2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Solving Sequential Manipulation Puzzles by Finding Easier Subproblems

Svetlana Levit[1,2], Joaquim Ortiz-Haro[1,3] and Marc Toussaint[1,2]

***Abstract*— We consider a set of challenging sequential manipulation puzzles, where an agent has to interact with multiple movable objects and navigate narrow passages. Such settings are notoriously difficult for Task-and-Motion Planners, as they require interdependent regrasps and solving hard motion planning problems.**

In this paper, we propose to search over sequences of easier pick-and-place subproblems, which can lead to the solution of the manipulation puzzle. Our method combines a heuristic-driven forward search of subproblems with an optimization-based Task-and-Motion Planning solver. To guide the search, we introduce heuristics to generate and prioritize useful subgoals. We evaluate our approach on various manually designed and automatically generated scenes, demonstrating the benefits of auxiliary subproblems in sequential manipulation planning.

I. Introduction

Manipulation planning in the presence of movable objects and an obstructed environment is challenging: the number of possible action sequences grows with the number of objects, leading to a hard combinatorial problem, whereas obstacles and narrow passages make motion planning difficult. Sequential manipulation requires reasoning about what objects to interact with, how to grasp, and where to place them. At the same time, collision constraints with the environment may introduce interdependencies between grasps, placements of objects, and robot trajectories, making planning especially hard.

Consider the puzzle in Fig. 1, where the 2-DoF robot agent (the yellow circle) has to move the blue cube into the red goal area. Due to the narrow entrances the cube can only be retrieved from behind the wall if the agent grasps it from the bottom or the top, which is not possible at the current position. Multiple regrasps are necessary to solve this problem: e.g. move the cube inside the left entrance (1b-1d), regrasp from the bottom (1e), move to the center (1f), regrasp and move to the goal (1g-1h).

As shown on this example, solving such sequential manipulation problems requires regrasping an object at different angles and positions, finding good intermediate placements for the objects, and planning trajectories. Such problems can be formulated as Task and Motion Planning. However, they pose the following challenges to TAMP planners:

This research has been supported by the German Research Foundation (DFG) under Germany's Excellence Strategy – EXC2002/1–390523135 "Science of Intelligence", and the German-Israeli Foundation for Scientific Research (GIF), grant I-1491-407.6/2019.

Correspondence to levit@tu-berlin.de
[1]Learning & Intelligent Systems Lab, TU Berlin, Germany
[2]Science Of Intelligence Cluster of Excellence
[3]Machines in Motion Laboratory, New York University, USA

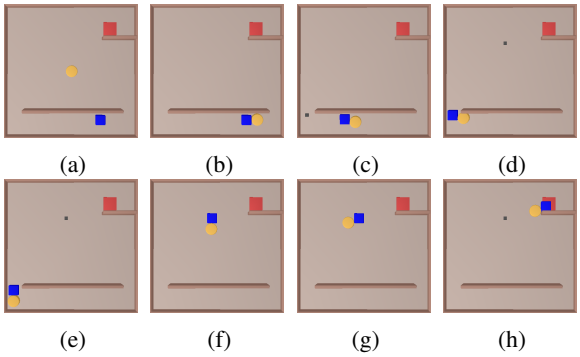


Fig. 1: Solving the Wall pick-and-place puzzle with subgoals. In order to bring the goal-object (blue) to the goal (red square), the agent (yellow circle) moves it and regrasps at several intermediate subgoal locations (gray dots).

- Motion planning with spatial non-convexities is a challenging problem by itself.
- For optimization-based TAMP approaches the non-convexity could lead to local optima.
- Manipulation planning is especially hard for narrow passages, where only very few grasp positions and orientations lead to feasible downstream paths and manipulation.

To address these challenges, we propose to solve the full TAMP problem by searching sequences of auxiliary pick-and-place subproblems, where objects need to be placed at intermediate locations: the subgoal locations. We introduce heuristics to guide the subproblem search and present several methods to generate subgoals for the pick-and-place subproblems. We evaluate their performance and capability to handle scenes with obstacles and narrow passages.

To summarize the contributions of our work, in this paper we present:

- A forward search method for solving sequential manipulation problems, integrating a subproblem search with an optimization-based TAMP solver.
- Two approaches to generate and select subgoal locations that serve as regrasp location candidates in a sequential manipulation problem.
- Heuristics to find configurations feasible for the optimization-based TAMP solver and useful to reach the goal.
- A set of pick-and-place puzzles which we created to evaluate our solver, together with solutions and performance metrics.

II. Related Works

Many challenges of our problem setting are inherent to Task-and-Motion Planning (TAMP) problems due to the dependencies between discrete logical decisions and continuous physical states. Recent TAMP approaches are thoroughly reviewed in [1]. TAMP solvers differ especially in the way physical constraints and motion planning are integrated into the task planning. Sample-based TAMP planners incrementally discretize the continuous configuration space. For instance, PDDL-Stream [2], combines sequential constrained sampling with PDDL Planning, whereas [3] integrates precomputed samples of feasible configurations into task planning.

