Learning to guide task and motion planning using score-space representation

Beomjoon Kim, Leslie Pack Kaelbling and Tomás Lozano-Pérez

Abstract— **In this paper, we propose a learning algorithm that speeds up the search in task and motion planning problems. Our algorithm proposes solutions to three different challenges that arise in learning to improve planning efficiency: what to predict, how to represent a planning problem instance, and how to transfer knowledge from one problem instance to another. We propose a method that predicts constraints on the search space based on a generic representation of a planning problem instance, called score space, where we represent a problem instance in terms of performance of a set of solutions attempted so far. Using this representation, we transfer knowledge, in the form of constraints, from previous problems based on the similarity in score space. We design a sequential algorithm that efficiently predicts these constraints, and evaluate it in three different challenging task and motion planning problems. Results indicate that our approach perform orders of magnitudes faster than an unguided planner.**

I. Introduction

Task and motion planning (TAMP) problems require a robot to decide how to manipulate objects in cluttered scenes to achieve a high-level goal such as setting the table. A variety of planners have been developed for TAMP problems ([1], [2], [3]). However, their worst-case computation time generally scales exponentially with problem size, and each new problem instance must be solved from scratch, making them inefficient for real world tasks. In contrast, humans are able to short-cut their planning process by learning to adapt previous planning experience to reduce the search space intelligently for new problem instances. This observation motivates the design of an algorithm that learns from experience to make predictions that guide the search of a planner. We face three important questions in designing the algorithm: (1) what to predict, (2) how to represent a problem instance, and (3) how to transfer knowledge from past experience to the current problem instance.

The first challenge is what to predict. Previous approaches to using learning to speed up planning have tried predicting a complete solution, or a subgoal that fully specifies robot configuration and world state, including object poses. However, because of the intricate relationship between object poses and the robot free-space, a small change in the environment may completely alter the space of feasible solutions. This non-smoothness in the relationship between a problem instance and its solution makes it difficult to predict a complete solution or a subgoal based on experience. This difficulty is illustrated in Figure 1 in a pick-and-place domain.

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA beomjoon@mit.edu, lpk@mit.edu, tlp@mit.edu.
We gratefully acknowledge support from NSF grants 1420927 and 1523767, from ONR grant N00014-14-1-0486, and from ARO grant W911NF1410433. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

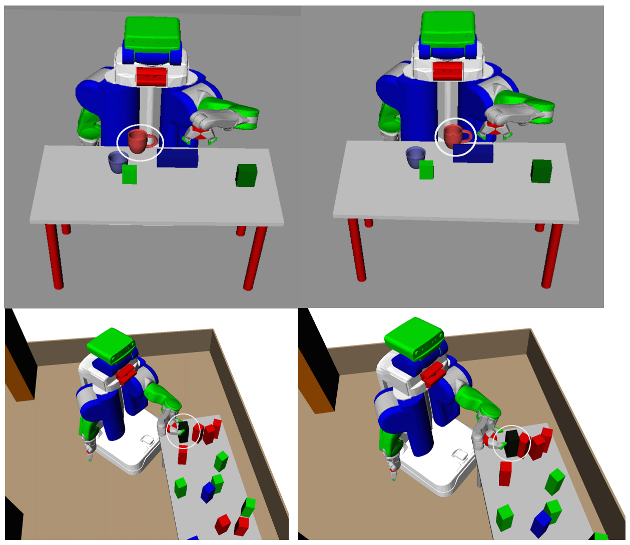


Fig. 1: Top: Obstacle poses differing by 0.02m require entirely different choice of grasp for picking the blue object. Bottom: Object placement pose choices differing by 0.05m make the difference between feasibility and infeasibility.

Building on these observations, we instead learn to predict constraints on the solution by finding a subset of decision variables that can be predicted reliably, while leaving the rest of the decision variables to be filled in by the planner. This decomposition is based on the intuition that constraints can generalize more effectively across problem instances than a complete solution or a subgoal. For instance, consider a robot trying to pick an object from a shelf, as shown in Figure 2 (right). A constraint that forces the robot to approach the object either from the side or the top, depending on whether the object is on the table or the shelf, can be used more reliably across different arrangements of obstacles and object poses than a detailed path plan and a pre-grasp configuration.

We will refer to the subset of decision variables that we predict as *solution constraints*. Solution constraints, when intelligently chosen, will effectively reduce the search space while preserving the robustness of the planner against changes in problem instances.

The second challenge is representing problem instances. In the TAMP problems that we are contemplating, problem instances may have different numbers of objects with different shapes and poses, making it difficult to design a generic

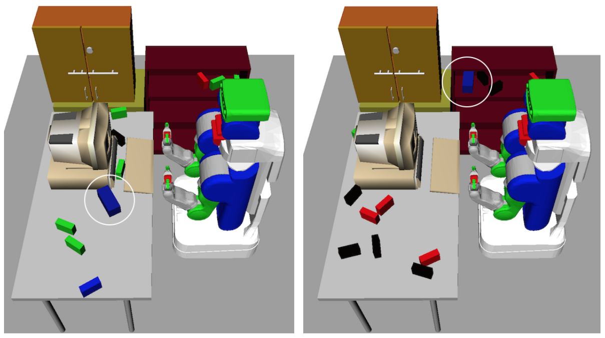


Fig. 2: Two instances of the grasp selection domain. The arrangement and number of obstacles vary randomly across different planning problem instances. The objective is to find an arm trajectory to a pre-grasp pose for the blue box, marked with a circle, whose pose is randomly determined in each problem instance

feature representation for learning.

