Learning from Experience in Manipulation Planning: Setting the Right Goals

Anca D. Dragan, Geoffrey J. Gordon, and Siddhartha S. Srinivasa

Abstract In this paper, we describe a method of improving trajectory optimization based on predicting good initial guesses from previous experiences. In order to generalize to new situations, we propose a paradigm shift: predicting *qualitative attributes* of the trajectory that place the initial guess in the basin of attraction of a low-cost solution. We start with a key such attribute, the choice of a goal within a goal set that describes the task, and show the generalization capabilities of our method in extensive experiments on a personal robotics platform.

1 Introduction

We are interested in pushing the boundaries of trajectory optimization for robots in complex human environments. Optimization techniques have been well-documented to struggle in these domains by getting stuck in high-cost local minima. We envision two possible means of alleviating the problem. The first is to improve the optimizer itself by widening the basin of attraction of low-cost minima. The second is to improve initialization and start with trajectories that are in the basin of attraction of low-cost minima. In our previous work [16], we made progress on the former by widening the basin of attraction of an efficient trajectory optimizer CHOMP [15] by taking advantage of the often available flexibility in goal selection. In this work, we focus on the latter: learning to generate initial trajectories that enable the optimizer to converge to low-cost minima.

So how does the robot acquire this trajectory-generating oracle? In designing the oracle, we take advantage of three key features: the optimization process itself, the repetition in the tasks, and the structure in the scenes. The optimization process relieves us from the need to produce low-cost initial trajectories. *The cost of the trajectory is irrelevant, as long as it lies in the basin of attraction of a low-cost*

The Robotics Institute, Carnegie Mellon University, e-mail: {adragan, siddh}@cs.cmu.edu
Machine Learning Department, Carnegie Mellon University, e-mail: {ggordon}@cs.cmu.edu

trajectory. Repetition or similarity in tasks allows the oracle to *learn from previous experience* how to produce trajectories. Finally, structure in the scenes suggests that we can use *qualitative attributes* to describe trajectories. For example, in a kitchen, we could say "go left of the microwave and grasp the object from the right." These attributes provide a far more compact representation of trajectories than a sequence of configurations. This work combines all three features and proposes a learning algorithm that, given a new situation, can generate trajectories in the basin of attraction of a low-cost trajectory by predicting the values of qualitative attributes that this trajectory should posses. As a consequence, instead of focusing on every single voxel of a scene at once, we first make some key decisions based on previous experience, and then refine the details during the optimization.

The idea of using previous experience to solve similar problems is not new. In Artificial Intelligence, it is known as Case-Based Reasoning [7, 8], where the idea is to use the solution to the most similar solved problem to solve a new problem. In the MDP domain, Konidaris and Barto [9] looked at transferring the entire value function of an MDP to a new situation. Stolle and Atkeson constructed policies for an MDP by interpolating between trajectories [10], and then used local features around states to transfer state-action pairs to a new problem [11, 12]. In motion planning, learning from experience has included reusing previous collision-free paths [13] or biasing the sampling process in randomized planners [14] based on previous environments.

Jetchev and Toussaint [1] explored trajectory prediction in deterministic and observable planning problems. They focused on predicting globally optimal trajectories: given a training dataset of situations and their globally optimal trajectories, predict the globally optimal trajectory for a new situation. Much like Case-Based Reasoning, their approach predicted an index into the training dataset of trajectories as the candidate trajectory [1, 2] or clustered the trajectories and predicted a cluster number [1, 3]. Since prediction is not perfect, a post-processing stage, where a local optimizer is initialized from the prediction is used to converge to the closest local minimum.

Our approach differers in two key ways. First, we take advantage of the necessity of the optimization stage, and focus on the easier problem of predicting trajectories that fall in the basin of attraction of low-cost minima. Second, by predicting low-dimensional attributes instead of whole past trajectories, we are able to generate trajectories beyond the database of previous experience, allowing us to generalize further away from the training set.

Although the dataset-indexing techniques are a promising start in the field of learning from experience for trajectory optimization, they are limited: they are reminiscent of earlier works in computer vision (e.g. [17]), where one way to classify an image is to find the closest image in the training set according to some features and predict its label (or find a set of closest images and verify their predictions in post-processing). In 2006, the vision community started thinking about learning the distance metric between images [18], and this is the state at which trajectory prediction is now. In 2009 however, the object recognition community started changing this classification paradigm and shifting towards a much more general way of rec-

ognizing objects based on a simple idea: predict attributes of the object instead of the object itself, and then use the attributes to predict the object [4, 5]. This not only improved recognition of known objects, but also *allowed learners to recognize objects they had never seen before*. A similar technique was used in [6] to recognize from brain scans words that a subject was thinking, by using physical attributes of the words as an intermediate representation. We propose to do the same for trajectory prediction: rather than predicting trajectories directly, we predict qualitative attributes of the trajectories first, such as where their goal point is or which side of an obstacle they choose, and then map these qualitative attributes into initial guesses for a local optimizer.