Using constraint propagation techniques, physical constraints can be incorporated in task planning to allow informed decisions [4]–[6]. Another approach to solve TAMP problems, instead of sampling, is to use optimization methods to resolve dependencies and plan with nonlinear and geometric constraints [7]–[9]. Most TAMP solvers focus on long-term planning for manipulating multiple objects in a static environment with few environment constraints, such as placing objects on big tables, with informative goals or task descriptions. In contrast, our problems require shorter action sequences with very precise manipulation, removing potential obstacles, and navigating narrow passages, without any guidance from the symbolic goal. We extend the optimization-based TAMP solver in [7] with a forward search over smaller subproblems, which can be solved with a sampling based motion planner.

Closely related to TAMP, multi-modal planning defines manipulation planning as a hybrid planning with grasps seen as modes [10]–[13]. To handle manipulation in constrained spaces, [14] explores the idea of probabilistic roadmaps for planning of grasps and placements. However, they only consider a single movable objects and do not include symbolic task planning. Similarly related is rearrangement planning, addressed by [15]–[18], with the important distinction that we do not know the target positions of any additional objects.

The problem settings closest to ours are Manipulation Among Movable Obstacles [19], [20] and Navigation Among Movable Obstacles [21], which can also be considered a special case of TAMP. These approaches often assume *monotone* manipulation cases, where every object is grasped only once. Many cluttered settings, including some of our puzzles, require multi-step interactions and finding intermediate positions for objects, before moving them to the goal.

The concept of subgoals has been extensively discussed in Reinforcement Learning (RL), e.g. [22]–[25]. While these works do not address TAMP, they provide interesting ideas for proposing subgoals. In particular, diverse density analysis [26] is an approach to characterize bottleneck regions, on which our method builds to propose subproblems.

In classical planning, landmarks [27], (related to our definition of subgoals) are used to guide search-based planning [28], [29], but are defined in the discrete logic domain. We draw on such ideas and transfer them to proposing manipulation subproblems for TAMP solving.

III. Problem Formulation and Notation

A. Logic Geometric Programming for TAMP

We adopt an optimization-based formulation of Task-and-Motion Planning, namely as Logic-Geometric Program (LGP) [7] [30]. Given the configuration space $\mathcal{X} = \mathbb{R}^n \times SE(3)^m$ of a n-DoF robot and m movable objects, LGP assumes that feasible paths are structured in phases (or modes), where smooth differentiable constraints describe the feasible motions in each mode and feasible transitions (switches) between modes, akin to multi-modal path planning [10]. A Logic-Geometric Program is a nonlinear mathematical program (NLP) of the form

$$\min_{\substack{K, s_{1:K} \\ x:[0,t_K]\to\mathcal{X}}} \int_0^{t_K} c(\bar{x}(t))\ dt \tag{1a}$$

$$\text{s.t.} \qquad x(0) = x_0, \qquad h_{\text{goal}}(x(t_K)) \le 0, \tag{1b}$$

$$\forall_k : \forall_t \in [t_{k\text{-}1}, t_k] : h_{\text{mode}}(\bar{x}(t), s_k) \le 0, \tag{1c}$$

$$h_{\text{switch}}(\bar{x}(t_k), s_{k\text{-}1}, s_k) \le 0, \tag{1d}$$

$$\forall_k : s_k \in \text{succ}(s_{k\text{-}1}),\ s_K \models \mathfrak{g} \tag{1e}$$

where $\bar{x}(t) = (x(t), \dot{x}(t), \ddot{x}(t))$. The decision variables of this NLP are the number of phases K, the symbolic mode sequence $s_{1:K}$, and the continuous path $x(t)$. The path is constrained to start at x_0 and end in a configuration $x(t_K)$ consistent with the goal constraints h_{goal}. In each phase, the path needs to fulfill mode constraints h_{mode} depending on the symbolic mode s_k, and each $\bar{x}(t_k)$ needs to fulfill mode-switch constraints h_{switch} depending on the symbolic mode transition. The times t_k of mode transitions are assumed to be fixed and equal spaced. Only certain mode transitions are possible logically, here notated via a successor function (usually given as a STRIPS-like task planning domain [31]). Finally, $\mathfrak{g}$ denotes a logical goal. All constraint functions h are assumed smoothly differentiable, and are here written as inequalities – but this is meant to also include equality constraints which an NLP solver handles differently.

B. Problem Formulation

The LGP (1) is a very general formulation for TAMP. In this work, we consider a subclass of problems where the available discrete actions are restricted to pick and place of any object o_m in the scene using a stable grasp and placement on the ground. In all problems, the goal $\mathfrak{g}$ is to put the goal object $o_{goal} \in \mathcal{O}$ on the red goal surface $goal$. We define this problem as $\mathcal{P}(x_0, \mathfrak{g})$ for initial configuration $x_0 \in \mathcal{X}$.

