This is also a bookcase like "Small-Shelf" and "Tall-Shelf", but the shelves are shorter and there is a divider, making it a much more challenging problem. **c)** An example environment from the "Table" dataset, which includes a table with several objects and is very different from the other datasets.

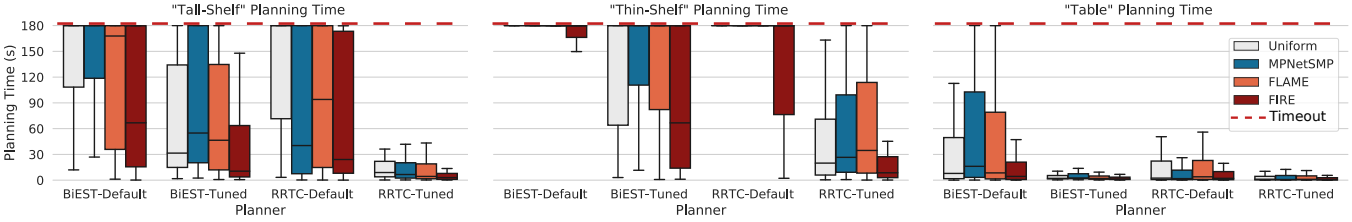

Fig. 6. Planning time (including retrieval) when testing in the three datasets shown in Fig. 5. All of the methods are only trained with the "Small-Shelf" dataset. The timeout was set to 180 seconds.

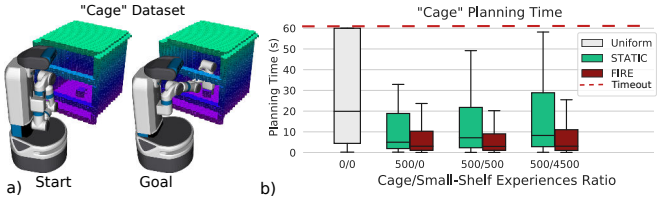

Fig. 7. **a)** An example problem from the "Cage" dataset. **b)** Planning time for 100 test examples from the "Cage" dataset using the RRTC-TUNED planner. The timeout was set to 60 seconds. The x-axis shows the number of experiences that exist in $\mathcal{DB}$ from "Small-Shelf" and from "Cage". Note that "Small-Shelf" and "Cage" have very different solution paths. In other words, the experiences from "Small-Shelf" do not transfer to "Cage".

poor biased sampling (if nothing is retrieved it defaults to UNIFORM). On the other hand, FIRE outperformed all other methods even in "Thin-Shelf" and "Table", demonstrating that the learned similarity function generalizes to problems that are significantly different than those in the training dataset. We also note that "Table" has a different x_{START} configuration than the training dataset "Small-Shelf". This demonstrates the usefulness of independently considering x_{START} and x_{GOAL} in the local primitives defined by FIRE.

C. Robustness to irrelevant experiences

1) Learning (Training): In this experiment, we do not retrain FIRE's similarity function and use the one obtained from training on "Small-Shelf" from Sec. V-A. However, now we add to $\mathcal{DB}$ example problems from both "Cage" and "Small-Shelf". Note that the problems from "Cage" and "Small-Shelf" are highly dissimilar in terms of solution paths. Thus, when solving a problem from "Cage" a good similarity function should not retrieve experiences generated from "Small-Shelf". The x-axis in Fig. 7b shows the ratio of example problems from "Cage" and "Small-Shelf". For example, $500/0$ denotes an experience database $\mathcal{DB}$ that has 500 examples from "Cage" and 0 examples from "Small-Shelf".

2) Evaluation (Testing): In this experiment, we tested on 100 example problems from the "Cage" dataset using RRTC-TUNED as the underlying planner. We compared with STATIC to illustrate how irrelevant experiences from "Small-Shelf" affect performance. The results in Fig. 7b show that although STATIC significantly outperforms UNIFORM, its performance degrades as we add irrelevant experiences in the training dataset. On the other hand, FIRE is robust to the irrelevant experiences from "Small-Shelf" added to $\mathcal{DB}$ since it maintains its good performance even with the $500/4500$ ratio. FIRE's similarity function was only trained on "Small-Shelf" while $\mathcal{DB}$ includes experiences from "Cage". This demonstrates that the learned latent space can successfully structure local primitives it was not trained on.

VI. Conclusion

In this work, we have proposed FIRE, a framework that learns a similarity function for motion planning problems with sensed environments. Using the learned similarity function, FIRE retrieves relevant experiences from a database in the form of "critical samples" that can informatively guide any sampling-based motion planner. Through our experiments, we demonstrated the generalization of FIRE outside its training dataset. Furthermore, FIRE can also learn incrementally without retraining by simply adding experiences in $\mathcal{DB}$, and can discriminate between relevant and irrelevant experiences.

