Learning problem space metrics for motion primitive selection

Marcela Poffald, Yajia Zhang, and Kris Hauser

Abstract— **Because computing solutions to optimal control and motion planning problems from scratch can be slow and inefficient, many researchers have proposed using a motion library, which contains precomputed example motions that can be adapted quickly to solve novel problems. This paper considers the task of selecting a "good" solution to adapt to a new problem, which we approach in a general machine learning framework. Our approach is to learn a metric in problem feature space for which proximity in space correlates highly with suitability for solution adaptation. With such a metric, motion primitive retrieval becomes a nearest neighbor query. We empirically compare the adaptation performance of several metric learning techniques on both synthetic and optimal motion planning datasets.**

I. INTRODUCTION

With recent developments in real-time robotics applications such as self-driving cars, human-robot interaction, and legged robots, it has become apparent that the computational expense of motion optimization is a major roadblock. These optimization problems are high-dimensional and time-sensitive, and solving these problems from scratch using traditional techniques is impractically computationally expensive. A promising approach for real-time optimization is that of reusing existing solution information so that new problems are not computed from scratch every time. The goal of such a method is to reduce online computational cost without sacrificing the quality of solutions. One such approach for this is the method of the motion library [5], which makes use of stored knowledge about optimal solutions in order to reduce online computation time required to generate new solutions.

The motion library approach in general consists of two phases: an offline phase, in which selected problems are solved and stored as motion primitives, and an online phase, in which a new problem is matched to a primitive and its solution is computed by adaptation from this primitive. In this paper we consider the primitive retrieval problem, that is, upon observing a new problem online, to select a solution to a previously-seen problem to be adapted to the new problem. The idea of this work is that for the adaptation to succeed well, the two problems should be, in some sense, similar. But, *measuring similarity between problems* is challenging. Naive approaches, e.g., euclidean distance in problem feature space, are unlikely to succeed without impractically huge motion libraries, because they fail to consider significant aspects of problem space, e.g., many irrelevant features, such as the positions of far-away obstacles. Other authors have relied on special-purpose heuristics, the design of which is more art than science. The primary contribution of this work is a new method for retrieving good motion primitives via metric learning on a problem feature space.

Unlike special-purpose heuristics, a learning approach allows our method to work with general problem representations and solution adaptation procedures. It relies on a relatively weak assumption, that the given solution adaptation method works better (i.e., more efficiently and yielding higher-quality solutions) when the source and destination problems are relatively similar, and works worse otherwise. In this work, we learn a metric that explicitly matches up new problems with the primitives in the library that are well-suited for solution adaptation. The online phase uses this metric to determine the best primitive using a nearest neighbor query. The intended result is that online solutions to novel problems will be adapted quickly and at a low computational expense.

Our distance metric learning approach uses a training set of problems that are well-paired with primitives in a given motion library. The training phase learns a metric that encourages the assignment of new problems to primitives for which adaptation yields high quality solutions, and discourages the pairing of problems for which adaptation is unsuccessful or solution is low-quality. We consider two general forms for the metric. A global approach uses a single metric function across the entire problem space, while a local approach assigns a local metric around each primitive. We evaluate their efficacy by looking at the resulting adaptation cost of the pairings generated by the 1NN classification to see if the overall cost decreases as a result of using the metrics. A decrease in adaptation costs indicates that the metric learning procedures are helping to encourage good pairings for adaptation.

Experiments compare several learning techniques against a baseline of Euclidean distance on both synthetic datasets and real motion planning datasets. Results suggest that learned distance metrics indeed improve performance significantly over the Euclidean baseline. We conclude with suggestions for future work in this area.

II. EXISTING WORK

Existing work in both the offline and online aspects of the motion library approach focus on the aspects of problem representation [11], [13], [6], [2] and adaptation [10], [6]. Some work has been done on the retrieval and use of appropriately-matched primitives, such as in [2] and [10]. In [2], the focus is on the construction of a hierarchical

*This research is partially supported by NSF Grant No. 1218534.

