# Guided Sequential Manipulation Planning Using a Hierarchical Policy

Hoai My Van*, Ozgur S. Oguz†‡, Zhehua Zhou* and Marc Toussaint‡§
*Chair of Automatic Control Eng., TU Munich, †Machine Learning & Robotics Lab, Uni. of Stuttgart
‡Max Planck Institute for Intelligent Systems, §Learning and Intelligent Systems Lab, TU Berlin, Germany

***Abstract*—We introduce a hierarchical policy structure that selects high-level actions for effective task and motion planning (TAMP) in sequential manipulation tasks. For such problems, scalability of the methods is a major challenge, due to the combinatorial complexity of possible discrete decisions. To overcome this, we propose to learn an upper-level policy that selects the next manipulation action, and a lower-level policy that decides on the end-effector and objects to be involved in the action given the encoded current state. We demonstrate the generalizability of our approach in various pick-and-place experiments. We further show that the time and space complexity is significantly reduced compared to a state-of-the-art TAMP framework especially for tasks involving many objects.**

## I. Introduction

Task and motion planning (TAMP) combines a symbolic planner, that conducts a logic based search for a sequence of high-level actions, and a geometric planner, that solves for the corresponding motion plan [1, 5, 7, 8, 9]. With this scheme, TAMP methods are able to generalize well over a large variety of tasks. However, as the dimensionality of the configuration space or complexity of the tasks to solve increases, the computation time of finding a feasible solution grows significantly [3, 4, 6, 10].

Recent work proposes a novel TAMP approach, the Logic-Geometric Program (LGP) [9], where a sequence of high-level actions imposes additional constraints on a nonlinear optimization problem, that can be solved for the optimal motion plan effectively. However, many possible high-level plan skeletons may turn out to be infeasible particularly in complex environments [3]. This results in a substantial amount of time being spent on attempting to solve the optimization problem of an infeasible skeleton. In essence, maintaining generalizability while simultaneously improving the scalability remains a major challenge.

Recent studies investigates learning based methods to determine feasibility of selected high-level actions [3, 10]. While the time complexity of the geometric planner is significantly improved, discrete action selection is not addressed. A hierarchical formulation of LGP defines subgoals within a bi-level optimization problem to improve the action selection performance [2]. However, acquiring such subgoals requires task specific abstractions, which might not be trivial for some manipulation problems. Thus, the complexity of the symbolic planner and its integration with a motion planner still remains as the main computational bottleneck especially for long-horizon problems.

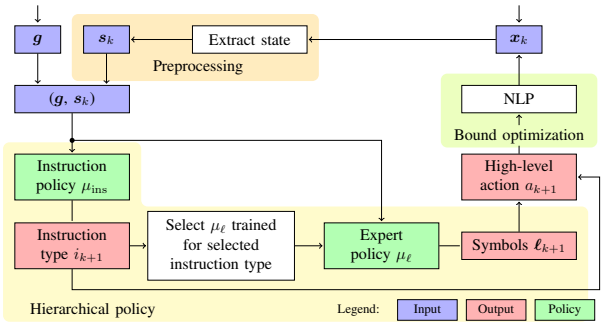



Fig. 1: Flow of the TAMP framework integrated with the hierarchical policy.

In this work, we propose to learn a hierarchical policy and integrate it within an LGP framework in order to improve the scalability of the symbolic planner (Fig. 1). The hierarchical policy serves as a heuristic for high-level action selection in order to guide the symbolic planner, such that a given objective is satisfied within a feasible horizon. The upper-level policy decides on the instruction type, such as grasping, while the lower-level policy, specifically trained for the instruction type, decides on the items involved in this action. We evaluate our approach in several pick-and-place scenarios. However, the proposed architecture is readily expandable to new tasks, by adjusting the hierarchical policy structure accordingly.

Overall, the main contributions of this work are:

- We propose a hierarchical policy that selects discrete decisions for sequential manipulation tasks effectively.
- We integrate this learned hierarchical policy with a non-linear program solver to tackle TAMP problems.
- We show that our approach generalizes to scenarios with unseen goals and outperforms a state-of-the-art TAMP solver for complex problems.

