Predicting Contextual Sequences via Submodular Function Maximization

Debadeepta Dey, Tian Yu Liu, Martial Hebert, Drew Bagnell

CMU-RI-TR-12-05

February 2012

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

© Carnegie Mellon University

***Abstract*—*Sequence optimization*, where the items in a list are ordered to maximize some reward has many applications such as web advertisement placement, search, and control libraries in robotics. Previous work in sequence optimization produces a static ordering that does not take any features of the item or context of the problem into account. In this work, we propose a general approach to order the items within the sequence based on the *context* (*e.g.*, perceptual information, environment description, and goals). We take a simple, efficient, reduction-based approach where the choice and order of the items is established by repeatedly learning simple classifiers or regressors for each "slot" in the sequence. Our approach leverages recent work on submodular function maximization to provide a formal regret reduction from submodular sequence optimization to simple cost-sensitive prediction. We apply our contextual sequence prediction algorithm to optimize control libraries and demonstrate results on two robotics problems: manipulator trajectory prediction and mobile robot path planning.**

I. Introduction

Optimizing the order of a set of choices is fundamental to many problems such as web search, advertisement placements as well as in robotics and control. *Relevance* and *diversity* are important properties of an optimal ordering or sequence. In web search, for instance, if the search term admits many different interpretations then the results should be interleaved with items from each interpretation [16]. Similarly in advertisement placement on web pages, advertisements should be chosen such that within the limited screen real estate they are diverse yet relevant to the page content. In robotics, *control libraries* have the same requirements for relevance and diversity in the ordering of member actions. In this paper, we apply *sequence optimization* to develop near-optimal control libraries. In the context of control libraries, a sequence refers to a ranked list of control action choices rather than a series of actions to be taken. Examples of control actions include grasps for manipulation, trajectories for mobile robot navigation or seed trajectories for initializing a local trajectory optimizer.

Control libraries are a collection of control actions obtained by sampling a useful set of often high dimensional control trajectories or policies. Examples of control libraries include a collection of feasible grasps for manipulation [5], a collection of feasible trajectories for mobile robot navigation [9], and a collection of expert-demonstrated trajectories for a walking robot (Stolle et. al. [20]). Similarly, recording demonstrated trajectories of experts aggressively flying unmanned aerial vehicles (UAVs) has enabled dynamically feasible trajectories to be quickly generated by concatenating a suitable subset of stored trajectories in the control library [8].

Such libraries are an effective means of spanning the space of all feasible control actions while at the same time dealing with computational constraints. The performance of control libraries on the specified task can be significantly improved by careful consideration of the *content* and *order* of actions in the library. To make this clear let us consider specific examples:

Mobile robot navigation. In mobile robot navigation the task is to find a collision-free, low cost of traversal path which leads to the specified goal on a map. Since sensor horizons are finite and robots usually have constrained motion models and non-trivial dynamics, a library of trajectories respecting the dynamic and kinematic constraints of the robot are precomputed and stored in memory. This constitutes the control library. It is desired to sample a subset of trajectories at every time step so that the overall cost of traversal of the robot from start to goal is minimized.

Trajectory optimization. Local trajectory optimization techniques are sensitive to initial trajectory seeds. Bad trajectory initializations may lead to slow optimization, suboptimal performance, or even remain in collision. In this setting, the control actions are end-to-end trajectory seeds that act as input to the optimization. Zucker [27] and Jetchev et al. [11] proposed methods for predicting trajectories from a precomputed library using features of the environment, yet these methods do not provide recovery methods when the prediction fails. Having a *sequence* of initial trajectory seeds provides fallbacks should earlier ones fail.

Grasp libraries. During selection of grasps for an object, a library of feasible grasps can be evaluated one at a time until a collision-free, reachable grasp is found. While a naive ordering of grasps can be based on force closure and stability criteria [2], if a grasp fails, then grasps similar to it are also likely to fail. A more principled ordering approach which takes into account features of the environment can reduce depth of the sequence that needs to be searched by having diversity in higher ranked grasps.

Current state-of-the-art methods in the problems we address either predict only a single control action in the library that has the highest score for the current environment, or use an ad-hoc ordering of actions such as random order or by past rate of success. If the predicted action fails then systems (e.g. manipulators and autonomous vehicles) are unable to recover or have to fall back on some heuristic/hard-coded contingency plan. Predicting a *sequence* of options to evaluate is necessary for having intelligent, robust behavior. Choosing the order of evaluation of the actions based on the context of the environment leads to more efficient performance.

A naive way of predicting contextual sequences would be to train a multi-class classifier over the label space consisting of all possible sequences of a certain length. This space is exponential in the number of classes and sequence length posing information theoretic difficulties. A more reasonable method would be to use the greedy selection technique by Steeter et al. [21] over the hypothesis space of all predictors which is guaranteed to yield sequences within a constant factor of the optimal sequence. Implemented naively, this remains expensive as it must explicitly enumerate the label space. Our simple reduction based approach where we propose to train multiple multi-class classifiers/regressors to mimic greedy selection given features of the environment is both efficient and maintains performance guarantees of the greedy selection.

Perception modules using sensors such as cameras and lidars are part and parcel of modern robotic systems. Leveraging such information in addition to the feedback of success or failure is *conceptually* straightforward: instead of considering

a sequence of control actions, we consider a sequence of classifiers which map features X to control actions A, and attempt to find the best such classifier at each slot in the control action sequence. By using contextual features, our method has the benefit of closing the loop with perception while maintaining the performance guarantees in Streeter et al.[21].