In this work, after providing the motivation for the idea of using attributes in trajectory prediction, we focus on one key such attribute of the trajectory: its end point. Most manipulation tasks are described by an entire region of goals rather than a single goal configuration, and our previous work [16] has shown that the choice of a goal significantly impacts the outcome of the optimizer. Therefore, by getting better at selecting good goals, we are able to initialize the optimizer in better basins of attraction. We take advantage of the fact that the robot has easy access to locally optimal trajectories though its local optimizer, and can obtain a rich dataset of multiple trajectories for a situation along with their performance. We compare several methods of learning to predict this attribute value, from predicting if a value is the best choice or not, to learning to rank these values and selecting the highest-ranking one as the prediction. We show that all these algorithms perform best when they take into account additional data about sub-optimal performance (the "error" in "trial-and-error"). Using this information, learners predict goals that achieve costs within 8-9% of the minimum cost in a test suite of reaching tasks with different starting points, target object poses and clutter configurations. We also study the generalization capabilities of the method, by evaluating its robustness to differences between training and test distributions; and, we show examples where relying on a library of previously executed trajectories is not suitable. We see this work as the foundation for a learning framework where reinforcement from past experience can be harnessed to guide trajectory optimizers in making the right decisions.

2 Framework

2.1 Trajectory Optimization

Although our work can use any trajectory optimizer that produces consistent solutions, we will revise here the details of a particular optimizer that has proven to be very efficient in a wide range of manipulation tasks. In recent work [16], we introduced Goal Set CHOMP, an optimizer that can bend trajectories out of collision while remaining smooth, and that exploits the entire goal set of configurations allowed by the task in order to find better solutions. This algorithm was an improve-

ment on the CHOMP optimizer [15], widening the basins of attraction of low-cost solutions by allowing trajectories to adapt their goals.

CHOMP optimizes a functional that trades off between a smoothness and an obstacle cost:

$$U[\xi] = \lambda f_{prior}[\xi] + f_{obs}[\xi] \quad \text{s.t. } h(\xi) = 0 \tag{1}$$

with the prior measuring a notion of smoothness such as sum squared velocities or accelerations along the trajectory ξ, the obstacle cost pushing all parts of the robot away from collision, and h capturing constraints on the trajectory.

We optimize the first-order Taylor Series expansion of U and h around ξ_t within a trust region shaped by a Riemannian metric A in the space of trajectories Ξ. This could be, for example, the Hessian of the prior cost, which will prefer smooth deformations as opposed to small deformations in the Euclidean norm. The resulting trajectory update rule looks like:

$$\xi_{t+1} = \arg\min_{\xi \in \Xi} U(\xi_t) + g_t^T(\xi - \xi_t) + \frac{\eta_t}{2}||\xi - \xi_t||_A^2 \quad \text{s.t. } h(\xi_t) + h'(\xi_t)(\xi - \xi_t) = 0 \tag{2}$$

A convenient representation of trajectories for CHOMP is as a vector of waypoints: $\xi = (\xi[1], \ldots, \xi[n])$. In this case, a typical constraint for CHOMP is a fixed goal: $\xi[n] = q_{\text{goal}}$. Goal Set CHOMP relaxes this assumption, and replaces the constraint by $h_n(\xi[n]) = 0$: the goal is restricted to a constraint surface instead of fixed.

2.2 Trajectory Attribute Prediction

The term "trajectory prediction" refers to the problem of mapping situations S (task descriptions) to a set of trajectories Ξ that solve them:

$$\tau : S \rightarrow \Xi \tag{3}$$

Previous work [1, 3] proposed solving this problem by learning to index into a dataset of examples. This approach is limited by the dataset of previously executed trajectories, much like, for example, the earlier work in object recognition was limited by labeled images it used. In our work, we combine the idea of using a lower dimensional representation of trajectories rather than the full dimensional representation with the ability to predict new trajectories that generalize to more different situations.

Our approach to solving the problem takes advantage of the capabilities of the optimizer. Since this optimizer is local, it will not produce the globally optimal trajectory independent of initialization, but it can produce various local minima with different costs. The training data set therefore contains not only the best trajectory found for the scene, but it can also include various other local optima. We also emphasize that trajectory prediction serves as an initialization stage for the optimizer,

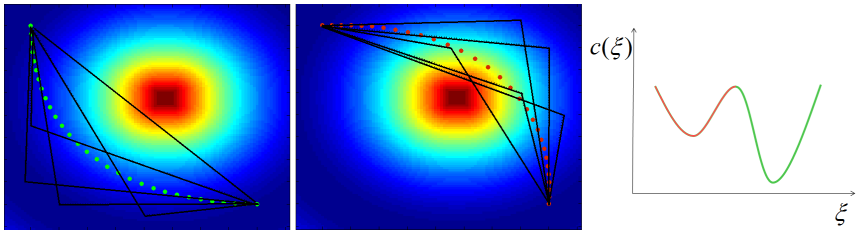

Fig. 1 A toy example that exemplifies the idea of attributes: there are two basins of attraction, and a simple attribute (the decision of going right vs. left) discriminates between them.

which leads to the following crucial observation: *In order to predict the optimal trajectory, we can predict any trajectory in its basin of attraction, and let the optimizer converge.*

So can we leverage this observation in such a way that we have the ability to predict new trajectories (rather than only the ones in the dataset) while avoiding the high dimensionality of the output space Ξ? We propose that there often exist some lower-dimensional trajectory attributes such that predicting these attribute values, rather than a full-dimensional trajectory, places the optimizer in the desired basin of attraction. The insight is that in producing a trajectory, a planner is faced with a few key decisions that define the topology of the trajectory. Once the right decisions are made, producing a good trajectory comes down to local optimization from any initialization that satisfies those decisions. This implies that we can reduce the problem of predicting a good trajectory to that of predicting these core attributes, and then mapping these core attributes to a trajectory. We will discuss each of these two subproblems in turn.

