Formalization, Development, and Baseline Analysis of a Task and Motion Planning Domain in RDDL

Miguel Iglesias Alcázar[1], **Fernando Fernández Rebollo**[2], **Yoonchang Sung**[3], **Yuqian Jiang**[3]
[1] Universidad Carlos III de Madrid
[2] University of Texas at Austin
[3] University of Texas at Austin
migigles@pa.uc3m.es, ffernand@inf.uc3m.es,
jiangyuqian@cs.utexas.edu yooncs8@cs.utexas.edu

Abstract

This paper introduces a new domain for Task and Motion Planning (TAMP) problems within the Relational Dynamic Influence Diagram Language (RDDL), developed in the pyRDDLGym. This domain accommodates a robotic arm navigating in a shelf environment, enabling both discrete and continuous actions essential for agile manipulation and navigation tasks. This domain formalized in RDDL, serves as a framework for planners implemented in RDDL. By integrating the JaxPlan algorithm and reinforcement learning, we refine decision-making policies for these tasks, demonstrating the framework's potential for advancing TAMP research. The introduced domain not only opens up new avenues for research in TAMP problems but also serves as a robust platform for developing algorithms or planners leveraging RDDL.

Introduction

The field of autonomous robotics is rapidly advancing, with an increasing demand for systems capable of executing complex tasks in diverse environments. A key aspect of these systems is their ability to plan and make intelligent decisions. This paper introduces a new domain for Task and Motion Planning (TAMP) problems within the Relational Dynamic Influence Diagram Language (RDDL), developed in the pyRDDLGym.

This domain features a robotic arm navigating in a shelf environment, enabling both discrete and continuous actions essential for agile manipulation and navigation tasks. The discrete aspects of this domain pertain to the planning component of the problem, determining the actions necessary for the agent to reach its goal state. In contrast, the continuous aspect arises from the arm's movements within the shelving environment.

The choice of RDDL over other languages as PDDLStream (Garrett, Lozano-Pérez, and Kaelbling 2020) for modeling this domain is based on RDDL's versatility in capturing uncertainties, its parameterization for both discrete and continuous actions, and its proven application in related domains, especially when dealing with stochastic TAMP domains. While PDDLStream excels in symbolic and geometric reasoning, RDDL's capacity to handle uncertainties aligns closely with the complexities of robotic actions in uncertain environments. RDDL also supports hybrid approaches of planning and RL.

The newly formulated domain is evaluated using the JaxPlan algorithm (Gimelfarb, Taitler, and Sanner 2024), provided as the baseline for the IPC competition. Furthermore, the domain was integrated and tested within the pyRDDLgym environment, providing a structured framework for testing and experimentation. This integration into pyRDDLgym offers a standardized environment, potentially benefiting future researchers by providing a structured framework for exploring and developing TAMP solutions.

The introduction of this domain not only opens up new avenues for research in TAMP problems but also serves as a robust platform for developing algorithms or planners leveraging RDDL. By integrating reinforcement learning techniques with planning methodologies, this work offers an exciting avenue for enhancing the decision-making capabilities of autonomous robots, setting the stage for future breakthroughs in robotic decision-making.

In summary, this research introduces a novel domain for TAMP problems within RDDL, providing a fertile ground for exploration and setting the stage for future breakthroughs in robotic decision-making. The use of RDDL, JaxPlan algorithm, and the pyRDDLgym environment provides a foundation for further exploration and development in the field of Task and Motion Planning.

Background

In this section, we review some of the necessary background on task and motion planning problems, RDDL language, and the deep reinforcement learning algorithm used in the project.

Task And Motion Planning

Task and Motion Planning (TAMP) (Garrett et al. 2021) seamlessly integrates high-level, discrete task planning with low-level, continuous motion planning, forming a foundational framework for autonomous decision-making in robotics.

In TAMP, discrete task planning involves symbolic representation and decision-making processes to determine action sequences within a finite state space. These actions, ranging from opening doors to manipulating objects, constitute the abstract layer of TAMP.

Concurrently, continuous motion planning focuses on orchestrating the physical movement trajectory of the robot, addressing challenges such as obstacle avoidance and path planning. This component deals with the nuances of navigation and manipulation in continuous space.

This fusion of discrete and continuous planning elements in TAMP provides a formal framework for addressing complex robotic challenges, bridging the gap between high-level decision-making and low-level action execution in autonomous systems.

RDDL

The Relational Dynamic Influence Diagram Language (RDDL) (Sanner 2010) is a robust language in AI planning that provides a comprehensive framework for defining state transitions, actions, and observations. It is structured into several sections, each serving a specific purpose in defining the planning problem.

RDDL files are divided into domain and instance files. The domain file describes the general characteristics of the problem, while the instance file specifies a particular instance of the problem.