In light of this, we propose a new type of representation for problem instances, called *score space*. We represent a problem instance in terms of a vector of the scores for a set of plans on that instance, where each of the plans is computed based on one of a fixed set of solution constraints.

The main advantage of the score space representation is that it gives direct information about similarity between problem instances and solution constraints, without any explicit state representation. For instance, consider the task in Figure 2. By computing a plan associated with the solution constraint that forces the robot to approach the object from the top and observing that it has a low score, we can learn that it is occluded from above, e.g. by a shelf, and can predict other more appropriate action choices. This information is not biased by the designer's choice of representation: having learned that approaching the object from the side did not work, a learning algorithm may try to approach the object from the top. Such reasoning is much more difficult when the learning algorithm is forced to reason in terms of a state-space representation of the problem, such as poses of obstacles or 2D or 3D images of the scene. The main disadvantage is that the score space information is computationally expensive to obtain, since it requires computing the plans associated with solution constraints.

This observation brings us to the third challenge of learning to plan, which is how to transfer knowledge from past experience to the current problem instance efficiently. We propose to solve this challenge by using the expectation and correlation of the scores of solution constraints from past problem instances to determine which solution constraint to try next. Our intuition is that the solution constraints that performed well or poorly together are more likely to do so in a new problem instance; for instance, in the previous example for picking an object from a shelf, grasps from the sides would have worked well in most of the problem instances, while grasps from the top or bottom would have worked poorly.

Our algorithm, BOX, is an upper-confidence-bound (UCB) type, experience-based, black-box function-optimization technique [4], [5] that learns to suggest appropriate solution constraints based on the scores of previously attempted plans. We evaluate BOX in three different TAMP domains of increasing complexity, to demonstrate its flexibility and reliability. In all these tasks, we show that BOX can accelerate planning by orders of magnitude over a basic planner that does not use solution constraints. We also provide a comparison to a sampler that picks solution constraints at random, and to a state-of-the-art black-box function optimization technique called deterministic optimistic optimization (DOO) [6] that does not use the score space representation. We find that BOX outperforms these other methods.

II. Related Work

There is a substantial body of work aimed at improving motion planning performance on new problem instances based on previous experience on similar problem instances [7], [8], [9], [10], [11], [12]. The approach is to store a large set of solutions to earlier instances so that, when presented with a new problem instance, one can (a) retrieve the most relevant previous solution and (b) adapt it to the new situation. These methods differ in the way that they find the most relevant previous solution and how it is adapted.

Several of these approaches define a similarity metric between problem instances and retrieve solutions based on this metric. For example, Hondál et al. [8] use the distance between the start and goal pairs as the metric, whereas, Hutchinson et al. [9] based their metric on descriptions of quickly generated low-quality solutions for the current and previous instance. Jetchev et al. [10] use a mapping into a task-relevant space and measure similarity in that space, via a learned metric.

Instead of defining solution similarity, Berenson et al. [7] define relevance of earlier solutions by measuring the degree of constraint violation, for example collision, in the current situation. A related idea is developed by Phillips et al. [12], where the search graph of past solutions is saved and the search for the current problem instance is biased towards the part of this past graph that is still feasible.

For more complex robotic planning, such as for mobile manipulation in cluttered environments, complete solutions are more difficult to adapt to new problem instances. In particular, the length of the plans is highly variable and they contain both discrete and continuous parameters.

Some earlier approaches [13], [14] have also focused on predicting partial solutions, in the form of a goal state or subgoals, instead of a complete solution. For instance, in the work of Dragan et al. [13], the objective is to learn from previous examples a classifier (or regressor) that, given a hand-designed feature representation of a planning problem instance, enables choosing a goal that leads to a good locally optimal trajectory. In the approach of Finney et al. [14], the goal is to learn a model that predicts partial paths or subgoals, from a given parametric representation of a planning problem instance, aimed at enabling a randomized motion planner to navigate through narrow passages.

Our approach can be seen as a method for choosing actions from a library; several methods have been proposed for this problem [15], [16], [6]. Dey et al. [15] propose a method that finds a fixed ordering of the actions in a library that optimizes a user-defined submodular function, for example, the probability that a sequence of candidate grasps will contain a successful one. Unlike our work, this method produces a static list, which does not change across different problem instances. Later, Dey et al. [16], generalized the approach by producing an ordered list of classifiers (operating over environment features) that select actions for a given problem instance. This approach again requires hand-designed features for the problem instances.

BOX is motivated by the *principle of optimism in the face of uncertainty* which is well surveyed by Munos et al. [6]. The main idea is to select the most "optimistic" item from the given set of items, by constructing an upper bound on the values of un-evaluated items. We build the upper bound using the statistics of the past planning problem instances, combined with the results of a subset of the available solution constraints so far on the current problem instance.

III. Problem Formulation

Our premise is that calling the planner with a solution constraint, although much more efficient than the completely unconstrained problem, takes a significant amount of time and may generate significantly suboptimal plans if the solution constraint used is not a good match for the problem instance. Given a new instance we will call the planner with a fixed number of solution constraints and return the best plan obtained. So, our problem is which solution constraints should be tried, and in what order.