## II. Hierarchical Policy For Action Selection

We assume a configuration space, $\mathcal{X} = \mathbb{R}^n \times SE(3)^m$, consisting of an $n$-dimensional robot and $m$ items. We define a set of symbols $\mathcal{L}$, that contains the logic representations of the end-effectors and the items that can be interacted with. Each symbol has at least one logic type $\tau \in \mathcal{T}$ that determines the actions the symbol can partake in. The state space $\mathcal{S} \subseteq \mathcal{X}$ is the subspace of the configuration space, that contains the world coordinates of all symbols in $\mathcal{L}$.

We assume a set of symbols $\mathcal{L}$, a set of goals $\mathcal{G}$, a set of actions $\mathcal{A}$ and a data set $\{\{((\boldsymbol{g}, \boldsymbol{s}_k), a_{k+1})_n\}_{k=0}^{K(n)-1}\}_{n=1}^{N}$,

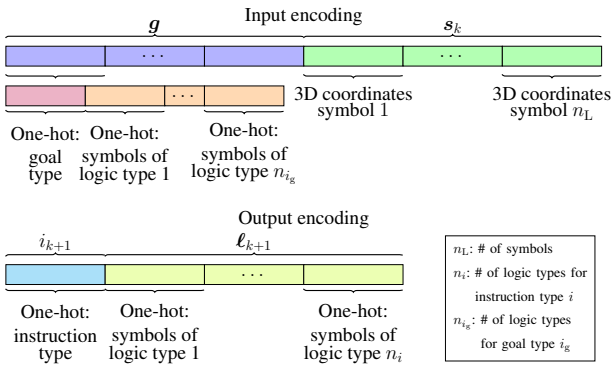



Fig. 2: Input and output encoding.

where $((\boldsymbol{g}, \boldsymbol{s}_k), a_{k+1}) \in (\mathcal{G}^{n_\mathrm{g}} \times \mathcal{S}) \times \mathcal{A}$. Here, $\boldsymbol{g} \in \mathcal{G}^{n_\mathrm{g}}$ is the logic state objective consisting of $n_\mathrm{g}$ goals, $\boldsymbol{s}_k \in \mathcal{S}$ is the state at step $k$ and $a_{k+1} \in \mathcal{A}$ is the action to execute to switch to the next step $k+1$. The data set consists of $N$ problems and $K(n)$ samples for each problem $n$. The aim of this work is to find a mapping $\mu : \mathcal{G}^{n_\mathrm{g}} \times \mathcal{S} \to \mathcal{A}$, such that the standard cross-entropy loss function $\mathrm{L}(\boldsymbol{a}_{k+1}, \mu(\boldsymbol{g}_k, \boldsymbol{s}_k))$ is minimized.

### A. Hierarchical Policy

A high-level action is represented by an instruction type-symbol tuple, i.e., $a_k = (i_k, \boldsymbol{\ell}_k) \in \mathcal{A} \subseteq \mathcal{I} \times \mathcal{L}^{n_i}$, where $i \in \mathcal{I}$ is the instruction type, e.g., grasp, and $\boldsymbol{\ell} \in \mathcal{L}^{n_i}$ are the $n_i$ symbols involved in the action.

Thus, the upper-level policy $\mu_\mathrm{ins} : \mathcal{G}^{n_\mathrm{g}} \times \mathcal{S} \to \mathcal{I}$ selects an instruction type $i_{k+1}$ such as grasp or place,

$$i_{k+1} = \mu_\mathrm{ins}(\boldsymbol{g}, \boldsymbol{s}_k), \tag{1}$$

and the lower-level policy $\mu_\ell : \mathcal{G}^{n_\mathrm{g}} \times \mathcal{S} \to \mathcal{L}^{n_i}$, specifically trained for the selected instruction type, selects the symbols,

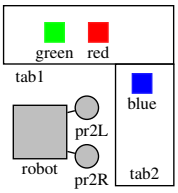
$$\boldsymbol{\ell}_{k+1} = \mu_\ell(\boldsymbol{g}, \boldsymbol{s}_k). \tag{2}$$