School of Informatics and Computing, Indiana University at Bloomington, Bloomington, IN 47405 USA { mpoffald,zhangyaj,hauserk } @indiana.edu

graph structure in which nodes in a branch of the structure all share similar courses of movement. The learning of new tasks is executed via $A^\star$ search of this graph, beginning at the node corresponding to the start position and searching for a path to the desired end position through the graph. In [10], a mixture-of-experts approach is used to retrieve primitives, each of which is weighted based on the probability of it being successful in the current context. This probability is modelled by an exponential distribution whose parameters are learned via linear regression, so that the resulting probabilities are based on their outcome in previous trials. Using these weights, primitives are then blended into solutions to the new problem. In this paper, we investigate the framing of the problem as a distance metric learning task using a 1NN classification approach for the assignment of new problems to a single primitive. This allows greater leeway in handling non-blendable primitives (e.g., ones of vastly different durations) or using more powerful adaptation procedures (e.g., nonlinear warps, or processes that perform light motion planning and validation).

Existing work in metric learning includes feature extraction approaches [16], [14] and global metric learning [12]. A method to improve accuracy of binary classification [3] uses the support vectors from SVM classification to estimate the orientation of class boundaries, and then generates a local weighting scheme from this information. In contrast to these methods, our method does not focus on binary classification, but rather focuses on identifying a single primitive from a library. Primitive selection can be considered an extreme version of multi-class classification, where every training point is its own class. Local metric approaches are also discussed in [1] [17], which use eigenvector analysis to devise local weighting schemes that reflect feature variance.

III. PROBLEM DESCRIPTION

This section will describe the primitive selection problem and the metric learning problem that we consider in this paper.

A. Primitive selection

The *primitive selection problem* can be stated follows. We are given a *primitive library* as a set of solved problems $P = \{(p_1,s_1),\ldots,(p_k,s_k)\}$, where each *primitive* (p_i,s_i) consists of a problem p_i annotated with a precomputed solution s_i. We will assume that problems are specified (or approximated) in a d-dimensional *feature space* $\mathbb{R}^d$. We are also given an *adaptation function* $s \leftarrow adapt((p_i,s_i),p)$ that attempts to adapt s_i to a new problem p to produce a new solution s. Since determining each solution from scratch is usually an expensive proposition, we assume the adaptation function is a much faster method that produces a near-optimal solution for novel problems suitable for real-time use. For example, one might simply use the prior solution directly ($s = s_i$), perform a deterministic warping function [9], optimize a cost function [4], or use the solution to bias a sampling-based motion planner [15].

The primitive selection problem asks to choose a primitive (p_i,s_i) to be adapted to a novel problem p. Denoting the selection procedure as $g : \mathbb{R}^d \rightarrow \{(p_1,s_1)\ldots,(p_k,s_k)\}$, the end result to the motion planning problem is determined by the procedure:

$$adapt(g(p),p) \tag{1}$$

To define a metric for the "goodness" of the selection / adaptation process, let us define the adaptation quality with a function $adaptcost((p_i,s_i),p)$ that gives a numerical quality score for the process of adapting a primitive (p_i,s_i) to problem p. Typical functions would most likely include a weighted combination of computational cost and success rate of the *adapt* procedure, and motion quality of the result $s \leftarrow adapt((p_i,s_i),p)$. Lower values are preferred. The adaptation cost should, for example, assign minimal cost to an instant, reliable, and optimal adaptation. The tradeoffs between these factors are application-dependent.

Theoretically, the ideal primitive is the result of the expression:

$$g^\star(p) = \arg\min_{(p_i,s_i)\in P} adaptcost((p_i,s_i),p). \tag{2}$$

However, it would be foolhardy to evaluate $g^\star$ directly, because it requires computing the *adapt* function in *adaptcost* for every primitive. Although each *adapt* operation is relatively fast compared to solving a problem from scratch, this calculation would be prohibitively expensive in large libraries. Hence, we wish to produce a $g(p)$ that approximates $g^\star(p)$ and is evaluated quickly.