The domain file consists of the following sections:

- **'types'**: This section defines the object types in the domain. These types can be used to parameterize state variables, action variables, and non-fluents.
- **'pvariables'**: This section declares parameterized state variables, action variables, and non-fluents. State variables represent the state of the world, action variables represent the actions that can be taken, and non-fluents represent static properties of the world.
- **'cpfs'**: This section describes the transition dynamics of the state variables. It defines how the state of the world changes in response to actions.
- **'reward'**: This section specifies the immediate reward function. It defines the immediate reward or cost received after taking an action in a particular state.
- **'state-action constraints'**: This section defines the constraints on the state and action variables. It specifies the legal actions in each state.

The instance file includes the following sections:

- **'objects'**: This section declares the objects of each type defined in the domain file.
- **'non-fluents'**: This section specifies the values of non-fluent variables. These values are assumed to be static and known.
- **'init-state'**: This section defines the initial state of the problem. It specifies the values of the state variables at the start of the planning problem.

RDDL's strength lies in its ability to accommodate both discrete and continuous scenarios seamlessly, making it well-suited for tasks involving robotic systems, where hybrid actions, featuring both discrete and continuous components, are common. Its expressive nature allows for the encapsulation of uncertainties and dynamic changes, enabling planners and learning algorithms to navigate these intricacies effectively.

In summary, RDDL serves as a powerful tool for formulating planning problems involving robots operating in environments with diverse and multifaceted characteristics. Its versatility and expressive capabilities continue to fuel innovations in the domain of AI planning by providing a robust foundation for addressing the challenges posed by dynamic and uncertain environments.

JaxPlan Algorithm

JaxPlan algorithm (Gimelfarb, Taitler, and Sanner 2024) handles complex, hybrid nonlinear domains by blending Tensorflow for symbolic computation and gradient-based optimization. Its strength lies in tackling complex planning tasks, excelling particularly in high-dimensional continuous action spaces.

This algorithm approaches planning in hybrid domains by optimizing cumulative rewards over a decision horizon, incorporating both discrete and continuous elements. The problem setup involves hybrid states (S) containing mixed discrete and continuous state vectors, actions (A) with constraints (C), and functions governing reward (R) and state transitions (T).

In this domain, each time step involves:

- s_t: The mixed discrete and continuous state vector at time t.
- a_t: The mixed discrete and continuous action vector at time t.
- $R(s_t, a_t)$: The reward function providing a non-positive reward at each step.
- $T(s_t, a_t)$: The function guiding state evolution over time.

The algorithm focuses on optimizing actions $(a_1, ..., a_{H-1})$ to maximize the accumulated reward value V. This is achieved through backpropagation, reversing the conventional training paradigm to optimize inputs (actions) with fixed parameters (transition and reward parameterization). The optimization process employs Mean Squared Error (MSE) as the loss function, facilitating efficient convergence, especially in large-scale neural networks. Batch optimization, involving running multiple instances in parallel, aids in exploring diverse solution spaces, mitigating the risk of local minima due to the non-convex nature of transition and reward functions.

The JaxPlan algorithm manages stochasticity through clever parametrization techniques, like the reparameterization trick, introduced in (Bueno et al. 2019). By transforming stochastic variables using deterministic functions, it ensures smooth gradient computations, crucial for efficient optimization in uncertain environments.

Domain definition

The Robotic Arm domain replicates a simplified environment mimicking a robot's operations in a shelf setting. It involves an abstract representation of a robot manipulating cans on shelves, inspired by real-world robotic tasks (figure 1). The domain comprises three primary components: the

arm, cans, and shelves. In this domain, two main areas are distinguished: the working and safe position. The working position is the actual shelves and in the safe position, the arm can change from one shelf to another.
[1]

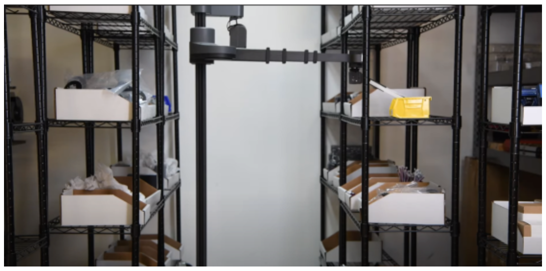

Figure 1: Real world robotic arm

State Definition

The state variables encompass spatial characteristics and task-related indicators. These include the dimensions and boundaries of shelves, sizes of objects, and positional attributes of the arm and cans. Additionally, relationships are included as can placement on shelves, the arm holding a can, and the arm's working position.

Actions

The domain involves both discrete and continuous actions. Discrete actions enable specific manipulations, including movements, shelf interactions, can operations, and shelf changes. Continuous actions include the movements of the arm in both dimensions. These actions allow the arm to perform precise movements, interact with objects, and navigate the environment.