We consider sequential manipulation problems requiring multiple regrasps, moving away objects and moving objects through tight spaces. However, $\mathfrak{g}$ does not contain any information to plan these manipulations, grasps and movements. Our approach is to find and solve auxiliary subproblems, which will guide our planner over a sequence of subgoals to a configuration x_{feas} from which $\mathfrak{g}$ can be solved.

C. Pick-and-Place Auxiliary Subproblems

Starting from a configuration $x_i \in \mathcal{X}$, the *pick-and-place auxiliary subproblem* $\tilde{\mathcal{P}}(x_i, g)$ is defined as the task of moving an object $o \in \mathcal{O}$ to a position $z \in SE(3)$ in a single pick-and-place sequence, with g being the object-position tuple (o, z) and defined as a *subgoal*.

$\tilde{\mathcal{P}}$ is an instance of an LGP problem with two phases $K = 2$ and a fixed sequence of mode-transitions: (`pick` o_1), (`place` o_1 `in` z). To pick an object, the robot should be able to touch it, with grasps possible from all angles. Once picked, the object is attached with a rigid joint until it is placed in z. The poses for pick and place are optimized jointly for the following constraints. The goal constraint h_{goal} is a *pose constraint* depending on z. The switch constraints h_{switch} are *touch* and *stable* for `pick`, and *above* and *stable-on* for the `place` action as defined in [30].

The solution of a subproblem $\tilde{\mathcal{P}}(x_i, g)$ is a collision-free path X from $x_i = X(0)$ to a final configuration $x^f = X(T)$ that meets the subgoal g. We use the following notation to indicate feasibility of a subproblem and return the final configuration:

$$x^f = \text{solve } \tilde{\mathcal{P}}(x_i, g), \tag{2}$$

The configuration x^f is a valid configuration that can be used later as a start for the original problem $\mathcal{P}(x^f, \mathfrak{g})$ or subsequent subproblems.

D. Subproblem Sequence

To solve the original problem (1), we aim to find a sequence of subgoals $[g_1, \ldots, g_L]$ such that,

$$x_0^f \equiv x_0, \tag{3}$$

$$\forall_{i \in [1,\ldots,L]} : x_i^f = \text{solve } \tilde{\mathcal{P}}(x_{i-1}^f, g_i), \tag{4}$$

$$x_{\text{goal}} = \text{solve } \mathcal{P}(x_L^f, \mathfrak{g}) \ . \tag{5}$$

The sequence of subproblems is represented as a graph in Fig. 2, which starts at node x_0, and is expanded using new subproblems (e.g. $\tilde{\mathcal{P}}(x_0, g_0)$ and $\tilde{\mathcal{P}}(x_0, g_1)$) or the original problem $\mathcal{P}(x_0, \mathfrak{g})$. The solid edges represent solution paths, leading to new valid configurations. Once $\mathcal{P}$ is solved for $\mathfrak{g}$, we can trace the complete solution from x_0 to x_{goal}.

IV. Solving Multi-Step Manipulation Problems using a Sequence of Subgoals

The main idea of our method is to perform a *forward search over possible pick-and-place subproblems* $\tilde{\mathcal{P}}$ to reach a configuration x_f, for which we can solve the original problem $\mathcal{P}(x_f, \mathfrak{g})$. We cannot derive such subproblems directly from the goal, because the symbolic goal only defines constraints on one object (o_{goal}) and these constraints are only related to its final position. Most importantly, the goal does not provide information on whether we need to move other objects, in what order or over which trajectories. Instead, we introduce heuristics in our search algorithm to come up with helpful and solvable subproblems.

In Alg. 1 we outline the main steps of our approach. At every iteration of the main loop, we select one of the configurations x from the list L using feasibility heuristics in `select_node`. We try to solve the original problem $\mathcal{P}(x, \mathfrak{g})$ with `solve`, which uses Multi-Bound Tree Search (MBTS) [32], which is a solver for LGP. It explores possible sequences of discrete actions using a tree search algorithm with multiple geometric bounds. These bounds allow to quickly test the feasibility of path and mode-switch configurations in (1) for a fixed action skeleton by solving informative relaxations of the trajectory optimization problem. If successful, `solve` returns a path X.

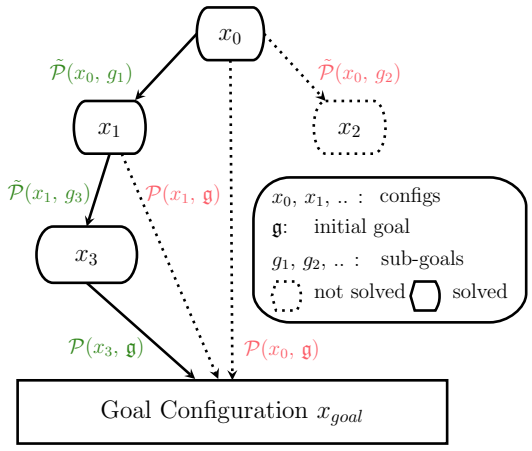


Fig. 2: Subproblem sequence. Solved subproblems and problems in green, unsolved in red.