B. Machine learning approaches

We can cast the primitive selection problem as a k-class classification problem, where each primitive is a class. To learn a classifier we are given a training set consisting of examples $D = \{(x_1,y_1,c_1),\ldots,(x_M,y_M,c_M)\}$ where each x_i is a problem sampled from problem space, y_i is a primitive, and $c_i = adaptcost(y_i,x_i)$. We also consider the optimal-primitive dataset $\tilde{D} = \{(x_1,y_1^\star),\ldots,(x_N,y_N^\star)\}$ where each $y_i^\star$ is the adaptation cost-minimizing primitive for x_i. The learning problem is to determine a k-class classifier g such that $g(p) = g^\star(p)$ over the entire problem space. (In a slight abuse of notation we will consider a primitive and its index as equivalent).

Whereas most existing metric learning techniques attempt to minimize misclassification rate:

$$\sum_{(x,y^\star)\in\tilde{D}} I[y^\star \neq g(x)] \tag{3}$$

for primitive selection, a more appropriate loss function is *adaptation cost*:

$$\sum_{(x,y^\star)\in\tilde{D}} adaptcost(g(x),x) - adaptcost(y^\star,x). \tag{4}$$

This is because adaptation cost captures the suboptimality of a given primitive selector when applied to x.

Standard machine learning techniques for binary classification can usually be adapted to the multiclass case either

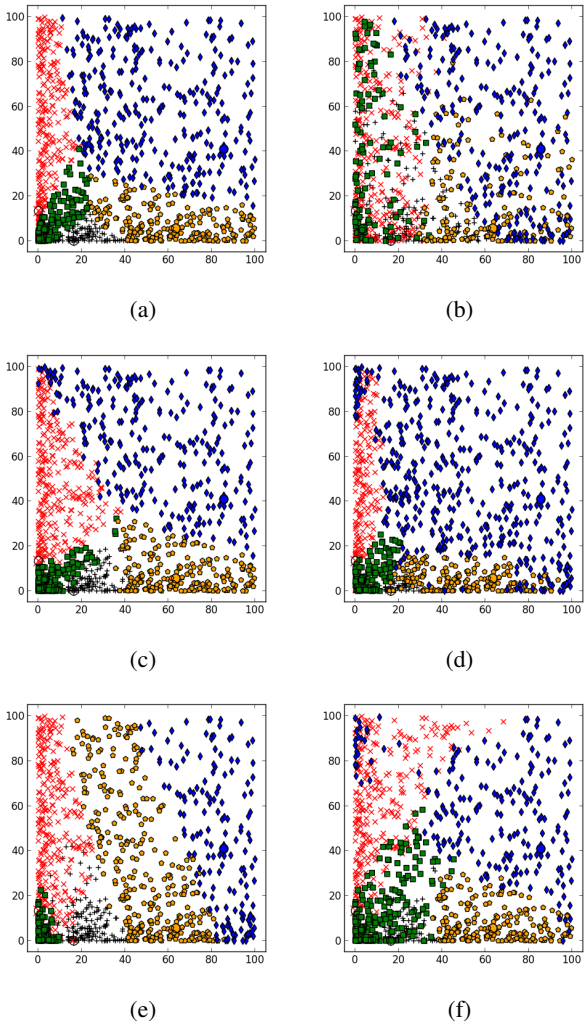

Fig. 1: Illustrating some of the compared metric learners on a synthetic 5-primitive dataset in 3 dimensions, where the 3rd dimension is irrelevant noise and is not depicted. (a) ground truth optimal primitives. (b) Euclidean distance predictions (48% accuracy). (c) Direct adaptation learning (82%). (d) Local KISSME (75%). (e) Global LMNN (61%). (f) Local LMNN (70%).

by training k one-vs-rest classifiers or $O(k^2)$ one-vs-one classifiers and then merging the results of all classifiers, e.g., via voting. However, this approach has several drawbacks: 1) training and evaluation is computationally expensive when k is high, 2) such models are not human interpretable, and 3) they cannot handle additions of primitives to the primitive library without re-training over the entire dataset. Such methods would prevent a robot from adapting its primitive library to its environment, online. Hence we prefer a nearest neighbor approach, for which nearest neighbor queries can be accelerated using a variety of indexing data structures (e.g., ball trees) or by using approximate nearest neighbors queries. However, since standard Euclidean distance is relatively poor at identifying similar problems, we adopt a metric learning approach.