The outlined examples present loss functions that depend only on the "best" action in the sequence, or attempt to minimize the prediction depth to find a satisfactory action. Such loss functions are monotone, submodular – *i.e.*, one with diminishing returns.[1] We define these functions in section II and review the online submodular function maximization approach of Streeter et al. [21]. We also describe our contextual sequence optimization (CONSEQOPT) algorithm in detail. Section III shows our algorithm's performance improvement over alternatives for local trajectory optimization for manipulation and in path planning for mobile robots.

Our contributions in this work are:

- We propose a simple, near-optimal reduction for contextual sequence optimization. Our approach moves from predicting a single decision based on features to making a *sequence* of predictions, a problem that arises in many domains including advertisement prediction [22, 16] and search.
- The application of this technique to the contextual optimization of control libraries. We demonstrate the efficacy of the approach on two important problems: robot manipulation planning and mobile robot navigation. Using the sequence of actions generated by our approach we observe improvement in performance over sequences generated by either random ordering or decreasing rate of success of the actions.
- Our algorithm is generic and can be naturally applied to any problem where ordered sequences (e.g., advertisement placement, search, recommendation systems, etc) need to be predicted and relevance and diversity are important.

II. CONTEXTUAL OPTIMIZATION OF SEQUENCES

A. Background

The control library is a set $\mathscr{V}$ of actions. Each action is denoted by $a \in \mathscr{V}$. [2] Formally, a function $f : \mathscr{S} \to \mathfrak{R}_+$ is monotone submodular for any sequence $S \in \mathscr{S}$ where $\mathscr{S}$ is the set of all sequences of actions if it satisfies the following two properties:

- (Monoticity) for any sequence $S_1, S_2 \in \mathscr{S}$, $f(S_1) \leq f(S_1 \oplus S_2)$ and $f(S_2) \leq f(S_1 \oplus S_2)$
- (Submodularity) for any sequence $S_1, S_2 \in \mathscr{S}$, $f(S_1)$ and any action $a \in \mathscr{V}$, $f(S_1 \oplus S_2 \oplus \langle a \rangle) - f(S_1 \oplus S_2) \leq f(S_1 \oplus \langle a \rangle) - f(S_1)$

[1]For more information on submodularity and optimization of submodular functions we refer readers to the tutorial [4].

[2]In this work we assume that each action choice takes the same time to execute although the proposed approach can be readily extended to handle different execution times.

where $\oplus$ denotes order dependent concatenation of sequences. These imply that the function always increases as more actions are added to the sequence (monotonicity) but the gain obtained by adding an action to a larger pre-existing sequence is less as compared to addition to a smaller pre-existing sequence (sub-modularity).

For control library optimization, we attempt to optimize one of two possible criteria: the cost of the best action a in a sequence (with a budget on sequence size) or the time (depth in sequence) to find a satisficing action. For the former, we consider the function,

$$f \equiv \frac{N_o - \min(\text{cost}(a_1), \text{cost}(a_2), \ldots, \text{cost}(a_N))}{N_o}, \qquad (1)$$

where cost is an arbitrary cost on an action (a_i) given an environment and N_o is a constant, positive normalizer which is the highest cost. [3] Note that the f takes in as arguments the sequence of actions $a_1, a_2, \ldots, a_N$ directly, but is also implicitly dependent on the current environment on which the actions are evaluated in $cost(a_i)$. Dey et al. [6] prove that this criterion is monotone submodular in sequences of control actions and can be maximized– within a constant factor– by greedy approaches similar to Streeter et al. [21].

For the latter optimization criteria, which arises in grasping and trajectory seed selectin, we define the monotone, submodular loss function $f : \mathscr{S} \to [0,1]$ as $f \equiv P(S)$ where $P(S)$ is the probability of successfully grasping an object in a given scenario using the sequence of grasps provided. It is easy to check [6] that this function is also monotone and submodular, as the probability of success always increases as we consider additional elements. Minimizing the depth in the control library to be evaluated becomes our goal. In the rest of the paper all objective functions are assumed to be monotone submodular unless noted otherwise.

While optimizing these over library actions is effective, the ordering of actions does not take into account the current context. People do not attempt to grasp objects based only on previous performance of grasps: they take into account the position, orientation of the object, the proximity and arrangement of clutter around the object and also their own position relative to the object in the current environment.

B. Our Approach

We consider functions that are submodular over sequences of either control actions themselves or, crucially, over classifiers that take as input environment features $\mathbf{X}$ and map to control actions $\mathscr{V}$. Additionally, by considering many environments, the expectation of f in equation (1) over these environments also maintains these properties. In our work, we always consider the expected loss averaged over a (typically empirical) distribution of environments.

In Algorithm 1, we present a simple approach for learning such a near-optimal contextual control library.

[3]For mobile robot path planning, for instance, $cost(a_i)$ is typically a simple measure of mobility penalty based on terrain for traversing a trajectory a_i sampled from a set of trajectories and terminating in a heuristic cost-to-go estimate, compute by, e.g. A*.