The policies are implemented as neural networks for classification. Each class represents a possible instruction type $i \in \mathcal{I}$ for the upper-level policy $\mu_\mathrm{ins}$ or a symbol $\ell \in \mathcal{L}$ for the lower-level policy $\mu_\ell$ respectively. The lower-level policy $\mu_\ell$ further consists of several sub-policies, each dedicated to determine the symbol $\ell_\tau \in \mathcal{L}_\tau \subseteq \mathcal{L}$ of one logic type $\tau$, e.g., the gripper that is used for a grasp. For the lower-level policy, we use a classifier chain. Thus, sub-policies make decisions serially based on previous decisions.

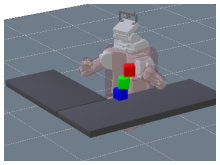
### B. Input And Output Encoding

A goal is represented by a goal type-symbol tuple, i.e., $g_j = (i_{\mathrm{g},j}, \boldsymbol{\ell}_{\mathrm{g},j}) \in \mathcal{G} \subseteq \mathcal{I}_\mathrm{goal} \times \mathcal{L}^{n_{i_\mathrm{g}}}$, where $i_\mathrm{g} \in \mathcal{I}_\mathrm{goal}$ is the goal type, e.g., held, and $\boldsymbol{\ell}_\mathrm{g} \in \mathcal{L}^{n_{i_\mathrm{g}}}$ are the $n_{i_\mathrm{g}}$ symbols associated to the goal type. Zero-padding is applied to shorter goal encodings in order to ensure consistent size for each goal.

The input is the tuple $(\boldsymbol{g}, \boldsymbol{s}_k)$, and the output is the next high-level action $a_{k+1}$ (Fig. 2). The state $\boldsymbol{s}_k$ contains the 3D world coordinates of all $n_\mathrm{L}$ symbols in the environment. The instruction and goal type are one-hot encoded. The associated symbols are one-hot vectors for each logic type.




(a) Schematic illustration.



(b) Final configuration for the objective (on blue red) (on red green).

Fig. 3: Pick-and-place setup: $\mathcal{L}_\mathrm{table} = \{\mathrm{red}, \mathrm{green}, \mathrm{blue}, \mathrm{tab1}, \mathrm{tab2}\}$, $\mathcal{L}_\mathrm{gripper} = \{\mathrm{pr2L}, \mathrm{pr2R}\}$, and $\mathcal{L}_\mathrm{box} = \{\mathrm{red}, \mathrm{green}, \mathrm{blue}\}$.

### C. Motion Planning For Sequential Manipulation Tasks

We integrated the hierarchical policy together with a nonlinear program (NLP) to realize a TAMP framework (Fig. 1). We adapted the NLP solver of [9] and modified its search strategy. First, the state $\boldsymbol{s}_k$ is extracted from the configuration $\boldsymbol{x}_k$. The encoded objective $\boldsymbol{g}$ and state $\boldsymbol{s}_k$ are concatenated to obtain the input for the hierarchical policy. The instruction policy $\mu_\mathrm{ins}$ maps the input to the instruction type $i_{k+1}$. The corresponding expert policy $\mu_\ell$ maps the input to the symbols $\boldsymbol{\ell}_{k+1}$ by using a classifier chain. The instruction type $i_{k+1}$ and the symbols $\boldsymbol{\ell}_{k+1}$ are concatenated to obtain the high-level action $a_{k+1}$.

Given a sequence $\boldsymbol{a}_{1:K} = (a_1, \ldots, a_K) \in \mathcal{A}^K$, we can construct an NLP $\mathcal{P}(\boldsymbol{a}_{1:K})$ to solve for a sequence of robot configurations according to [9]. The NLP can be solved for different levels of detail, i.e., multiple bounds. During sequential action search, we use a coarse bound, which only solves for the key frames of a sequential manipulation task, i.e., the configuration before and after each action. The next configuration $\boldsymbol{x}_{k+1}$ is obtained by solving the NLP (Fig. 1),

$$\boldsymbol{x}_{k+1} = \mathcal{P}_\mathrm{coarse}(\boldsymbol{a}_{1:k+1})|_{k+1}. \tag{3}$$