C. Nearest neighbors selection

In particular, a nearest neighbor approach defines a primitive selection model:

$$g(p) = \arg\min_i d(p_i, p) \tag{5}$$

where d is a problem space metric $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$.

For this model to have a low adaptation cost, the problem domain must satisfy a few assumptions:

1) A problem's solution is the best primitive for adapting to the same problem.
2) For each primitive (p_i, s_i), the set $P_i = \{p | i = g^\star(p)\}$, i.e., those problems for which primitive (p_i, s_i) yields the lowest-cost adaptation, is localized about p_i.
3) All points in the Voronoi cell of p_i have a low adaptation cost.

The first two assumptions are reasonable because adapting solutions to problems that are "far" will be less likely to succeed. To clarify the third assumption, for each primitive (p_i, s_i), define the the set $P_{i,\varepsilon} = \{p | adaptcost(g(p), p) - adaptcost(g^\star(p), p) < \varepsilon\}$, i.e., those problems for which primitive (p_i, s_i) yields an adaptation that is "good enough", with tolerance parameter ε. If, then, the Voronoi cell of p_i is contained within $P_{i,\varepsilon}$, we can ensure that the nearest neighbors approach yields high quality adaptations everywhere. This assumption is justifiable, as long as ε is sufficiently high or the motion library populates problem space relatively densely.

D. Metric Learning

We now turn to the question of generating a good distance metric d. The simplest form of metric is a squared Mahalanobis distance:

$$d_M(x, y; \mathbf{M}) = (x - y)^T \mathbf{M} (x - y) \tag{6}$$

with $\mathbf{M}$ a positive semidefinite matrix.

This approach is *global* in that the metric is applied uniformly across the space. We also consider a *local* nearest neighbors approach that defines a different metric per primitive:

$$d(p_i, y) \equiv d_{\mathbf{M}^i}(p_i, y) = (p_i - y)^T \mathbf{M}^i (p_i - y) \tag{7}$$

where each $\mathbf{M}^i$, $i = 1, \ldots, k$ is a positive semidefinite matrix that defines the localized distance metric about each primitive p_i. At testing time, the classifier compares a new problem to each primitive using that primitive's distance metric, and is assigned to the one to which its the closest. In other words,

$$g(p) = \arg \min_{(p_i, s_i) \in P} d_{\mathbf{M}^i}(p_i, p) \tag{8}$$

The main advantage of a local metric is that it is able to adapt much more closely to the decision boundaries between near and far primitives. However, learning is much more susceptible to overfitting, and the learned model is of size $O(kd^2)$, which may be prohibitively large to store in memory when the library or feature space is large.

Within these two metric classes we compare the following metric learning methods:

1) Euclidean. $\mathbf{M}$ is the identity matrix.
2) Covariance learning. A quickly-trained method based on sphering the covariance of points near a primitive. We test it only in local form.
3) KISSME [8]. A method inspired by likelihood ratio tests, which determines $\mathbf{M}$ as a difference between precision matrices of "near" and "far" points. Although the original publication only considers global metrics we compare both global and local forms.
4) Large Metric Nearest Neighbors (LMNN) [12]. Optimizes a loss function that measures the margin between "near" and "far" points. This method is applied in both global and local forms.
5) Direct adaptation loss minimization. A gradient descent is applied to (4). Applied only in global form.

We note that covariance learning, KISSME, and LMNN are trained by splitting training data into "near" and "far" sets. To preprocess the dataset, we define a problem x as "near" to the primitive y if the adaptation cost $adaptcost(y,x)$ is within a fraction of ε of the optimal (amongst examples in D involving x). In this paper we use $\varepsilon = 0.1$. For each primitive p_j we define the set of near problems as $T_j = \{x \mid (x, p_j, z) \in D \text{ and } z \leq (1+\varepsilon)g^\star(x)\}$. We also define the set of far problems as $\bar{T}_j = \{x \mid (x, p_j, z) \in D \text{ and } z > (1+\varepsilon)g^\star(x)\}$.