Algorithm 1 `Algorithm for training` ConSeqOpt `using classifiers`

Input: `sequence length N, multi-class cost sensitive classifier routine` π, `dataset` D `of` $|D|$ `number of environments and associated features` $\mathbf{X}$, `library of control actions` $\mathscr{V}$
Output: `sequence of classifiers` $\pi_1, \pi_2, \ldots, \pi_N$

1: **for** $i = 1$ **to** N **do**
2: $\quad \mathbf{M}_{\mathbf{L_i}} \leftarrow \texttt{computeTargetActions}(\mathbf{X}, \mathbf{Y}_{\pi_1, \pi_2, \ldots, \pi_{i-1}}, \mathscr{V})$
3: $\quad \pi_i \leftarrow \texttt{train}(\mathbf{X}, \mathbf{M}_{\mathbf{L_i}})$
4: $\quad \mathbf{Y}_{\pi_i} \leftarrow \texttt{classify}(\mathbf{X})$
5: **end for**

Algorithm 2 `Algorithm for training` ConSeqOpt `using regressors`

Input: `sequence length N, regression routine` $\Re$, `dataset` D `of` $|D|$ `number of environments, library of control actions` $\mathscr{V}$
Output: `sequence of regressors` $\Re_1, \Re_2, \ldots, \Re_N$

1: **for** $i = 1$ **to** N **do**
2: $\quad \mathbf{X_i}, \mathbf{M_{B_i}} \leftarrow \texttt{computeFeatures\&Benefit}(D, \mathbf{Y}_{\Re_1, \Re_2, \ldots, \Re_{i-1}}, \mathscr{V})$
3: $\quad \Re_i \leftarrow \texttt{train}(\mathbf{X_i}, \mathbf{M_{B_i}})$
4: $\quad \tilde{\mathbf{M}}_{\mathbf{B_i}} \leftarrow \texttt{regress}(\mathbf{X_i}, \Re_i)$
5: $\quad \mathbf{Y}_{\Re_i} = \texttt{argmax}(\tilde{\mathbf{M}}_{\mathbf{B_i}})$
6: **end for**

C. Algorithm for Contextual Submodular Sequence Optimization

Figure 1 shows the schematic diagram for algorithm 1 which trains a classifier for each slot of the sequence. Define matrix $\mathbf{X}$ to be the set of features from a distribution of example environments (one feature vector per row) and matrix $\mathbf{Y}$ to be the corresponding target action identifier for each example. Let each feature vector contain L attributes. Let D be the set of example environments containing $|D|$ examples. The size of $\mathbf{X}$ is $|D| \times L$ and size of $\mathbf{Y}$ is $|D| \times 1$. We denote the i^{th} classifier by π_i. Define $\mathbf{M}_{\mathbf{L_i}}$ to be the matrix of marginal losses for each environment for the i^{th} slot of the sequence. In the parlance of cost-sensitive learning $\mathbf{M}_{\mathbf{L_i}}$ is the example-dependent cost matrix. $\mathbf{M}_{\mathbf{L_i}}$ is of dimension $|D| \times |\mathscr{V}|$. Each row of $\mathbf{M}_{\mathbf{L_i}}$ contains, for the corresponding environment, the loss suffered by the classifier for selecting a particular action $a \in \mathscr{V}$. The most beneficial action has 0 loss while others have non-zero losses. These losses are normalized to be within $[0-1]$. We detail how to calculate the entries of $\mathbf{M}_{\mathbf{L_i}}$ below. Classifier inputs are the set of feature vectors $\mathbf{X}$ for the dataset of environments and the marginal loss matrix $\mathbf{M}_{\mathbf{L_i}}$.

For ease of understanding let us walk through the training of the first two classifiers π_1 and π_2.

Consider the first classifier training in Figure 1 and its inputs $\mathbf{X}$ and $\mathbf{M}_{\mathbf{L_1}}$. Consider the first row of $\mathbf{M}_{\mathbf{L_1}}$. Each element of this row corresponds to the loss incurred if the corresponding action in $\mathscr{V}$ were taken on the corresponding environment whose features are in the first row of $\mathbf{X}$. The best action has 0 loss while all the others have relative losses in the range $[0-1]$ depending on how much worse they are compared to the best action. This way the rest of the rows in $\mathbf{M}_{\mathbf{L_1}}$ are filled out. The cost sensitive classifier π_1 is trained. The set of features $\mathbf{X}$ from each environment in the training set are again presented to it to classify. The output is matrix $\mathbf{Y}_{\pi_1}$ which contains the selected action for the 1^{st} slot for each environment. As no element of the hypothesis class performs perfectly this results in $\mathbf{Y}_{\pi_1}$, where not every environment had the 0 loss action picked.

Consider the second classifier training in Figure 1. Consider the first row of $\mathbf{M}_{\mathbf{L_2}}$. Suppose control action id 13 was selected by classifier π_1 in the classification step for the first environment, which provides a gain of 0.6 to the objective function f i.e. $f[13] = 0.6$. For each of the control actions a present in the library $\mathscr{V}$ find the action which provides maximum marginal improvement i.e. $a_{max} = argmax_a(f([13,a]) - f([13])) = argmax_a(f([13,a]) - 0.6$. Additionally convert the marginal gains computed for each a in the library to proportional losses and store in the first row of $\mathbf{M}_{\mathbf{L_2}}$. If a_{max} is the action with the maximum marginal gain then the loss for each of the other actions is $f([13,a_{max}]) - f([13,a])$. a_{max} has 0 loss while other actions have $>= 0$ loss. The rest of the rows are filled up similarly. π_2 is trained, and evaluated on same dataset to produce Y_{π_2}.

This procedure is repeated for all N slots producing a sequence of classifiers $\pi_1, \pi_2, \ldots, \pi_N$. The idea is that a classifier must suffer a high loss when it chooses a control action which provides little marginal gain when a higher gain action was available. Any cost-sensitive multi-class classifier may be used.

