PhyRoGen: Synthetic Generation of Physical Robot Manipulation Puzzles Using Procedural Content Generation

Lennart Julian Droß and Andreas Orthey and Marc Toussaint

***Abstract*— Robot manipulation of physical puzzles is an important requirement for automatic assembly and disassembly tasks. However, to enable robots to solve physical puzzles, manipulation skills need to be learned, which requires large training datasets, the generation of which is often time consuming and tedious. To overcome this problem, we propose the Physical Robot Manipulation Puzzle Generation framework (PhyRoGen), which leverages procedural content generation (PCG) for automated generation of synthetic datasets of manipulation puzzles. PhyRoGen is a general-purpose puzzle generator, which can generate physical puzzles with interlocking object dependencies, where one articulated object must be manipulated before another can be moved. Based upon PhyRoGen, we define six concrete generators which we use to generate 24 physical puzzles. By using a benchmarking framework, we are able to solve all puzzles in 1 to 300 seconds using sampling-based planning algorithms. Finally, we demonstrate that every generated puzzle is manipulatable by using a KUKA LBR iiwa robot in a physical simulation. This shows that our framework is able to procedurally generate unique, solvable robot manipulation puzzles, which is a crucial ingredient to benchmark manipulation algorithms and to develop robust foundation models.**

I. INTRODUCTION

Large datasets are essential for developing and evaluating robot manipulation algorithms, both for benchmarking motion planners [1], [2] and for training learning-based systems [3], [4]. However, manually curating such datasets is tedious, limits the diversity of generated scenarios, and makes it difficult to produce systematic variations [5, p. 2].

To tackle this problem, we leverage procedural content generation (PCG) to algorithmically create synthetic datasets. While some works have proposed PCG to generate datasets for robot manipulation problems [6], [1], they do not generate problems with *interlocking object dependencies*. These are problems where one articulated object must be manipulated before another can be moved. For example, a door (with a revolute joint) may need to be opened before a cube (on a prismatic joint) behind it can be moved. Throughout this paper, the term *joint* refers to the kinematic connection between object links in the puzzle (e.g., a door hinge or a sliding rail), not to the joints of the robot manipulator. We call such problems *physical manipulation puzzles* [7], [8].

To create manipulation puzzles, we introduce the *Phy*sical *Ro*bot Manipulation Puzzle *Gen*eration Framework (PhyRoGen). PhyRoGen consists of a general maker algorithm, which takes as input sequences of objects, joints, transformations, and returns as output a kinematic chain representing a manipulation puzzle. By using this framework, we can show that a variety of interesting puzzles can be created, including lockbox puzzles [7], simple escape rooms [8], and puzzles which require re-grasping sequences [9]. In total, we create 24 puzzles, and showcase that they are solvable using a benchmark framework to compare planning algorithms [2], [1]. Finally, we devise a method to manipulate those puzzles using a KUKA LBR iiwa robot. This is shown in Fig. 1, where we showcase two puzzles generated with PhyRoGen. The mobile manipulator robot is able to manipulate the objects to move the green object to a desired goal configuration, while removing red objects whenever they block the execution.

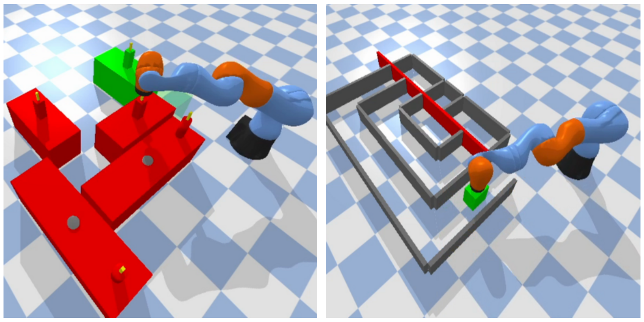


Fig. 1: A KUKA mobile manipulator solves a procedurally generated manipulation puzzle with interlocking object dependencies. **Left:** The grid world environment, where a green slider has to be moved, but is blocked by multiple red objects. **Right:** The Move N Times environment, where a green cube has to be moved out of a maze which is blocked by a sliding prismatic door, requiring multiple re-grasping sequences.

To summarize, our contributions are the following:

- Puzzle generation: We develop the PhyRoGen puzzle generation framework and implement it using Blender [10] in combination with the Phobos [11] plugin.
- Puzzle benchmarking: We create six generators to produce a total of 24 puzzles, and benchmark them using sampling-based puzzle-solving algorithms using Robowflex [12], [13] and OMPL [14].
- Puzzle manipulation: We show how puzzles can be manipulated and solved using a simulated KUKA LBR iiwa mobile robot platform using PyBullet [15] and OMPL [14].

A complete overview of the modules involved is shown in Fig. 2. All material related to this work, including puzzle URDF files, images, benchmark results (graphs and database files), manipulation videos, and source code is freely available[1].

II. RELATED WORK

PCG [16] has been used in the creation of digital games since the 1980s with games like *Rogue* which use procedurally generated environments [17]. This has lead to the genre of *roguelike* games [18], which include the creation of virtual game spaces [19], game bits (textures, sound, game objects), game systems (ecosystems, road networks, entity behavior), and game scenarios (puzzles, levels, stories)[20], [17]. Recently, PCG systems have been used to generate complete game universes like in *No Man's Sky* or *Minecraft* [21], [22].

To avoid repetitions and keep players engaged, game development has also focused on procedural puzzle generation [23]. An exemplary physical puzzle is *Sokoban* [24], a PSPACE-complete problem [25], which has caught the attention of the gaming community [26], [27], [28], [24]. While Sokoban puzzles present interesting challenges for robots, they are not well suited for physical robot exploration, because unrecoverable states can be easily reached and are difficult to recognize.