If a sequence of actions $\boldsymbol{a}_{1:K}$ satisfies the objective, the corresponding optimal motion plan is solved with a dense discretization by a detailed bound,

$$\boldsymbol{x} = \mathcal{P}(\boldsymbol{a}_{1:K}). \tag{4}$$

## III. Experiments & Results

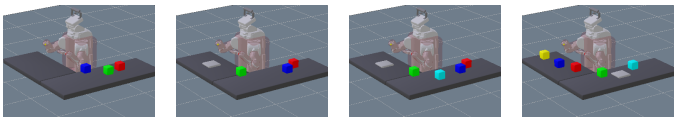
We evaluated the proposed approach in a pick-and-place scenario and verified the effectiveness of the hierarchical policy to find a feasible sequence of high-level actions for a given objective. We defined $\mathcal{T} = \{\mathrm{gripper}, \mathrm{box}, \mathrm{table}\}$, $\mathcal{A} = \{(\mathrm{grasp\ gripper\ box}), (\mathrm{place\ gripper\ object\ table})\}$, $I = \{\mathrm{grasp}, \mathrm{place}\}$, and $\mathcal{G} = \{(\mathrm{held\ box}), (\mathrm{on\ box\ table})\}$. Three neural networks were trained. The upper-level policy $\mu_\mathrm{ins}$ determines the instruction type $i_{k+1}$. The lower-level policies $\mu_\mathrm{grasp}$ and $\mu_\mathrm{place}$ select the symbols $\boldsymbol{\ell}_{k+1}$.

### A. Generalizability

We evaluated the proposed approach for generalizability (Fig. 4). The position of the boxes can vary on the tables. We defined 15 objectives consisting of one goal and 72 objectives consisting of two goals. The policies were trained for 40 objectives consisting of two goals for 51 initial configurations and tested for 20 new configurations for all, including the 47 unseen, objectives.

| | Default | | Objective Adaptation | |
|---|---|---|---|---|
| | All | Unseen | All | Unseen |
| Shortest solution | 91.9% | 90.3% | 98.5% | 98.2% |
| Feasible solution | 8.1% | 9.7% | 1.5% | 1.8% |
| No solution | 0% | 0% | 0% | 0% |
| Reattempts due to | | | | |
| Infeasible solution | 3.2% | 2.2% | 3.4% | 2.5% |
| Maximum depth | 1.1% | 1.6% | 0% | 0% |

TABLE I: Generalizability over all and unseen objectives. The results were averaged over all configurations and objectives. *Default* refers to the proposed approach without any modification. A shortest solution contains the minimal number of high-level actions. Feasible solution is a solution that is not a shortest one. See *Appendix* for additional details.



(a) (b) (c) (d)

Fig. 4: Setups for comparison with various colored boxes and a gray tray.

Our proposed approach always found a solution and the policy is able to generalize over unknown combinations of goals. The shortest solution was found in a majority of cases, 91.9% for all, and 90.3% for unseen objectives (Table I). Replanning was required in a few cases ($< 5\%$), indicating that a solution was usually found at the first attempt. The performance is further improved by adapting the objective $\boldsymbol{g}$ to only consist of the $n_\mathrm{u}$ unsatisfied goals, i.e., $\boldsymbol{g}_\mathrm{ad} \in \mathcal{G}^{n_\mathrm{u}} \subseteq \mathcal{G}^{n_\mathrm{g}}$. With this modification, the shortest solution was found in over 98% of cases (Table I-"Objective Adaptation"). Furthermore, no reattempts due to maximum depth were required.

### B. Scalability

We compared our approach to the original LGP formulation [9] with respect to scalability. The policy trained for the initial setup was used with some modifications due to the network architecture. If there are more than two goals, only a subset of unsatisfied goals is selected, i.e., $\boldsymbol{g} \in \mathcal{G}^2 \subseteq \mathcal{G}^{n_\mathrm{u}}$. If there are more than three boxes, only boxes included in $\boldsymbol{g}$ are selected. If $\boldsymbol{g}$ contains more than three boxes, $\boldsymbol{g}$ is changed to consist of one goal.