In the future, we would like to improve FIRE by bounding its memory requirements and treating biased samples differently from uniform samples [22], [24]. Additionally, we would like to investigate how the same ideas apply to other problems that include motion planning such as task and motion planning or kinodynamic planning.

REFERENCES

[1] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. on Control Syst. Tech.*, vol. 17, no. 5, pp. 1105–1118, 2009.

[2] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. Konidaris, "Robot motion planning on a chip," in *Robotics: Science and Syst.*, 2016.

[3] N. T. Dantam, Z. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *Int. J. of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[4] J. F. Canny, *The Complexity of Robot Motion Planning*. MIT Press, 1988.

[5] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, 1996.

[6] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *Int. J. of Computational Geometry and Applications*, vol. 9, no. 4n5, pp. 495–512, 1999.

[7] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2000.

[8] O. Salzman, "Sampling-based robot motion planning," *Communications of the ACM*, vol. 62, no. 10, pp. 54–63, 2019.

[9] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," *IEEE J. Robot. Autom.*, vol. 34, no. 2, pp. 111–127, 2013.

[10] D. Coleman, I. A. Sucan, M. Moll, K. Okada, and N. Correll, "Experience-based planning with sparse roadmap spanners," in *IEEE Int. Conf. Robot. Autom.*, pp. 900–905, 2015.

[11] C. Chamzas, A. Shrivastava, and L. E. Kavraki, "Using local experiences for global motion planning," in *IEEE Int. Conf. Robot. Autom.*, pp. 8606–8612, May 2019.

[12] C. Chamzas, Z. Kingston, C. Quintero-Peña, A. Shrivastava, and L. E. Kavraki, "Learning Sampling Distributions Using Local 3D Workspace Decompositions for Motion Planning in High Dimensions," in *IEEE Int. Conf. Robot. Autom.*, June 2021.

[13] J.-M. Lien and Y. Lu, "Planning motion in environments with similar obstacles," *Robotics: Science and Syst.*, 2009.

[14] G. Tang and K. Hauser, "Discontinuity-sensitive optimal control learning by mixture of experts," in *IEEE Int. Conf. Robot. Autom.*, pp. 7892–7898, 2019.

[15] W. Merkt, V. Ivan, T. Dinev, I. Havoutis, and S. Vijayakumar, "Memory clustering using persistent homology for multimodality-and discontinuity-sensitive learning of optimal control warm-starts," *IEEE Trans. Robot.*, 2020.

[16] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," *IEEE Int. Conf. Robot. Autom.*, vol. 3, pp. 4420–4426, 2003.

[17] D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *Int. J. of Robotics Research*, vol. 25, no. 7, pp. 627–643, 2006.

[18] V. Boor, M. H. Overmars, and a. V. D. Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *IEEE Int. Conf. Robot. Autom.*, vol. 2, pp. 1018–1023, May 1999.

[19] J.-M. Lien, S. L. Thomas, and N. M. Amato, "A general framework for sampling on the medial axis of the free space," in *IEEE Int. Conf. Robot. Autom.*, vol. 3, pp. 4439–4444, 2003.

[20] H. Kurniawati and D. Hsu, "Workspace-based connectivity oracle: An adaptive sampling strategy for prm planning," in *Int. Wksp. on the Algorithmic Foundations of Robotics*, pp. 35–51, Springer, 2008.

[21] M. Zucker, J. Kuffner, and J. A. Bagnell, "Adaptive workspace biasing for sampling-based planners," *IEEE Int. Conf. Robot. Autom.*, pp. 3757–3762, 2008.

[22] D. Molina, K. Kumar, and S. Srivastava, "Learn and link: Learning critical regions for efficient planning," in *IEEE Int. Conf. Robot. Autom.*, pp. 10605–10611, 2020.

[23] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *IEEE Int. Conf. Robot. Autom.*, pp. 7087–7094, May 2018.

[24] B. Ichter, E. Schmerling, T.-W. E. Lee, and A. Faust, "Learned critical probabilistic roadmaps for robotic motion planning," in *IEEE Int. Conf. Robot. Autom.*, pp. 9535–9541, 2020.

[25] R. Kumar, A. Mandalika, S. Choudhury, and S. S. Srinivasa, "LEGO: Leveraging experience in roadmap generation for sampling-based planning," *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2019.

[26] R. K. Jenamani, R. Kumar, P. Mall, and K. Kedia, "Robotic motion planning using learned critical sources and local sampling," *arXiv preprint arXiv:2006.04194*, 2020.

