Learning Efficient Constraint Graph Sampling
for Robotic Sequential Manipulation

Joaquim Ortiz-Haro[1,*], Valentin N. Hartmann[1,3], Ozgur S. Oguz[1,2], Marc Toussaint[2,3]

Abstract— Efficient sampling from constraint manifolds, and thereby generating a diverse set of solutions for feasibility problems, is a fundamental challenge. We consider the case where a problem is factored, that is, the underlying nonlinear program is decomposed into differentiable equality and inequality constraints, each of which depends only on some variables. Such problems are at the core of efficient and robust sequential robot manipulation planning. Naive sequential conditional sampling of individual variables, as well as fully joint sampling of all variables at once (e.g., leveraging optimization methods), can be highly inefficient and non-robust. We propose a novel framework to learn how to break the overall problem into smaller sequential sampling problems. Specifically, we leverage Monte-Carlo Tree Search to learn assignment orders for the variable-subsets, in order to minimize the computation time to generate feasible full samples. This strategy allows us to efficiently compute a set of diverse valid robot configurations for mode-switches within sequential manipulation tasks, which are waypoints for subsequent trajectory optimization or sampling-based motion planning algorithms. We show that the learning method quickly converges to the best sampling strategy for a given problem, and outperforms user-defined orderings or fully joint optimization, while providing a higher sample diversity. Video: https://youtu.be/mCNdvjTbHNI

I. INTRODUCTION

Highly constrained problems are at the core of a wide variety of different applications (e.g. scheduling [1], hardware verification [2], or robotic manipulation planning [3]). To robustly tackle such problems, they are represented as constrained optimization or satisfaction problems, and one aims to find a diverse set of feasible solutions.

Focusing on problems in continuous domains, there are two main approaches to generate solutions: The first is to use a nonlinear solver to generate a joint assignment for all variables simultaneously. If the problem is highly nonlinear, however, joint optimization is sensitive to the initial guess, and is prone to converging to an infeasible point, or repeatedly to the same local optima. The second direction is to decompose the problem using sequential assignment of subsets of variables. However, this approach is often impossible due to joint constraints on variables. Defining dedicated constraint manifold sampling operations and a sampling order to address this issue requires in-depth

*joaquim.ortiz@ipvs.uni-stuttgart.de
[1]Machine Learning & Robotics Lab, University of Stuttgart, Germany
[2]Max Planck Institute for Intelligent Systems, Germany
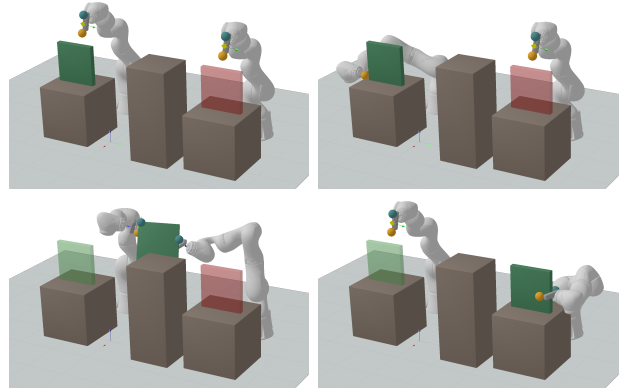[3]Learning and Intelligent Systems Lab, TU Berlin, Germany

Fig. 1: A sequence of mode-switches in the *Handover* problem. Goal: Move the green box to the target location (red). Robot A picks the box (top-right) and hands it to robot B (bottom-left) to place it in the goal position (bottom-right).

knowledge of the problem domain. While defining good decompositions is feasible for simple or structured settings, in general, they are suboptimal for complex scenarios.

We demonstrate our work on problems arising in robotic sequential manipulation, which requires finding feasible solutions that satisfy the kinematic and dynamic constraints imposed by the robot and the environment, possibly arising from future actions and goals. Such problems implicitly encode discrete structures due to the stable interaction modes that describe the contact- or kinematic switches. These mode-switches are discrete snapshots of the configuration variables when the constraint activity changes, e.g., when the robot picks up an object (Fig. 1). Generating a diverse set of feasible mode-switches, and then optimizing the trajectories while considering the whole set can alleviate infeasibility and suboptimality issues that occur when only using one possible mode-switching sequence.

As example, consider the mode-switch configurations of a pick-and-place problem, defined with three sets of variables: the robot configurations (joint values) when picking and placing the object, and the relative transformation between the gripper and the object. Some possible sampling sequences to generate a full sample are: (i) optimize all variables jointly, (ii) sample the relative transformation first, and then compute the robot configurations, or (iii) compute the pick configuration and relative transformation first, and then the place configuration.

In this work, we present an algorithm to efficiently generate a diverse set of solutions for high-dimensional

nonlinear programs containing many infeasible local optima by exploiting its factored structure. Our method relies on the factored graph representation of the task to reason online about the best sequence of constrained conditional sampling operations.