We formulate the problem as a black-box function optimization problem over a discrete space of candidate solution constraints, and use upper bounds constructed from experience on previous problem instances as well as the accumulated experience on this instance to determine which constraint to try next.

Formally, we have a sample space for problem instances, Ω, whose elements ω are distributed according to $P(\omega)$; a space of possible planning solutions, $\mathscr{X}$; and a space of solution constraints, Θ. The plan solution space includes all possible assignments to all of the decision variables, and the solution constraint space includes all possible assignments to a *subset* of decision variables for the given planning problem. The function $J(\omega,x)$ specifies the score of a solution $x \in \mathscr{X}$ on problem instance $\omega \in \Omega$. We assume a planner $\pi : \Omega \rightarrow \mathscr{X}$ that, given a problem instance ω can return a solution $\pi(\omega) \in \mathscr{X}$ that is either feasible or locally optimal depending on the nature of the problem. In addition, we assume that, given a solution constraint θ, the planner π will return $\pi(\omega,\theta) \in \mathscr{X}$,which is a plan subject with the solution constraint θ; in general this solution will not be optimal (so in general $J(\omega,\pi(\omega)) > J(\omega,\pi(\omega,\theta))$), unless θ was perfectly suited to the problem instance ω, but constraining the plan to satisfy θ will make it significantly more efficient to compute. With a slight abuse of notation, we will denote $J(\omega,\theta) = J(\omega,\pi(\omega,\theta))$, the evaluation of the solution constraint.

Let $\hat{\Theta} = \{\theta_1,\ldots,\theta_m\} \subseteq \Theta$ be a set of samples from Θ. Now, we formulate our problem as follows: given a "training set" of example problem instances $\omega_1,\ldots,\omega_n$ sampled identically and independently from $P(\omega)$, a discrete set of solution constraints $\hat{\Theta}$, and the score function $J(\cdot,\cdot)$, generate a high-scoring solution to the "test" problem instance w_{n+1}.

Interesting problems that we do not explicitly address in this paper are: how to select the subset of decision variables for specifying constraints θ, and how to select $\hat{\Theta}$ from Θ. In this paper, we take the simple approach of solving the training problem instances and then extracting the θ values corresponding to a set of hand-chosen decision variables as solution constraints. The details of constructing $\hat{\Theta}$ are provided in Algorithm 2.

IV. Algorithm

Instead of designing a problem-dependent representation for problem instances, we represent a problem instance with as a vector of scores of solution constraints $\hat{\Theta}$, where

$$\Phi(\omega) = [J(\omega,\theta_1),\cdots,J(\omega,\theta_m)]$$

$\Phi(\omega)$ here is a random vector that maps a sample from the sample space of problem instances to $\mathbb{R}^m$. Using this representation, our training data constructed from n problem instances can be represented with a $n \times m$ matrix

$$\mathbf{D} = \begin{bmatrix} \Phi(\omega_1) \\ \Phi(\omega_2) \\ \vdots \\ \Phi(\omega_n) \end{bmatrix}$$

that we call the *score matrix*. Now, given a new problem instance, ω, our goal is to take advantage of one or more solution constraints in $\hat{\Theta}$ to find a high scoring plan without evaluating all of solution constraints in $\hat{\Theta}$. To do this, we will develop a procedure that evaluates $J(\omega,\theta)$ by computing a plan $\pi(\omega,\theta)$ for $k << m$ values of θ.

We begin by making use of the intuition that some solution constraints (via the plans they generate) are inherently more useful than others, independent of the problem instance. This leads to a naive score-space approach, STATIC, that tries solution constraints in $\hat{\Theta}$ in a static order according to the empirical mean scores in the $1 \times m$ vector computed by

$$\hat{\mu} := \frac{1}{n}\sum_{i=1}^{n}\mathbf{D}_i \tag{1}$$

where i indicates the row of the score matrix, and then returns the highest scoring plan obtained from trying the top k solution constraints.

This simple approach does not take advantage of the fact that there are correlations among the scores of solution constraints across problem instances; that is, the score of a solution constraint that has been already tried on this problem instance can inform us about the scores of other untried but correlated solution constraints. In order to exploit correlation, we assume that the random vector Φ is distributed according to a multivariate Gaussian distribution, $\mathcal{N}(\mu, \Sigma)$.

Now the score matrix is used to estimate the parameters for the prior distribution of Φ, $\hat{\mu}$ and $\hat{\Sigma}$, where $\hat{\mu}$ is defined in equation 1 and

$$\hat{\Sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{D}_i - \hat{\mu})^T(\mathbf{D}_i - \hat{\mu})$$

is a $m \times m$ covariance matrix. This prior distribution is updated given evidence about a new problem instance, in the form of score values.

Algorithm 1 contains detailed pseudo-code for an algorithm based on these ideas, called BOX, which stands for Blackbox Optimization with eXperience. It takes as input: w_{n+1}, the "test" planning problem instance; C, a constant governing the magnitude of the bounds; k, the number of solution constraints to evaluate; $\hat{\Theta}$, the set of solution constraints in the training set; $\hat{\mu}$ and $\hat{\Sigma}$, the parameters for prior distribution of $\Phi(w_{n+1})$; J, the scoring function; and π, the planner.

