List Prediction Applied To Motion Planning

Abhijeet Tallavajhula[1], Sanjiban Choudhury[1], Sebastian Scherer[1], Alonzo Kelly[1]

***Abstract*— There is growing interest in applying machine learning to motion planning. Potential applications are predicting an initial seed for trajectory optimization, predicting an effective heuristic for search based planning, and even predicting a planning algorithm for adaptive motion planning systems. In these situations, providing only a *single* prediction is unsatisfactory. It leads to many scenarios where the prediction suffers a high loss. In this paper, we advocate *list* prediction. Each predictor in a list focusses on different regions in the space of environments. This overcomes the shortcoming of a single predictor, and improves overall performance. A framework for list prediction, CONSEQOPT, already exists. Our contribution is an extensive domain-specific treatment. We provide a rigorous and clear exposition of the procedure for training a list of predictors. We provide experimental results on a spectrum of motion planning applications. Each application contributes to understanding the behavior of list prediction. We observe that the benefit of list prediction over a single prediction is significant, irrespective of the application.**

I. INTRODUCTION

There have been a number of efforts to introduce procedures from machine learning to motion planning. These procedures offer the ability to adapt motion planning algorithms to a specific environment. Adaptation is performed in a data-driven manner, and past experience in the form of solved motion planning environments can be leveraged to train the learning procedure.

In this work, we consider adaptation in the form of selecting from a fixed set of options. Assume we have a library of elements, and each element is an option for a motion planning algorithm. Given an environment, we want to predict which element to use. For instance, in a trajectory optimization algorithm, the library elements are seed trajectories, and the environment is the configuration of obstacles. The objective is to find a learning procedure that has good prediction performance, i.e, how well can the learnt procedure predict the best element in the library?

Machine learning procedures provide guarantees on expected performance, which is the average performance over a probability distribution of environments. A predictor trained on this distribution has interesting behavior. It predicts the correct element on most environments, but has extremely low performance on environments which are infrequent. This is particularly unsatisfactory for motion planning, which demands good prediction performance on all environments.

In this work, we do not tease a worst-case performance guarantee out of learning theory. Instead, we advocate a procedure that takes steps towards meeting the motion planning performance demand—list prediction. This involves training not one, but multiple predictors. They are trained sequentially. Importantly, they focus on different environments. As illustrated in Fig. 1, the second predictor will focus on environments where the first predictor had low performance. The third will focus on environments where the first two had low performance, and so on.

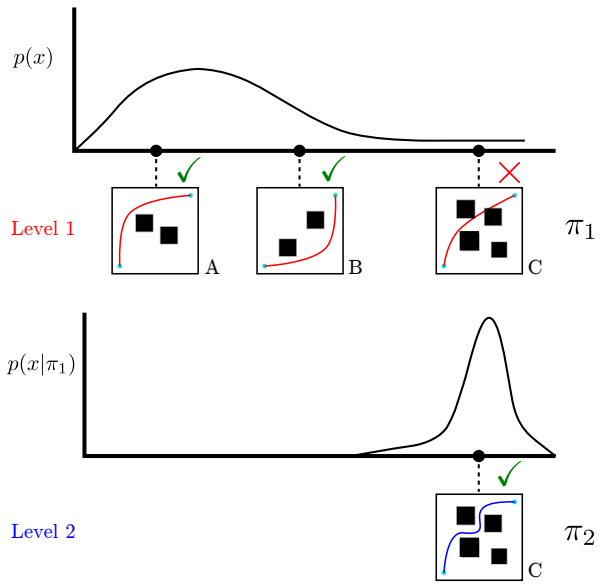

Fig. 1: An illustration of how list prediction works for seed prediction. The first level predictor, π_1, performs well on environments such as A and B which are likely under the distribution $p(x)$. It performs poorly on infrequent environments like C . At the second level, unsolved environments are likely under the distribution $p(x|\pi_1)$. C is solved by predictor π_2.

From a learning viewpoint, a list of predictors will always perform better than a single predictor. A list also addresses the motion planning concern of not only how, but also where performance is low—the second predictor will attempt to do well on the infrequent environments which the first predictor ignored. A list is also compatible with motion planning applications which have the scope to evaluate multiple options in parallel, and select the best one. The size of the list, henceforth called the budget, can be chosen to reflect the computational resources available to the motion planning systems to make a decision within a time period.

List prediction is not a new topic, having been introduced to motion planning as CONSEQOPT [1]. We use CONSEQOPT as a base, but make the following contributions

1) Clear derivation of the training procedure for list prediction in the framework of loss-sensitive multiclass classification.

[1]The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA atallav1@andrew.cmu.edu, sanjiban,basti,alonzo@cmu.edu

2) Analyzing the behavior of predictors at different levels across a spectrum of motion planning applications.

The overview of the paper is as follows—in Section II we describe the problem formally, at the end of which it is evident that training a list of predictors is justified. Section III is devoted to implementation, where we spend time discussing exactly how prediction is performed. Some properties of list prediction are highlighted in Section IV. In Section V, we explore the operation of CONSEQOPT, developing intuition and visualizing predictions. Our experiments are on the entire spectrum of planning problems—from predicting seeds for trajectory optimization, to heuristics for search based planners, to planning algorithms for adaptive motion planning systems. We believe that our insights will benefit any effort undertaken in applying prediction to motion planning, clarifying understanding and arguing for the well-grounded procedure of list prediction.