The time and space complexity was evaluated for seven problems (Table II) in four different setups (Fig. 4). The goal comprised either to stack boxes, or to place them on the tray or another table. For Prob. b.1, c.1 and d.1 the boxes do not have to be stacked. Thus, for the original LGP approach, the branching factor $b$ was reduced by not classifying the symbols in $\mathcal{L}_\mathrm{box}$ as the logic type $\mathrm{table}$. Additionally, for Prob. b.1 and c.1, $\mathrm{tab1}$ was not classified as the logic type $\mathrm{table}$, as the boxes were never to be placed on it. Note that $b$ was not changed for the proposed approach. We compared computation time and the tree size until the first feasible solution was found. The number of maximum attempts was four and the maximum depth was 20. The symbolic planner was stopped at a maximum tree size of $4 \cdot 10^5$ due to the memory limit.

| Prob. | Target | # of objectives | $K_\mathrm{min}$ |
|---|---|---|---|
| a.1 | Stack boxes on table | 9 | 4 |
| b.1* | Place boxes on tray/tab2 | 8 | 6 |
| b.2 | Stack boxes on tray | 6 | 6 |
| c.1* | Place boxes on tray/tab2 | 6 | 8 |
| c.2 | Stack boxes on tray | 6 | 8 |
| d.1* | Place boxes on tray/other table | 6 | 10 |
| d.2 | Stack boxes on tray | 6 | 10 |

TABLE II: Problems used for comparison of scalability (setups as in Fig. 4). *: Reduced branching factor used for the original LGP formulation [9]. $K_\mathrm{min}$ is the minimum number of high-level actions required to solve the problem.

| | | Original work | | Proposed approach | |
|---|---|---|---|---|---|
| | | mean | std | mean | std |
| Prob. a.1 | Time [s] | 2.1 | 0.8 | 2.7 | 0.9 |
| | Tree size | 400.7 | 127.6 | 26.1 | 6.0 |
| Prob. b.1* | Time [s] | 4.0 | 1.1 | 5.0 | 1.0 |
| | Tree size | 3096.0 | 41.4 | 48.0 | 0.0 |
| Prob. b.2 | Time [s] | 8.55 | 1.0 | 3.6 | 0.2 |
| | Tree size | 43630.3 | 437.5 | 47.0 | 0.0 |
| Prob. c.1* | Time [s] | 112.3 | 8.5 | 12.4 | 3.2 |
| | Tree size | 216682.5 | 1998.3 | 101.0 | 38.0 |
| Prob. c.2 | Time [s] | – | – | 8.5 | 3.3 |
| | Tree size | – | – | 98.8 | 35.8 |
| Prob. d.1* | Time [s] | – | – | 18.0 | 5.6 |
| | Tree size | – | – | 176.7 | 59.9 |
| Prob. d.2 | Time [s] | – | – | 8.3 | 1.1 |
| | Tree size | – | – | 117.7 | 0.5 |

TABLE III: Results for comparison of scalability.

The original work is faster for $K_\mathrm{min} = 4$ (Table III). Due to the computational effort of the hierarchical policy and additional optimization, less nodes can be discovered within the same time. For $K_\mathrm{min} = 6$, both approaches perform similarly, with the original work being faster if $b$ is reduced, and slower if otherwise. For Prob. c.1, the original work requires significantly more time and larger tree size to find a solution, even though $b$ is reduced. Starting from Prob. c.2, the original LGP formulation fails to find a solution as the maximum tree size is reached. In contrast, the hierarchical policy maintains a feasible computation time and tree size over all problems. Note that the computation time for our approach is smaller for the stacking problems as the optimization problem is less complex in these cases. Prob. c.2 required more time than Prob. d.2 due to more reattempts.

## IV. Conclusion

In this work, we propose a hierarchical policy for high-level action selection and integrate it in a TAMP framework. Our approach is able to generalize over unseen objectives. Furthermore, both the time and the space complexity are significantly reduced compared to a state-of-the-art TAMP approach. In future work, the proposed approach has to be evaluated on different tasks. Furthermore, feasibility is currently only determined by path optimization after the symbolic planning is finalized. Thus, an efficient feasibility classifier [3] can be integrated into our proposed high-level action selection in order to realize a more effective TAMP solver.