The algorithm iterates over solution constraints: first it selects a solution constraint, then it uses it to construct a new plan, then the score of that plan combined with the prior computed from $\mathbf{D}$ is used to determine the next solution constraint to evaluate. We use I_t to denote the indices of solution constraints that have been tried up to time t, $\bar{I}_t = I \setminus I_t$ to denote the ones not tried, $i^{(t)}$ to denote the index of the solution constraint chosen at time t, $x^{(t)}$ to denote the associated plan, $J^{(t)}$ to denote the score of that plan on the given problem instance, $J^{1:t}$ and $J^{\overline{1:t}}$ to denote the scores of tried and untried solution constraints up to time t, respectively. At time t, we can rearrange the covariance matrix $\hat{\Sigma}$ as

$$\begin{bmatrix} \hat{\Sigma}_{\bar{I}_t,\bar{I}_t} & \hat{\Sigma}_{\bar{I}_t,I_t} \\ \hat{\Sigma}_{I_t,\bar{I}_t} & \hat{\Sigma}_{I_t,I_t} \end{bmatrix}$$

where the subscript represents a set of rows and columns of the matrix $\hat{\Sigma}$. This way, the top-left block matrix is the covariance among tried solution constraints, the top-right and bottom-left represent covariance among tried and untried solution constraints, and the bottom-right represents the covariance among the untried solution constraints.

Line 2 of Algorithm 1 selects the next solution constraint to try based on the principle of *optimism in the face of uncertainty*, by selecting the one for which the upper bound of the α-percent confidence interval is highest. The next three lines generate a plan using the chosen solution constraint, and then evaluate it. At iteration t, given the experience of trying $\theta^{(1)}, \cdots, \theta^{(t-1)}$ and getting scores $J^{1:t} := [J^{(1)}, \cdots, J^{(t)}]$, our posterior on the scores of of the untried solution constraints, denoted $J^{\overline{1:t}}$, is

$$J^{\overline{1:t}} | J^{1:t} \sim \mathcal{N}(\hat{\mu}^{(t+1)},\ \hat{\Sigma}^{(t+1)})$$

where

$$\begin{aligned} \hat{\mu}^{(t+1)} &= \hat{\mu}_{\bar{I}_t} + \hat{\Sigma}_{\bar{I}_t,I_t}(\hat{\Sigma}_{I_t,I_t})^{-1}(J^{1:t} - \hat{\mu}_{I_t}) \\ \hat{\Sigma}^{(t+1)} &= \hat{\Sigma}_{\bar{I}_t,\bar{I}_t} - \hat{\Sigma}_{\bar{I}_t,I_t}(\hat{\Sigma}_{I_t,I_t})^{-1}\hat{\Sigma}_{I_t,\bar{I}_t} \end{aligned} \tag{2}$$

Algorithm 1 BOX($w_{n+1}, C, k, \hat{\Theta}, \hat{\mu}, \hat{\Sigma}, J, \pi$)

```
for  $t = 1$ to $k$  do
    $i^{(t)} = arg\max_{i \in I^{\overline{1,\cdots,t}}} \hat{\mu}_i^{(t)} + C \cdot \sqrt{\hat{\Sigma}_{ii}^{(t)}}$ // $i^{th}$ entry and $i^{th}$ diagonal entry
    $\theta^{(t)} = \hat{\Theta}_{i^{(t)}}$
    $x^{(t)} = \pi(w_{n+1}, \theta^{(t)})$
    $J^{(t)} = J(w_{n+1}, x^{(t)})$
    Compute $\hat{\mu}^{(t+1)}$ and $\hat{\Sigma}^{(t+1)}$ using eqn. 2
end for
$t^* = arg\max_{t \in \{1,\cdots,k\}} J^{(t)}$
return $x^{(t^*)}$
```

Algorithm 2 GenerateTrainingData($n, \pi, J, [\omega_1, \cdots, \omega_n]$)

```
for $\omega$ in $[\omega_1, \cdots, \omega_n]$ do
    $x_i = \pi(\omega)$ // repeat to get multiple solutions if desired
    $\theta_i = extractConstraint(x_i)$ // elements of $\hat{\Theta}$
end for
for $\omega$ in $[\omega_1, \cdots, \omega_n]$ do
    for $\theta$ in $\hat{\Theta}$ do
        $J(\omega, \theta) = J(\omega, \pi(\omega, \theta))$ // elements of $\mathbf{D}$
    end for
end for
return $\mathbf{D}, \hat{\Theta}$
```

The constant C governs the size of the confidence interval on the scores. We arbitrarily choose $C = 1.96$ to ensure 95% confidence interval in our experiments, but it could be tuned via cross validation for better performance. The number of evaluations k should be chosen based on the desired trade-off between computation time and solution quality.

In order to create the score matrix $\mathbf{D}$ and solution constraints $\hat{\Theta}$, we run Algorithm 2. This algorithm takes as input n, the number of training problem instances, π a planning algorithm that can solve problem instances w_{n+1} without additional constraints, J, the scoring function for a plan, and $[\omega_1, \cdots, \omega_n]$, a set of training sample problem instances drawn iid from $P(\omega)$. For each problem instance, a solution is generated using π, and a constraint is extracted from the solution and added to set $\hat{\Theta}$. The process of extracting constraints is domain-dependent; several examples are illustrated in section V. Each new solution constraint is used to generate a solution $\pi(w_{n+1}, \theta)$ whose score $J(w_{n+1}, \theta)$ is stored in the $\mathbf{D}$ matrix.