II. FORMULATION

A. Element prediction

We start with predicting a single element. Let $x \in X$ denote a motion planning environment sampled from a distribution $p(x)$. We assume a library $\mathcal{L}$ of L elements is given, such that $\mathcal{L} = \{\xi_j\}, j = 1\colon L$, $\xi_j \in \Xi$. Each element has a cost when applied in an environment. We denote this cost by a scalar, $c(x, \xi)$. If planning time was not a constraint, we would step through the library and pick the lowest cost element. Instead, we want to predict an element which will have low cost. Since prediction can only do as well as the best element in the library, we define loss as a relative cost. The loss of predicting element ξ_j is

$$l(x, \xi_j) = c(x, \xi_j) - \min_{\xi \in \mathcal{L}} c(x, \xi) \tag{1}$$

A predictor π, belonging to a class of predictors Π, takes an environment as input and outputs a library element, i.e, $\pi : X \to \Xi$. The loss of a predictor is

$$l(x, \pi) = c(x, \pi(x)) - \min_{\xi \in \mathcal{L}} c(x, \xi) \tag{2}$$

Finally, the risk of π is the expected loss

$$R(\pi) = \int l(x, \pi) p(x) dx \tag{3}$$

We would like to find the minimum risk predictor. Assume we have training data in the form of environments and element costs, $\mathcal{D} = \{(x_i, c(x_i, \xi_j))\}, i = 1\colon N, j = 1\colon L$, $x_i \sim p(x)$, $\xi_j \in \mathcal{L}$. Training environments are i.i.d samples. We also assume that the costs of all elements in the library are available. The objective can then be stated as finding the predictor which minimizes the empirical risk

$$\hat{R}(\pi) = \frac{1}{N} \sum_{i=1}^{N} l(x_i, \pi) \tag{4a}$$

$$\pi = \arg\min_{\tilde{\pi} \in \Pi} \hat{R}(\tilde{\pi}) \tag{4b}$$

Observe that predicting one of L elements given x is like classifying x into one of L classes. So π is a multiclass classifier. The 'correct' class is the minimum loss element, but misclassification losses differ. Specifically, finding π is a case of loss-sensitive multiclass classification[1].

The loss l is non-convex, which makes it difficult to directly minimize the empirical risk. The solution approach detailed in [2], which we follow, is to replace l with a convex function, known as the surrogate loss. This is a well-known procedure in learning. To recall the simple case of binary classification, the $0-1$ loss is also non-convex. Binary classification is solved by replacing the loss with a convex surrogate. The step here can be thought of as a more general version of using surrogates. As per the analysis in [2], minimizing the convex surrogate risk implies minimizing the true risk. Using the surrogate requires the predictor π to be defined in terms of a scoring function. Let $s(x, \xi) \in \mathbb{R}$ be a function which assigns a score to library element ξ in environment x. The predictor then picks the element with the highest score

$$\pi(x) = \arg\max_j s(x, \xi_j) \tag{5}$$

Risk will be minimized in terms of the scoring function s, and therefore π. The predictor space Π also depends on the space of scoring functions. For any environment, the scores are required to be centered over the library elements. Intuitively, this means that the scores are required to be well-behaved. Otherwise, we would be free to assign arbitrarily high scores to low-loss elements, or arbitrarily low scores to high-loss elements. This constraint is expressed as

$$\sum_{j=1}^{L} s(x, \xi_j) = 0, \forall\, x \tag{6}$$

The last ingredient is $\psi(t) \in \mathbb{R}$, a convex function of t. The surrogate loss is then defined as

$$l_\psi(x, s) = \sum_{j=1}^{L} l(x, \xi_j) \psi(s(x, \xi_j)) \tag{7}$$

Note that the scores $s(x, \xi)$ don't enter the loss l_ψ directly, but through the convex function ψ. The surrogate loss is large when either the true loss $l(x, \xi_j)$ is large, or $\psi(s(x, \xi_j))$ is large. Intuitively, minimizing this loss encourages high scores to be assigned to the low-loss elements. The empirical surrogate risk is defined in terms of the surrogate loss. With the following shorthand: $l_{ij} = l(x_i, \xi_j)$, $s_{ij} = s(x_i, \xi_j)$, the transformed optimization problem is

$$\hat{R}_\psi(s) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} l_{ij} \psi(s_{ij}) \tag{8a}$$

$$s = \arg\min_{\tilde{s}} \hat{R}_\psi(\tilde{s}) \tag{8b}$$

We detail the choices of the convex surrogate $\psi(t)$, scorer $s(x, \xi)$, and the optimization scheme in Section III.

[1] Called *cost*-sensitive classification in the learning literature. Since we use costs to refer to element costs, we use the term *loss*-sensitive classification.

B. List prediction

We now move on to list prediction. An algorithm for this task, ConseqOpt, was presented in [1]. We broadly follow their treatment to find a list of predictors. The length of the list, or budget B, is fixed and decided by computational resources. The list of predictors is denoted by $\langle \pi^1 \colon \pi^B \rangle$. The list of elements predicted for environment x is $\langle \pi^1(x) \colon \pi^B(x) \rangle$. We use superscripts for levels in the list. Definitions in II-A are extended to lists. The loss in (2) is extended to take a list of predictors as argument

$$l(x, \langle \pi^1 \colon \pi^B \rangle) = \min_k c(x, \pi^k(x)) - \min_{\xi \in \mathcal{L}} c(x, \xi) \tag{9}$$

Observe that the loss only cares about the lowest cost element in the predicted list, $\min_k c(x, \pi^k(x))$. The risk of a list of predictors is the expected loss

$$R(\langle \pi^1 \colon \pi^B \rangle) = \int l(x, \langle \pi^1 \colon \pi^B \rangle) p(x) dx \tag{10}$$