Games also play an important role as learning environments for reinforcement learning (RL) agents [29]. This includes Arcade games [30], [31] and real-time strategy games [32]. To generalize those RL systems, it becomes important to generate synthetic worlds. Systems like Procgen [33] or MiniHack [34] can generate different game environments with simple point-based reward functions. Despite their usefulness for RL agents, those systems are not suitable for robotics due to the focus on mostly 2-dimensional spaces and simplified action spaces.

In the last decade, robot learning has also become a central focus of the robotics community [3], [35]. Similarly to RL agents, robot learning requires large datasets to create robust systems [36]. To generate such datasets, researchers have focused on perturbating existing problems with automatic domain randomization [37]. However, to generalize to unseen environments, it becomes crucial to be able to learn on automatically generated worlds. Therefore, several generator algorithms have been proposed like ProcTHOR [6], which is a PCG framework for interactive physical environments, MotionBenchMaker [1], a tool to generate motion planning datasets, or ClutterGen [38], which can generate diverse sets of tables with objects for manipulation. Those datasets can be used in learning benchmarks like Meta-World [39] or RLBench [40], which are specifically designed to be used in robotics frameworks. Those generators, however, do not generate environments with interlocking object dependencies, where one articulated object blocks another. This makes them less suitable for task and motion planning (TAMP) that involves physical exploration.

While robot learning algorithms have mostly been applied to robot manipulation tasks [37], [3], work on learning for TAMP algorithms [41], [42] has only recently been started [43]. In TAMP, an algorithm has to reason over both discrete, logical constraints and continuous state spaces connecting them [44], [45]. One category of TAMP algorithms tackles physical puzzles [7], where objects have dependencies preventing them from being manipulated in a random order [8]. This makes physical puzzles differ from tabletop scenarios [46] or top-down rearrangement planning scenarios [47].

Our proposed puzzle generator is complementary to both learning and TAMP planners in that PhyRoGen creates environments with interlocking object dependencies, such that manipulation puzzles can be created, where a robot has to reason about object interactions [8]. This is an important requirement both for learning frameworks and for advancing TAMP algorithms.

III. PHYROGEN: PHYSICAL ROBOT MANIPULATION PUZZLE GENERATION FRAMEWORK

The PhyRoGen framework consists of three parts. First, we define kinematic chain generators, which are functions to generate a sequence of kinematic chains from an integer seed. Based upon them, we detail the PhyRoGen maker algorithm, which is a general-purpose kinematic chain generator maker, taking as input objects, joints, and transformations, and returning a kinematic chain. Finally, we depict six concrete kinematic chain generators, which we use to build 24 scenarios of randomly generated manipulation puzzles.

A. Kinematic Chain Generators

A kinematic chain generator is a function which gets as input an integer *seed* in $\mathbb{N}$, plus a set of objects, a set of joints, a set of transformations, and a number of objects. As output, a generator produces a kinematic chain consisting of links and joints together with their relative transformations. We define a generator to have the following attributes:

- **Deterministic**. For a given seed and identical input, the generator always produces the same kinematic chain.
- **Interdependent**. Each generator produces kinematic chains which represent puzzles which are solvable only if all joints are actuated. That means there are no superfluous joints added.
- **Solvable and Manipulatable**. Each kinematic chain should lead to a puzzle generated which is solvable and eventually manipulatable by a mobile manipulator.

B. PhyRoGen Maker Algorithm

Here we present the PhyRoGen maker algorithm. This is an iteration-based algorithm, which aims to create a new object in each iteration, and which produces as output a kinematic chain representing a physical puzzle.

The complete algorithm is detailed in Algorithm 1. As input, we are given the tuple (`Objects`, `Joints`, `Transforms`, N, `Seed`), whereby `Objects` is either a finite set of objects or an infinite sequence of objects, `Joints` is either a finite set of joints or an infinite sequence of joints, `Transforms` is either a finite set of transformations or an infinite sequence of transformations, N is the number of objects of the algorithm, and `Seed` is the unique seed

---

[1] Overview page: sites.google.com/view/robot-manipulation-puzzles

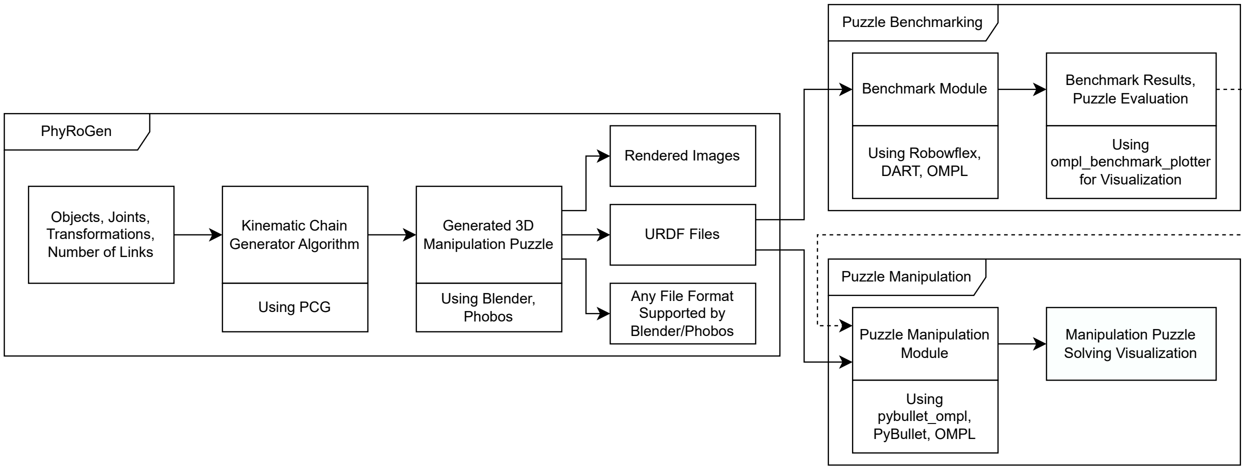


Fig. 2: Systems overview about this paper, including the puzzle generation (left), and the puzzle benchmarking and manipulation modules (right).