E. Local covariance metrics

The simplest local metric learning method uses the distribution of each class to sphere the points belonging to each class. The method computes the covariance matrix of near pairs centered about p_j:

$$\mathbf{A}_j = 1/|T_j| \sum_{x \in T_j} (x - p_j)(x - p_j)^T \tag{9}$$

and sets $\mathbf{M}^j = \mathbf{A}_j^{-1}$. In practice, it is important to avoid singularities and protect against overfitting by adding a small multiple of the identity matrix λI to $\mathbf{A}_j$ before performing the inversion (we use $\lambda = 0.1$).

F. KISSME learning

KISSME is a refinement of the covariance learning technique that also learns the distribution of far points, and discriminates the class of a new point using a likelihood ratio test. It computes $\mathbf{A}_j$ as above as well as the covariance of far points:

$$\mathbf{B}_j = 1/|\bar{T}_j| \sum_{x \in \bar{T}_j} (x - p_j)(x - p_j)^T \tag{10}$$

and then defines the metric as the closest positive semidefinite matrix to

$$\tilde{\mathbf{M}}_j = |T_j|\mathbf{A}_j - |\bar{T}_j|\mathbf{B}_j \tag{11}$$

where $\mathbf{M}_j$ is determined by projecting $\tilde{\mathbf{M}}_j$ to the subspace of positive semidefinite matrices.

G. LMNN learning

The global and local LMNN algorithms described in [12] search the cone of positive semidefinite matrices for a metric that maximizes the margin between inputs belonging to one class and the nearest inputs of any other class. Both the global and local metric learning procedures utilize this same framework. We discuss the global method first, and then outline the modified local version.

The basic idea of the method is that if $x_i \in T_j$ and $x_l \in \bar{T}_j$, we would like the matrix $\mathbf{M}$ to measure x_i as being closer to p_j than x_l is. LMNN also attempts to enforce a unit margin of 1 so that a perfect classifier would satisfy:

$$\begin{aligned} &(p_j - x_i)^T\mathbf{M}(p_j - x_i) + 1 < (p_j - x_l)^T\mathbf{M}(p_j - x_l) \\ &\forall x_i \in T_j, \forall x_l \notin T_j, \forall p_j \in P \end{aligned} \tag{12}$$

Because perfect classifiers are rare, LMNN uses a slack variable approach. Thinking of the target neighbors for each primitive as a cluster, LMNN penalizes the number of points that invade the perimeter of clusters to which they were not assigned. These intruding points are known as *impostors*. Formally, an impostor is a point x_l which violates (12) by being closer to a primitive p_j than a true neighbor of that primitive, x_i, as measured by the metric $\mathbf{M}$. Formally, an impostor is any point $x_l \in \bar{T}_j$ for which:

$$(p_j - x_l)^T\mathbf{M}(p_j - x_l) \leq (p_j - x_i)^T\mathbf{M}(p_j - x_i) + 1 \tag{13}$$

$$\text{for some } x_i \in T_j \tag{14}$$

holds.

So, LMNN minimizes the following loss function over the values of $\mathbf{M}$:

$$\begin{aligned} loss(\mathbf{M}) =& (1-\mu) \sum_{p_j, x_i \in T_j} (p_j - x_i)^T\mathbf{M}(p_j - x_i) + \\ & \mu \sum_{p_j, x_i \in T_j, x_l \in \bar{T}_j} \max(1 + (p_j - x_i)^T\mathbf{M}(p_j - x_i) - \\ & (p_j - x_l)^T\mathbf{M}(p_j - x_l), 0) \end{aligned} \tag{15}$$

The first summand of this equation penalizes distances between each primitive p_j and its target neighbors T_j, and acts to keep the value of $\mathbf{M}$ small. The second term penalizes encroachment of impostors x_l into each p_j's cluster by summing the amount by which each impostor violates the margin in (12). The tradeoff between these two objectives is controlled by the parameter $\mu \in [0,1]$. The LMNN learning procedure uses a gradient-descent method to minimize this function over the cone of positive semidefinite matrices.