2.3 Attributes

To explain the idea of attribute prediction, we start with the toy world from Figure 1: a point robot needs to get from a start to a goal while minimizing the cost in (1). If we run CHOMP in this world, we get two solutions depending on the initial trajectory: a low and a high cost one. In order to converge to the low-cost trajectory, we can start with any trajectory to the right of the obstacle. Predicting the optimal trajectory reduces to predicting a single bit of information: right vs. left of the obstacle.

In higher dimensional problems, there are many basins of attractions and instead of globally optimal trajectories we can talk about good local minima vs. high-cost and sometimes infeasible local minima. In this setting, it is often the case that the lower-cost basins are still described by simple decisions (i.e. low-dimensional, even discrete, trajectory attributes). Figure 2 shows an example where going above an obstacle vs. around it will determine whether the optimizer converges to a low cost trajectory vs. a high cost one. In this case, a single bit of information will place the optimizer in a good basin of attraction. An optimizer like CHOMP can be initialized

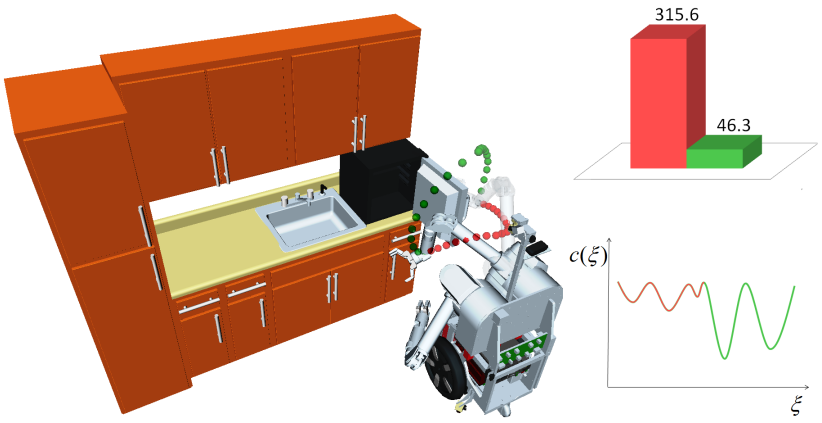

Fig. 2 High-dimensional problems are described by many basins of attraction, but there are often attributes of the trajectory that can discriminate between low cost basins and high cost basins. In this case, such an attribute is around vs. above the fridge door.

with a simple trajectory that satisfies this property, such as the one in Figure 3, and, as exemplified in the same figure, will bend it out of collision to a low-cost trajectory.

Based on this observation, we propose changing the trajectory prediction paradigm to a trajectory attributes prediction problem where we first predict key attributes that a good trajectory should have:

$$\tau : S \rightarrow A(\Xi, S) \tag{4}$$

Here, $A(\Xi, S)$ denotes the trajectory attributes, which are conditioned on the situation, e.g. "in front of the shelf" or "elbow up around the cabinet". These attributes implicitly define a subset of trajectories $\Xi_A \subseteq \Xi$, and as a second step the optimizer is initialized from any trajectory $\xi \in \Xi_A$. The overall framework is

$$S \rightarrow A(\Xi, S) \rightarrow \xi \in \Xi_A \rightarrow \xi^*$$

with ξ^* the locally optimal trajectory in the basin of attraction of ξ.

Constructing a trajectory from a set of attributes ($A(\Xi, S) \rightarrow \xi \in \Xi_A$) can be seen as solving a simple constrained optimization problem: starting from a straight line trajectory, we want to keep it short while satisfying certain constraints on a few of its way-points. Since this problem is convex, generating a trajectory from attributes is very fast. As an example of such a problem, "above X and then to the left of Y" translates into two constraints on two way-points of a piecewise linear trajectory. The example from Figure 3 is an instantiation of that, with one constraint on one mid-point, above the fridge door, which generates two straight line segments in configuration space. Similarly, a goal attribute will be a constraint on the final end-point of the trajectory.

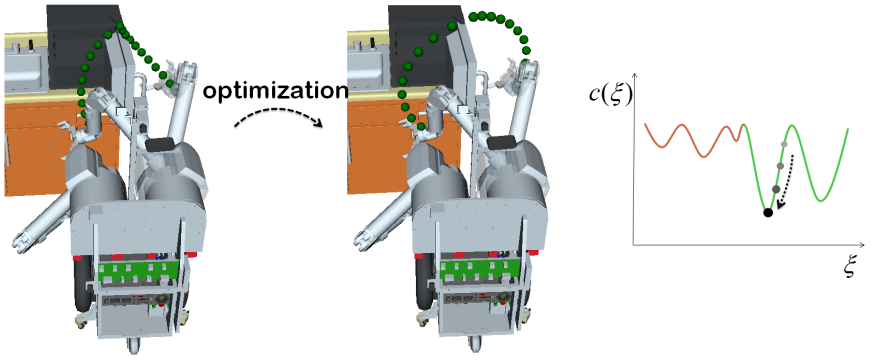

Fig. 3 Once the right choice is made (above the fridge door), we can easily create a trajectory that satisfies it. This trajectory can have high cost, but it will be in the basin of attraction of a low-cost solution, and running a local optimizer (e.g. CHOMP) from it produces a successful trajectory.