If $\mathcal{P}(x, \mathfrak{g})$ was not solved, we try to reach configurations from which it may be solvable. We generate subgoals for objects reachable to the agent using `propose_subgoals`, giving us a set of auxiliary pick-and-place subproblems $\tilde{\mathcal{P}}$. In comparison to the original problem $\mathcal{P}$, solving $\tilde{\mathcal{P}}$ does not require a search over different discrete actions. We solve the auxiliary subproblems using a fixed-action-sequence solver `sub_solve`: a combination of nonlinear optimization and sample based motion planning. First, we compute the mode-switch configurations for the grasp and the placement of the object using nonlinear optimization with a randomized initialization. Second, we use bi-directional Rapidly-exploring Random Trees (RRTs) [33], to compute a collision-free path between the mode-switch configurations, similar to [34].

The solutions of $\tilde{\mathcal{P}}$ give us trajectories leading to new valid configurations, which are added to the list L of start configurations to try. The search concludes, if either we solve $\mathcal{P}$ or we do not have any more configurations to expand.

A. Prioritizing configurations with `select_node`

Intuitively, we would like to prioritize configurations from which the LGP-Solver is more likely to solve the problem $\mathcal{P}$. We introduce the score function $s(x) = 10v_o + 5v_g + v_{\text{dist}}$ to select configurations, where

- v_o is 1, if there is a line-of-sight between o_{goal} and the goal, and 0 otherwise.

Algorithm 1 Forward subproblem search

Input: Start configuration x_0, logical goal $\mathfrak{g}$
$L \leftarrow \{x_0\}$ ▷ Initialize list of valid configurations
while L not empty **do**
  $x \leftarrow$ `select_node`(L) ▷ s. IV-A
  $X \leftarrow$ `solve` $\mathcal{P}(x, \mathfrak{g})$ ▷ Try to reach goal
  if X `found` **then**
    Return X_{full} ▷ Trace solution back to x_0
  for $o \in \mathcal{O}$ **do**
    if not `reachable`(x, o) **then**
      Continue
    $Z \leftarrow$ `propose_subgoals`(x, o) ▷ s. IV-B
    for $z \in Z$ **do**
      $g \leftarrow (o, z)$ ▷ New subgoal g
      $X \leftarrow$ `sub_solve` $\tilde{\mathcal{P}}(x, g)$
      if X `found` and not `rej`(L, x^f) **then**
        `Insert`(L, x^f) ▷ add last element of X
Return `None` ▷ No solution found

- v_g is 1, if there is a line-of-sight between o_{goal} and the agent, and 0 otherwise.
- v_{dist} measures proximity d of o_{goal} to the goal, and is set to 5 if $d < 0.2$, 2 if $d < 0.4$, and 0 otherwise.

We only expand a configuration twice, if all other configurations in L have been tried. If every configuration has been tried twice, we stop the search.

B. Generating Subproblems with `propose_subgoals`

First, to decide on the object o, we try to find a collision-free path to the object center with bi-directional RRT to check if the object is reachable to the agent. Hard to reach objects can still be deemed unreachable, as we limit the RRT step budget.

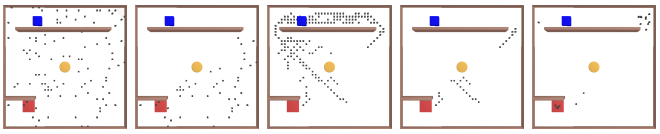
Second, for a given object o we generate a set of subgoal positions Z with the `propose_subgoals` method:

1) Generate position candidates.
2) Filter candidates with `Filter` (see IV-B.1).
3) From the filtered positions sample a small subset (Z).

We evaluated several methods to generate the initial subgoal positions:

- `Rnd`: randomly generated
- `Btl`: using bottleneck analysis
- `Hum`: from human demonstrations

`Btl`: to find optimal subgoal positions we look for *bottlenecks* in the scene, i.e. entrances or narrows passages you need to pass to reach other areas. Similar to *diverse density* analysis in [26], we look for areas with a higher number of possible paths leading through them. First, we rasterize the scene and construct a grid-like graph, with vertices connected only if there is a path between them for the object o. Then we calculate the shortest paths from vertices in a small area around the current object to all other vertices in the graph. We use the number of paths passing through a vertex relative to the number of vertices as a measure of *paths density* of this vertex. Compared to random sampling, the `Btl` method



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

Fig. 3: Subgoals in the scene Wall. a) `Rnd` b) `Rnd` filtered by heuristics. c) `Btl` d) `Btl` after filtering e) `Hum`

is far more likely to propose a position inside a narrow space than inside a larger area, which helps to guide the subgoal generation to the bottlenecks.