Thus, our approach learns a decomposition based on the given microscopic factorization, optimizing for the success rate and computation time of sampling. This maximizes the number of solutions that fulfill the constraints in a fixed computational time.

Our main contributions are:

- A new framework to model constrained conditional sampling in constraint graphs and computation operations as a stochastic decision process,
- Pruning-strategies that leverage the factored constraint graph representation of the nonlinear program to reduce the space of possible computations, and
- An algorithm based on Monte-Carlo Tree Search to find sequences of computational operations that maximize the number of generated solutions in a fixed computational time.

II. RELATED WORK

A. Problem Decomposition and Constraint Graphs

In continuous optimization, decompositions using the underlying problem structure are used to improve the performance of the algorithms both from a theoretical and practical perspective [4]–[7]. Examples in robotic applications demonstrate attempts to decompose complex nonlinear problems in a sequence of easier subproblems, where the solution of the simpler problems is used to guide optimization and sampling methods towards the solution of the full problem [8]–[11].

In constraint satisfaction problems (CSP), constraint graphs are often used to reason about the problem structure, and to decompose problems into simpler ones. Algorithms solving CSPs make use of the graph structure, e.g. identifying connected components or trees, to efficiently assign variables [12]. A general overview on CSPs can be found in [13]. There is a rising interest for sampling solutions satisfying CSPs [2], [14], [15]. Generating multiple solutions to the problem at hand improves the applicability of these methods in settings where some constraints can not be modeled or only evaluated a posteriori [16].

However, to the best of our knowledge, the current literature related to the problem of sampling from a CSP focuses mostly on CSPs with finite, discrete domains.

In this work, we use the problem structure of the continuous constraint satisfaction problem in the form of a constraint graph to reduce the set of possible computation sequences. We reason about the best way to generate *multiple diverse* samples, instead of only generating a *single solution* to a constraint satisfaction problem.

B. Meta-Decision Processes

The optimization over computational operations relates directly to optimal meta-decision-making [17]. Meta-decision-making has various applications in heuristic search [18]–[21],

and other resource bound planning processes [22], [23]. More recently, meta-reasoning has been applied in reinforcement learning [24], or in temporal planning [25] amongst others. The robotics community is applying similar ideas in their domain, namely in path planning. In [26], deciding which edge to evaluate next in a lazy motion planner is explicitly regarded as decision problem.

We use bandit algorithms [27], [28] to reason about the sequence of computational operations to generate solutions to a nonlinear program. The decisions we take are the discrete choices of which variables should be computed and assigned next.

C. Robotic Sequential Manipulation

Since we demonstrate our contributions on sequential manipulation problems, we briefly give an overview over the approaches taken to solve such problems. Task and motion planning (TAMP) is concerned with finding action sequences, and corresponding motion plans to fulfil some symbolic and/or geometric goal [29]–[31].

A comprehensive review on the history and current developments of TAMP can be found in [32]. In the following, we highlight some work that is specifically relevant for us: Logic geometric programming [3], [9], [33], [34] solves several optimization problems, of which one tries to find the mode switches for a previously determined action sequence. In [35], the full TAMP problem is regarded as a factored transition graph, which is then sampled to find solutions to the whole problem. Multi-modal motion planning [36], [37] extends motion planning to solve manipulation planning [38], [39] by incorporating mode-switches as the intersection of overlapping manifolds.

Generally, current TAMP approaches assume that mode-switches can be efficiently generated with either joint optimization or user defined sampling sequences. In our work, we focus on problems where this assumption does not hold due to obstacles, nonlinear constraints, and strong dependencies between variables, and present a method to efficiently sample the solution manifold for such cases.

Our method can be directly integrated as an inner module in TAMP approaches, enabling them to solve more challenging and constrained problems. In fact, in cluttered scenarios such as a construction site, generating the mode-switches becomes one of the computational bottlenecks of the whole TAMP pipeline [40].

III. CONSTRAINT GRAPH SAMPLING

A. Constraint Sampling Program Formulation

Consider a nonlinear program (NLP) without cost term in its general form,

$$\text{find } x \in \mathcal{X} \text{ s.t. } h_{\text{ineq}}(x) \leq 0, \ h_{\text{eq}}(x) = 0 \qquad (1)$$