During test time, for a given environment features are extracted, and the classifiers associated with each slot of the sequence outputs a control action to fill the slot. This sequence can then be evaluated as usual. This procedure is formalized in Algorithm 1. In `computeTargetActions` the previously

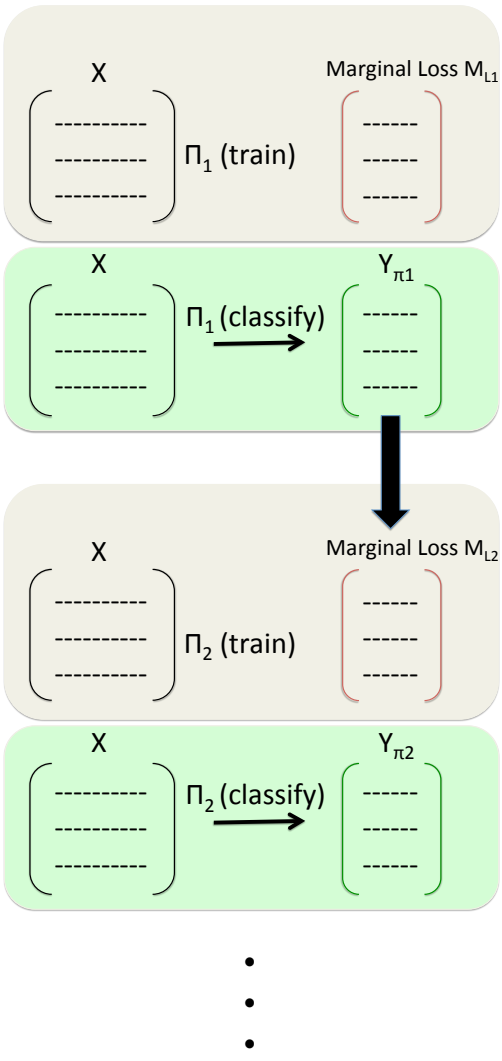


Fig. 1: Schematic of training sequence of classifiers for regret reduction of contextual sequence optimization to multi-class, cost-sensitive, classification

detailed procedure for calculating the entries of the marginal loss matrix M_{L_i} for the i^{th} slot is carried out, followed by the training step in `train` and classification step in `classify`.

Algorithm 2 has a similar structure as algorithm 1. This alternate formulation has the advantage of being able to add actions to the control library without retraining the sequence of classifiers. Instead of directly identifying a target class, we use a squared-loss regressor in each slot to produce an estimate of the marginal benefit from each action at that particular slot. Hence $\mathbf{M_{B_i}}$ is a $|D| \times |\mathscr{Y}|$ matrix of the actual marginal benefit computed in a similiar fashion as $\mathbf{M_{L_i}}$ of Algorithm 1, and $\tilde{\mathbf{M}}_{\mathbf{B_i}}$ is the estimate given by our regressor at i^{th} slot. In line 2 we compute the feature matrix $\mathbf{X_i}$. In this case, a feature vector is computed *per action* per environment, and uses information from the previous slots' target choice $\mathbf{Y}_{\mathfrak{R}_i}$. For feature vectors of length L, $\mathbf{X_i}$ has dimensions $|D||\mathscr{Y}| \times L$. The features and marginal benefits at i^{th} slot are used to train regressor $\mathfrak{R}_\mathbf{i}$, producing the estimate $\tilde{\mathbf{M}}_{\mathbf{B_i}}$. We then pick the action a which produces the maximum $\tilde{\mathbf{M}}_{\mathbf{B_i}}$ to be our target choice $\mathbf{Y}_{\mathfrak{R}_i}$, a $|D|$ length vector of indices into $\mathscr{Y}$ for each environment.

D. Reduction Argument

We establish a formal regret reduction [3] between cost sensitive multi-class classification error and the resulting error on the learned sequence of classifiers. Specifically, we demonstrate that if we consider the control actions to be the classes and train a series of classifiers– one for each slot of the sequence– on the features of a distribution of environments then we can produce a near-optimal sequence of classifiers. This sequence of classifiers can be invoked to approximate the greedy sequence constructed by allowing additive error in equation (3).

Theorem 1. *If each of the classifiers ($\pi_\mathbf{i}$) trained in Algorithm 1 achieves multi-class cost-sensitive regret of r_i, then the resulting sequence of classifiers is within at least $(1-\frac{1}{e})\max_{S\in\mathscr{S}} f(S) - \sum_{i=1}^{N} r_i$ of the optimal such sequence of classifiers S from the same hypothesis space.* [4]

Proof: (Sketch) Define the loss of a multi-class, cost-sensitive classifier π over a distribution of environments D as $l(\pi, D)$. Each example can be represented as $(x_n, m_n^1, m_n^2, m_n^3, \ldots, m_n^{|\mathscr{Y}|})$ where x_n is the set of features representing the n^{th} example environment and $m_n^1, m_n^2, m_n^3, \ldots, m_n^{|\mathscr{Y}|}$ are the per class costs of misclassifying x_n. $m_n^1, m_n^2, m_n^3, \ldots, m_n^{|\mathscr{Y}|}$ are simply the n^{th} row of M_{L_i} (which corresponds to the n^{th} environment in the dataset D). The best class has a 0 misclassification cost and while others are greater than equal to 0 (There might be multiple actions which will yield equal marginal benefit). Classifiers generally minimize the expected loss $l(\pi, D) = \mathop{\mathbb{E}}_{(x_n, m_n^1, m_n^2, m_n^3, \ldots, m_n^{|\mathscr{Y}|})\sim D} [C_{\pi(x_n)}]$ where $C_{\pi(x_n)} = m_n^{\pi(x_n)}$ denotes the example-dependent multi-class misclassification cost. The best classifier in the hypothesis space Π minimizes $l(\pi, D)$

$$\pi^* = \mathop{\text{argmin}}_{\pi\in\Pi} \mathop{\mathbb{E}}_{(x_n, m_n^1, m_n^2, m_n^3, \ldots, m_n^{|\mathscr{Y}|})\sim D} [C_{\pi(x_n)}] \tag{2}$$

The regret of π is defined as $r = l(\pi, D) - l(\pi^*, D)$. Each classifier associated with i^{th} slot of the sequence has a regret r_i.