1) Filtering subgoal positions with `Filter`: from the set of generated positions, we select the most promising ones with heuristics from IV-A: we calculate the feasibility score of a hypothetical solution configuration, assuming the object o was successfully moved to the subgoal position z. Then we filter the positions based on their scores. Filtering helps to find positions, where o_{goal} would be either close to the *goal* or in line-of-sight of it. A distribution of the subgoal positions before and after filtering is visualized in Fig. 3.

Finally, from the filtered positions we randomly sample a subset of subgoals, adding a trivial subgoal, where the position is equal to the current position of the object.

C. Rejecting similar configurations with `rej`

After a subproblem is solved, we add the last configuration of the solution path to the list L. To avoid repeating problems, we calculate a similarity value for configurations, based on the discretized positions of movable objects, and allow at most two similar configurations to be added.

V. Experiments

We evaluated the solver on pick-and-place puzzles of different complexity, addressing common TAMP challenges. While we formulated our problem in a general way with object poses in $SE(3)$, we consider a robot with 2-DoF to highlight the challenges of interdependencies between grasps and trajectories. Our benchmark problems include:[1]

- 9 manually designed puzzles, challenging for the solver due to multiple obstacles and bottlenecks, Fig. 4.
- 360 PCG (Procedural Content Generation) scenes: automatically generated maze-style puzzles with two objects, allowing us to evaluate how well the approach performs in general settings, Fig. 5.

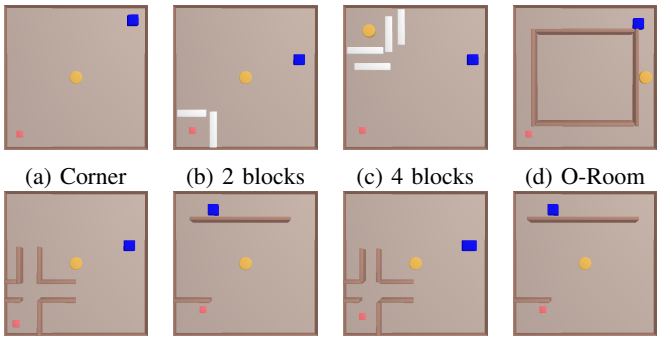
To create the PCG scenes, we used the PCGRL Framework, which can generate solvable gridworld game levels of varying difficulty [35] [36]. We converted the gridworld scenes (Sokoban levels) to continuous 3D TAMP problems.

We measured the following metrics:

1) *Completeness*: solution ratio over multiple runs.
2) *Performance*: solution time.
3) *Success rate*: percentage of solved PCG scenes.

The *performance* and *completeness* experiments are repeated 10 times on 9 manually designed puzzles and 12

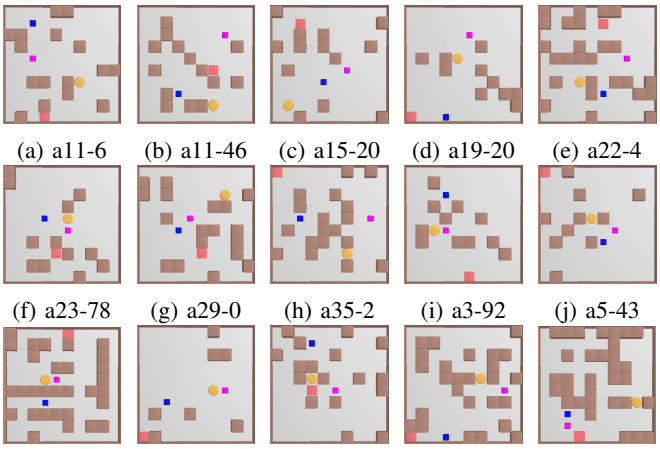
[1]Available at https://github.com/svetlanalevit/manipulation-puzzles.



(a) Corner (b) 2 blocks (c) 4 blocks (d) O-Room

(e) Maze-Easy (f) Wall-Easy (g) Maze (h) Wall

Fig. 4: Puzzle scenes: the agent (yellow circle) needs to bring the goal-object (blue) to the goal (red square). Movable obstacles in white, static walls in brown.

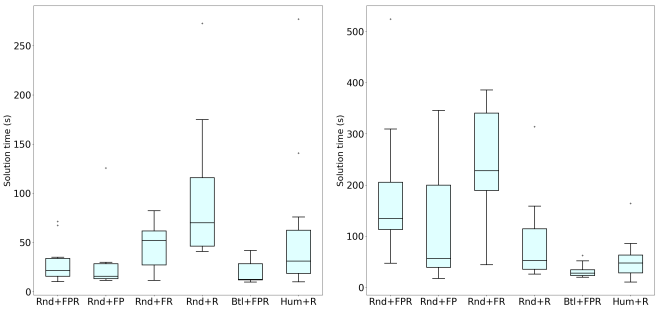


(a) a11-6 (b) a11-46 (c) a15-20 (d) a19-20 (e) a22-4

(f) a23-78 (g) a29-0 (h) a35-2 (i) a3-92 (j) a5-43

(k) a4-4 (l) a19-18 (m) a23-6 (n) a31-4 (o) a4-2

Fig. 5: Proceduraly Generated Scenes (subset) with walls (brown) and two movable objects: the goal-object (blue) and an obstacle-object (pink).