[27] I. Patil, B. K. Rout, and V. Kalaichelvi, "Prediction of bottleneck points for manipulation planning in cluttered environment using a 3d convolutional neural network," *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)*, pp. 358–364, 2019.

[28] R. Terasawa, Y. Ariki, T. Narihira, T. Tsuboi, and K. Nagasaka, "3d-cnn based heuristic guided task-space planner for faster motion planning," in *IEEE Int. Conf. Robot. Autom.*, pp. 9548–9554, IEEE, 2020.

[29] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Trans. Robot.*, 2020.

[30] T. Jurgenson and A. Tamar, "Harnessing reinforcement learning for neural motion planning," in *Robotics: Science and Syst.*, 2019.

[31] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song, "Learning to plan in high dimensions via neural exploration-exploitation trees," in *Int. Conf. on Learn. Repr.*, 2020.

[32] M. Farber, "Topological complexity of motion planning," *Discrete and Computational Geometry*, vol. 29, no. 2, pp. 211–221, 2003.

[33] J. J. Rice and J. Schimmels, "Multi-homotopy class optimal path planning for manipulation with one degree of redundancy," *Mechanism and Machine Theory*, vol. 149, p. 103834, 2020.

[34] M. Stolle, H. Tappeiner, J. Chestnutt, and C. G. Atkeson, "Transfer of policies based on trajectory libraries," in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pp. 2981–2986, IEEE, 2007.

[35] T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, "Memory of motion for warm-starting trajectory optimization," *IEEE Robot. Autom. Letters*, vol. 5, no. 2, pp. 2594–2601, 2020.

[36] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *IEEE Int. Conf. Robot. Autom.*, pp. 3671–3678, 2012.

[37] È. Pairet, C. Chamzas, Y. R. Petillot, and L. E. Kavraki, "Path planning for manipulation using experience-driven random trees," *IEEE Robot. Autom. Letters*, vol. 6, p. 3295–3302, Apr. 2021.

[38] S. Finney, L. P. Kaelbling, and T. Lozano-Pérez, "Predicting Partial Paths from Planning Problem Parameters," in *Robotics: Science and Syst.*, 2007.

[39] T. F. Iversen and L.-P. Ellekilde, "Kernel density estimation based self-learning sampling strategy for motion planning of repetitive tasks," in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pp. 1380–1387, IEEE, 2016.

[40] P. Lehner and A. Albu-Schaeffer, "Repetition sampling for efficiently planning similar constrained manipulation tasks," *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pp. 2851–2856, 2017.

[41] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *International workshop on similarity-based pattern recognition*, pp. 84–92, Springer, 2015.

[42] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 3637–3645, 2016.

[43] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3DMatch: Learning local geometric descriptors from rgb-d reconstructions," in *CVPR*, 2017.

[44] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and Freight: Standard platforms for service robot applications," in *Wksp. on Autom. Mobile Service Robots*, 2016.

[45] A. Orthey, A. Escande, and E. Yoshida, "Quotient-space motion planning," in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pp. 8089–8096, IEEE, 2018.

[46] B. Raveh, A. Enosh, and D. Halperin, "A little more, a lot better: Improving path quality by a path-merging algorithm," *IEEE Trans. Robot.*, vol. 27, no. 2, pp. 365–371, 2011.

[47] D. Chicco, "Siamese neural networks: An overview," *Artificial Neural Networks*, pp. 73–94, 2020.

[48] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, vol. 2, pp. 1735–1742, 2006.

[49] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[50] M.-F. Balcan, A. Blum, and N. Srebro, "A theory of learning with similarity functions," *Machine Learning*, vol. 72, no. 1-2, pp. 89–112, 2008.

[51] C. Chamzas, C. Quintero-Peña, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, "MotionBenchMaker: A Tool to Generate and Benchmark Motion Planning Datasets," *IEEE Robot. Autom. Letters*, vol. 7, p. 882–889, Apr. 2022.

[52] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[53] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *IEEE Int. Conf. Robot. Autom.*, vol. 2, pp. 995–1001, 2000.

[54] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robot. Autom. Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[55] S. Chitta, I. Sucan, and S. Cousins, "Moveit!," *IEEE Robot. Autom. Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[56] Z. Kingston and L. E. Kavraki, "Robowflex: Robot motion planning with moveit made easy," *arXiv preprint arXiv:2103.12826*, 2021.

[57] M. Moll, I. A. Şucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robot. Autom. Magazine*, vol. 22, no. 3, pp. 96–102, 2015.