V. Experiments

We demonstrate the effectiveness of score-space algorithms STATIC and BOX in three robotic planning domains, each of which has several decision variables and different types of solution constraints. The first domain requires the robot to move to a pre-grasp configuration for an object, which involves choosing the configuration as well as a collision-free path in a domain in which the number and poses of obstacles vary. The second domain requires the robot to pick an object that is randomly placed on one of the tables in a large room, which requires choosing a base and a pre-grasp configuration, and collision-free path from the initial configuration to the pre-grasp configuration. The problem instances in this domain again vary with respect to obstacle and object poses. The last domain requires the robot to pick an object, move to another room by going through a narrow passage, and then place the object on a table in that room. Solving this problem requires choosing a pre-grasp configuration, a placement pose for the object on the table, a pre-place configuration, and paths that connects all these configurations. The problem instances vary in terms of size of the object being manipulated, and the poses of obstacles.

In each of these domains, once the solution constraints have been specified, only paths need to be planned, and so $\pi(\cdot,\theta)$ is a path planner. To implement the RAWPLANNER, $\pi(\omega)$, we simply randomly sample the solution constraints from their original spaces, such as $\mathbb{R}^2$ for object pose on a table instead of from $\hat{\Theta}$, and then use $\pi(\omega,\theta)$ with the sampled solution constraint.

We are interested in both running time and solution quality. We compare score-space algorithms, STATIC and BOX, with RAWPLANNER as well as two other methods that generate plans by selecting a subset of size k of the solution constraints from $\hat{\Theta}$ and return the highest scoring one. RAND selects k of the θ_i values at random from $\hat{\Theta}$ DOO is an adaptation of DOO [6] to optimization of a black-box function over a discrete set, which is $\hat{\Theta}$ in our case. Like BOX, it alternates between evaluating θ_j and constructing upper bounds on the unevaluated θ_i for k rounds. It assumes that the function is Lipschitz continuous with constant λ, and uses the bound $J^{\omega}(\theta_i) \leq J^{\omega}(\theta_j) + \lambda \cdot l(\theta_i,\theta_j)$ for some semi-metric l, $\lambda \in \mathbb{R}$. We use the Euclidean metric for l, and $\lambda = 1$.

To show that score-space algorithms can work with different planners, we show results using two different planners: bidirectional RRT implemented in OpenRAVE [17], seeded with a fixed randomization seed value, and Trajopt [18]. In the last domain, where there is a narrow-passage path planning problem, Trajopt could not find a feasible path without being given a good initial solution, so we omit it.

In each domain, we report the results using two plots, the first showing the time to find the first feasible solution and the second showing how the solution quality improves as the algorithms are given more time. Each data point on each plot is produced using leave-one-out cross-validation. That is, given a total data set of n problem instances and associated solutions, we report the average of n experiments, in which $n-1$ of the instances are used as training data and the remaining one is used as a test problem instance. In all cases, we seek short paths, and so use a score function that measures the trajectory length, assigning a large cost if the plan is infeasible in the problem instance. Given a plan $x = (c_1,\cdots,c_l)$ where c_i denotes a configuration of the robot,

$$J^{\omega}(x) = \begin{cases} -\sum_{i=1}^{l-1} ||c_{i+1} - c_i|| \text{ if } x \text{ feasible in } \omega \\ d, \text{ otherwise} \end{cases} \tag{3}$$

where $||\cdot||$ denote a suitable distance metric between configurations and $d = min(\mathbf{D}) - mean(\mathbf{D})$. This is our strategy for finding an artificial scale-sensitive minimum score for failing to solve a problem.

A. Picking an object in a cluttered scene

Our first problem domain is to find an arm motion to grasp an object that lies randomly either on a desk or bookshelf, where there also are randomly placed obstacles. Neither the grasp of the object nor the final configuration of the robot is specified, so the complete planning problem includes choosing a grasp, performing inverse kinematics to find a pre-grasp configuration for the chosen grasp, and then solving a motion planning problem to the computed pre-grasp configuration.

A planning problem instance for this domain is defined by an arrangement of several objects on a table. Figure 2 shows two instances of this problem, which are also part of the training data. There are up to 20 obstacles in each problem instance. The robot's active degrees of freedom (DOF) are its left and right arms, each of which has 7 DOFs, and torso height with 1 DOF, for a total of 15 DOF. $\hat{\Theta}$ consists of 81 different grasps per each arm, computed using OpenRAVE's grasp model function. Notice that since our search space for solution constraints is discrete, RAWPLANNER is equivalent to RAND.

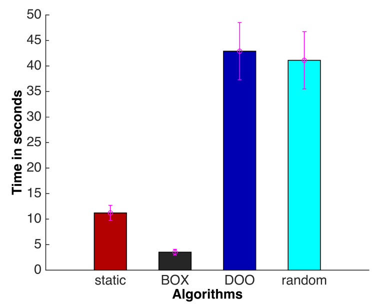
Given a solution constraint θ, which is a grasp (pose of robot hand with respect to the object) and an arm to pick the object with, it remains for $\pi(\omega,\theta)$ to find an IK solution and motion plan, which can be expensive, but predicting a good grasp makes the overall process much more efficient. The trajectory of the arm to the pre-grasp configuration, with the base fixed, is scored according to eqn. 3, with a score of d assigned to problem instances and constraints for which no solution is found within a fixed amount of computation.