3 Learning to Select Good Goals

Most manipulation tasks are described by an entire region of goals rather than one particular configuration that the robot needs to be in such that it can achieve the task. Goal sets appear in reaching for an object, placing it on a surface, or handing it off to a person. In our previous work, we introduced Goal Set CHOMP, a trajectory optimizer that can take advantage of the goal set in order to obtain lower-cost solutions. However, this optimizer is still local, and the initial goal choice (the goal the initial trajectory ends at) still has a high impact on the final cost of the trajectories. Figure 4 plots this final cost for a variety of initial choices, in the problem of reaching for a target object in a small amount of clutter. Because of the large difference illustrated in the figure, the choice of a goal is a crucial component in the optimizer's initialization process. Once a choice is made, a trajectory $\xi \in \Xi_A$ that satisfies this attribute value can be, for example, the straight line trajectory from the start to that goal. For any given situation, we can run an optimizer like Goal Set CHOMP with a straight line trajectory to each of the goals in a discretization of the goal set. In this section, we describe several different ways of taking advantage of this data in order to learn to predict what goal to initialize the optimizer with in order to minimize cost.

3.1 Some Words on Features

To enable learning, we designed features that capture potential factors in deciding how good a goal is. These are indicators of how much free space there is around the goal and how hard it is to reach it. A subset of these features are depicted in Figure 5. We constructed these indicators with simplicity in mind, as a test of what can be done with very little input. We do however believe that much higher performance

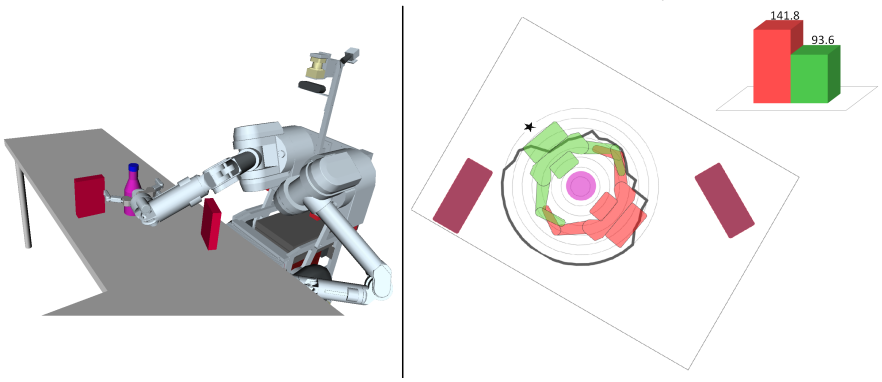

Fig. 4 Left: the robot in one of the goal configurations for grasping the bottle. Right: for the same scene, the black contour is a polar coordinate plot of the final cost of the trajectory Goal Set CHOMP converges to as a function of the goal it starts at; goals that make it hard to reach the object are associated with higher cost; the bar graph shows the difference in cost between the best goal (shown in green and marked with *) and the worst goal (shown in red).

is achievable with a larger set of features, followed perhaps by a feature selection approach. We are also excited about the possibility of producing such features from a much rawer set using feature learning, although important questions, such as informing the algorithm about the kinematics of the robot, are still to be teased out.

3.1.1 A Minimal Set of Features

- The distance in configuration space from the starting point to the goal: $||\xi[n] - \xi[0]||$. Shorter trajectories tend to have lower costs, so minimizing this distance can be relevant to the prediction.
- The obstacle cost of the goal configuration: the sum of obstacle costs for all body points on the robot, $\sum_b c(x_b(\xi_n))$, with c an obstacle cost in the work-space and x_b the forward kinematics function at body point b.
- The obstacle cost of the straight-line trajectory from the start to the goal $\bar{\xi}$: $\sum_i \sum_b c(x_b(\bar{\xi}[i]))$. If the straight line trajectory goes through the middle of obstacles, it can potentially be harder to reach a collision-free solution.
- The goal radius: a measure of the free space around the goal in terms of how many goals around it have collision-free inverse kinematics solutions. For example, the goal set of grasping a bottle can be expressed as a Workspace Goal Region [20] with a main direction of freedom in the yaw of the end effector (this allows grasping the bottle from any angle, as in Figure 4). In this case, the feature would compute how many goals to the left and to the right of the current one have collision-free inverse kinematics solutions, and select the minimum of those numbers as the goal radius. The closer the clutter will be to the goal, the smaller

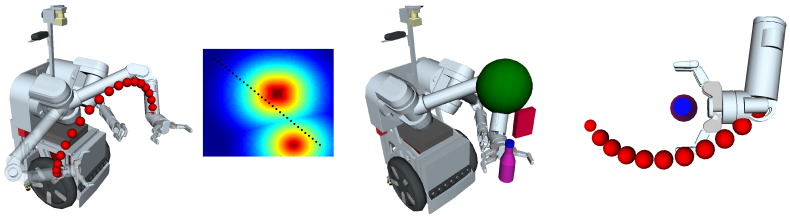

Fig. 5 Five of the features we are using. From left to right: length of the straight line trajectory, cost of the goal and of the straight line trajectory, free space radius around the elbow and collision with the target object.

this radius will be. It has the ability to capture the clutter at larger distances than the second feature can.