(a) PCG Scene a3-92 (b) Wall

Fig. 6: Runtime comparison over 10 runs (excl. runs without a solution). For `Btl`: the time without subgoal generation.

PCG problems. In Table I we report the solution times, the standard deviation and the solution rate, if it is below 100%. We used a single i7-11700@2.5 GHz CPU for the tests. The *success rate* experiments are repeated once on all PCG problems and we count the number of solved problems.

We compared three methods to generate subgoals (s. IV-B): `Rnd`, `Btl`, and `Hum`. Using random subgoals `Rnd`, we evaluated the importance of the individual components:

- **F** - `Filter`, heuristics based filtering of subgoals during subgoal generation, s. `Filter` in IV-B.1.
- **P** - `Prio`: heuristics based selection of next configurations for node expansion in `select_node`.
- **R** - `Reject`: diversity based rejection of new configurations from solved subproblems.

In total, we evaluated the following combinations of methods (as reported in Tab. I):

1) `Rnd+FPR`: the full method with random subgoals (`Filter`: yes, `Prio`: yes, `Reject`: yes).
2) `Rnd+FP`: without rejection of similar configrations (`Filter`: yes, `Prio`: yes, `Reject`: no).
3) `Rnd+FR`: without heuristics for configuration prioritization (`Filter`: yes, `Prio`:no, `Reject`: yes)
4) `Rnd+R`: without heuristics (`Filter`: no, `Prio`: no, `Reject`: yes).
5) `Btl+FPR`: subgoals from the bottleneck analysis, (`Filter`: yes, `Prio`: yes, `Reject`: yes)
6) `Hum+R`: manual subgoals, without heuristics (`Filter`: no, `Prio`: no, `Reject`: yes)
7) `MBTS`: baseline method, Multi-Bound Tree Search.

VI. Results

Comparison to the MBTS Baseline: Table I summarizes the results of our evaluations. We find that our method (Rnd+FPR) consistently solves the benchmark problems while the baseline solver (MBTS) fails (dashes) at problems with objects hidden behind walls and narrows passages (e.g. Maze, Maze-Easy, O-Room). As our method tries direct MBTS first, the runtimes for solved problems are the same.

We further analyzed the *success rate* over all 360 PCG problems and found 99.7% for Rnd+FPR vs 73.6% for MBTS. The improvement is especially visible for problems where o_{goal} is behind obstacles: in 122 out of 360 PCG scenes the o_{goal} is at least partially occluded from the agent by an obstacle, while the other 238 have no occlusion (direct line-of-sight). With the subgoals search, we can solve 100% of non-occluded and 99.1% of occluded problems. With MBTS 88.24% and 45.08% respectively. The scene a4-2 (Fig. 5o) was not solved, as all objects were marked as unreachable.

Comparison of the subgoal generation methods: Table I also reports the results for the three subgoal generation methods described in IV-B in the columns `Rnd+FPR`, `Btl+FPR` and `Hum+R`. Overall, the `Rnd+FPR` (random subgoals with heuristics) performs better than `Btl` subgoals in terms of solution time and solution rate for most scenes: it shows faster average time in 16 out of 21 scenes, and similar in 4, and has in all but one scenes 100% solution rate. The observed time difference is mostly due to the overhead to calculate the bottleneck subgoals. However, for scenes Wall and Maze, where the subgoals need to be placed in specific small areas, the `Btl` and `Hum` subgoals lead to faster search times (excluding the time to create or generate subgoals), as can be seen in Fig. 6b.

TABLE I: Average solution time over 10 runs (s). `Rnd`: random subgoals, `Btl`: bottleneck subgoals, `Hum`: subgoals from human demonstrations. Ablations: `Rnd+FP`: no rejection of similar configs, `Rnd+FR`: no heuristics to select next config, `Rnd+R`: no heuristics. Solution rate in percent, if below 100% and '-', if not solved. For `Btl+FPR`: in square brackets the time without computing subgoals.