where x are variables in some continuous space $\mathcal{X}$ of dimensionality k (e.g., the vector space $\mathbb{R}^k$), $h_{\text{eq}} : \mathcal{X} \to \mathbb{R}^{m_{\text{eq}}}$ and $h_{\text{ineq}} : \mathcal{X} \to \mathbb{R}^{m_{\text{ineq}}}$ are differentiable vector functions representing the constraints.
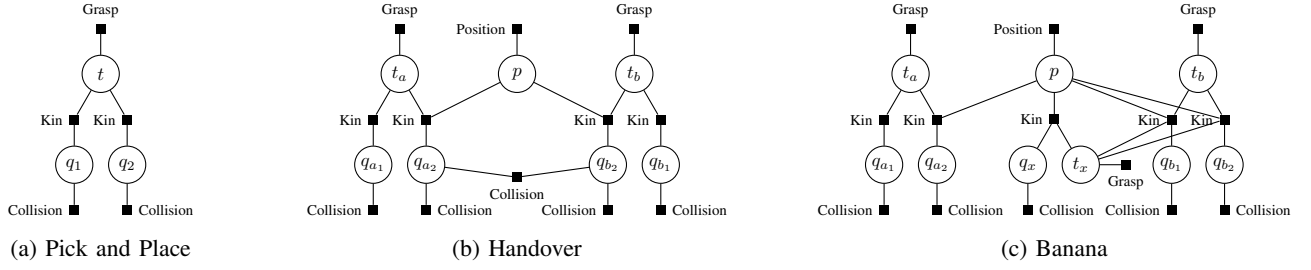
Fig. 2: Constraint graphs in manipulation planning, *Pick-and-place* is used to illustrate how the space of possible computations can be efficiently pruned. We also used two complex manipulation scenarios, *handover* and *banana*, to evaluate our method.

Let $\mathcal{M} = \{x \in \mathcal{X} : h_{\text{ineq}}(x) \leq 0, h_{\text{eq}}(x) = 0\}$ be the feasible set of (1): a nonlinear and possibly disconnected manifold. Our goal is to generate a diverse set of discrete samples $\{x^{(i)} : x^{(i)} \in \mathcal{M}\}$ that is an ϵ-cover. An ϵ-cover is a finite set of samples $\{x^{(i)} : x^{(i)} \in \mathcal{M}\}$ s.t. $\max_{x \in \mathcal{M}} \min_{x^{(i)}} \|x - x^{(i)}\| < \epsilon$.

In general, a solution $x^{(i)}$ can not be generated with a closed-form sampling expression. Thus, the standard approach to generate a solution is to sample on the ambient space $x_s \in \mathcal{X}$ and project it to $\mathcal{M}$ with a local optimizer.

B. Constraint Graphs

We use the knowledge about the problem structure (variables, constraints and their dependencies) to transform the general formulation of the NLP into a constraint graph: We assume the space $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ (and thus $x = \{x_1, ..., x_n\}$) can be factored into n variables.

Similarly, the constraints $h_{\text{eq}}(x), h_{\text{ineq}}(x)$ are factored into l sets of constraints, $\{h_1, h_2, \ldots, h_l\}$ with $h_j : \mathbb{R}^{a_j} \to \mathbb{R}^{b_j}$ where $a_j, b_j \in \mathbb{N}$ are the dimensions of the domain and co-domain of the constraint h_j, and each constraint h_j depends only on a subset of variables. Such factorizations naturally arise in many applications, where each variable has some semantic and geometric meaning.

Using this factorization, the problem can be represented graphically as a constraint graph [13], which are a special instance of factor graphs [41].

We show examples of such constraint graphs for robotics manipulation in Fig. 2. The problems involve robot configurations q, object positions p and relative transformations t between the robotic gripper and the object. These variables are constrained by the kinematic chain of the robot (*Kin*), collision avoidance (*Collision*), grasping (*Grasp*), and position constraints (*Position*). By combining these basic blocks, we can model complex tasks such as handovers or sequential manipulation with intricate dependencies.

Our approach can readily also handle cost objectives in the original NLP (1), at which point we would more naturally speak of factor graphs. Note that, while cost objectives can be included, from here on, we focus on feasibility problems.

A key observation is that constraint graphs are typically sparse, with few edges (i.e., each constraint is only applied to a limited number of variables). This property will be exploited in the next section to design efficient algorithms.

IV. SEQUENTIAL SAMPLING IN CONSTRAINT GRAPHS AS A MARKOV DECISION PROCESS

We first observe that a complete assignment of the variables $x = \{x_1, \ldots, x_n\}$ can be computed in different ways: Variables could be assigned jointly, one by one, or following a sequence of blocks of different sizes. We define the computational state $s \in P(S)$ to refer to the indices of x that have been assigned, thus S denotes the set of indices $\{1, ..., n\}$, with P being the powerset-operator. Hence, any full assignment for x can be computed following a sequence of assignments $H = (s_0, s_1, \ldots, s_g)$ where $s_i \subset s_{i+1}$, and $s_0 = \{\}, s_g = S$. We use $H_i = (s_0, ..., s_i)$ to denote a partial sequence up until the i-th assignment.