Algorithm 1 PhyRoGen Maker

```
Input: $(\texttt{Objects}, \texttt{Joints}, \texttt{Transforms}, N, \texttt{Seed})$
Output: Kinematic Chain $\texttt{K}$
1:  $\texttt{K} \leftarrow$ InitializeKinematicChain()
2:  for $n \in \{1, \ldots, N\}$ do
3:      while True do
4:          $\texttt{obj} \leftarrow$ GetItem($\texttt{Objects}, n, \texttt{Seed}$)
5:          $\texttt{jnt} \leftarrow$ GetItem($\texttt{Joints}, n, \texttt{Seed}$)
6:          $\texttt{tf} \leftarrow$ GetItem($\texttt{Transforms}, n, \texttt{Seed}$)
7:          $\texttt{Kt} \leftarrow$ NewKinematicChain($\texttt{K}, \texttt{obj}, \texttt{jnt}, \texttt{tf}$)
    // Joint should invalidate puzzle in its default state
8:          $\texttt{Kt} \leftarrow$ FixJoint($\texttt{Kt}, \texttt{jnt}$)
9:          if ValidatePuzzle($\texttt{Kt}$) then
10:             continue
11:         end if
    // Puzzle should be valid when joint is variable
12:         $\texttt{Kt} \leftarrow$ UnfixJoint($\texttt{Kt}, \texttt{jnt}$)
13:         if Not ValidatePuzzle($\texttt{Kt}$) then
14:             continue
15:         end if
16:         $\texttt{K} \leftarrow \texttt{Kt}$
17:         break
18:     end while
19: end for
20: return $\texttt{K}$
```

in $\mathbb{N}_{>0}$ which acts as the randomization, i.e. which objects, joints, and transformations to use in each iteration. As output, the algorithm returns a kinematic chain `K`, representing a physical puzzle.

To achieve this, the following steps are carried out. First, we initialize an empty kinematic chain (Line 1). We then run N iterations of the algorithm, whereby in each iteration we enter a while loop (Line 3) which ends when a new kinematic chain has been found (Line 16–17). To find a new kinematic chain, we first get a random object `obj`, joint `jnt`, and transformations `tf` from the input sequences (Line 4–6). The `GetItem` function either choose a random element from a set by using the seed, or returns the n-th element in the sequence. Those items are used to extend the current kinematic chain `K` to create a new temporary kinematic chain `Kt` (Line 7). We then conduct two validations steps. First, we check if the puzzle is solvable when the new object is at its default position (Line 8–11). If yes, we reject this solution and continue, since the object would not contribute to a solution and would thereby be superfluous. This validation step uses the RRT-Connect planner [48] on the temporary kinematic chain `Kt` until a first solution is found or a timeout is reached. Second, we check if there exists a solution to the puzzle when the object is movable along its joint (Line 12–15). If there exists no solution, then the puzzle is not solvable, and we have to reject the kinematic chain. Once the new kinematic chain has passed both checks, we update the global kinematic chain `K` (Line 16) and exit the while loop (Line 17). After all iterations have passed, we return the kinematic chain (Line 20).

For every chain generated, we define a physical puzzle problem as the problem of moving the first link from its default position to a dedicated goal position. There are three cases: If the object has an attached prismatic joint, we define the goal as the opposite end of its limits. If a revolute joint is present, we define the goal as a 90 degree rotation from its current position in either clockwise or counterclockwise direction. If the joint is $SE(2)$, we define the goal as being a random placement outside of a bounding box around the final generated puzzle.

C. Six Kinematic Chain Generators

Given the general-purpose the PhyRoGen maker algorithm, we present six concrete input tuples, each leading to a different set of kinematic chain generators. The chosen six input tuples of objects, joints, transformations, and number of objects are shown in Table I. We have indicated in this table infinite sequences as gray colored cells and finite sets as white colored cells. This input directly defines the kinematic chain generators through PhyRoGen, and can be used to generate an infinite sequence of manipulation puzzles by changing the input seed.

The kinematic chains are built directly in the 3D modeling software Blender [10] using a Python script [49]. From this 3D description, we can both render images of the puzzles and export them to the Unified Robot Description Format (URDF) by using the Blender extension Phobos [11].

IV. PUZZLE BENCHMARKS AND PUZZLE MANIPULATION

Once all puzzles have been created, we use a puzzle solver to find solutions and execute them using a robot manipulator. To this end, we use sampling-based motion planning methods [50] to solve the puzzles, whereby we rely on the less-action RRT [8] (LA-RRT), which is a multi-object planner to compute a minimal number of moves to solve a given puzzle.

To enable this, we provide an open source implementation. This implementation uses Robowflex [12], and combines it with the Open Motion Planning Library (OMPL) [14], using the internal benchmarking tools [2].