Streeter et al. [21] consider the case where the i^{th} decision made by the greedy algorithm is performed with additive error ε_i. Denote by $\hat{S} = \langle \hat{s}_1, \hat{s}_2, \ldots, \hat{s}_N \rangle$ a variant of the sequence S in which the i^{th} argmax is evaluated with additive error ε_i. This can be formalized as

$$f(\hat{S}_i \oplus \hat{s}_i) - f(\hat{S}_i) \geq \max_{s_i \in \mathscr{Y}} f(\hat{S}_i \oplus s_i) - f(\hat{S}_i) - \varepsilon_i \tag{3}$$

[4] When the objective is to minimize the time (depth in sequence) to find a satisficing element then the resulting sequence of classifiers $f(\hat{S}_{\langle N \rangle}) \leq 4 \int_0^\infty 1 - \max_{S\in\mathscr{S}} f(S_{\langle n \rangle}) dn + \sum_{i=1}^{N} r_i$.

where $\hat{S}_0 = \langle\rangle$, $\hat{S}_i = \langle \hat{s}_1, \hat{s}_2, \hat{s}_3, \ldots, \hat{s}_{i-1} \rangle$ for $i \geq 1$ and s_i is the predicted control action by classifier π_i. They demonstrate that, for a budget or sequence length of N

$$f(\hat{S}_{\langle N \rangle}) \geq (1 - \frac{1}{e}) \max_{S \in \mathscr{S}} f(S) - \sum_{i=1}^{N} \varepsilon_i \tag{4}$$

assuming each control action takes equal time to execute.

Thus the i^{th} argmax in equation (3) is chosen with some error $\varepsilon_i = r_i$. An ε_i error made by classifier π_i corresponds to the classifier picking an action whose marginal gain is ε_i less than the maximum possible. Hence the performance bound on additive error greedy sequence construction stated in equation (4) can be restated as

$$f(\hat{S}_{\langle N \rangle}) \geq (1 - \frac{1}{e}) \max_{S \in \mathscr{S}} f(S) - \sum_{i=1}^{N} r_i \tag{5}$$

■

Theorem 2. *The sequence of squared-loss regressors ($\Re_i$) trained in Algorithm 2 is within at least $(1 - \frac{1}{e}) \max_{S \in \mathscr{S}} f(S) - \sum_{i=1}^{N} \sqrt{2(|\mathscr{V}| - 1) r_{reg_i}}$ of the optimal sequence of classifiers S from the hypothesis space of multi-class cost-sensitive classifiers.*

Proof: (Sketch) Langford et al. [14] show that the regret reduction from multi-class classification to squared-loss regression has a regret reduction of $\sqrt{2(|k| - 1) r_{reg}}$ where k is the number of classes and r_{reg} is the squared-loss regret on the underlying regression problem. In Algorithm 2 we use squared-loss regression to perform multi-class classification thereby incurring for each slot of the sequence a reduction regret of $\sqrt{2(|\mathscr{V}| - 1) r_{reg_i}}$ where $|\mathscr{V}|$ is the number of actions in the control library. Theorem 1 states that the sequence of classifiers is within at least $f(\hat{S}_{\langle N \rangle}) \geq (1 - \frac{1}{e}) \max_{S \in \mathscr{S}} f(S) - \sum_{i=1}^{N} r_i$ of the optimal sequence of classifiers. Plugging in the regret reduction from [14] we get the result that the resulting sequence of regressors in Algorithm 2 is within at least $(1 - \frac{1}{e}) \max_{S \in \mathscr{S}} f(S) - \sum_{i=1}^{N} \sqrt{2(|\mathscr{V}| - 1) r_{reg_i}}$ of the optimal sequence of multi-class cost-sensitive classifiers.

■

III. Case Studies

A. Robot Manipulation Planning via Contextual Control Libraries

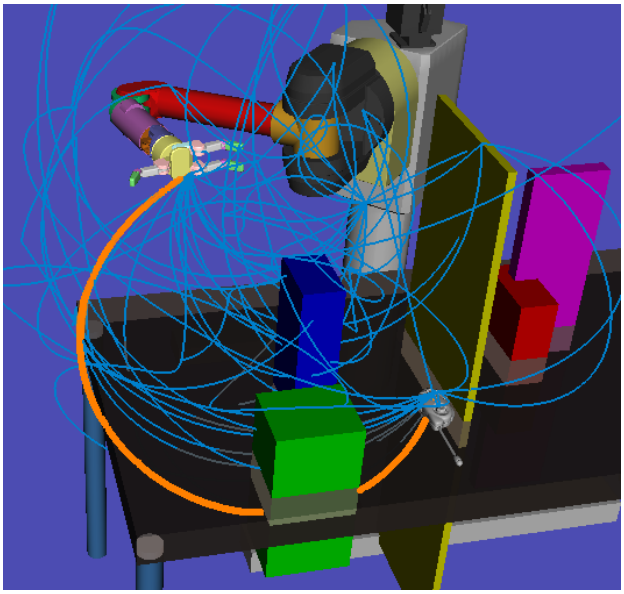
We demonstrate the application of ConSeqOpt for manipulation planning on a 7 degree of freedom manipulator.