A transition $s_i \to s_j$ in a sequence implies a computational operation $x_{s_j} = o_{s_i, s_j}(x_{s_i})$ that assigns numerical values to the new variables, given x_{s_i}, which is kept fixed. We call a transition *valid*, if $s_i \subset s_j$. Thus, $\mathcal{Y}(s_i) = \{s_j : s_i \subset s_j\}$ denotes the states that can be reached with a valid transition from s_i. Generally, the transition operations are implemented either with direct conditional sampling or using optimization methods initialized with a randomized guess. For instance, in the pick-and-place problem presented in the introduction, examples of such transitions are generating 6-DOF grasps or solving inverse kinematics (i.e. computing robot joint values for a given end-effector position).

Algorithm 1 shows how to generate solutions from a constraint graph by choosing a valid transition and associated operation at each step, and therefore fixing the sequence of assignments. The performance of the algorithm depends heavily on how transitions are selected (line 8 – our method described in Section V-A is in orange).

Each transition $s_i \to s_j$ is a conditional sampling operation, which generates samples with an unknown but defined conditional probability density $p_{s_j|s_i}(x_{s_j}|x_{s_i})$, defined on the feasible manifold of x_{s_j}. There are no assumptions on the shape of the distribution. We assume that the probability density is zero everywhere in case the transition conditioned on x_{s_i} is not feasible. The joint probability density $p_{H_j}(x_{s_j}) = p_{s_j|s_i}(x_{s_j}|x_{s_i})p_{H_i}(x_{s_i})$ thus depends on the history $H_j = (s_0, .., s_i, s_j)$ of transitions to reach s_j.

A valid transition $s_i \to s_j$ between computational states does not always produce an assignment that satisfies the associated constraints for x_{s_j}. There are two sources of infeasibility: *(i)* the problem conditioned on the previously

assigned variable x_{s_i} could be infeasible, or *(ii)* the sampling operation could fail to generate a solution even if one exists.

We can define an ad hoc success rate estimate for transitions between the current sequence H_i and the next computational state s_{i+1} (equivalently, between H_i and $H_{i+1} = H_i \cup s_{i+1}$) as follows:

$$Pr\left(H_i \rightarrow H_{i+1}\right) = \hat{p}_{H_i,H_{i+1}} \; \frac{\int_{\Omega_{i,i+1}} p_{H_i}(x_{s_i})d\Omega}{\int_{\Omega_i} p_{H_i}(x_{s_i})d\Omega}, \quad (2)$$

where Ω_i is the feasible space for x_{s_i} and $\Omega_{i,i+1} \subseteq \Omega_i$ is the feasible space for x_{s_i} for which a joint sample $x_{s_{i+1}}$ exists. In this estimate, the success rate $\hat{p}_{H_i,H_{i+1}} \in (0,1)$ represents the probability that a solution satisfying the constraints will be found if one exists, which we estimate as constant for a given transition, rather than depending on the previous assignment x_{s_i}. Note that the success rate could be low for certain transitions, while it could be close to 1 for others.

A. Computational State Transition as a Markov Decision Process

We can now define a Markov Decision Process (MDP): The state space is $\{H_i\} \cup \{\neg\}$, i.e., the sets of partial sequences starting from s_0 plus an infeasible state $\neg$. The action space contains the possible transitions between sequences, i.e., adding a new computational state. Each action has a success rate $p_a = Pr(H \rightarrow H')$ of reaching the new sequence as defined in (2), (i.e. the probability of generating an assignment satisfying the constraints) and probability $1 - p_a$ of going to the infeasible state $\neg$. We note that the MDP has a maximum horizon length of n transitions, i.e., the size of the variable set, which is equivalent to assigning all the variables one by one.

We use the reward structure R to obtain a concept of optimality that directly relates to our objective of maximizing the number of samples: We introduce a reward $r_g = 1$ for reaching the goal, i.e., reaching S, and a stochastic cost, i.e., a negative reward, r_t on each transition, which is the time the transition takes. The cost r_t is not only added at the final node of the sequence, but directly at the intermediate nodes. We weigh the two costs linearly, obtaining λr_t and $(1-\lambda)r_g$ with $\lambda \in (0,1)$.

V. Choosing Computational operations with Monte-Carlo Tree Search

We use Monte-Carlo Tree Search (MCTS, [42]) on the previously defined MDP to find the optimal sequence of computational operations. MCTS incrementally builds a tree of possible transition sequences to find the most promising. This is achieved by randomly choosing transitions starting from the initial state s_0 of the MDP, executing its corresponding conditional sampling operation and weighting the generated sequences using the obtained reward. Thus, in our setting, the nodes of the tree built by the MCTS are sequences of computational states.

The reward structure of the MDP guides the tree search algorithm to choose transitions that have a low computational cost and high probability of producing a full sample. In

Algorithm 1 Framework to generate solutions from a constraint graph, with our contributions in orange.