To show that the generated puzzles are not only solvable, but can also be manipulated, we use a custom method to manipulate them. This method works as follows. First we import the results of the lowest-cost run of LA-RRT for each puzzle (after the time limit) and extract both the object sequences and the object paths. For each object path, we compute an inverse kinematics (IK) solution for the robot at designated grasp points on the objects. To execute those motions, we compute how the grasp point moves when the object is moved along its path. We then use the end-effector Jacobian [51] to generate a joint direction at the grasp point which moves the object along the computed object path. To deal with singularities, we observe possible joint flips on the robot during the execution. If a singularity occurs, we release the grasp, and re-grasp at a new grasp position at which the robot is not in a singularity at the next object position. This process is continued until the object has attained its final position.

Implementation-wise, we use a KUKA LBR iiwa on a mobile platform for manipulation of the puzzles. We simulate this in PyBullet [15] with an interface to OMPL [14]. All computations are using the internal IK solver of PyBullet to compute joint configurations at grasp points. To plan connections to grasp points, we use the Batch Informed Trees (BIT*) [52] planner in OMPL. BIT* is used for two tasks. First, to reach the IK solutions, which have been found at a grasp point. Second, to compute motions to new grasp points after singularities are detected. This is a slightly different problem since we need a motion from a grasp point to another grasp point.

V. DEMONSTRATION

The demonstration is divided into three parts. In the first part, we show how our generators can generate different puzzle variations. In the second part, we use different sampling-based motion planning algorithms to benchmark solving those puzzles. Finally, we execute each puzzle with a KUKA LBR iiwa robot by using the best solution found by the benchmark planners.

Our code is available on GitHub[2]. All experiments were conducted on a workstation PC with an Intel Core i7-6700 3.4 Ghz quad-core processor, 16 GB memory, running the Ubuntu 18.04 LTS operating system.

A. Puzzle Generation

We show that our generators can generate unique physical manipulation puzzles using our framework. To generate the puzzles, we ran each generator and generate the first four seeds for each puzzle. The runtimes to generate them are Simple Sliders (4 seconds), Grid World (5 seconds), Continuous Space (83 seconds), Lockbox Random (13 seconds), Move N Times (5 seconds), and Rooms (17 seconds). This generates 24 unique physical puzzles as shown in Fig. 3.

B. Puzzle Benchmarking

To show that the generated puzzles are solvable, we use four sampling-based planning algorithms on all 24 puzzles. The four planning algorithms are LA-RRT [8], RRT* [53], LBT-RRT [54], and BIT* [52]. Note that we are using optimal sampling-based planners [55], since we want to optimize the total action cost, which is the total number of actions (individual joint moves) required to solve the puzzle. Each planner is run on each puzzle whereby we repeat the experiments 100 times per planner with the following time limits: Simple Sliders (40 seconds), Grid World (40 seconds), Continuous Space (20 seconds), Lockbox Random (600 seconds), Move N Times (20 seconds), and Rooms (60 seconds). We collect both the time to find the first feasible solution and track the cost over time.

The averaged results are shown in Fig. 4. It can be seen that LA-RRT is able to solve 23 out of 24 scenarios with a success rate of 100%. The only outlier is Lockbox Random 1, where LA-RRT only reaches 95% success rate at the time limit. BIT* is able to solve 24 out of 24 scenarios with a success rate of 100%. The two remaining planners LBTRRT and RRT* can solve 7 out of 24 and 6 out of 24 with 100%. In terms of optimality, LA-RRT reaches the lowest cost of all planners consistently over all 24 scenarios. This is expected, since LA-RRT is specifically designed to solve manipulation puzzles with a minimal number of actions.

C. Puzzle Manipulation

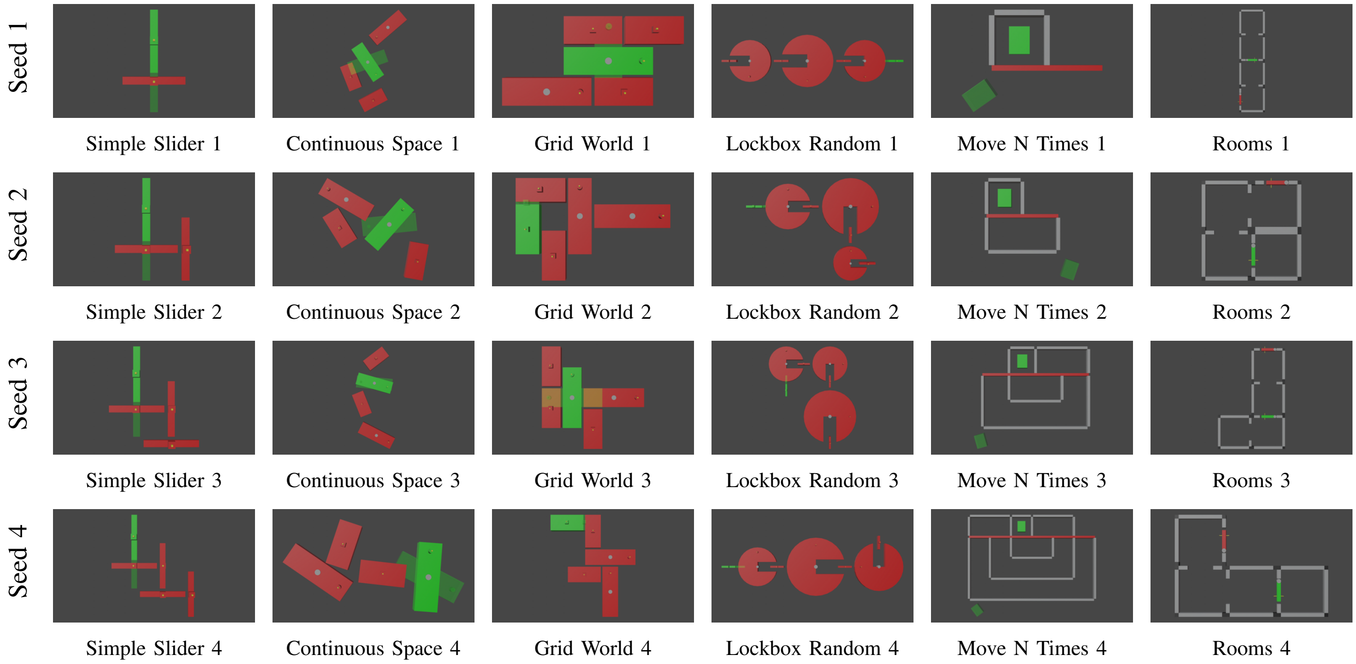
By using the method described in Sec. IV, we can show that all 24 puzzles can eventually be executed using a mobile manipulator robot. The results are depicted in Fig. 5, where we show snapshots of how the robot is manipulating each puzzle. For each scenario, we show four frames, whereby Frame 1 depicts the start configuration of the puzzle. Frame 2 and Frame 3 show intermediate manipulation steps, where the robot moves some of the dependent obstacles out of the way. Eventually, Frame 4 shows the last action where the