- The elbow room: the maximum radius of a collision-free sphere located at the elbow, indicating how much free space the elbow has around it for that particular goal configuration. Configurations that restrict the motion of the elbow are potentially harder to reach.
- The target collision amount: the percent of the last m configurations of the initial trajectory that are colliding with the target object $\frac{1}{m}\sum_{i=N-m+1}^{N} collides(\xi[i]^{sg})$. Here, $collides(q) = 1$ when q is in collision with the target object and 0 otherwise. This feature is another factor in how easy it is to reach the goal — if the initial trajectory passes through the target object, bending it out of collision could be too difficult.

3.1.2 Domain Adaptation

Among the features, the distance from the start as well as the initial trajectory cost can differ substantially between different scenes, and so may cause difficulty for generalization. A classical approach to deal with this problem is standardization, which we can not do directly because of the large difference between our training and test set statistics. The test set contains some scenes that are considerably harder, and some that are far easier than any in the training set: training data will never capture the entire diversity of the situations the robot will face. We still need to generalize to these situations, so we normalize the distance and cost features in a situation — this makes all situations have the same range of costs, allowing the learner to distinguish among them. We then add in the mean values of these two features, to give the learner access to how difficult the scene is, and only then standardize. More sophisticated domain adaptation strategies (e.g., [21]) are an area of future work.

3.2 Learners

We are comparing several learners, differing in the model they use (linear vs. non-linear), how they use the data and whether they focus on predicting the best cost or on fitting the cost in general.

3.2.1 Classification

a) The Vanilla Version: The easiest way to approach the problem of deciding which goal is optimal is to directly predict if a goal will be optimal or not. For every situation, we assign the goal corresponding to the minimum final cost the value 1, and 0 to all the other goals.

We can now train a standard classifier, such as a Support Vector Machine, to predict optimality of a goal. In a new scene, given a set of goal configurations, this classifier will select any number of goals to be optimal, and we will select a random one of these as the initial guess for the optimizer. If the classifier predicts that none of these goals are optimal, then we select randomly among all goals, i.e. the classifier has not given the optimizer any information.

b) The Data-Efficient Version: Since we have access to costs and not just to the binary decision of "is optimal", another approach is to allow the classifier to predict any goal within a certain percent of the minimum cost. This can help by softening the data for the classifier, but there is of course a trade-off with predicting higher cost goals. We determined the value for this trade-off (the percent cutoff) on a validation set.

3.2.2 Inverse Optimal Control

a) The Vanilla Version: A different way to look at the problem is to treat the best goals as expert demonstrations. In Inverse Optimal Control, we want to create a cost function that explains why the experts are optimal – in our case, we want a cost function c_{IOC} in feature space such that the best goal does have the best cost in every situation. Once we have this function, we can apply it to the goals in a new scene and choose the goal $g^* = \arg\min_g c_{IOC}(f_g)$ (here f_g denotes the features associated with goal g).

Taking the Maximum Margin Planning approach introduced in [19], we want to find a cost function $c_{IOC} = w^T f$ that makes the optimal goal have the lowest cost by some margin. To improve generalization, we will require a larger margin for goals that are farther away from the expert: in particular, we define $l(g, g')$ to be the *structured margin*, which is zero when $g = g'$ and large when g and g' are far apart. Then saying that some goal g is optimal means $w^T f_g \leq w^T f_{g'} \quad \forall g'$. Adding in our structured margin, penalizing constraint violations with a slack variable, and regularizing w, we have:

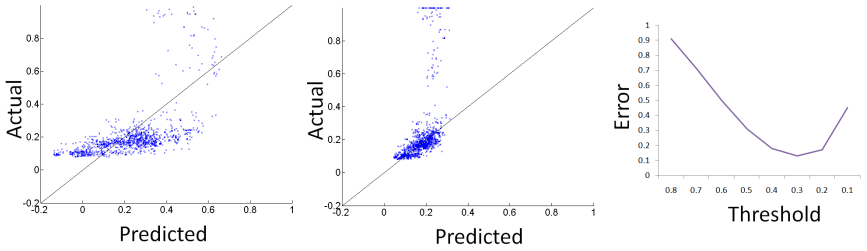

Fig. 6 From left to right: the actual vs. predicted cost without thresholding, the actual vs. predicted cost with thresholding, and the dependence of the fit error of a validation set of medium and low cost examples on the threshold (on the left of the minimum, the regressors pays too much attention to high costs, on the right it uses too little data.

$$\min_w \sum_s \left(w^T f_{g^s_{exp}} - \min_i (w^T f_{g^s_i} - l(g^s_i, g^s_{exp})) \right) + \frac{\lambda}{2} ||w||^2 \tag{5}$$

where g^s_i denotes goal i in situation s, and $l(g,g') = ||f_g - f'_g||^2$ is the structured margin which penalizes solutions from being far away in *feature space* from the expert in situation s. Overall, w pays a penalty for allowing non-expert goals to have low costs.