The increased power of a list of predictors over a single predictor may seem to be offset by the difficult task of finding the optimal list. The optimal list requires a search over all lists of length B

$$\langle \pi^{1*} \colon \pi^{B*} \rangle = \arg \min_{\langle \tilde{\pi}^1 \colon \tilde{\pi}^B \rangle \in \Pi^B} R(\langle \tilde{\pi}^1 \colon \tilde{\pi}^B \rangle) \tag{11}$$

However, the risk is a monotone supermodular function. We don't spend time on supermodular and submodular functions here, but see [1] for details. The implication of this property is that there exists a simple algorithm for finding a near-optimal list: greedy selection. Selecting the first predictor π^1 is exactly the minimization in (4b). The second predictor is selected as $\pi^2 = \min_{\tilde{\pi} \in \Pi} R(\langle \pi^1, \tilde{\pi} \rangle)$, or minimizing the risk after fixing the predictor at the first level. In general, the greedy procedure is

$$R^k(\pi) = R(\langle \pi^1 : \pi^{k-1}, \pi \rangle) \tag{12a}$$

$$\pi^k = \arg \min_{\tilde{\pi} \in \Pi} R^k(\tilde{\pi}),\ k = 1 \colon B \tag{12b}$$

We refer to $R^k(\pi)$ as the risk at level k. It is a function of $\langle \pi^1 \colon \pi^{k-1} \rangle$. We also use the shorthand $l^k_{ij} = l(x_i, \langle \pi^1 \colon \pi^{k-1}, \xi_j \rangle)$ for the losses at the level k. They calculate the loss of element ξ_j, given the list of predictors $\langle \pi^1 \colon \pi^{k-1} \rangle$. For example, if one of the earlier predictors has already predicted the lowest loss element for environment x_i, then it does not matter which element is predicted at level k, and all the marginal losses will be zero, $l^k_{ij} = 0\ \forall j$. We again use the convex relaxation, optimizing the empirical surrogate risks at each level. The predictor π^k is defined in terms of a scoring function s^k at that level

$$\hat{R}^k_\psi(s) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} l^k_{ij} \psi(s_{ij}) \tag{13a}$$

$$s^k = \arg \min_{\tilde{s}} R^k_\psi(\tilde{s}) \tag{13b}$$

$$\pi^k(x) = \arg \max_j s^k(x, \xi_j),\ k = 1 \colon B \tag{13c}$$

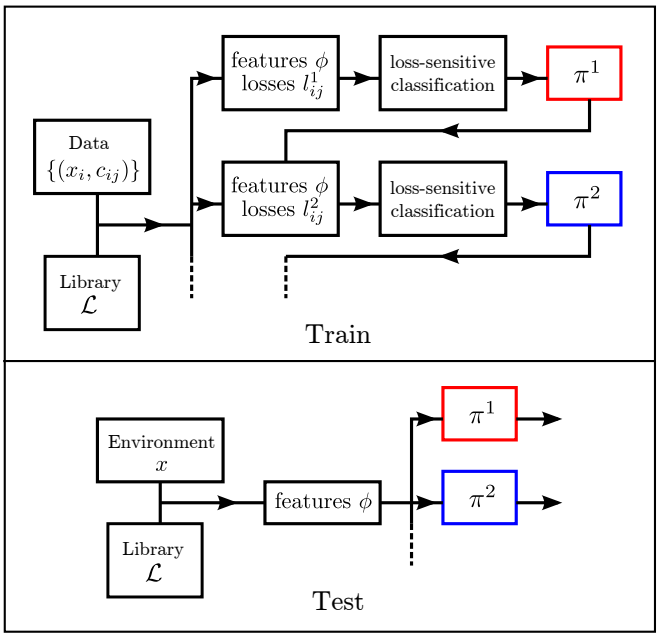

Fig. 2: A flowchart of the train and test steps of list prediction.

We close this section with an observation about the risks. If X is the space of environments, let $X|\neg 1$ be the space of environments where loss is non-zero under π^1. Similarly, $X|\neg 1 \colon \neg(k-1)$ is the space of environments where loss is nonzero under the list $\langle \pi^1 \colon , \pi^{k-1} \rangle$, or the space of unsolved environments at level k. Then

$$\begin{aligned} R(\langle \pi^1 \colon \pi^{k-1}, \pi \rangle) &= \int_X l(x, \langle \pi^1 \colon \pi^{k-1}, \pi \rangle) p(x) dx \\ &= \int_{X|\neg 1 \colon \neg(k-1)} l(x, \langle \pi^1 \colon \pi^{k-1}, \pi \rangle) p(x) dx \end{aligned} \tag{14}$$

It is only the space of unsolved environments that appear in the risk at level k. The environments already correctly classified are of no concern.

III. Implementation

A. Train

The data $\mathcal{D}$ consists of N environments x_i. We run elements of the library $\mathcal{L}$ on each environment, resulting in L costs c_{ij} for x_i. This gives us information about the performance of all elements over a number of environments. A further step is to extract features $\phi \in \mathbb{R}^d$ from the data. As in other learning procedures, the features encode the information relevant to prediction. The scoring function $s(x, \xi)$ was referred to in an abstract form in Section II. Here we make the scorer concrete. It is chosen to be a linear function of the features ϕ. We discuss how to solve loss-sensitive classification without referencing k, the list level. There are two cases for the features.