The experiments were run on a data set of 1800 problem instances. Figure 3a compares the time required by each method to find the first feasible plan with RRT as the path planner, and Figure 3d compares the time with TrajOpt as the path planner. In both of the plots, we can observe that the score-space algorithms STATIC and BOX outperform all other algorithms in terms of finding a good solution with a given amount of time. BOX performs about three times faster than STATIC, showing the advantage of using the correlation information. Compared to DOO and RAND, BOX is more than nine times faster. DOO does only slightly better than RAND, which illustrates that in the space of grasps, the Euclidean metric is not effective.

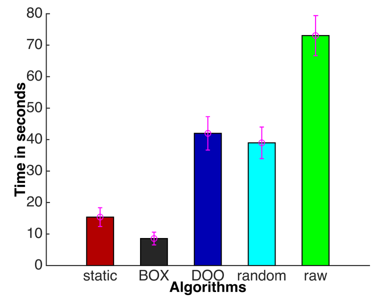
Figure 4a compares the solution quality vs time when RRT is used; figure 4d compares the same quantities when TrajOpt is used. Here, the score-space algorithms again outperform the other algorithms, with BOX outperforming STATIC.

B. Picking randomly placed object in a cluttered scene

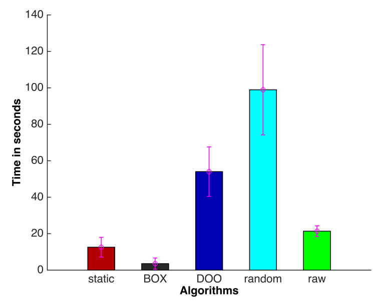
In this experiment, we evaluate how the score-space algorithms perform when we construct the matrix $\hat{\Theta}$ by sampling from a continuous space. Here, the robot needs to search for a base configuration, a left arm pre-grasp configuration, and a feasible path between these configurations to pick an object.



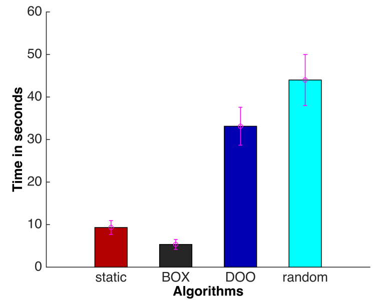
(a) Grasp and arm selection (RRT)



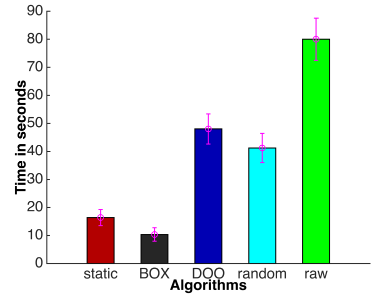
(b) Grasp and base selection (RRT)



(c) Pick-and-place (RRT)

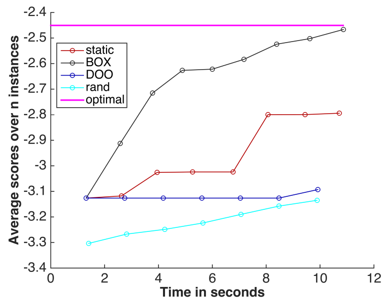


(d) Grasp and arm selection (TrajOpt)

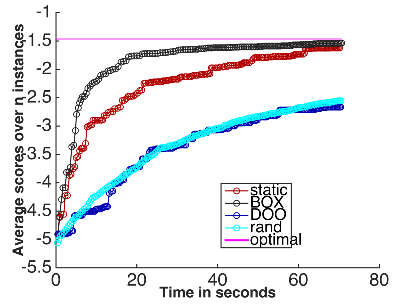


(e) Grasp and base selection (TrajOpt)

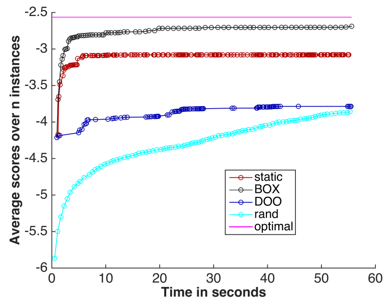
Fig. 3: LOOCV estimate of time to find first feasible solution, for each method in different domains. Bars indicate 95% confidence interval on mean. The top row uses RRT and the bottom row uses TrajOpt



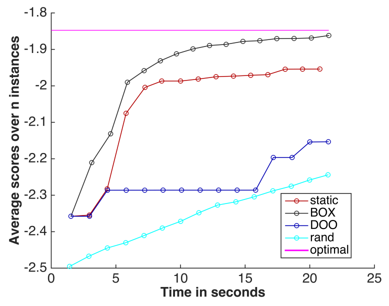
(a) Grasp selection (RRT)



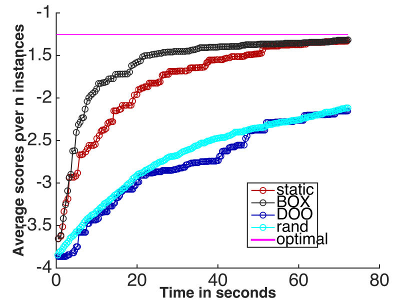
(b) Grasp and base selection (RRT)