---

[2] [Redacted due to double-blind review]

TABLE I: Input Sequence Configurations. Gray cells represent infinite sequence inputs, while the remaining white cells represent finite sets.

Name	Objects	Joints	Transformation	Number of Objects
Simple Slider	{T-shaped slider}	{Prismatic}	$\{0, 180\}$	Seed+1
Continuous Space	{Cuboid}	{Revolute}	$[0, 90] \times [0, 1]$	4
Grid World	{Cuboid}	{Revolute, Prismatic}	$\{0, 90, 180, 270\}$	5
Lockbox Random	{Cuboid, Disk, Cuboid, Disk, ...}	{Prismatic, Revolute, Prismatic, ...}	$\{0, 90, 180, 270\}$	7
Move N Times	{Cuboid, Door(N), Corridor(1), ...}	{SE(2), Prismatic, Fixed, Fixed, ...}	$\{0, 180, 0, \ldots\}$	Seed+2
Rooms	{Room, Door Room}	{Fixed}	$\{0, 90, 180, 270\}$	4



Simple Slider 1 Continuous Space 1 Grid World 1 Lockbox Random 1 Move N Times 1 Rooms 1

Simple Slider 2 Continuous Space 2 Grid World 2 Lockbox Random 2 Move N Times 2 Rooms 2

Simple Slider 3 Continuous Space 3 Grid World 3 Lockbox Random 3 Move N Times 3 Rooms 3

Simple Slider 4 Continuous Space 4 Grid World 4 Lockbox Random 4 Move N Times 4 Rooms 4

Fig. 3: Visualization of puzzle environments for the first four seeds using the input from Table I.

robot moves the green object (the only one with a dedicated goal configuration) towards its goal position. The robot used is a KUKA LBR iiwa on a mobile manipulator base, having three degrees of freedom (dof) for the base and 7-dof for the manipulator arm. Full videos for all 24 scenarios are available[3]. The complete execution and planning time for the scenarios is between 4s (Simple Sliders 1) and 51s (Move N Times 4).

VI. CONCLUSION

We proposed to leverage procedural content generation (PCG) to generate synthetic datasets of manipulation puzzles suitable for learning-based approaches that involve physical exploration.

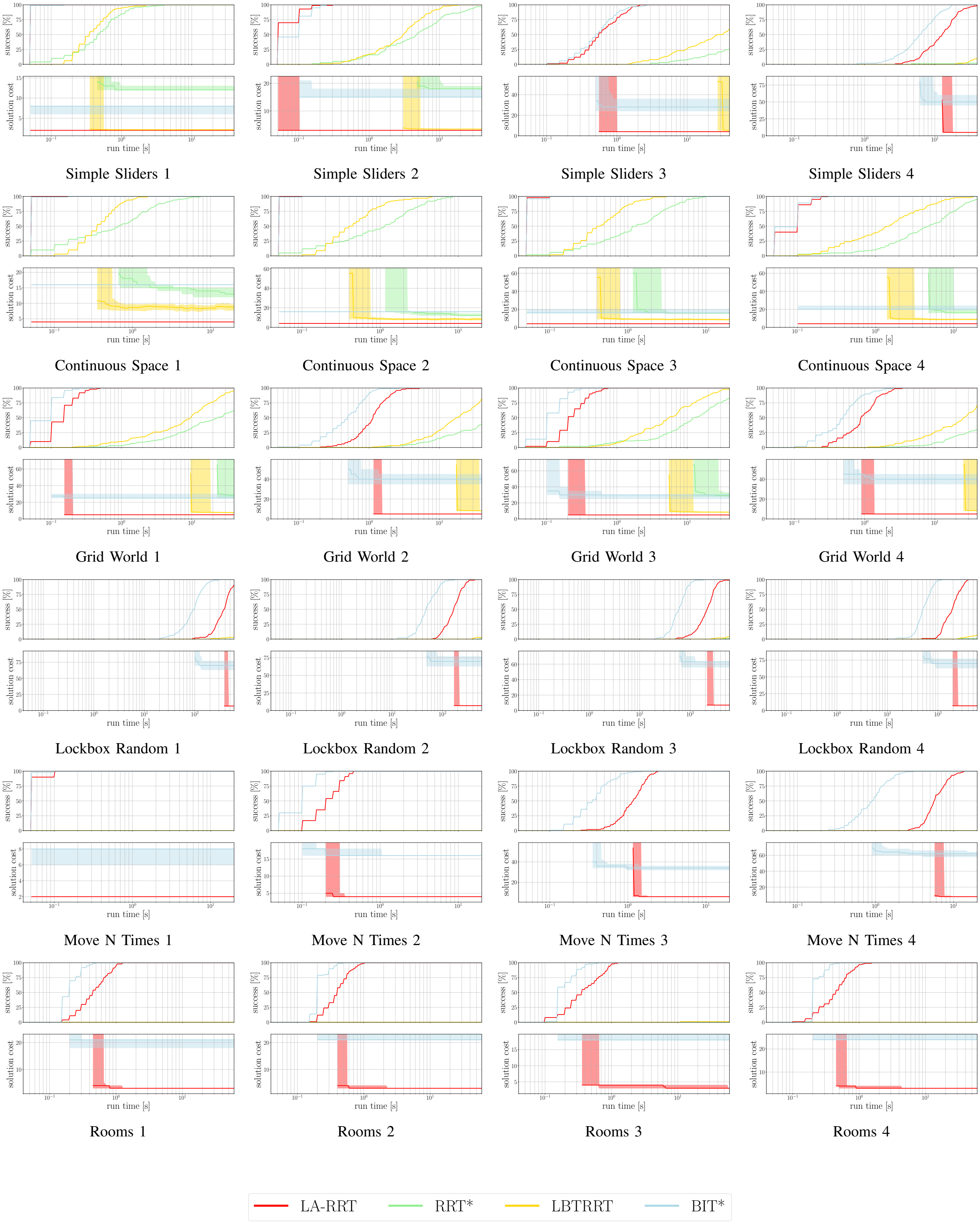
Compared with previous works, such as ProcTHOR [6] and MotionBenchMaker [1], which also leverage PCG to generate manipulation problems, our method uses PCG specifically to generate interlocking object dependencies to create physical manipulation puzzles. The additional aspect of exploring and reasoning about the physical constraints of environments makes our methods suitable for learning-based TAMP [56].