1) $\phi = \phi(x, \xi)$: Features are computed on an environment-element pair. There is unique information for each element in the context of an environment. $\phi_{ij} = \phi(x_i, \xi_j)$ is an abbreviation. $\hat{\phi}_{ij} = \phi_{ij} - \frac{1}{L} \sum_{l=1}^{L} \phi_{il}$ are the centered features, required for (6), the constraint that scores

sum to zero. Scores are linear in features, $s_{ij} = \beta^T \hat{\phi}_{ij}$, $\beta \in \mathbb{R}^d$. We choose two standard convex forms for ψ: hinge, $\psi(t) = (1+t)_+$, and square, $\psi(t) = (1+t)^2$. Plugging all terms into the expression for the empirical surrogate risk (8a), the optimization problems are

$$J(\beta) = \sum_{i=1}^{N} \sum_{j=1}^{L} l_{ij}(1 + \beta^T \hat{\phi}_{ij})_+ + \frac{\lambda}{2}\beta^T\beta \tag{15a}$$

$$J(\beta) = \sum_{i=1}^{N} \sum_{j=1}^{L} l_{ij}(1 + \beta^T \hat{\phi}_{ij})^2 + \frac{\lambda}{2}\beta^T\beta \tag{15b}$$

$$\min_{\beta \in \mathbb{R}^d} J(\beta) \tag{15c}$$

$J(\beta)$ is the sum of the empirical surrogate risk and a regularization term. λ is the regularization weight and controls for overfitting the training data. In binary classification, using a hinge ψ leads to an SVM. The form in (15a) is similar to an SVM. In binary classification, a square ψ leads to regression. Similarly, the form in (15b) corresponds to loss-weighted regression. We try to regress from the features to the losses, but focus on cases where losses l_{ij} are high.

2) $\phi = \phi(x)$: Features are computed on the environment only. We abbreviate $\phi_i = \phi(x_i)$. Since the features have no information about the library elements, the scorers use a set of weights, $\beta_j \in \mathbb{R}^d$, $j = 1\colon L$, one for each library element. $\hat{\beta}_j = \beta_j - \frac{1}{L}\sum_{l=1}^{L} \beta_l$ are the centered weights, required for constraint (6). With this choice, $s_{ij} = \hat{\beta}_j^T \phi_i$. The optimization problems are

$$J(\beta_{1:L}) = \sum_{i=1}^{N} \sum_{j=1}^{L} l_{ij}(1 + \hat{\beta}_j^T \phi_i)_+ + \frac{\lambda}{2} \sum_{j=1}^{L} \beta_j^T \beta_j \tag{16a}$$

$$J(\beta_{1:L}) = \sum_{i=1}^{N} \sum_{j=1}^{L} l_{ij}(1 + \hat{\beta}_j^T \phi_i)^2 + \frac{\lambda}{2} \sum_{j=1}^{L} \beta_j^T \beta_j \tag{16b}$$

$$\min_{\beta_{1:L} \in \mathbb{R}^d} J(\beta_{1:L}) \tag{16c}$$

All the above optimization problems are convex optimization problems. In this paper, we solve them using the primal subgradient method. By introducing a superscript k into the losses, l_{ij}^k, and weights, β_j^k, in the above, we get the objectives, J^k, to be optimized at each level.

B. Test

After training, we obtain a list of scorers, which in turn defines a list of predictors, $\langle \pi^1 \colon \pi^B \rangle$. The scorers are used for prediction as in (13c). During testing, we are handed a query environment x. We compute features ϕ based on x and the library $\mathcal{L}$. The features are fed to each predictor, resulting in a list of elements $\langle \pi^1(x) \colon \pi^B(x) \rangle$. Training and testing for list prediction are summarized in Figure 2.

IV. Discussion

We highlight some aspects of the training procedure in this section.

A. Focus on unsolved environments

Rewriting the risk at level k as in (14), we see the mathematical reason for the intuition that deeper in the list, the focus is on unsolved environments. This is also clear from the optimization setup (15, 16). If x_i has been classified correctly before level k, $l_{ij}^k = 0 \, \forall \, j$, and x_i is ignored by the optimization. Earlier predictions in the list tend to be generic, while subsequent predictions focus on the specifics of the training data.

The predictor space Π plays a role in performance. A weak space may not be able to deal with the unsolved environments. Increasing the length of the list will not help, as $X|\neg 1 \colon \neg(k-1)$ will not reduce significantly with k. On the other hand, a powerful space Π may achieve sufficiently low risk in the first level. The issue of Π is, however, orthogonal: predicting a list will never have higher risk than predicting a single element, and is always helpful.

B. Cost regression

Given the training data, an approach that comes to mind is to regress directly from features ϕ to costs c, and pick the element with the lowest cost. We refer to this procedure as cost regression. If successful, cost regression provides a lot of information. It would estimate the minimum cost that can be achieved by the library. It would also allow arbitrary addition of elements to the library.

In this work, we assume the library is given, and focus only on prediction. This allows us to deal with losses instead of costs. Cost regression is aiming for a more difficult task than necessary. The next approach that suggests itself is to regress from features ϕ to losses l. This is close, but not correct. The right procedure following from using the convex surrogate is loss-weighted regression, as seen in (15b). With classification, we have not found scope for adding new elements to the library. When the library is modified, the list of predictors must be trained again.

C. Element costs

Element costs may have to be preprocessed to capture the quantity of interest. For example, in trajectory seed optimization, a large finite cost may have to be assigned to 'failed' seeds. Similarly, if we consider any cost below a limit as 'solving' trajectory planning, thresholding is required. Once costs are decided, a fixed notion of loss is used throughout list prediction.