1: **Input:** Constraint graph, time budget T.
2: $L = \{\}$ (empty list of samples)
3: **while** current time $t \leq T$ **do**
4:     feasible = True, $x = \{\}$ (empty sample)
5:     $s = s_0$ (initial empty computation state)
6:     **while** $s \neq S$ **and** feasible **do**
7:         $s' \leftarrow$ *Choose next state from* $\mathcal{Y}(s)$ using UCT
8:         $x_{s'}$, feasible, reward $\leftarrow o_{s,s'}(x_s)$
9:         *Update UCT tree with* reward
10:         $s \leftarrow s'$
11:     **if** feasible **then**
12:         append x to L
13: **Output:** List of valid samples L

TABLE I: Number of transitions $s_i \rightarrow s_j$ in three scenarios of manipulation planning, before and after transition pruning.

	Variables	Transitions Complete	Transitions after Pruning	Pruning Ratio [%]
Pick-Place	3	19	8	57.89
Handover	7	2059	163	92.09
Banana	9	19171	534	97.21

practice, under the assumption that λ is chosen well[1], this means that the number of generated samples is maximized.

A. Upper Confidence Tree (UCT)

In particular, we use the UCT algorithm [28] to balance the exploitation of known sequences with the exploration of new sequences. UCT expands the child node i that maximizes

$$Q_i + c\sqrt{\frac{\ln N_i}{n_i}}, \quad (3)$$

where Q_i is the current estimate of the expected reward of node i, N_i is the number of simulations that evaluated the parent node, n_i is the number of rollouts that evaluated node i, and c is a constant chosen by the user. The estimate of the Q function of the nodes of the trees converges as the number of rollouts increases. Hence, the UCT algorithm incrementally builds a search tree, estimates the Q values of each transition, and chooses actions using the upper confidence bound (3), leading to Algorithm 1.

We note that, if at least one of the sampling sequences assigns non zero probability density to the whole feasible manifold, so does our algorithm, since UCT never stops exploring all possible sequences.

B. Pruning the Sampling Tree using the Constraint Graph

Since the number of sequences and transitions grows exponentially with the dimension of the problem (see Table I), it is not possible to naively apply MCTS on the previously defined MDP. However, most real world problems have a

---

[1] An estimate of the average computational cost $\hat{c} = -\sum r_t$ can be used as a guidance for choosing a suitable $\lambda = \frac{1}{\hat{c}+1}$.

sparse structure that can be leveraged to significantly reduce the set of available sequences and transitions.

We provide two examples for the transitions that can be pruned: Consider a pick and place problem represented with the variables $\{q_1, q_2, t\}$, where q_1, q_2 are the robot configurations when picking and placing, and t is the relative transformation between the end-effector and the object (also see the contraint graph in Fig. 2a).

- We prune $q_1 \rightarrow (q_1, t)$ and $q_2 \rightarrow (q_2, t)$. If q_1 or q_2 is sampled independently at random, the probability to grasp the object correctly is zero.
- We prune $t \rightarrow (q_1, q_2)$ because q_1 and q_2 are independent if t is fixed, and thus it does not make sense to generate q_1 and q_2 jointly.

Thus, we combine knowledge of the constraint graph structure and the local information in each variable and constraint, i.e., the number of available degrees of freedom, and the number of equality constraints, to prune a transition $s_i \rightarrow s_j$ if it fulfills any of the two criteria:

a) Zero probability of success: In general, the probability of sampling variables from the ambient space such that equality constraints are fulfilled, is zero. Hence, a transition can be pruned if the number of new, linearly independent, equality constraints exceeds the number of new degrees of freedom that are added in that transition.

b) Equivalence under conditional independence: Variables in the constraint graph are conditionally independent if all paths that connect them with each other contain already assigned variables. Given the partial assignment of a computational state s_i, we delete a transition that jointly samples variables that are conditionally independent with respect to the ones assigned in s_i. In such a case, joint sampling means choosing an ordering of two conditionally independent processes, which is already represented in the computation graph

We show how the number of transitions is reduced in three example scenarios in robotics manipulation in Table I.

C. Problem Families and Tree Warmstart

Obstacle and object configurations have an impact on the computational time and the success rate of the conditional sampling operations, potentially changing the optimal sampling sequence. Thus, the best sampling order depends on both the constraint graph and the concrete problem instance.

MCTS provides a good framework to include the information gathered on solutions of similar problems as a warmstart.

We propose to warmstart the tree search by initializing Q_i of each node with the average of the Q values of the previous problems, and set n_i to an *equivalent count visit* n_{equiv}, that models how confident we are with the warmstart, following the approach presented in [43].