Recent work [18, 11] has shown that by relaxing the hard constraint of avoiding obstacles into a soft penalty term on collision, simple local optimization techniques can quickly lead to smooth, collision-free trajectories suitable for robot execution. Often the default initialization trajectory seed is a simple straight-line initialization in joint space [18]. This heuristic is surprisingly effective in many environments, but suffers from local convergence and may fail to find a trajectory when one exists. In practice, this may be tackled by providing cleverer initialization seeds [11, 27]. While these methods reduce the chance of falling into local minima, they do not have any alternative plans should the chosen initialization seed fail. A contextual ranking of a library of initialization trajectory seeds can provide feasible alternative seeds should earlier choices fail. Proposed initialization trajectory seeds can be developed in many ways including human demonstration [17] or use of a slow but complete planner[13].

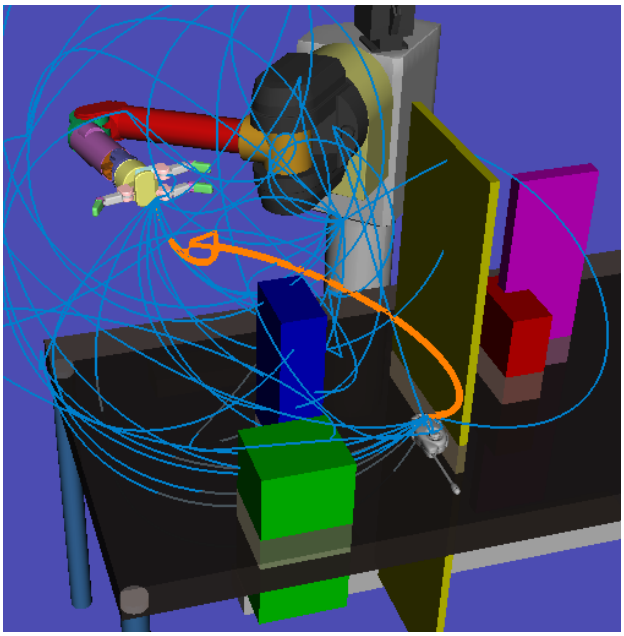
For this experiment we attempt to plan a trajectory to a pre-grasp pose over the target object in a cluttered environment using the local optimization planner CHOMP [18] and minimize the total planning and execution time of the trajectory. A training dataset of $|D| = 310$ environments and test dataset of 212 environments are generated. Positions of the target object and obstacles on the table are randomly assigned. To populate the control library, we consider initialization trajectories that move first to an "exploration point" and then to the goal. The exploration points are generated by randomly perturbing the midpoint of the original straight line initialization in joint space. The resulting initial trajectories are then piecewise straight lines in joint space from the start point to the exploration point, and from the exploration point to the goal. Half of the seed trajectories are prepended with a short path to start from a left-arm configuration, and half are in right-arm configuration. This is because the local planner has a difficult time switching between configurations, while environmental context can provide a lot of information about which configuration to use. 30 trajectories are generated and form our action library $\mathscr{V}$. Figure 2a shows an example set for a particular environment. Notice that in this case the straight-line initialization of CHOMP goes through the obstacle and therefore CHOMP has a difficult time finding a valid trajectory using this initial seed.

In our results we use a small number $(1-3)$ of slots in our sequence to ensure the overhead of ordering and evaluating the library is small. When CHOMP fails to find a collision-free trajectory for multiple initializations seeds, one can always fall back on slow but complete planners. Thus the contextual control sequence's role is to quickly evaluate a few good options and choose the initialization trajectory that will result in the minimum execution time. We note that in our experiments, the overhead of ordering and evaluating the library is negligible as we rely on a fast predictor and features computed as part of the trajectory optimization, and by choosing a small sequence length we can effectively compute a motion plan with expected planning time under 0.5s. We can solve most manipulation problems that arise in our manipulation research very quickly, falling back to initializing the trajectory optimization with a complete motion planner only in the most difficult of circumstances.

For each initialization trajectory, we calculate 17 simple feature values which populate a row of the feature matrix $\mathbf{X_i}$: length of trajectory in joint space; length of trajectory in task space, the xyz values of the end effector position at the exploration point (3 values), the distance field values used by CHOMP at the quarter points of the trajectory (3 values), joint values of the first 4 joints at both the exploration point



(a) The default straight-line initialization of CHOMP is marked in orange. Notice this initial seed goes straight through the obstacle and causes CHOMP to fail to find a collision-free trajectory.



(b) The initialization seed for CHOMP found using CONSEQOPT is marked in orange. Using this initial seed CHOMP is able to find a collision free path that also has a relatively short execution time.

Fig. 2: CHOMP initialization trajectories generated as control actions for CONSEQOPT. Blue lines trace the end effector path of each trajectory in the library. Orange lines in each image trace the initialization seed generated by the default straight-line approach and by CONSEQOPT, respectively.

(4 values) and the target pose (4 values), and whether the initialization seed is in the same left/right kinematic arm configuration as the target pose. During training time, we evaluate each initialization seed in our library on all environments in the training set, and use their performance and features to train each regressor $\Re_i$ in CONSEQOPT. At test time, we simply run Algorithm 2 without the training step to produce $Y_{\Re_1,\ldots,\Re_N}$ as the sequence of initialization seeds to be evaluated. Note that while the first regressor uses only the 17 basic features, the subsequent regressors also include the difference in feature values between the remaining actions and the actions chosen by the previous regressors. These difference features improve the algorithm's ability to consider trajectory diversity in the actions chosen.