The local LMNN procedure is similar, but any distance involving a point $x_i \in T_j$ uses the metric assigned to the cluster T_j, $\mathbf{M}^j$. In this manner, all metrics are trained in relation to each other as a global optimization, and scaling problems between metrics are averted. The new loss function, with these cluster-centered metrics, is

$$loss(\mathbf{M}^1 \dots \mathbf{M}^k) = (1-\mu) \sum_{p_j, x_i \in T_j} (p_j - x_i)^T \mathbf{M}^j (p_j - x_i) + \mu \sum_{p_j, x_i \in T_j, x_l \in \bar{T}_j} [1 + (p_j - x_i)^T \mathbf{M}^j (p_j - x_i) - (p_j - x_l)^T \mathbf{M}^l (p_j - x_l)]_+ \quad (16)$$

Again, gradient descent is used to minimize loss.

IV. Direct adaptation loss minimization

Because they were designed for classification problems, KISSME and LMNN attempt to reduce misclassification rate rather than the adaptation loss. We present a new formulation that considers the loss of the closest primitive only against the optimal primitive. This is similar to LMNN with an ε of 0, except that loss is measured only with respect to the worst impostor. The new loss function:

$$loss(\mathbf{M}) = \sum_{x_i} [\min_{p_j} (p_j - x_i)^T \mathbf{M} (p_j - x_i) - (y_i^\star - x_i)^T \mathbf{M} (y_i^\star - x_i)]_+ \quad (17)$$

and is minimized using gradient descent over positive semidefinite matrices.

V. Experiments

For each learning method we evaluate adaptation loss (4) for problems with variety of sizes and amounts of training data. We note that although learning times were reasonable for all tested cases (from seconds up to several hours), our tests involve relatively small datasets ($k \leq 100$ and $|D| \leq 100,000$). In the future, with much larger primitive libraries, it is likely that one may choose a less accurate, quickly-learned method over one that has higher accuracy but impractical learning time. We note that our implementation is in Python, so learning times could be significantly sped up using a compiled language.

A. Synthetic dataset

We begin with a synthetic dataset that captures the ability of a metric learner to adapt to nonlinearities and noise in the mapping from problem space to adaptation costs. We first generate k primitives $\tilde{p}_1, \dots, \tilde{p}_k$ and N points $\tilde{x}_1, \dots, \tilde{x}_N$ at random in $[0,1]^{10}$, and then apply a nonlinear transformation $f(\vec{\tilde{x}})$ to each point to obtain the primitive library $p_1, \dots, p_k$ and training points $x_1, \dots, x_N$. As a proxy of $adaptcost(x_i, p_j)$ we use the Euclidean distance $\|\tilde{x}_i - \tilde{p}_j\|$ in the original problem space. In other words, the best primitive for a given point is the nearest primitive in the pre-transformation space.

As a deterministic nonlinear transformation we simply apply a power to each component:

$$f(\vec{\tilde{x}}) = [\tilde{x}_1^\alpha, \tilde{x}_2^\alpha, \dots, \tilde{x}_{10}^\alpha] \quad (18)$$

with the *Basic* dataset using $\alpha = 2$. We also test a *High Warp* transformation with $\alpha = 5$ to investigate the effect of greater nonlinearity on metric learner.

As a third *Noisy* transformation we consider the effect of added irrelevant dimensions:

$$f(\vec{\tilde{x}}) = [\tilde{x}_1^\alpha, \tilde{x}_2^\alpha, \dots, \tilde{x}_{10}^\alpha, \varepsilon_1, \varepsilon_2, \varepsilon_3] \quad (19)$$

where each $\varepsilon_1, \varepsilon_2, \varepsilon_3$ are random variables drawn from [0,1].

On a dataset with $k = 10$ primitives, 1000 training points, and 100 testing points, we obtain the following average loss:

Learner	Basic	High Warp	Noisy
Random	0.34	0.37	0.38
Euclidean	0.030	0.15	0.065
Direct learning (global)	0.017	0.11	**0.019**
Covariance local	**0.016**	**0.089**	0.038
KISSME global	0.025	0.12	0.030
KISSME local	0.13	0.11	0.13
LMNN global	0.034	0.15	0.026
LMNN local	0.048	0.16	0.078

We observe that local methods are better at the High Warp datasets due to its high nonlinearity. Irrelevant dimension of the noisy dataset most significantly degrade performance of the Euclidean and local covariance methods.