V. Experiments

We now discuss our experiments applying list prediction to a spectrum of motion planning applications. The presentation of each application follows a pattern. We first describe the application setting and motivation for list prediction. We then walk through the experiment setup. Details such as the size of training data are collected in Table I. Due to space constraints, the details are kept concise. Further information can be found in [3]. Since we have already described the implementation, we jump straight to the results after the setup.

TABLE I: Application Details

Application	Library Length L	List Budget B	Feature Dimension d	Train Data N	Validation Data	Test Data
Seed Prediction 2D	39	3	73	700	200	100
Seed Prediction 7D	30	3	17	310	112	100
Heuristic Prediction	101	3	1620	675	193	96
Planner Prediction	100	3	1764	579	166	82

Empirical risks for all applications are in Tables II-V. We observe that the risk of a list is significantly lower than the risk of a single element, irrespective of the application. In addition, the risk is lower when using the hinge surrogate loss compared to the square surrogate loss. Each subsection is accompanied by figures showing sample lists predicted for the application.

A. Seed Prediction for Trajectory Optimization of 2D Point Robot

1) Motivation: For the problem of planning a trajectory from a start to goal configuration, local trajectory optimization is an approach where an initial seed joining the start to goal is optimized. While these methods are fast, their solution quality is heavily dependent on the initial seed. Often these methods converge to a bad local minimum around the initial seed, e.g, passing through the middle of obstacles. The effectiveness of a seed is not known apriori. A computationally expensive approach is to optimize every element in a library of seed trajectories. List prediction can be used instead to predict a small set of elements.

2) Environment x and distribution $p(x)$: We consider a point robot in a 2D environment. The environment x consists of square obstacles generated from a uniform random distribution $p(x)$, see Figure 3. The start and goal configuration are fixed for all environments.

3) Element ξ and library $\mathcal{L}$: An element is a seed trajectory that connects the start to the goal. The library $\mathcal{L}$ consists of diverse trajectories that are optimal on environments drawn from $p(x)$.

4) Costs $c(x, \xi)$: A seed ξ is used as an input to a local optimizer in environment x. The cost of a trajectory is the sum of a smoothness term and an obstacle proximity term. CHOMP [4] was used as the local optimizer. $c(x, \xi)$ is the cost of the trajectory that results after the seed ξ is optimized.

5) Features ϕ: Features $\phi(x, \xi)$ are computed on a pair of environment and seed. The optimization problem for this case is (15). ϕ is a vector containing information about downsampled gradients around ξ and in a local region around it.

6) Results: The distribution $p(x)$ places large probability mass on environments where obstacles are clustered. The predictor at the first level predicts simple seeds that go around these clusters. But there are environments which require the optimal seed to be in a complicated homotopy class. These are infrequent under $p(x)$, so they are ignored by the first predictor. Subsequent predictors focus on these environments, making customized predictions. Figures 3 and 4 make this point with specific examples.

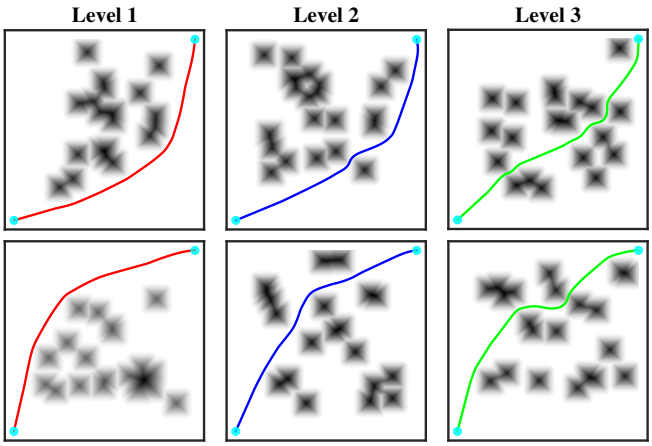

Fig. 3: Training phase for Section V-A. The objective is to optimize a trajectory from start to goal (cyan dots) in an obstacle field (grayscale image). The examples shown are problems solved by predictors at different levels and the trajectories shown are post-optimization. The level 1 predictor (red) learns a simple classification rule to solve a large number of problems—it predicts seeds that simply go around a cluster of obstacles to achieve the lowest cost. The level 2 predictor (blue) focusses on and solves environments that level 1 did not solve. It learns to predict seeds in better homotopy classes. The level 3 predictor (green) focusses on corner cases, e.g the two instances shown have the optimal trajectory passing through a narrow gap that is surrounded by local minima. The level 3 predictor learns seeds that are optimized into this narrow gap.

B. Heuristic Prediction in Search Based Planning

1) Motivation: Heuristics are essential to improving the runtime performance of search based planning. Recent approaches such as MHA* [5] create a framework that allows the use of an inadmissible heuristic as long as it is anchored by weighted A*. MHA* allows heuristics to have a lot of flexibility and effectively act as modules that expand promising states only. Given a library of inadmissible heuristics, list prediction can be used to predict a small subset of heuristics.

2) Environment x and distribution $p(x)$: The objective is to the plan the motion of a 4 link arm from start to goal. The environment x consists of two rectangular blocks, one above and one below the arm. $p(x)$ is such that the horizontal positions of the blocks are uniformly random. See Figure 5.

3) Element ξ and library $\mathcal{L}$: An element is an inadmissible heuristic. A heuristic is an exponential kernel on a chosen state, called an 'attractor state'. The heuristic penalizes arm states away from the attractor state [2]. A library of heuristics is generated by randomly sampling attractor states. We also add an element to the library corresponding to no heuristic—MHA* reverts to weighted A*.

[2]The kernel is defined only in 3 out of the 4 dimensions, i.e, the function is invariant to the base joint.