While we have shown that our method is able to generate unique manipulation puzzles, we still face some limitations.

- **Narrow Passages** Our current method struggles to generate environments containing narrow passages between objects, because motion planning methods often cannot handle them efficiently [57]. We could solve this by using planners combining sampling-based and optimization-based frameworks [58].
- **Tree-Dependencies** Our current framework generates environments with sequential (chain-like) dependencies between objects. However, puzzles often require tree-like dependency structures, where multiple objects independently block a common target, which we aim to generate in future work using a backtracking algorithm.
- **Online Learning** Our current framework generates data which can then be used for learning. However, we envision that our framework is used in an online manner, where the difficulty of environments is adjusted based

---

[3] sites.google.com/view/robot-manipulation-puzzles



Simple Sliders 1

Simple Sliders 2

Simple Sliders 3

Simple Sliders 4

Continuous Space 1

Continuous Space 2

Continuous Space 3

Continuous Space 4

Grid World 1

Grid World 2

Grid World 3

Grid World 4

Lockbox Random 1

Lockbox Random 2

Lockbox Random 3

Lockbox Random 4

Move N Times 1

Move N Times 2

Move N Times 3

Move N Times 4

Rooms 1

Rooms 2

Rooms 3

Rooms 4

Fig. 4: Visualization of benchmark results across six generators, each with four variations, and with the legend below.