(c) Pick-and-place (RRT)



(d) Grasp selection (TrajOpt)



(e) Grasp and base (TrajOpt)

Fig. 4: Solution score versus run time for different algorithms in various domains. The time axis goes until the first algorithm reaches 95% of the optimal score, marked with magenta. This optimal line is obtained by taking the θ from $\hat{\Theta}$ that achieved maximum score for each problem instance. The top row uses RRT and the bottom row uses TrajOpt

A planning problem instance is again defined by the arrangement of objects. Figure 5 shows three different training problem instances. We have 20 rectangular boxes as obstacles, all resting on the two tables both of which remain fixed in all instances. For each of the red obstacles and the blue target object, the (x,y) location and orientation in the plane of the table are randomly chosen subject to the constraint that they are not in collision. It is possible that the problem instances will be infeasible (the target object is too occluded or kinematically unreachable by the robot). The robot always starts at the same initial configuration.

The robot's active DOFs include its base configuration, torso height, and left arm configuration, for a total of 11 DOF. A solution constraint for this domain consists of the robot base configuration to pick the target object, (x,y,ψ), where ψ is an orientation of the robot, as well as one of 81 grasps from the previous section.

The solution constraints in this case are the grasp g, and the base configuration k. Given a planning problem instance with no constraints, the RAWPLANNER for this domain performs three sampling procedures, backtracking among them as needed to find a feasible solution.

1) Sample a base configuration, $k=(x,y,\psi)$, from a circular region of free configuration space, with radius equal to the length of the robot's arm, centered at the location of the object.
2) Sample, without replacement, from the 81 grasps until a legal one is found, i.e. one for which there is an IK solution in which the robot is holding the target object using that grasp in a collision-free configuration.
3) Use bidirectional RRT or TrajOpt to find a path for the arm and torso between the configurations found in steps 1 and 2.

We assume that the configuration from step 1 is reachable from the initial configuration. To extract a solution constraint from the resulting plan, we simply return the base configuration from step 1 and the grasp from step 2.

Unlike RAWPLANNER which has to search for k and g, $\pi(\omega,\theta)$ simply solves the inverse kinematics and motion planning problems as in the previous example. The trajectory of the arm to the pre-grasp configuration, with the base fixed according to the constraint, is scored according to equation 3, with a score of d assigned to problem instances and constraints for which no feasible solution is found within a fixed number of iterations of the RRT. The experiments were run on a data set of 1000 problem instances. The set $\hat{\Theta}$ contained 1000 pairs of grasp and robot base configuration, each extracted from a different problem instance.

Figures 3b and 3e show the time required by each method to find the first feasible plan, using RRT and TrajOpt as the planner. The score-space algorithms perform orders of magnitude better than the other algorithms, with BOX again outperforming better than STATIC. DOO and RAND do provide some advantage by using previously stored solution constraints compared to RAWPLANNER. RAWPLANNER has to sample in the continuous space of base configurations and check whether an IK solution and feasible path exist by running IK and path planning. This causes a significant increase in time to find a solution.

Figure 4b compares the solution quality vs time when RRT is used and Figure 4e compares the same quantities when TrajOpt is used. Again, the score-space approaches outperform all other methods, with BOX performing better than STATIC, by using the correlation information from the score space. DOO and RAND perform similarly, mainly because that simple Euclidean distance is not effective for the hybrid space of base configuration and grasps.

C. Moving object to a different room

In this last experiment, with problem instances are shown in figure 6, we introduce solution constraints involving the placements of objects. Here, the robot needs to pick a large object (shown in black) up off of a table in one room, carry it through a narrow door, and place the object on a table. The initial poses of the target object and the robot are fixed, but problem instances vary in terms of the initial poses of 28 obstacles on both the starting and final tables, which are chosen uniformly at random on the table-tops subject to non-collision constraints, and the length of the target object, which is chosen at random from three fixed sizes.

The robot's active DOFs are the same as the previous problem domain. The solution constraints in this domain consists of grasp g to pick the object, o, the placement pose of the object on the table in the back room, k_b, the pre-placement base configuration of the robot for placing the object at pose o, and k_{sg}, the subgoal base configuration for path planning through the narrow passage to k_b from initial configuration.

Given a problem instance with no constraint, RAWPLANNER performs four sampling procedures, similarly to the previous domain.

1) Sample a grasp g, without replacement, from the 81 grasps until a legal one is found.
2) Plan a path for the arm and torso to the pre-grasp configuration found in step 1. If none is found, choose another grasp.
3) Sample a collision-free object pose o on the table in the other room.
4) Sample k_b, the pre-placement base configuration, from a circular region of free configuration space around o, with radius equal to the length of the robot's arms. If none is found, go back to the previous step.
5) Plan a path from the initial configuration to k_b. If none is found, go back to the previous step.
6) Plan a path from k_b to a place configuration for putting the object down at o. If none is found, go back to the previous step.

In contrast, given a solution constraint, $\pi(\omega,\theta)$ simply solves for inverse kinematics and path plans.

The experiments were run on a data set of 1500 problem instances, with 500 instances per rod size. The set $\hat{\Theta}$ contained 1000 tuples of solution-constraint values, obtained first by running Algorithm 2 and then randomly subsampling them to reduce the size to 1000.