Next, to evaluate susceptibility to overfitting, we varied the proportion of training points per primitive:

Dataset	primitives	training	testing
$\times 10$	10	10,000	1,000
baseline	10	1,000	1,000
/ 10	10	100	1,000

and the sparseness of the dataset, i.e., by subsampling edges in the training set:

Dataset	primitives	training	testing	edges
baseline	10	1,000	1,000	10,000
20%	10	1,000	1,000	2,000
5%	10	1,000	1,000	500

Using the Noisy dataset as a baseline, we obtain the following average test loss for the modified datasets:

Learner	$\times 10$	baseline	/ 10	20%	5%
Random	0.36	0.38	0.33	0.36	0.36
Euclidean	0.078	0.065	0.068	0.064	0.079
Direct	**0.019**	**0.019**	**0.028**	**0.023**	**0.071**
Cov. l	0.025	0.038	0.080	0.079	0.16
KISSME g	0.04	0.030	0.045	0.051	0.45
KISSME l	0.13	0.13	0.13	0.16	0.39
LMNN g	0.039	0.026	0.037	0.041	0.18
LMNN l	0.058	0.078	0.12	0.28	0.39

Obviously more data is beneficial overall, but local methods in particular degrade faster with smaller datasets. This is fairly expected, since they are prone to overfitting.

B. Motion Primitive Dataset

Next, we test the performance of metric learning algorithm on a toy motion planning scenario. We consider a 4-link robot that moves from a fixed start to goal configuration among circular obstacles (see Fig. 2 left).

Problems are sampled by randomizing the number of obstacles and their positions. The problem feature vector is obtained by first dividing the 2D task space into a 10×10 grid, and recording the distance from each grid point to the nearest obstacle, or 0 if it is within the obstacle (Fig. 2 right). An asymptotically-optimal motion planner PRM* [7] is run

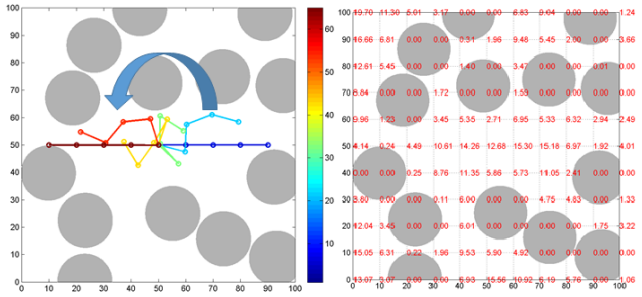

Fig. 2: Left: A toy motion planning problem that asks to connect the start configuration (blue) to the end configuration (dark red) without colliding with obstacles (dark circles). Right: Feature representation of a problem. Each vertex in the grid yields a feature equal to its distance to the nearest obstacle.

with a $1\,s$ cutoff on each problem to obtain a solution path. If no solution is found the problem is discarded.

To adapt a motion primitive s to a novel problem q, we first consider applying s directly via collision checking with the obstacles in q, up to some resolution e. If there is no collision, we admit s as a solution. If a collision occurs on some intermediate configuration(s), then we randomly perturb the joint angles to retract the configurations in collision. The perturbation angles are drawn from a zero mean Gaussian distribution with standard deviation 5°. This procedure is performed recursively, and terminates if no solution is found within a given cutoff time t (0.1 s in our implementation). Our adaptation costs are measured by the failure rate, as measured by 100 adaptation attempts (hence, we wish to maximize adaptation success rate).

We evaluated all learners on a library of 100 primitives, with 500 test problems, 500 training problems, and $|D|$ = 10,000, obtaining the following results:

learner	avg train loss	avg test loss
Random	0.29	0.28
Euclidean	0.33	0.33
Direct learning (global)	0.007	0.24
Covariance local	0.042	**0.11**
KISSME global	0.30	0.29
KISSME local	**0.0053**	**0.11**
LMNN global	0.17	0.25
LMNN local	0.10	0.22

suggest that direct learning and LMNN methods appear to have overfit to the data. The local covariance learning and local KISSME methods performed the best, yielding nearly indistinguishable performance on the test set. Global KISSME did not perform well, which suggests that local methods are needed to learn nonlinearities in the data set.