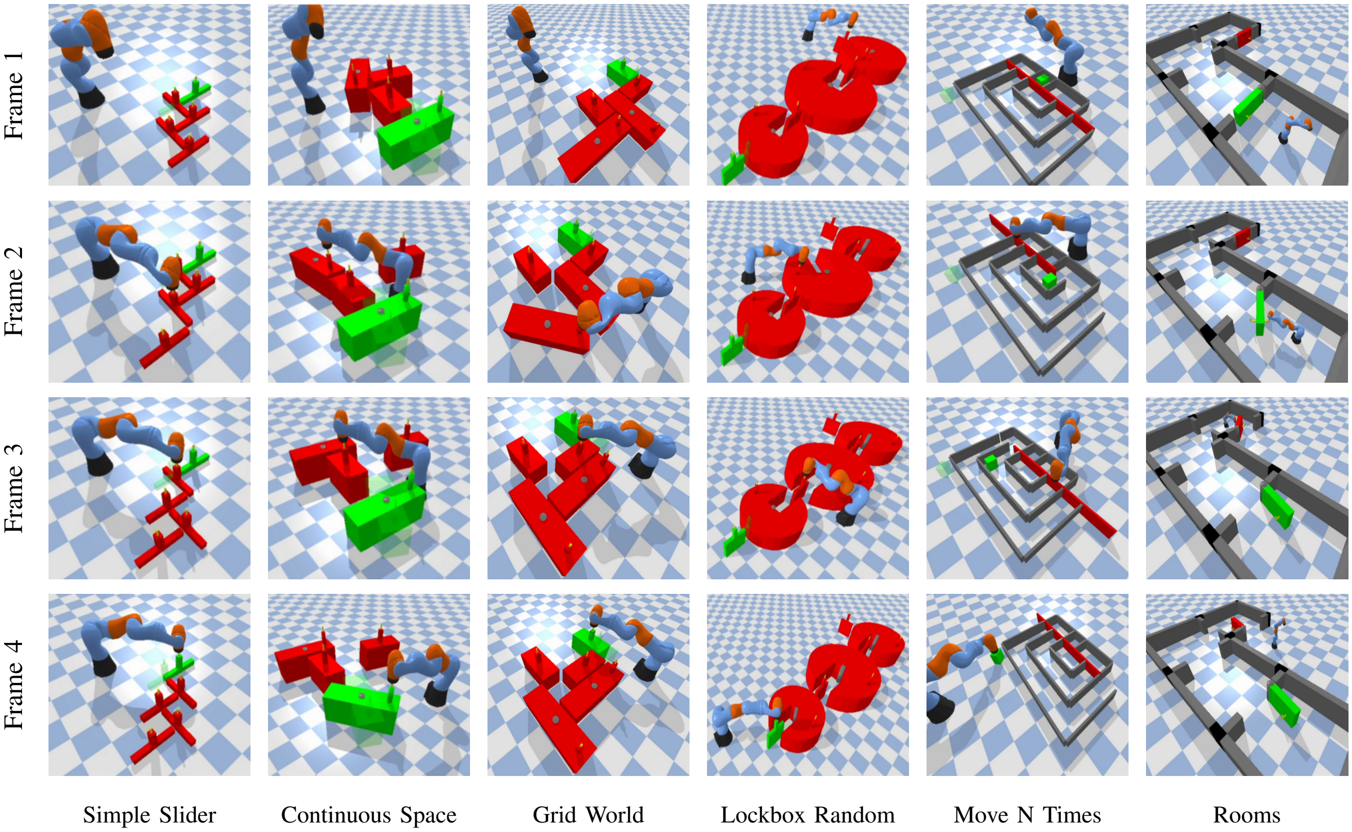


Fig. 5: Visualization of manipulation tasks across all six environments. The manipulation task for the mobile manipulator robot is to move the green object to its goal position (transparent green), while red objects block the green object and have to be moved out of the way (interlocking dependencies). The `Rooms` environment is different in that the robot has to move out of the room itself, but is blocked by all objects (doors).

on the progress of the robot [59].

- **Baseline Comparisons** Our evaluation demonstrates solvability and manipulatability of the generated puzzles, but does not yet include comparisons against other generation frameworks or ablation studies. A systematic comparison with manually curated benchmarks would provide stronger evidence for the utility of procedurally generated puzzles.
- **Real-World Validation** All experiments are conducted in simulation. Transferring the generated puzzles to a physical robot would validate that the puzzles and solutions transfer to a real robot.

Despite limitations, our proposed method is able to reliably generate synthetic datasets of unique manipulation puzzles in seconds. The difficulty and solvability of the 24 puzzle samples is verified by extensive benchmarking and manipulation simulations. Overall, our method yields good performance in terms of speed of generation (4s to 83s), speed of solvability (1s to 300s), and quality of generated puzzles (all are solvable using a realistic mobile manipulator). We therefore believe that this framework is a useful tool to create large datasets for benchmarking and training of learning-based manipulation robots [3], [4].

REFERENCES

[1] C. Chamzas, C. Quintero-Peña, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, “Motionbenchmaker: A tool to generate and benchmark motion planning datasets,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 882–889, apr 2022.

[2] M. Moll, I. A. Sucan, and L. E. Kavraki, “Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization,” *IEEE Robotics and Automation Magazine*, vol. 22, no. 3, pp. 96–102, 2015.

[3] O. Kroemer, S. Niekum, and G. Konidaris, “A review of robot learning for manipulation: Challenges, representations, and algorithms,” *Journal of Machine Learning Research*, vol. 22, no. 30, pp. 1–82, 2021.

[4] R. Firoozi, J. Tucker, S. Tian *et al.*, “Foundation models in robotics: Applications, challenges, and the future,” *International Journal of Robotics Research*, vol. 44, no. 5, pp. 701–739, 2025.

[5] K. Zhang, P. Yun, J. Cen *et al.*, “Generative artificial intelligence in robotic manipulation: A survey,” *CoRR*, vol. abs/2503.03464, 2025.

[6] M. Deitke, E. VanderBilt, A. Herrasti *et al.*, “Procthor: Large-scale embodied ai using procedural generation,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 5982–5994, 2022.

[7] M. Baum, M. Bernstein, R. Martin-Martin *et al.*, “Opening a lockbox through physical exploration,” in *IEEE International Conference on Humanoid Robots*. IEEE, 2017, pp. 461–467.

[8] 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*, vol. 8, no. 8, pp. 4529–4536, 2023.

[9] 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.

[10] B. O. Community, “Blender,” Blender Foundation, Stichting Blender Foundation, Amsterdam, 2021.

[11] K. von Szadkowski and S. Reichel, “Phobos: A tool for creating complex robot models,” *Journal of Open Source Software*, vol. 5, no. 45, p. 1326, 2020.

[12] Z. Kingston and L. E. Kavraki, “Robowflex: Robot motion planning with moveit made easy,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2022, pp. 3108–3114.

[13] J. Lee, M. X. Grey, S. Ha *et al.*, “Dart: Dynamic animation and robotics toolkit,” *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[14] I. A. Sucan, M. Moll, and L. E. Kavraki, “The open motion planning library,” *IEEE Robotics and Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[15] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” 2021.

[16] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, “What is procedural content generation? mario on the borderline,” in *International Workshop on Procedural Content Generation in Games*, 2011, pp. 1–6.

[17] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[18] J. Harris, *Exploring Roguelike Games*. CRC Press, 2020.

[19] B. M. F. Viana and S. R. dos Santos, “Procedural dungeon generation: A survey,” *Journal on Interactive Systems*, vol. 12, no. 1, pp. 83–101, 2021.

[20] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, “Procedural content generation for games: A survey,” *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 9, no. 1, pp. 1–22, 2013.

[21] J.-B. Hervé and C. Salge, “Comparing pcg metrics with human evaluation in minecraft settlement generation,” in *Proceedings of the International Conference on the Foundations of Digital Games*, 2021, pp. 1–15.