VI. EXPERIMENTAL RESULTS

A. Scenarios

We evaluate our approach on two complex sequential manipulation problems that pose different challenges in terms of nonlinearity and sparsity:
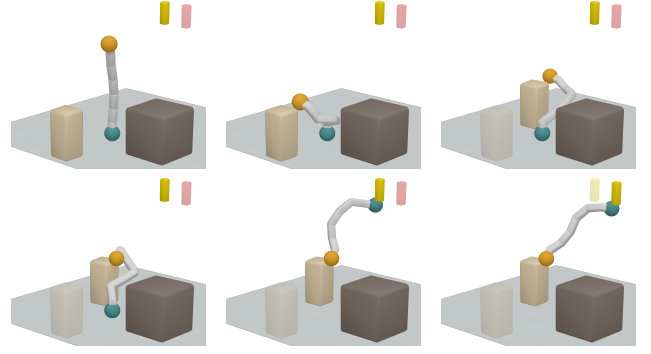

Fig. 3: One possible sequence of mode switches in the *banana* problem. The dark brown object is an obstacle.

TABLE II: Sampling sequences in *Banana* and *Handover* scenarios. Sampling order is left to right, each tuple (.) denotes joint sampling of the subset.

Expert 1	Handover	$(q_{a_1}, q_{a_2}, t_a, q_{b_1}, q_{b_2}, t_b, p)$
Joint Opt.	Banana	$(q_{a_1}, q_{a_2}, t_a, p, q_x, t_x, q_{b_1}, q_{b_2}, t_b)$
Expert 2	Handover	$p, t_a, t_b, q_{a_1}, q_{a_2}, q_{b_2}, q_{b_1}$
One-by-one	Banana	$p, t_a, t_b, t_x, q_{a_1}, q_{a_2}, q_x, q_{b_1}, q_{b_2}$
Expert 3	Handover	$(q_{a_1}, t_a), (q_{a_2}, p, q_{b_2}, t_b), (q_{b_1})$
Sequential	Banana	$(q_{a_1}, t_a), (q_{a_2}, p), (q_x, t_x), (q_{b_1}, t_b), q_{b_2}$
Mode-switches		
Tree:	Handover	$t_b, t_a, q_{a_1}, q_{b_1}, (q_{a_2}, p), q_{b_2}$
Best found		$(q_{b_1}, t_b), (q_{b_2}, p), (q_{a_2}, t_a), q_{a_1}$
Sequence	Banana	$(q_{a_1}, t_a), (q_{a_2}, p), (q_{b_1}, q_{b_2}, t_b, t_x), q_x$
(Examples)		$(p, q_{b_2}, t_b, t_x), q_x, (q_{a_1}, t_a), q_{a_2}, q_{b_1}$

- *Handover*: Two robots (7D) have to collaborate to place a box in the goal configuration. Grasping is modeled with a two-finger gripper. Figure 1 shows a possible solution and Fig. 2b shows the constraint graph.
- *Banana:* The hanging 'banana' has to be moved to a goal position. The robot (8D) can interact with the world using both sides of the kinematic chain as gripper. The box has to be moved, and used as a tool to stand on, to be able to reach the goal. Grasping is modeled as 'grasp by touch'. Fig. 3 shows snapshots from a solution sequence and Fig. 2c shows the constraint graph.

For both problems, 8 different instances[2] (i.e., varying obstacle and object configurations) are evaluated[3] multiple times, and if not stated otherwise, the average results are reported in the following analyses. The different instances can have a big impact on the computation times, and thus the optimal sampling sequence.

B. Computation operations

One of the motivations of this work is that both the computation time and the success rate of conditional sampling operations must be considered to design efficient algorithms. As an illustrative example, we report the average computational time and success rate of some sampling operations (which can return infeasible results) in the first instance

---

[2]For the problem instances, please check the accompanying video.

[3]Experiments are run using a Intel(R) Core(TM) i5-4200U 1.60GHz CPU.

(a) *Handover*: Sample rate   (b) *Banana*: Sample rate   (c) *Handover*: Coverage   (d) *Banana*: Coverage
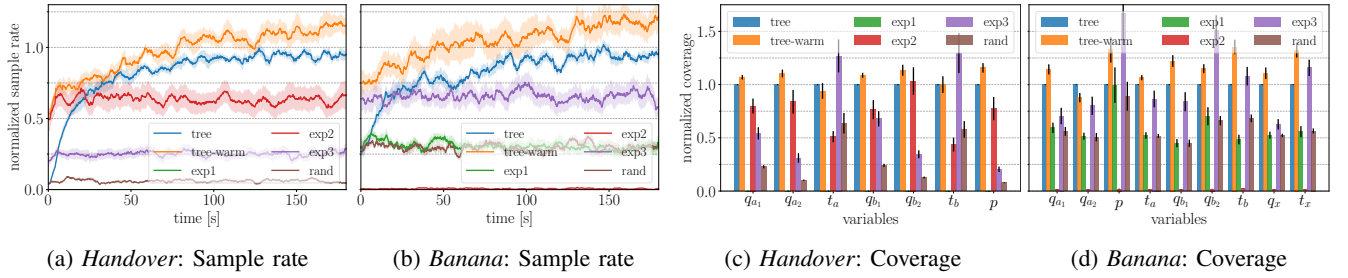
Fig. 4: Sampling rates and approximated projected coverages (normalized by *tree*), averaged over all problems instances.

of the *handover* scenario: Relative Transformation (0.68ms, 100%), Inverse Kinematics (1.55ms, 87%), Grasp of fixed object (5.8ms, 55%), Pick and Place (54ms, 46%), Handover mode-switch (68ms, 44%).