TABLE II: Seed Trajectory Prediction for 2D Point Robot

List Size	Hinge Loss	Square Loss
Single Element	0.1073	0.1106
3 Elements	0.0715	0.0721

TABLE III: Seed Trajectory Prediction for 7D Manipulator

List Size	Hinge Loss	Square Loss
Single Element	15.454	13.013
3 Elements	3.6085	3.6799

TABLE IV: Heuristic Prediction

List Size	Hinge Loss	Square Loss
Single Element	0.0976	0.0933
3 Elements	0.0325	0.0360

TABLE V: Planner Prediction

List Size	Hinge Loss	Square Loss
Single Element	0.2222	0.2281
3 Elements	0.0222	0.0281

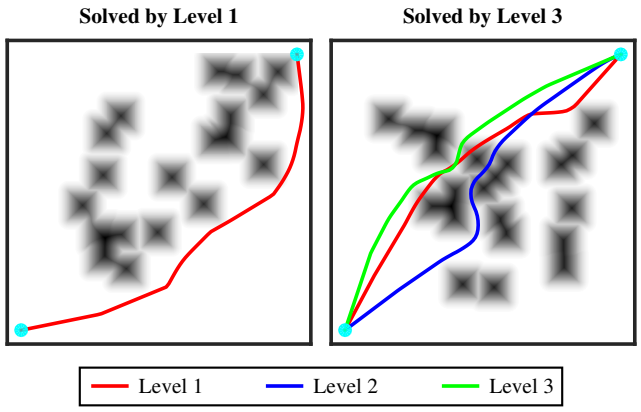

Fig. 4: Test predictions for Section V-A. The environment on the left is solved by the level 1 predictor (red) which predicts an initial seed that goes around the obstacles on optimization. On the other hand, the environment on the right is solved only by level 3. The optimal trajectory passes through a narrow gap with a kink while there are many local minima surrounding this trajectory. Level 1 (red) makes a naive prediction that gets stuck cutting across obstacles. Level 2 (blue) comes closer to solving it but chooses a wrong homotopy class. Level 3 (green) solves the environment by predicting a seed which is optimized into the narrow gap.

4) Costs $c(x, \xi)$: The element ξ is used as a heuristic input to MHA*. MHA* plans on a lattice created by discretizing each joint space 12 times. $c(x, \xi)$ is set to be the number of states expanded when using ξ (a maximum of 10000 expansions are allowed). The cost is scaled from 0 to 10.

5) Features ϕ: Features $\phi(x)$ are computed on the environment only. The optimization problem for this case is (16). ϕ is a vector of Histogram of Gradients on the image of the environment.

6) Results: Under $p(x)$, environments frequently have a sufficient gap between the two blocks for the arm to pass through. The predictor at the first level predicts attractor states corresponding to simple arm 'tucking' configurations. Environments in which the blocks are close together, leading to a narrow gap, are infrequent. These environments require a complicated 'tucking' attractor state. The subsequent predictors solve such environments, as seen in Figure 5.

C. Planner Prediction in Adaptive Motion Planning

1) Motivation: The effectiveness of a planning algorithm to plan a trajectory in an environment within a time constraint depends on the configuration of obstacles. The notion of a list of planners to create a planner ensemble has shown promising results [8].

2) Environment x and distribution $p(x)$: The objective is to the plan the motion of a 2D point robot from start to goal. The environment x consists of circular obstacles. $p(x)$ is such that the positions and radii of the obstacles are sampled uniformly. See Figure 6.

3) Element ξ and library $\mathcal{L}$: Each element is a sampling based motion planning algorithm. A library of such algorithms is generated by varying tree growing strategies, sampling strategies and heuristics.

4) Costs $c(x, \xi)$: The planner ξ is used to plan a trajectory within a time constraint of 0.05s. $c(x, \xi)$ is set to be equal to the path length of the solution. The cost is affinely transformed to $[0, 20]$. If no feasible path was found, $c(x, \xi)$ is set to 20.

5) Features ϕ: Features $\phi(x)$ are computed on the environment only. The optimization problem for this case is (16). ϕ is a vector of Histogram of Gradients on the image of the environment.

6) Results: The first predictor predicts planners such as BIT*, RRT-Connect and Informed-RRT*. These planners don't make strong assumptions about structure in the environment, which results in good performance over a wide range of environments. Environments with structure are infrequent under $p(x)$. We observe that subsequent predictors predict planners which exploit structure. See Figure 6.

VI. Related Work

With a formulation for list prediction in place, we can discuss related work in a common language. Jetchev and Toussaint [9] was an early work on predicting seeds for trajectory planning. Cost regression, which we defined as directly regressing from features to costs, and classification were implemented. Classification was found to perform better. Our work uses the formalism of loss-sensitive classification to arrive at both regression and classification. In IV-B, we also reason about cost regression being a more difficult task than classification. Dragan et al. [10] predicted the usefulness of end-effector goals for trajectory planning on a manipulator. Their work was not limited to using a library of elements. However, we offer justifications for some heuristics they considered. For example, [10] used a threshold on costs to focus on relevant environments in