Scene	Rnd+FPR	Rnd+FP	Rnd+FR	Rnd+R	Btl+FPR	Hum+R	MBTS
Cube-Free	0.2±0.01	0.2±0.01	**0.2**±0.01	0.2±0.01	0.2±0.01		0.2±0.01
Corner	0.3±0.01	0.3±0.01	**0.3**±0.01	0.3±0.01	0.3±0.01		0.3±0.01
2 blocks	2.5±0.01	2.6±0.01	2.5±0.01	2.5±0.04	**2.5**±0.01		2.5±0.02
4 blocks	38.4±23.01	35.2±20.57	41.7±15.45	**30.4**±23.22	165.8±81.87[55.8]	176.6±68.29	-
Maze-Easy	**54.0**±10.96	56.0±10.91	84.7±38.45	155.4±86.73	143.9±94.29[85.7](80.0%)	55.2±9.27	-
Wall-Easy	**1.8**±0.16	2.1±0.29	7.8±9.89	9.4±8.73	8.3±2.19[2.4]	3.5±4.45	-
O-Room	68.3±56.10	100.7±80.62	92.9±76.65	254.5±170.08	259.2±128.40[102.6]	**61.1**±30.37	-
Maze	926.2±615.78	1750.7±1075.61(90.0%)	1454.8±1133.81	2867.7±1781.67(40.0%)	1772.5±1319.81[839.4](80.0%)	**154.9**±162.50(80.0%)	-
Wall	190.1±130.60	125.6±111.11	240.0±108.18	91.8±84.86	59.7±25.21[32.6](90.0%)	**55.9**±41.72	-
a11-6	33.4±9.71	**31.6**±4.61	56.3±35.00	123.8±94.97	67.7±21.98[33.1]		-
a15-20	**11.2**±1.08	12.0±3.19	18.7±9.77	57.2±44.36	42.0±15.49[14.2]		-
a19-20	0.4±0.01	0.4±0.01	**0.3**±0.01	0.3±0.01	0.3±0.01		0.4±0.01
a22-4	**45.9**±0.22(20.0%)	46.7±0.32(20.0%)	147.7±91.98	120.9±66.96	170.5±54.49[81.4](30.0%)	144.4±72.16	-
a23-78	**13.9**±5.03	22.5±27.59	48.5±56.25	135.1±100.35	60.2±28.71[22.4]		-
a29-0	31.9±18.89	29.2±5.11	**28.1**±11.44	47.1±34.09	46.7±1.95[22.5]		-
a35-2	**9.8**±2.32	10.6±1.75	14.3±8.99	60.9±55.65	55.5±24.94[17.6]		-
a3-92	30.2±21.01	**29.3**±32.81	46.3±23.11	99.4±70.47	48.2±22.97[20.0]	62.9±76.90	-
a5-43	**21.1**±5.47	29.3±24.47	41.0±16.94	67.3±35.52	36.6±7.27[16.5]		-
a19-18	15.7±4.15	**14.2**±5.41	21.4±11.76	47.1±28.99	42.1±23.61[16.7]		-
a23-6	73.1±42.11	**61.6**±39.50	148.3±102.77	86.9±47.62	84.4±35.56[37.4]	69.8±60.94	-
a31-4	**22.0**±3.86	24.0±12.99	25.2±12.94	43.1±15.09	50.8±3.41[20.5]		-

Discussion of feasibility heuristics: Filtering subgoals with `Filter` provides a significant performance boost, as reported in columns `Rnd+FR` and `Rnd+R`. While we sample the same amount of subgoals in both methods, with filtering we select subgoal positions with o_{goal} closer to the goal or in line-of-sight of it. However, as seen in Fig. 6, filtering of random subgoals leads to highly varying results in the Wall scene, where success strongly depends on finding the narrow opening, while filtering leaves only few candidates in this area (Fig. 3b).

Prioritizing configurations with a higher score, as described in Sec. IV-A, leads to a faster solution in 16 out of 21 scenes, as can be seen on `Rnd+FPR` and `Rnd+FR` results of Tab. I and both scenes in Fig. 6.

Rejecting similar configurations: We compare the runtime of the full version `Rnd+FPR` to the ablation version `Rnd+FP`, where we do not reject similar configurations. In most scenes the search finishes before we encounter similar configurations, except for Maze, Wall and O-Room, which take longer. For Maze we observed a higher number of rejected configurations in `Rnd+FPR`, and for this problem the non-rejecting ablation `Rnd+FP` is slower. However, for the scene Wall, we see the opposite effect in some of the runs, with the lower median time for `Rnd+FP` (Fig. 6b). This indicates that it can be beneficial to try similar configurations for problems, where the objects are initially hard to reach.

Comparison to human-baseline subgoals: Only two scenes (Maze and Wall) significantly benefit from the `Hum` subgoals, and the scene O-Room is slightly better than `Rnd+FPR`. These are hard problems with narrow passages, but only a single object without additional obstacles. We conclude that having a *good initial guess* for the subgoals only helps if the scene remains unchanged and has only few possible solutions. In most other problems (especially 4-blocks and the PCG scenes) the positions of the agent and the objects can be very different in every solution. Therefore, *generating subgoals dynamically for every configuration leads to better average performance than having a set of good but fixed subgoals*.

VII. Conclusions

Integrating subproblem search with a baseline LGP-Solver enables us to plan for scenes which are not feasible for the baseline solver alone: containing occluded objects, obstacles and multiple narrow passages. By using simple heuristics to generate subgoals and select configurations, we can significantly decrease the time to solution. We have shown that our approach is applicable to a wide variety of layouts.