We compare CONSEQOPT with two other methods of ranking the initialization library: a random ordering of the actions, and an ordering by sorting the output of the first regressor. Sorting by the first regressor is functionally the same as maximizing the absolute benefit rather than the marginal benefit at each slot. We compare both the number of CHOMP failures as well as the average execution time of the final trajectory. For execution time, we assume the robot can be actuated at 1 rad/second for each joint and use the shortest trajectory generated using the N seeds ranked by CONSEQOPT as the performance. If we fail to find a collision free trajectory and need to fall back to a complete planner (RRT [13] plus trajectory optimization), we apply a maximum execution time penalty of 40 seconds due to the longer computation time and resulting trajectory.

The results over 212 test environments are summarized in Figure 3. With only simple straight line initialization, CHOMP is unable to find a collision free trajectory in 162/212 environments, with a resulting average execution time of 33.4s. While a single regressor ($N = 1$) can reduce the number of CHOMP failures from 162 to 79 and the average execution time from 33.4s to 18.2s, when we extend the sequence length, CONSEQOPT is able to reduce both metrics faster than a ranking by sorting the output of the first regressor. This is because for $N > 1$, CONSEQOPT chooses a primitive that provides the maximum marginal benefit, which results in trajectory seeds that have very different features from the previous slots' choices. Ranking by the absolute benefit tends to pick trajectory seeds that are similar to each other, and thus are more likely to fail when the previous seeds fail. At a sequence length of 3, CONSEQOPT has only 16 failures and an average execution time of 3 seconds. **A 90% improvement in success rate and a 75% reduction in execution time.** Note that planning times are generally negligible compared to execution times for manipulation hence this improvement is significant. Figure 2b shows the initialization seed found by CONSEQOPT for the same environment as in Figure 2a. Notice that this seed avoids collision with the obstacle between the manipulator and the target object enabling CHOMP to produce a successful trajectory.

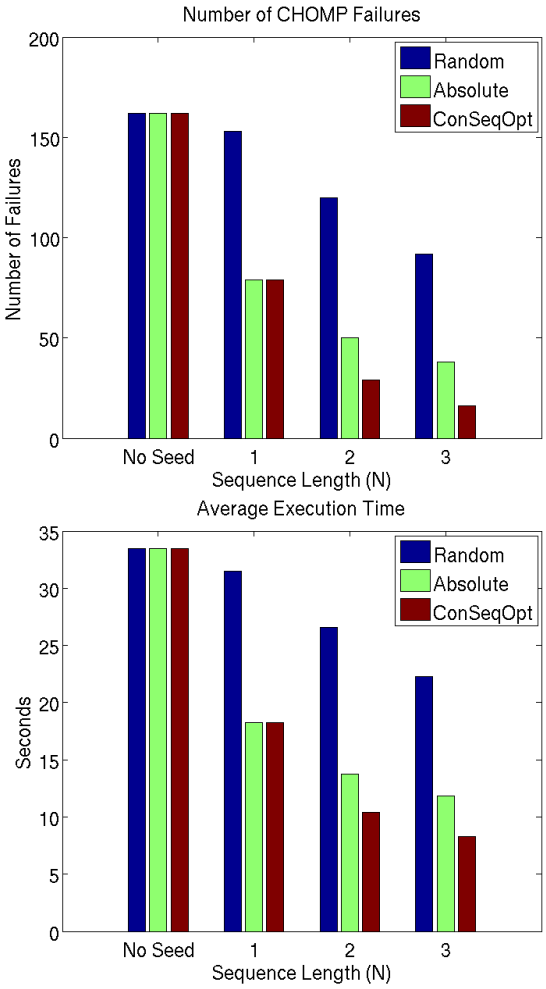


Fig. 3: Results of CONSEQOPT for manipulation planning in 212 test environments. The top image shows the number of CHOMP failures for three different methods after each slot in the sequence. CONSEQOPT not only significantly reduces the number of CHOMP failures in the first slot, but also further reduces the failure rate faster than both the other methods when the sequence length is increased. The same trend is observed in the bottom image, which shows the average execution time of the chosen trajectory. The 'No Seed' column refers to the straight-line heuristic used by the original CHOMP implementation

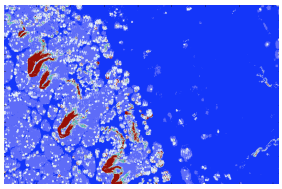
B. Mobile Robot Navigation

An effective means of path planning for mobile robots is to sample a budgeted number of trajectories from a large library of feasible trajectories and traverse the one which has the lowest cost of traversal for a small portion and repeat the process again. The sub-sequence of trajectories is usually computed offline [9, 7]. Such methods are widely used in modern, autonomous ground robots including the two highest placing teams for DARPA Urban Challenge and Grand Challenge [25, 15, 24, 23], LAGR [10], UPI [1], and Perceptor [12] programs. We use CONSEQOPT to maximize this function and generate trajectory sequences taking the current environment features.