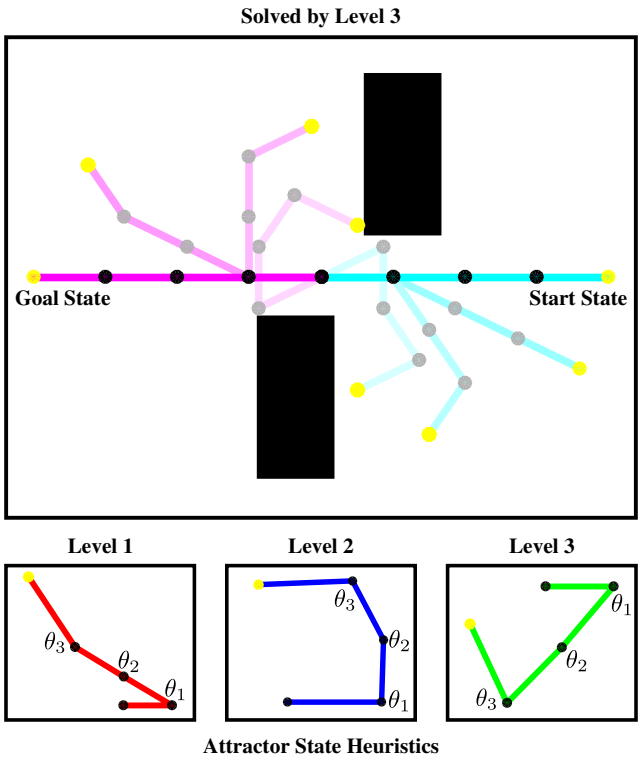

Fig. 5: Sample lists predicted for Section V-B. MHA* is used to plan a trajectory for a 4 link arm from a start state (cyan) to goal state (magenta). The environment has two blocks creating a gap. To pass through, the arm has to tuck in and roll out again. Heuristics used are attractor states. The figure shows an infrequent environment for which the gap is narrow. The predictor at level 1 predicts a heuristic (red) that naively tucks the arm in the direction of the gap. However the upper block hinders this approach. The predictor at level 2 predicts a heuristic (blue) that brings the end effector closer to the base joint, however, the gap is small enough that this too is ineffective. The predictor at level 3 predicts a heuristic (green) that tucks the arm in a non-trivial configuration that allows it to pass through the gap. Snapshots from the trajectory resulting from the third heuristic are shown.

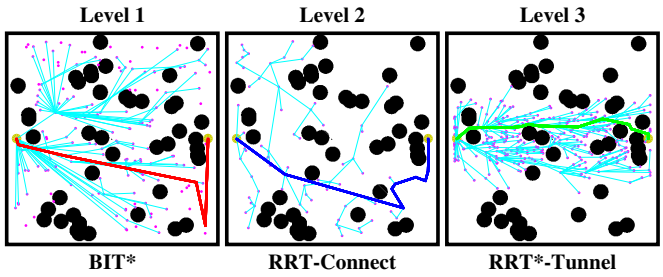

Fig. 6: Sample lists predicted for Section V-C. A planning algorithm is used to plan a trajectory from start to goal (beige). The environment consists of random circular obstacles. In this environment, the goal happens to be blocked off by a wall of 3 obstacles. The predictor at level 1 naively predicts BIT* [6], given its effectiveness in solving frequently occurring environments. However, the wall makes it difficult for BIT* to find a good solution in the time budget. The predictor at level 2 predicts RRT-Connect [7], given its effectiveness on environments where BIT* fails—however this too cannot plan around the wall. The predictor at level 3 predicts RRT*-Tunnel [8], as it concentrates sampling in a tunnel around the initial straight-line solution, and finds a path through the gap in the wall.

the training data. The threshold itself required tuning. In our approach, weighting training data by losses, as in (15), naturally follows from using a surrogate loss. Importantly, both [9] and [10] provided only single predictions. While a stronger predictor class Π was suggested to increase performance, a list was not. As our results show, predicting lists can lead to significant improvements.

CONSEQOPT for list prediction appeared in Dey et al. [1]. The algorithm was presented along with a theoretical analysis. Direct regression from features to losses was used for prediction in [1]. In contrast, our discussion IV-B points out that loss-weighted regression is what follows from the square surrogate loss. We also investigate the behavior of list prediction in motion planning more thoroughly. [1] considered seed prediction on a 7D manipulator[3], while we show results on a larger variety of problems. A follow up to CONSEQOPT was Ross et al. [11]. It considered the online setting, with data streaming in. In that setting, it was shown that training a single predictor for predicting a list works well.

We believe the concepts of libraries, loss-sensitive classification and list prediction will be of benefit to existing work in motion planning. Specifically, they will benefit applications that allow multiple predictions to be made and evaluated, either sequentially or in parallel.

One such area that can benefit is trajectory prediction. Zucker [12] generates a 'behavior library' of optimized trajectories and predicts the best trajectory given a query. Berenson et al. [13] generates a library of past plans and uses a heuristic to select one that can be repaired easily to solve a new environment. Pan et al. [14] predicts if a seed trajectory will be successful for local optimization. Poffald et al. [15] uses a library of motion primitives and predicts the best primitive that can be adapted for a new environment. Jain et al. [16] learns a preference function on a library of trajectories. List prediction fits seamlessly into all these frameworks.

A niche where list prediction would make a direct impact is in the generation of diverse trajectory library. Current approaches resort to greedy maximization of coverage to generate a library of trajectories [17], [18], [19]. Such problems would benefit greatly from machine learning techniques generating lists of diverse trajectories, as shown in [1].

Wzorek et al. [20] predicts a motion planning strategy, from a library, that can be applied to repair a plan. Palmieri and Arras [21] learns a distance metric for an RRT to predict the nearest neighbor. Pan et al. [22] learns a collision probability for queries to reduce the number of collision checking. Morales et al. [23] predicts a motion planner to apply to different sections of a planning problem. Vernaza and Lee [24] learn a descent direction for trajectory optimization. Even in these applications, it is possible to use list prediction for improving performance.