Taking the subgradient of (5) yields the following update rule:

$$w \leftarrow w - \alpha \left(\sum_s \left(f_{g^s_{exp}} - f_{g^{s*}} \right) + \lambda w \right) \tag{6}$$

$$\text{where} \quad g^{s*} = \arg\min_{g_i} (w^T f_{g^s_i} - l(g^s_i, g^s_{exp})) \tag{7}$$

This algorithm is targeted at identifying the minimum cost goal (7), ignoring the costs associated with all other goals. It gains efficiency as it does not waste resources trying to explain what happens with other goals. Whether this focus on the expert pays off or not will be established in Section 4.

b) The Data-Efficient Version: With IOC, there exists a way of introducing the true cost information (which we do have, unlike typical IOC problems which are only given expert examples), without losing the focus on the expert. By changing the margin l_s to be the true cost difference between the goal and the expert goal rather than the distance in features, $l(g^s_i, g^s_{exp}) = U(\xi^{final}_{g_{exp}}) - U(\xi^{final}_{g_i})$, the algorithm will ensure that the minimum with respect to its new cost is close in true cost to the expert, i.e. has low cost. In future work, we are interested in combining these two distance metrics and using a cutoff on cost difference as in the next section, 3.2.3.

3.2.3 Regression

a) The Vanilla Version A third way to predict the minimum cost goals is to predict the final cost associated to each of the goals:

$$f^s_{g_i} \rightarrow U(\xi^{final}_{g_i})$$

with $\xi^{final}_{g_i}$ the final trajectory obtained by initializing the optimizer with the straight line to goal g_i, and choose the best one:

$$g^* = \arg\min_{g_i} U(\xi^{final}_{g_i})$$

This is sometimes referred to as arg min-regression. We looked at three different regressors:

- Linear Regression: $w = F^{\dagger}C$, with F a matrix concatenating every feature vector on every situation, one per row, and C a vector concatenating all the final costs obtained by Goal Set CHOMP, one per row.
- Gaussian Process: A wide Gaussian radial basis kernel performed the best, since we need far knowledge transfers.
- Neural Network: We used the Back-Propagation Neural Network with one hidden layer. We determined the number of nodes in this layer, as well as the weight decay coefficient based on performance on a validation set.

b) The Data-Efficient Version: Looking at the initial performance of Linear Regression on the training set (Figure 6, left), it becomes apparent that there are a lot of data points with very high cost, and predicting that cost accurately is not only unnecessary, but leads to not being able to identify the good solutions from the mediocre ones. This suggests that even these regressors should not use all the data, but rather focus their efforts on discriminating among the lower-cost solutions by truncating the cost at some threshold. We selected this threshold based on a validation set as shown in Figure 6 (right). The plot shows that a very low threshold degrades performance by confusing the learner to pay attention to the high-cost outliers, and a very high threshold also degrades performance by starving the learner of data. Figure 6 (center) portrays the new predictions based on the learned threshold, forming a much better fit for the solutions we are interested in, while keeping the high-cost predictions sufficiently high. We also tried to do the thresholding per scene instead of on the entire training data, but this did not cause a significant improvement, because the effect on how well the regressors can fit the data is minimal.

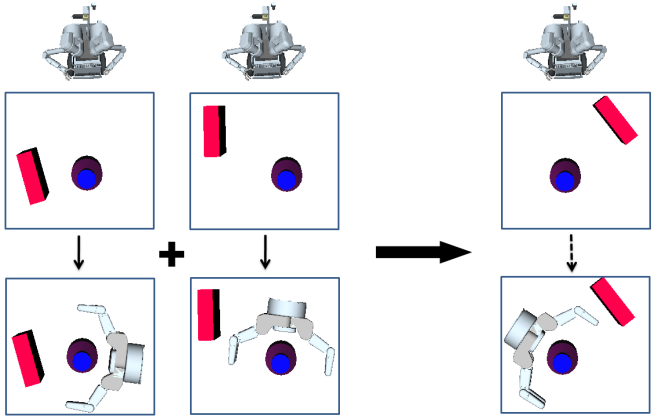

Fig. 7 Two training situations along with their corresponding best goal, and a test situation in which the correct goal is predicted. If the learner were constrained to the set of previously executed trajectories, it would not have been able to generalize to this new scene.

4 Experimental Results

4.1 Generalization From Limited Data

In a first experiment, we wanted to test how well we can generalize to new situations, going beyond the exemplars already executed. We used only two scenes for training, shown in Figure 7, where the goal was the grasp the bottle while avoiding the table holding the object, as well as the box placed next to the target. We ran CHOMP to each goal in a discretization of the goal set, and recorded the final cost. Figure 7 shows the goals that produced the best cost for each of the scenes. We then trained a neural network to predict this cost given only the first three features from Section 3.1.1.

For testing, we moved the object to a very different location than in the training examples, also shown in Figure 7. With a Nearest-Neighbor approach, the robot would identify one of the training scenes as closest, and initialize the optimizer from the best final trajectory for that scene. In this case, all the trajectories go to a goal that is sub-optimal or even colliding with the environment. The trajectory attributes approach, however, allows us to go beyond these previously executed trajectories. The learner predicts that goal shown on the right of Figure 7 will produce the best cost. This goal has never been optimal in the training examples, yet because it stays away from clutter while maintaining a short distance from the starting configuration, the learner will recognize it as better than the other choices. Indeed, when initializing the optimizer from the straight line trajectory to that goal, the final cost is only 1% higher than the best path we were able to find using multiple initializations of Goal Set CHOMP to the different goals.