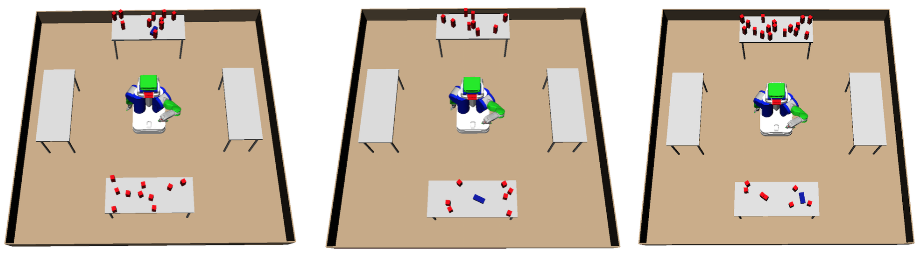


Fig. 5: Three instances in which the robot must select base configuration, grasp, and paths, to pick the target object (blue). The poses of the objects are randomly varied between instances.

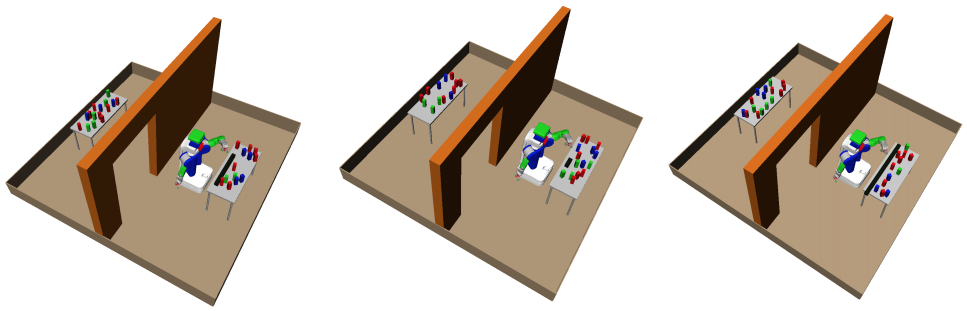


Fig. 6: Three problem instances from most complex domain. The robot's initial configuration and the black object's initial pose are fixed across different planning scenes, but other objects's poses and the black object's length vary.

Figure 3c shows the time required by each method to find the first feasible plan. Again, the score-space algorithms significantly outperform the other algorithms andBOX outperforms STATIC. One noticeable difference between this domain and the previous two is that an ineffective solution constraint takes a long time to evaluate, because computing a path plan or IK solution for an infeasible . This is evident in performance of RAND and DOO which perform worse than RAWPLANNER as they tend to choose solution constraints that are infeasible and expensive to evaluate.

Figure 4c shows the average solution score as a function of computation time. The graphs show a similar trend as in the previous experiments, with score-space algorithms outperfoming the other algorithms, and BOX performing better than STATIC. The fact that this domain requires a significant amount of time to try an ineffective solution constraint is again evident in DOO's plot, where consecutive dots have a large gap between them. BOX and STATIC are able to avoid this issue by exploiting the score-space information.

These experiments demonstrate that our approach can significantly improve the speed of planning in complex TAMP problems. It is founded on two key ideas: (1) that predicting a few critical aspects of a solution may simultaneously generalize better than predicting entire solutions while still providing substantial computational leverate and (2) that the degree of success of previously-tried solutions is a good representation for the similarity of problem instances which affords useful generalization.

REFERENCES

[1] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *IJRR*, vol. 32, no. 9-10, 2013.

[2] T. Lozano-Pérez and L. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," *IROS*, 2014.

[3] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," *ICRA*, 2014.

[4] R. Munos, "From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning," *Foundations and Trends in Machine Learning*, 2014.

[5] N. Srinivas, A. Krause, S. KaKade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *ICML*, 2010.

[6] R. Munos., "Optimization of deterministic functions without the knowledge of its smoothness." *Advances in Neural Information Processing Systems*, 2011.

[7] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," *ICRA*, 2012.

[8] J. Hodál and J. Dvořák, "Using case-based reasoning for mobile robot path planning," *Journal of Engineering Mechanics*, 2008.

[9] S. Pandya and S. Hutchinson, "A case-based approach to robot motion planning," *IEEE Intl. Conf. on Systems, Man and Cybernetics*, 1992.

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

[11] J. Lien and Y. Lu, "Planning motion in environments with similar obstacles," *RSS*, 2009.

[12] M. Phillips, B. Cohen, S. Chita, and M. Likhachev, "E-graphs: Bootstrapping planning with experience graphs," *RSS*, 2012.

[13] A. Dragan, G. Gordon, and S. S. Srinivasa, "Learning from experience in manipulation planning: Setting the right goals," *ISRR*, 2011.

[14] S. Finney, L. P. Kaelbling, and T. Lozano-Pérez, "Predicting partial paths from planning problem parameters," *RSS*, 2007.

[15] D. Dey, T. Y. Liu, B. Sofman, and J. A. Bagnell, "Efficient optimization of control libraries," *AAAI*, 2012.

[16] D. Dey, T. Y. Liu, B. Sofman, M. Hebert, and J. A. Bagnell, "Contextual sequence prediction via submodular function optimization," *RSS*, 2012.

[17] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, CMU Robotics Institute, August 2010.

[18] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *IJRR*, 2014.