[3]We ran experiments on the dataset from [1] for seed prediction on a 7D manipulator. Results are in Table III.

VII. Conclusion

We have presented comprehensive arguments advocating list prediction. We set up a framework, developed intuition, and demonstrated results on a variety of planning problems. To establish a firm base, each step in list prediction that we considered is theoretically justified. However, there are a number of modifications to the procedure, which may help improve performance in practice.

In this work, the same features ϕ were used at all levels. We could instead propagate information down levels to improve performance. For example, in seed prediction, knowing which seed was predicted at the first level, and failed, might enable better predictions at the second level. One way to propagate information is to construct different features at different levels, ϕ^k. ϕ^k includes information from $\langle \pi^1(x) \colon \pi^{k-1}(x) \rangle$, so π^k has explicit knowledge of the past predictions. Note that the space of predictors at each level, Π^k, is now different.

Building the set $\mathcal{L}$, or library generation, is higher than list prediction in the hierarchy of decision making. The simplest method of library generation is to sample the space of elements, as in our heuristics library. It may be beneficial to generate the library more intelligently.

Our trajectory seeds library, for example, consisted of optimal trajectories in environments drawn from $p(x)$. At any given iteration of the library creation process, an environment is randomly sampled. If the library does not contain a seed which when optimized leads to an acceptable solution, an expensive global optimization routine is invoked to solve the optimization. The result is then appended to the library. The process is repeated till the library size crosses a size limit.

The library elements are thus associated with environments. This 'library information' could also be appended to the features ϕ. See [3] for results of these strategies on our datasets. A formal investigation into library generation is an interesting direction for future work.

VIII. Acknowledgement

Sanjiban Choudhury acknowledges the support from ONR grant N000141310821. Abhijeet Tallavajhula was supported by the National Science Foundation under Grant Number 1317803. The authors particularly thank Debadeepta Dey for a number of discussions, and sharing the grasp seed prediction dataset.

References

[1] D. Dey, T. Y. Liu, M. Hebert, and J. A. Bagnell, "Contextual sequence prediction with application to control library optimization," in *Robotics Science and Systems, RSS*, 2013.

[2] B. Ávila Pires, C. Szepesvari, and M. Ghavamzadeh, "Cost-sensitive multiclass classification risk bounds," in *International Conference on Machine Learning, ICML*, 2013.

[3] A. Tallavajhula and S. Choudhury, "List prediction for motion planning: Case Studies," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-15-25, September 2015.

[4] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[5] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic A*," in *Seventh Annual Symposium on Combinatorial Search*, 2014.

[6] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *IEEE International Conference on Robotics and Automation, ICRA*, 2015.

[7] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation, ICRA*, 2000.

[8] S. Choudhury, S. Arora, and S. Scherer, "The Planner Ensemble: Motion planning by executing diverse algorithms," in *IEEE International Conference on Robotics and Automation, ICRA*, 2015.

[9] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," *Autonomous Robots*, vol. 34, no. 1-2, pp. 111–127, 2013.

[10] A. Dragan, G. Gordon, and S. Srinivasa, "Learning from experience in manipulation planning: Setting the right goals," in *Proceedings of the International Symposium on Robotics Research (ISRR) 2011*, July 2011.

[11] S. Ross, J. Zhou, Y. Yue, D. Dey, and J. A. Bagnell, "Learning policies for contextual submodular prediction," *arXiv preprint arXiv:1305.2532*, 2013.

[12] M. Zucker, "A data-driven approach to high level planning," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-09-42, January 2009.

[13] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *IEEE International Conference on Robotics and Automation, ICRA*, 2012.

[14] J. Pan, Z. Chen, and P. Abbeel, "Predicting initialization effectiveness for trajectory optimization," in *IEEE International Conference on Robotics and Automation, ICRA*, 2014.

[15] M. Poffald, Y. Zhang, and K. Hauser, "Learning problem space metrics for motion primitive selection," in *IROS Workshop on Machine Learning in Planning and Control of Robot Motion*, 2014.

[16] A. Jain, B. Wojcik, T. Joachims, and A. Saxena, "Learning trajectory preferences for manipulators via iterative improvement," in *Advances in neural information processing systems*, 2013, pp. 575–583.

[17] C. Green and A. Kelly , "Toward optimal sampling in the space of paths," in *13th International Symposium of Robotics Research*, November 2007.

[18] L. H. Erickson and S. M. LaValle, "Survivability: Measuring and ensuring path diversity," in *IEEE International Conference on Robotics and Automation, ICRA*, 2009.

[19] S. Aurora, S. Choudhury, D. Althoff, and S. Scherer, "Emergency maneuver library - ensuring safe navigation in partially known environments," in *IEEE International Conference on Robotics and Automation, ICRA*, 2015.

[20] M. Wzorek, J. Kvarnström, and P. Doherty, "Choosing replanning strategies for unmanned aircraft systems," in *International Conference on Automated Planning and Scheduling, ICAPS*.

[21] L. Palmieri and K. O. Arras, "Distance metric learning for rrt-based motion planning with constant-time inference," in *IEEE International Conference on Robotics and Automation, ICRA*, 2015.

[22] J. Pan, S. Chitta, and D. Manocha, "Faster sample-based motion planning using instance-based learning," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 381–396.

[23] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato, "A machine learning approach for feature-sensitive motion planning," in *Algorithmic Foundations of Robotics VI*. Springer, 2005, pp. 361–376.

[24] P. Vernaza and D. D. Lee, "Learning dimensional descent for optimal motion planning in high-dimensional spaces," in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.