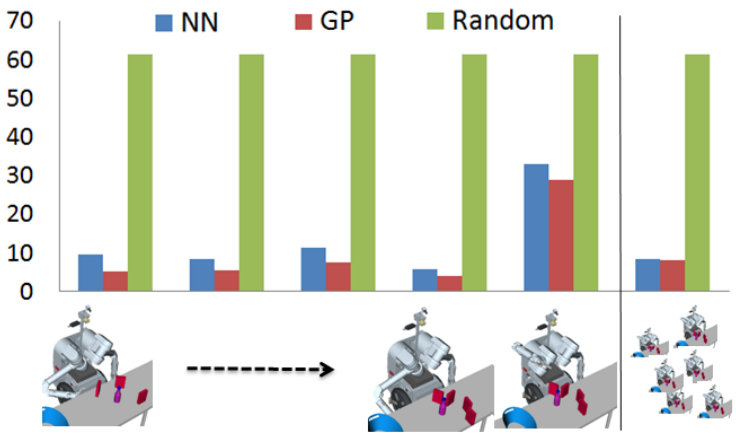

Fig. 8 The loss over the minimum cost on the same test set when training on scenes that are more and more different, until everything changes drastically in the scene and performance drops significantly. However, the loss decreases back to around 8% when training on a wide range of significantly different scenes, showing that the algorithm can do far transfers if given enough variety in the training data.

4.2 Generalization Dependence on Train-Test Similarity

In this next experiment, we were interested in testing how far away from the training data we can transfer knowledge to. We created one testing situation, and trained two of the regressors (the Neural Network and the Gaussian Process) on situations that are more and more different from the testing one. In Figure 8, we plot the performance in these cases as the percent of degradation of cost over the minimum that Goal Set CHOMP can reach — the final cost corresponding to initializing the optimizer with a straight line trajectory to the best goal. These performances, averaged across 15 different clutter configurations, are compared with our baseline: what happens if we randomly choose a collision-free goal, without any learning?

In the first setting, we train and test on the same dataset. Both the Neural Network and the GP perform drastically better than the no-learning baseline. We then change the situation slightly: first the clutter configurations change, then the target object position changes by approx. 20cm, followed by the starting configuration of the robot. In the last but one test, we change all these situation descriptors drastically, and the performance decreases significantly, although the learning algorithms still outperform the baseline. Finally, we show that more variety in the training set can lead to better generalization. When we increase the number of examples in the training set — we still train on very different situations, but we provide a wider range with more possible starting configurations and target poses — we notice that the performance again improves to about 8% for both regressors. The random choice baseline does of course not take into account this data and performs the same, around 62% degradation over the minimum cost.

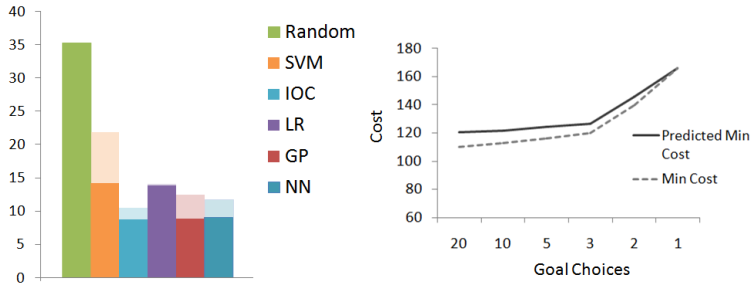

Fig. 9 Left: Percentage loss over the best cost for all the methods. Solid bars are the data-efficient versions, and transparent bars are the vanilla algorithms, which perform worse. Right: The predicted minimum cost vs. the true minimum cost as function of the number of choices considered.

4.3 Main Experiment

We are also interested in a realistic evaluation of the day-to-day performance of our system, as well as establishing which learning approach from Section 3.2 is most suitable for our problem. Should the learner focus on just the optimal goal or or should it also focus on the sub-optimal goals and their performance?

We created a large set of training examples, comprising of 90 situations varying in the starting configuration, target object pose, and clutter distribution. In each situation, we ran Goal Set CHOMP starting from the straight line trajectory to each of the collision-free goals in the discretized goal set (a total of 1154 examples) and recorded the final cost. We also created a test set of 108 situations (1377 examples) that differ in all three components from the training data.

Figure 9(Left) shows the percentage of cost degradation over the minimum, averaged across all testing situations, for the five approaches from Section 3.2. The solid bars are the data-efficient versions of the algorithms: the regressors use thresholds established on a separate validation set, IOC uses the cost distance for the structured margin, and the classifier predicts goals close to the minimum as well. The vanilla versions of these methods, shown with transparent bars, always perform worse than their data-efficient counterparts.