## REFERENCES


[1] N. Dantam, Z. Kingston, S. Chaudhuri, and L. Kavraki. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*, 03 2018. doi: 10.1177/0278364918761570.

[2] D. Driess, O. S. Oguz, and M. Toussaint. Hierarchical task and motion planning using logic-geometric programming (HLGP). *RSS Workshop on Robust TAMP*, 2019. URL http://dyalab.mines.edu/2019/rss-workshop/driess.pdf.

[3] D. Driess, O. S. Oguz, J. S. Ha, and M. Toussaint. Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[4] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges. Robust task and motion planning for long-horizon architectural construction planning. URL http://arxiv.org/abs/2003.07754.

[5] L. P. Kaelbling and T. Lozano-Perez. Integrated task and motion planning in belief space. *International Journal of Robotics Research*, 32(9-10), 2013. URL http://lis.csail.mit.edu/pubs/tlp/IJRRBelFinal.pdf.

[6] B. Kim, L. P. Kaelbling, and T. Lozano-Perez. Learning to guide task and motion planning using score-space representation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2810–2817, 2017.

[7] 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*, 33(14):1726–1747, 2014. doi: 10.1177/0278364914545811. URL https://doi.org/10.1177/0278364914545811.

[8] T. Lozano-Perez and L. P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014. URL http://lis.csail.mit.edu/pubs/tlpk-iros14.pdf.

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

[10] A. Wells, N. Dantam, A. Shrivastava, and L. Kavraki. Learning feasibility for task and motion planning in tabletop environments. *IEEE Robotics and Automation Letters*, PP:1–1, 01 2019. doi: 10.1109/LRA.2019.2894861.


## APPENDIX

The experiments were conducted on Ubuntu 18.04 on a 16GB RAM machine. The parameters used for training are displayed in Table IV. The upper-level policy as well as each sub-policy of the lower-level policies are represented by a feed-forward neural network with $n_{\mathrm{l}}$ hidden layers with $n_{\mathrm{h}}$ nodes, a dropout rate of $r$ and a L2 regularization of $w$. The policies were trained using early stopping and the Adam optimizer. For each objective, all feasible sequences of high-level actions were used for training. Each sequence provided several samples. Each sample consisted of an input $(\boldsymbol{g}, \boldsymbol{s}_k)$ and the consecutive high-level action $a_{k+1}$. Samples with objectives with two goals were duplicated for each sequence of goals. Objectives with one goal are implemented as the same goal twice. From all training samples, 20% were randomly selected as validation data.

The training and testing sets for the experiments in subsection III-A are detailed in Table V. A reattempt occured up to 3 times when either a feasible motion path could not be solved by the NLP with the action sequence provided by the hierarchical policy, or the predefined maximum depth of the search tree was reached. Here, a maximum depth of 8 was defined. For the experiments in subsection III-B, each problem was evaluated for one initial configuration and for different objectives, e.g., different orders of a stack. The number of tested objectives is displayed in Table II-"# of objectives".

| | $n_{\mathrm{l}}$ | $n_{\mathrm{h}}$ | $r$ | $w$ | Epochs | Batch size | Learning rate |
|---|---|---|---|---|---|---|---|
| $\mu_{\mathrm{ins}}$ | 2 | 64 | 0.3 | | 200 | | |
| $\mu_{\mathrm{grasp}}$ | 3 | 80 | 0.3 | $1\mathrm{e}{-4}$ | 250 | 128 | $1\mathrm{e}{-3}$ |
| $\mu_{\mathrm{place}}$ | 3 | 80 | 0.1 | | 250 | | |

TABLE IV: Neural network and training parameters.

| | Initial configurations | Objectives: one goal | Objectives: two goals |
|---|---|---|---|
| Training Set | 52 | 0/15 | 40/72 |
| Testing Test | 20 (new) | 15/15 | 72/72 |

TABLE V: Training and testing set for generalizability.