In practice, our algorithm finds the optimal balance between *(i)* choosing simple operations which are fast, and have a high success rate, but can potentially induce infeasibility in future assignments and *(ii)* optimizing variables jointly, which considers joint feasibility but is slower and often has a lower success rate.

The conditional sampling operations are implemented by randomizing the initial guess of a nonlinear solver without cost term, and solved using KOMO [44]. The initial guess covers the whole ambient space of the current partial assignment such that all possible partial solutions have a non-zero probability (see Section IV).

C. Number of Samples and Approximate Coverage

We compare the number of samples generated by our algorithms (*tree* and *tree-warm*), against *expert*-sequences (see Table II for the specific sequences) and *Rand*. Expert-sequences are fixed sampling orders that represent different user-defined strategies. *Rand* chooses computational operations at random, without learning.

a) Number of samples: We plot the evolution of the sampling rate (including MCTS overhead for our algorithms) in our two scenarios in Figs. 4a and 4b. Asymptotically, *tree* outperforms all experts, which have a disparate performance across our two scenarios. Due to the exploration of possible sampling sequences, in the beginning the sample rate of *tree* is lower compared to some of the *expert*-sequences. On the other hand, *tree-warm* alleviates this issue due to the warmstart, and improves its sample rate as time advances. We also observe that the *expert*-sequences are more sensitive to the different problem instances.

The final sample rates achieved by *tree-warm* on the different instances vary between 7-15 samples/s in *handover* and 5-10 samples/s in *banana*. We show examples for some of the discovered sequences by *tree* in Table II.

b) Approximate Coverage: We evaluate the coverage of the solution manifold achieved by our algorithms *tree* and *tree-warm*, which should achieve good coverage (i.e., a diverse set of samples) by design. Since the coverage of a nonlinear manifold embedded in a high-dimensional space can not be evaluated reliably with a relatively low number

of samples, we evaluate the 'projected coverage' as a proxy-measure: We project the full samples to each variable of the sequence and compare the coverage in each of these subspaces. We discretize these subspaces and count the number occupied cells, i.e., cells that are occupied by multiple samples are only counted once. Although an approximation, it provides useful and interpretable information about which parts of the solution-manifold are covered.

The results shown in Figs. 4c and 4d are normalized by the number of occupied cells in *tree*. While some experts achieve better coverage on a subset of variables, coverage of *tree* and *tree-warm* mostly outperform the others. This confirms our hypothesis that, with correct design and implementation of the sampling operations, maximizing the number of samples is a good heuristic to maximize the coverage.

VII. CONCLUSION

We proposed a framework to reason about optimal decompositions of factored nonlinear programs and demonstrated it on feasibility problems arising in robotics manipulation planning. Our algorithm chooses the computational decisions, i.e., which subset of variables to conditionally sample next, to maximize the number of generated samples in a fixed computational time. We use the method to efficiently generate a diverse set of samples for mode-switches in robotic manipulation, which is essential for guiding trajectory optimization, or to set goal and start configurations for sampling-based motion planning. A limitation is that the algorithm must explore inefficient computations and needs some time to converge to the best sampling order, which could prevent usage when the computational budget is very limited. As demonstrated, this is alleviated by warmstarting the algorithm using information from previous runs.

Our framework would naturally allow to also include cost factors in the NLP. We want to briefly comment on why we chose to neglect cost terms: Our approach is tailored to provide a diverse set of feasible samples that can be used to seed higher-level optimization or path finding problems, in particular, to escape potential infeasible local optima. The diversity and uniform coverage of samples is an essential ingredient for the completeness of such overall solvers. Including a cost term in the NLP would be straight-forward but compromise the diversity of generated samples and make the overall approach again prone to local optima.

References

[1] M. D. Johnston and G. Miller, "Spike: Intelligent scheduling of hubble space telescope observations," *Intelligent Scheduling*, 1994.

[2] R. Dechter, K. Kask, E. Bin, R. Emek, *et al.*, "Generating random solutions for constraint satisfaction problems," in *AAAI*, 2002, pp. 15–21.

[3] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proc. of Robotics: Science and Systems (R:SS 2018)*, 2018.

[4] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations research*, vol. 8, no. 1, pp. 101–111, 1960.

[5] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numer. Math.*, vol. 4, no. 1, p. 238–252, Dec. 1962. [Online]. Available: https://doi.org/10.1007/BF01386316

[6] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[7] D. P. Bertsekas, "Convexification procedures and decomposition methods for nonconvex optimization problems," *Journal of Optimization Theory and Applications*, vol. 29, no. 2, pp. 169–197, 1979.