Here, average loss is equivalent to the expected failure rate of adapting a selected motion primitive to a new problem. Clearly, local methods perform better than global ones on this dataset, with KISSME leading to a reduction of the failure rate by 67% beyond Euclidean distance. The significance of this is that if one were to use the motion primitive approach, with a from-scratch plan as a backup in case primitive adaptation failed, the total running time would be $0.89 \times 0.1\,s + 0.11 \times 1\,s = 0.199\,s$ on average. This gives a five-fold speedup beyond planning from scratch.

VI. CONCLUSION

In this paper, we tested metric learning methods for selecting motion primitives in a motion library for adaptation to novel problems. We compared several local and global metric learning algorithms on synthetic datasets as well as a toy motion planning example. Our experiments suggest that learned metrics can reduce the cost of a selected primitive by several times beyond the one selected by Euclidean distance. Local metrics appear to perform better on large datasets but global metrics guard against overfitting. Future work will include tests on much larger libraries of motion primitives with higher-dimensional feature vectors to better assess performance and scalability trends. We will also investigate algorithms that can more flexibly balance fitting and overfitting.

REFERENCES

[1] Anna Atramentov and Steven M LaValle. Efficient nearest neighbor searching for motion planning. In *Int. Conf. on Robotics and Automation*, volume 1, pages 632–637. IEEE, 2002.

[2] Miha Deniša, Tadej Petric, Tamim Asfour, and Aleš Ude. Synthesizing compliant reaching movements by searching a database of example trajectories. In *Int. Conf. on Humanoid Robots*, 2013.

[3] Carlotta Domeniconi and Dimitrios Gunopulos. Efficient local flexible nearest neighbor classification. In *SDM*, 2002.

[4] Keith Grochow, Steven L Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. *ACM Transactions on Graphics (TOG)*, 23(3):522–531, 2004.

[5] Kris Hauser. Toward a "google" for robot motions. In *Proc. Workshop on Robotics Challenges and Vision*, pages 5–8, 2013.

[6] Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, 34(1-2):111–127, 2013.

[7] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Robotics Research*, 30(7):846–894, 2011.

[8] Martin Koestinger, Martin Hirzer, Paul Wohlhart, Peter M. Roth, and Horst Bischof. Large scale metric learning from equivalence constraints. In *Int. Conf. on Computer Vision and Pattern Recognition*, 2012.

[9] Alex X Lee, Sandy H Huang, Dylan Hadfield-Menell, Eric Tzeng, and Pieter Abbeel. Unifying scene registration and trajectory optimization for learning from demonstrations with application to manipulation of deformable objects. In *Int. Conf. Intelligent Robots and Systems*, 2014.

[10] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *Int. J. Robotics Research*, 32(3):263–279, 2013.

[11] Martin Stolle, Hanns Tappeiner, Joel Chestnutt, and Christopher G Atkeson. Transfer of policies based on trajectory libraries. In *Int. Conf on Intelligent Robots and Systems*, pages 2981–2986. IEEE, 2007.

[12] Kilian Weinberger, John Blitzer, and Lawrence Saul. Distance metric learning for large margin nearest neighbor classification. *Advances in neural information processing systems*, 18:1473, 2006.

[13] Elly Winner and Manuela Veloso. Automatically acquiring planning templates from example plans. In *Proc. AIPS-2002 Workshop on Exploring Real-World Plans*, 2002.

[14] Jie Xu, Jian Yang, and Zhihui Lai. K-local hyperplane distance nearest neighbor classifier oriented local discriminant analysis. *Information Sciences*, 232:11–26, 2013.

[15] Katsu Yamane, James J Kuffner, and Jessica K Hodgins. Synthesizing animations of human manipulation tasks. *ACM Transactions on Graphics (TOG)*, 23(3):532–539, 2004.

[16] Liu Yang and Rong Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2, 2006.

[17] Liu Yang, Rong Jin, Rahul Sukthankar, and Yi Liu. An efficient algorithm for local distance metric learning. In *AAAI*, volume 2, 2006.