For most scenes, the subgoals generated by the random heuristic lead to similar or better solution rates and times compared to subgoals from human demonstrations. This shows that our approach does not require the subgoals to be in optimal positions, but it benefits from the ability to dynamically generate diverse subgoals. The two scenes (Maze and Wall) where manual subgoals provide an advantage, are also the ones where a bottleneck heuristic performs better.

Our approach results in very interesting puzzle solutions, combining object repositioning, navigating through passages and refined multi-step object interactions. However, the solutions could be potentially optimized in terms of the path lengths or the number of object interactions.

In our evaluations we focused on pick-and-place actions, but the approach is also applicable to other interactions, as pushing.

REFERENCES

[1] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *arXiv preprint arXiv:2010.01083*, 2020.

[2] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 440–448.

[3] J. Ferrer-Mestres, G. Francès, and H. Geffner, "Combined task and motion planning as classical AI planning," *CoRR*, 2017.

[4] 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*, 2014.

[5] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artificial Intelligence*, vol. 247, 05 2015.

[6] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 3684–3691.

[7] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *International Joint Conference on Artificial Intelligence*, 2015.

[8] J. Ortiz-Haro, E. Karpas, M. Katz, and M. Toussaint, "Conflict-directed diverse planning for logic-geometric programming," in *Proceedings of ICAPS*, 2022.

[9] ——, "A conflict-driven interface between symbolic planning and nonlinear constraint solving," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 518–10 525, 2022.

[10] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[11] W. Vega-Brown and N. Roy, "Asymptotically optimal planning under piecewise-analytic constraints," in *Algorithmic Foundations of Robotics XII*. Springer, 2020, pp. 528–543.

[12] 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.

[13] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. V. Wichert, and W. Burgard, "Optimal, sampling-based manipulation planning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3426–3432.

[14] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

[15] J. Ota, "Rearrangement of multiple movable objects - integration of global and local planning methodology," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 2, 2004, pp. 1962–1967 Vol.2.

[16] R. Wang, Y. Miao, and K. E. Bekris, "Efficient and high-quality prehensile rearrangement in cluttered and confined spaces," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1968–1975.

[17] K. Ren, P. Chanrungmaneekul, L. E. Kavraki, and K. Hang, "Kinodynamic rapidly-exploring random forest for rearrangement-based nonprehensile manipulation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 8127–8133.

[18] S. B. Bayraktar, A. Orthey, Z. Kingston, M. Toussaint, and L. E. Kavraki, "Solving rearrangement puzzles using path defragmentation in factored state spaces," *IEEE Robotics and Automation Letters*, 2023.

[19] D. M. Saxena, M. S. Saleem, and M. Likhachev, "Manipulation planning among movable obstacles using physics-based adaptive motion primitives," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6570–6576, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231846605

[20] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2007, pp. 3327–3332.

[21] M. Stilman and J. J. Kuffner, *Planning Among Movable Obstacles with Artificial Constraints*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 119–135. [Online]. Available: https://doi.org/10.1007/978-3-540-68405-3_8

[22] E. Chane-Sane, C. Schmid, and I. Laptev, "Goal-conditioned reinforcement learning with imagined subgoals," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1430–1440.

[23] S. Nasiriany, V. Pong, S. Lin, and S. Levine, "Planning with goal-conditioned policies," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[24] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese, "Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 4583–4590.

[25] T. Jurgenson, O. Avner, E. Groshev, and A. Tamar, "Sub-goal trees a framework for goal-based reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 5020–5030. [Online]. Available: https://proceedings.mlr.press/v119/jurgenson20a.html

[26] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *International Conference on Machine Learning*, 2001. [Online]. Available: https://api.semanticscholar.org/CorpusID:1223826

[27] J. Hoffmann, J. Porteous, and L. Sebastia, "Ordered landmarks in planning," *Journal of Artificial Intelligence Research*, vol. 22, pp. 215–278, 2004.

[28] R. F. Pereira, N. Oren, and F. Meneguzzi, "Landmark-based approaches for goal recognition as planning," *Artificial Intelligence*, vol. 279, p. 103217, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370219300013

[29] E. Karpas and C. Domshlak, "Cost-optimal planning with landmarks," in *International Joint Conference on Artificial Intelligence*, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:59814028

[30] 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.

[31] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3, pp. 189–208, 1971. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370271900105

[32] 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 (ICRA)*, 2017.

[33] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 2. IEEE, 2000, pp. 995–1001.

[34] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, "Long-horizon multi-robot rearrangement planning for construction assembly," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 239–252, feb 2023.

[35] S. Stolz, "Procedural environment generation for task and motion planning problems," Bachelor's Thesis, 2023.

[36] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "Pcgrl: Procedural content generation via reinforcement learning," 2020.