[8] D. Driess, O. Oguz, and M. Toussaint, "Hierarchical task and motion planning using logic-geometric programming (hlgp)," RSS Workshop on Robust Task and Motion Planning, 2019.

[9] M. Toussaint and M. Lopes, "Multi-bound tree search for logic-geometric programming in cooperative manipulation domains," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2017.

[10] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 586–601, 2018.

[11] A. Orthey, A. Escande, and E. Yoshida, "Quotient-space motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 8089–8096.

[12] M. Mouhoub and B. Jafari, "Heuristic techniques for variable and value ordering in csps," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 457–464.

[13] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[14] V. Gogate and R. Dechter, "A new algorithm for sampling csp solutions uniformly at random," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2006, pp. 711–715.

[15] S. Ermon, C. Gomes, and B. Selman, "Uniform solution sampling using a constraint solver as an oracle," in *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'12. Arlington, Virginia, USA: AUAI Press, 2012, p. 255–264.

[16] E. Danna, M. Fenelon, Z. Gu, and R. Wunderling, "Generating multiple solutions for mixed integer programming problems," in *International Conference on Integer Programming and Combinatorial Optimization*. Springer, 2007, pp. 280–294.

[17] S. Russell and E. Wefald, "Principles of metareasoning," *Artificial intelligence*, vol. 49, no. 1-3, pp. 361–395, 1991.

[18] D. O'Ceallaigh and W. Ruml, "Metareasoning in real-time heuristic search," in *Eighth Annual Symposium on Combinatorial Search*, 2015.

[19] F. Lieder, D. Plunkett, J. B. Hamrick, S. J. Russell, N. Hay, and T. Griffiths, "Algorithm selection by rational metareasoning as a model of human strategy selection," in *Advances in neural information processing systems*, 2014, pp. 2870–2878.

[20] S. Zilberstein, "Metareasoning and bounded rationality," *AAAI Workshop - Technical Report*, 01 2008.

[21] E. Karpas, O. Betzalel, S. E. Shimony, D. Tolpin, and A. Felner, "Rational deployment of multiple heuristics in optimal state-space search," *Artificial Intelligence*, vol. 256, pp. 181–210, 2018.

[22] M. E. Bratman, D. J. Israel, and M. E. Pollack, "Plans and resource-bounded practical reasoning," *Computational intelligence*, vol. 4, no. 3, pp. 349–355, 1988.

[23] M. Boddy and T. L. Dean, *Solving time-dependent planning problems*. Brown University, Department of Computer Science, 1989.

[24] R. Pascanu, Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. Reichert, T. Weber, D. Wierstra, and P. Battaglia, "Learning model-based planning from scratch," *arXiv preprint arXiv:1707.06170*, 2017.

[25] M. Cashmore, A. Coles, B. Cserna, E. Karpas, D. Magazzeni, and W. Ruml, "Temporal planning while the clock ticks," in *ICAPS*, 2018.

[26] A. Mandalika, S. Choudhury, O. Salzman, and S. Srinivasa, "Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, no. 1, 2019, pp. 745–753.

[27] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[28] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[29] I. A. Sucan and L. E. Kavraki, "On the advantages of using task motion multigraphs for efficient mobile manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, CA, 2011, pp. 4621–4626.

[30] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Robotics: Science and Systems*, 2016.

[31] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *The International Journal of Robotics Research*, vol. 33, no. 14, pp. 1726–1747, 2014.

[32] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," 2020.

[33] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," 2020.

[34] D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint, "Learning geometric reasoning and control for long-horizon tasks from visual input," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[35] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Sampling-based methods for factored task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1796–1825, 2018. [Online]. Available: https://doi.org/10.1177/0278364918802962

[36] K. Hauser and J.-C. Latombe, "Multi-modal motion planning in non-expansive spaces," *The International Journal of Robotics Research*, vol. 29, no. 7, pp. 897–915, 2010.

[37] Z. Kingston, A. M. Wells, M. Moll, and L. E. Kavraki, "Informing multi-modal planning with synergistic discrete leads," in *IEEE International Conference on Robotics and Automation*, 2020, pp. 3199–3205.

[38] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[39] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual review of control, robotics, and autonomous systems*, vol. 1, pp. 159–185, 2018.

[40] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges, "Robust task and motion planning for long-horizon architectural construction planning," in *Proc. of the Int. Conf. on Intelligent Robots and Systems*, 2020.

[41] B. J. Frey, F. R. Kschischang, H.-A. Loeliger, and N. Wiberg, "Factor graphs and algorithms," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 35. Citeseer, 1997, pp. 666–680.

[42] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[43] S. Gelly and D. Silver, "Combining online and offline knowledge in uct," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 273–280.

[44] M. Toussaint, "KOMO: Newton methods for k-order Markov constrained motion problems," e-Print arXiv:1407.0414, 2014.