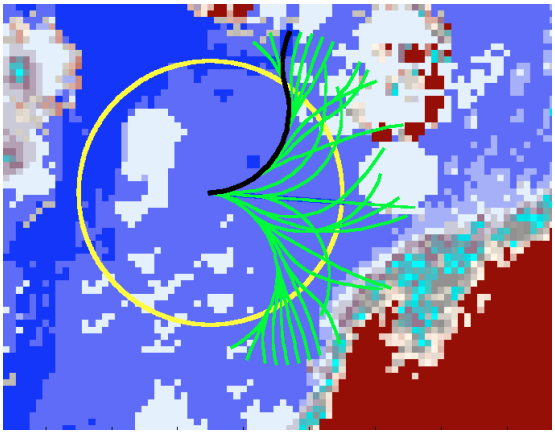
Figures 4a and 4b shows a section of Fort Hood, TX and the corresponding robot cost-map respectively. We simulated a robot traversing between various random starting and goal locations using the maximum-discrepancy trajectory [9] sequence as well as sequences generated by CONSEQOPT using Algorithm 1. A texton library [26] of 25 k-means cluster centers was computed for the whole overhead map. At runtime the texton histogram for the image patch around the robot was used as features. Online linear support vector machines (SVM) with slack re-scaling [19] were used as the cost-sensitive classifiers for each slot. We report a 9.6% decrease over 580 runs using $N = 30$ trajectories in the cost of traversal as compared to offline precomputed trajectory sequences which maximize the area between selected trajectories [9]. Our approach is able to choose which trajectories to use at each step based on the appearance of terrain (woods, brush, roads, etc.) As seen in Figure 4c at each time-step CONSEQOPT the trajectories are so selected that most of them fall in the empty space around obstacles.



(a) Overhead color map of portion of Fort Hood, TX



(b) Cost map of corresponding portion



(c) Robot traversing the map using CONSEQOPT generating trajectory sequences which try to avoid obstacles in the vicinity

REFERENCES

[1] J.A. Bagnell, D. Bradley, D. Silver, B. Sofman, and A. Stentz. Learning for autonomous navigation. *Robotics Automation Magazine, IEEE*, 17(2):74 –84, June 2010.

[2] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J Kuffner. Grasp planning in complex scenes. In *IEEE-RAS Humanoids*, December 2007.

[3] A. Beygelzimer, V. Dani, T. Hayes, J. Langford, and B. Zadrozny. Error limiting reductions between classification tasks. In *ICML*. ACM, 2005.

[4] Andreas Kraus Carlos Guestrin. Beyong convexity: Submodularity in machine learning. URL www.submodularity.org.

[5] E. Chinellato, R.B. Fisher, A. Morales, and A.P. del Pobil. Ranking planar grasp configurations for a three-finger hand. In *ICRA*, volume 1, pages 1133–1138. IEEE, 2003.

[6] Debadeepta Dey, Tian Yu Liu, Boris Sofman, and J. Andrew (Drew) Bagnell. Efficient optimization of control libraries. Technical Report CMU-RI-TR-11-20, Robotics Institute, Pittsburgh, PA, June 2011.

[7] L.H. Erickson and S.M. LaValle. Survivability: Measuring and ensuring path diversity. In *ICRA*, pages 2068–2073. IEEE, 2009.

[8] E. Frazzoli, MA Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. In *Decision and Control, 2000.*, volume 1, 2000.

[9] C. Green and A. Kelly. Optimal sampling in the space of paths: Preliminary results. Technical Report CMU-RI-TR-06-51, Robotics Institute, Pittsburgh, PA, November 2006.

[10] LD Jackel et al. The DARPA LAGR program: Goals, challenges, methodology, and phase I results. *JFR*, 2006.

[11] N. Jetchev and M. Toussaint. Trajectory prediction: learning to map situations to robot trajectories. In *ICML*. ACM, 2009.

[12] Alonzo Kelly et al. Toward reliable off road autonomous vehicles operating in challenging environments. *IJRR*, 25 (1):449–483, May 2006.

[13] Jr. Kuffner, J.J. and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *ICRA*, volume 2, pages 995 –1001, 2000.

[14] John Langford and Alina Beygelzimer. Sensitive error correcting output codes. In *Learning Theory*, Lecture Notes in Computer Science. 2005.

[15] M. Montemerlo et al. Junior: The stanford entry in the urban challenge. *JFR*, 25(9):569–597, 2008.

[16] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th ICML*, 2008.

[17] N. Ratliff, J.A. Bagnell, and S. Srinivasa. Imitation learning for locomotion and manipulation. Technical Report CMU-RI-TR-07-45, Robotics Institute, Pittsburgh, PA, December 2007.

[18] Nathan Ratliff, Matthew Zucker, J. Andrew (Drew) Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *ICRA*, May 2009.

[19] B. Scholkopf and A.J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. the MIT Press, 2002.

[20] M. Stolle and C.G. Atkeson. Policies based on trajectory libraries. In *ICRA*, pages 3344–3349. IEEE, 2006.

[21] M. Streeter and D. Golovin. An online algorithm for maximizing submodular functions. In *NIPS*, pages 1577–1584, 2008.

[22] M. Streeter, D. Golovin, and A. Krause. Online learning of assignments. In *NIPS*, pages 1794–1802. Citeseer, 2009.

[23] Sebastian Thrun et al. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, 23:661–692, September 2006.

[24] Christopher Urmson et al. A robust approach to high-speed navigation for unrehearsed desert terrain. *JFR*, 23 (1):467–508, August 2006.

[25] Christopher Urmson et al. Autonomous driving in urban environments: Boss and the urban challenge. *JFR*, June 2008.

[26] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *ICCV*, 2005.

[27] Matthew Zucker. A data-driven approach to high level planning. Technical Report CMU-RI-TR-09-42, Robotics Institute, Pittsburgh, PA, January 2009.