[22] M. Awiszus, F. Schubert, and B. Rosenhahn, “Wor(l)d-gan: Toward natural-language-based pcg in minecraft,” *IEEE Transactions on Games*, vol. 15, no. 2, pp. 182–192, 2022.

[23] B. De Kegel and M. Haahr, “Procedural puzzle generation: A survey,” *IEEE Transactions on Games*, vol. 12, no. 1, pp. 21–40, 2019.

[24] Y. Zakaria, M. Fayek, and M. Hadhoud, “Procedural level generation for sokoban via deep learning: An experimental study,” *IEEE Transactions on Games*, vol. 15, no. 1, pp. 108–120, 2022.

[25] J. Culberson, “Sokoban is pspace-complete,” *IEICE Technical Report*, 1997.

[26] B. Kartal, N. Sohre, and S. J. Guy, “Data-driven sokoban puzzle generation with monte carlo tree search,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 12, 2016, pp. 58–64.

[27] J. Taylor, *The Procedural Generation of Interesting Sokoban Levels*. University of North Texas, 2015.

[28] S. Karman *et al.*, “Generating sokoban levels that are interesting to play using simulation,” Master’s thesis, Utrecht University, 2018.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[30] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[32] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[33] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, “Leveraging procedural generation to benchmark reinforcement learning,” in *Proceedings of the International Conference on Machine Learning*. PMLR, 2020, pp. 2048–2056.

[34] M. Samvelyan, R. Kirk, V. Kurin *et al.*, “Minihack the planet: A sandbox for open-ended reinforcement learning research,” in *Neural Information Processing Systems*, 2021.

[35] X. Xiao, J. Liu, Z. Wang *et al.*, “Robot learning in the era of foundation models: A survey,” *Neurocomputing*, p. 129963, 2025.

[36] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[37] OpenAI, I. Akkaya, M. Andrychowicz *et al.*, “Solving rubik’s cube with a robot hand,” *ArXiv Preprint*, 2019.

[38] Y. Jia and B. Chen, “Cluttergen: A cluttered scene generator for robot learning,” in *Proceedings of the Conference on Robot Learning*, 2024.

[39] T. Yu, D. Quillen, Z. He *et al.*, “Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning,” in *Proceedings of the Conference on Robot Learning*. PMLR, 2020, pp. 1094–1100.

[40] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, “Rlbench: The robot learning benchmark and learning environment,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3019–3026, 2020.

[41] L. P. Kaelbling and T. Lozano-Pérez, “Hierarchical task and motion planning in the now,” in *IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1470–1477.

[42] C. R. Garrett, R. Chitnis, R. Holladay *et al.*, “Integrated task and motion planning,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 265–293, 2021.

[43] D. Driess, F. Xia, M. S. M. Sajjadi *et al.*, “PaLM-e: An embodied multimodal language model,” in *Proceedings of the International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. e. a. Krause, Ed., vol. 202. PMLR, 23–29 Jul 2023, pp. 8469–8488.

[44] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, “Differentiable physics and stable modes for tool-use and manipulation planning,” in *Proceedings of Robotics: Science and Systems*, 2018.

[45] Z. Kingston, M. Moll, and L. E. Kavraki, “Exploring implicit spaces for constrained sampling-based planning,” *International Journal of Robotics Research*, vol. 38, no. 10-11, pp. 1151–1178, 2019.

[46] K. Gao, D. Lau, B. Huang, K. E. Bekris, and J. Yu, “Fast high-quality tabletop rearrangement in bounded workspace,” in *IEEE International Conference on Robotics and Automation*. IEEE, 2022, pp. 1961–1967.

[47] T. Tsesmelis, L. Palmieri, M. Khoroshiltseva *et al.*, “Re-assembling the past: The repair dataset and benchmark for real world 2d and 3d puzzle solving,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 30 076–30 105, 2024.

[48] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 2000, pp. 995–1001.

[49] C. Conlan, *The Blender Python API: Precision 3D Modeling and Add-on Development*. Apress, 2017.

[50] A. Orthey, C. Chamzas, and L. E. Kavraki, “Sampling-based motion planning: A comparative review,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, no. 1, pp. 285–310, 2024.

[51] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 4th ed. Upper Saddle River, NJ: Pearson, 2017.

[52] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, “Batch informed trees (bit*): Informed asymptotically optimal anytime search,” *International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.

[53] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[54] O. Salzman and D. Halperin, “Asymptotically near-optimal rrt for fast, high-quality motion planning,” *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 473–483, 2016.

[55] J. D. Gammell and M. P. Strub, “Asymptotically optimal sampling-based motion planning methods,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 295–318, 2021.

[56] Z. Zhao, S. Cheng, Y. Ding *et al.*, “A survey of optimization-based task and motion planning: From classical to learning approaches,” *IEEE/ASME Transactions on Mechatronics*, 2024.

[57] A. Orthey and M. Toussaint, “Section patterns: Efficiently solving narrow passage problems in multilevel motion planning,” *IEEE Transactions on Robotics*, vol. 37, pp. 1891–1905, 2021.

[58] J. Kamat, J. Ortiz-Haro, M. Toussaint, F. T. Pokorny, and A. Orthey, “Bitkomo: Combining sampling and optimization for fast convergence in optimal motion planning,” in *International Conference on Intelligent Robots and Systems*, 2022, pp. 4492–4497.

[59] L. Gisslén, A. Eakins, C. Gordillo, J. Bergdahl, and K. Tollmar, “Adversarial reinforcement learning for procedural content generation,” in *Proceedings of the IEEE Conference on Games*. IEEE, 2021, pp. 1–8.