The best performer is our version of data-efficient IOC — this algorithm focuses on predicting the expert rather than fitting cost, while taking into account the true cost and ensuring that non-expert predictions have low cost. Although both IOC and LR are linear, the advantage of IOC over LR is its expert prediction focus. The non-linear regressors have similar performance as IOC, and their advantage is a better fit of that data. The SVM is focusing on low-costs with a linear kernel, so its performance is, as expected, close to LR.

In these experiments, we had a fairly fine discretization of the goal set per scene. It makes sense to ask if we could get away with fewer choices. Figure 9(Right) indicates that the answer is yes: with 5 goals, for example, we can predict the minimum cost better, and we this minimum is not a lot larger than the one considering, say, 20 goals.

5 Conclusion

In this paper, we proposed moving away from the learning from experience paradigm of predicting trajectories from a library. We proposed instead to predict the important attributes of trajectories that will place the initial guess for a trajectory optimizer in a good basin of attraction. We presented a first step towards this trajectory prediction paradigm by focusing on a very important attribute of the trajectory: the choice of a goal. We showed that the learner can can generalize well by predicting this attribute, and presented results emphasizing the importance of learning from experience in practice using our attribute prediction framework. The next step in our work is to identify the set of attributes that are required in order to differentiate between basins of attraction, based on both the optimizer and the current situation the robot is in. We see this as a very exciting challenge combining the machine learning and manipulation worlds that will pave the road towards a more semantic way of planning, focused on a hierarchy influenced by previous experiences of the robot.

References

1. Jetchev, N., and Toussaint, M. (2009). Trajectory prediction. Proceedings of the 26th Annual International Conference on Machine Learning - ICML 09, 1-8. New York, New York, USA: ACM Press
2. Dey, D., Liu, T., Sofman, B., and Bagnell, J. (2011). Efficient Optimization of Control Libraries. Technical Report (CMU-RI-TR-11-20)
3. Jetchev, N., and Toussaint, M. (2010). Trajectory prediction in Cluttered Voxel Environments. IEEE International Conference on Robotics and Automation - ICRA2010, Anchorage, AK, USA
4. Lampert, C., Nickisch, and H., Harmeling, S. (2009). Learning To Detect Unseen Object Classes by Between-Class Attribute Transfer. IEEE Conference on Computer Vision and Pattern Recognition (2009), 951-958
5. Farhadi, A., Endres, I., Hoiem, D., and Forsyth, D. (2009). Describing objects by their attributes. IEEE Conference on Computer Vision and Pattern Recognition (2009), 1778-1785.
6. Palatucci, M., Hinton, G., Pomerleau, D., and Mitchell, T. M. (2009). Zero-Shot Learning with Semantic Output Codes. Advances in Neural Information Processing Systems, 22, 1-9.
7. Veloso, M. M. (1992). PhD. Learning by Analogical Reasoning in General Problem Solving. (CMU-CS-92-174)
8. Lopez De Mantaras, R., Mcsherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., and Faltings, B. (2006). Retrieval, reuse, revision and retention in case-based reasoning. The Knowledge Eng. Review, 20(03), 215.
9. Konidaris, G., and Barto, A. (2006). Autonomous shaping: Knowledge transfer in reinforcement learning. Proceedings of the 23rd international conference on Machine learning (pp. 489-496)
10. Stolle, M., and Atkeson, C. G. (2006). Policies based on trajectory libraries. IEEE ICRA, (May), 3344-3349
11. Stolle, M., and Atkeson, C. G. (2007). Knowledge Transfer Using Local Features. IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007.
12. Stolle, M., Tappeiner, H., Chestnutt, J., and Atkeson, C. G. (2007). Transfer of policies based on trajectory libraries. IEEE/RSJ International Conference on Intelligent Robots and Systems. 2981-2986
13. Branicky, M., Knepper, R., and Kuffner, J. (2008). Path and trajectory diversity: Theory and algorithms. IEEE Int. Conf. on Robotics and Automation (ICRA), 1359 - 1364
14. Martin, S., Wright, S., and Sheppard, J. (2007). Offline and online evolutionary bi-directional RRT algorithms for efficient re-planning in dynamic environments. IEEE CASE. 1131-1136
15. Ratliff, N., Zucker, M., Bagnell, J. A., and Srinivasa, S. (2009). CHOMP: Gradient optimization techniques for efficient motion planning. IEEE ICRA. 489-494
16. Dragan, A. D., Ratliff, N., and Srinivasa, S. S. (2011). Manipulation Planning with Goal Sets Using Constrained Trajectory Optimization. IEEE International Conference on Robotics and Automation (2011)
17. Lowe, D. G. (1999). Object recognition from local scale-invariant features. Proceedings of the Seventh IEEE International Conference on Computer Vision, 2([8), 1150-1157 vol.2
18. Frome, A., Singer, Y., and Malik, J. (2006). Image Retrieval and Classification Using Local Distance Functions. In Advances in Neural Information Processing Systems (NIPS), 2006.
19. Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. (2006). Maximum margin planning. Proceedings of the 23rd international conference on Machine learning ICML 06, 729-736.
20. Berenson, D., Srinivasa, S. S., Ferguson, D., Collet, A., and Kuffner, J. J. (2009). Manipulation planning with Workspace Goal Regions. Proceedings of the IEEE ICRA. 618-624
21. Blitzer, J., Daume, H., ICML Tutorial on Domain Adaptation, http://adaptationtutorial.blitzer.com/