Task Objective

The primary objective of the robotic arm within this domain is to consolidate all the cans onto a designated shelf. This involves strategic decision-making to select the appropriate shelf and precise movements to transfer the cans effectively.

Constraints

Various constraints govern the arm's actions to ensure safe and efficient operations. Constraints include limitations on arm movements within shelf boundaries, avoidance of collisions with objects, and specific conditions for can interactions, such as lifting and placement criteria.

Modeling and formalization

The domain previously defined is formalized in RDDL, in this section we describe the most relevant sections of the domain file.

[1]https://www.youtube.com/watch?v=srAzGAcahTA

Actions

In our domain, we can find the following actions. *move_x* and *move_y*: real value action that moves the robotic arm a certain amount in the x and y axis respectively. *retract-off-shelf* and *extend-to-shelf* boolean actions that move the arm from the working position to the safe position or vice versa. *pick-up* and *put-down* boolean actions that make the arm pick up or put down a can. *change-shelf* boolean action that makes the arm change from one shelf to another.

Intermediate fluents

The intermediate fluents do not belong to the domain, they were created to compute certain preconditions or to be used as auxiliary functions to make the implementation easier and more readable. Among these fluents we can find the following. *safe-position* boolean fluent that tells whether the arm is in the safe position. *arm-free* returns if the arm is holding a can. *break-extension* and *break_x* check if the arm collides with a can when moving forward (*break-extension*) or moving horizontally (*break_x*). These fluents are the most complex ones as there are so many different cases to be considered and the implementation in RDDL is not trivial at all. For instance, it has to be checked the case where the arm collides with another can when it is not holding a can and the case where it is holding a can. This second case can be divided into subcases depending on the position of the can being held by the arm with respect to the can to be avoided when moving and also the size of both cans.

put-down-conditions and *pick-up-conditions* check if the arm can put down or pick up a can. The latter is a rather complex fluent as it has to check that the can to be picked up lies within some boundaries so the arm is close enough and in a position where it can pick up the can and also it has to check that no other can is in between the arm and the can to be picked up. Finally *different-can* just checks whether two can are the same one.

State fluents

State fluents are the fluents forming the states of the domain, we can find the following. *on-shelf* boolean fluent that determines whether a can is on a shelf. *holding* tells if the arm is holding a can. *working-shelf* is a boolean fluent that returns true if the arm is working on a specific shelf. *x_position_a*, *y_position_a*, *x_position_c*, *y_position_a* are real value fluents that define the coordinates of the cans and the arm.

Reward

The reward is just defined as 1 if a goal state is reached, that is, if all the cans are placed on the same shelf, and 0 otherwise.

Stochastic version

Expanding upon the deterministic planning domain, this section delves into the stochastic iteration of the robotic arm environment. Here, uncertainties in action execution are introduced to mimic real-world variations and complexities encountered during task execution. The conceptual framework and formalization of this stochastic domain are detailed below.

Conceptual model

The stochastic rendition retains the core characteristics of its deterministic counterpart while integrating probabilistic elements to encapsulate uncertainties linked with the robot's actions. Notably, actions involving picking up cans and arm movements are adjusted to account for the inherent variability in execution.

In the stochastic domain, the success probability of the arm picking up a can is contingent upon the distance between the arm and the can itself. The likelihood of success diminishes as the distance increases, reflecting the scenario where grasping a closer can is more probable due to enhanced accuracy.

Additionally, arm movements in the stochastic domain follow a stochastic process. Rather than deterministic positional updates, the new position post-movement action is sampled from a normal distribution. This distribution has a mean equivalent to the previous position plus the intended movement, with a variance of 1. This incorporation introduces variability in movement execution, mirroring the inherent uncertainties in motion control.

Formalization

The formalization of the stochastic domain entails integrating these probabilistic aspects into the existing RDDL formalization. These adaptations primarily involve modifying the success probabilities of actions and determining new positions post-movement actions.

The incorporation of probabilistic success probabilities and stochastic movement processes enhances the domain's realism. This augmentation facilitates a more accurate representation of uncertainties in action execution, thereby broadening the domain's applicability and adaptability to real-world scenarios.

Implementing Baseline Algorithm

In this section, we present how the JaxPlan can be applied to our arm domain. The algorithm utilizes a recurrent neural network (RNN) (Medsker and Jain 2001) that operates based on the differentiability of state and reward functions. However, the functions expressed in RDDL lack differentiability, posing a critical challenge to the algorithm's implementation. To address this limitation, Fuzzy Logic techniques are employed, transforming non-differentiable RDDL functions into differentiable forms. This adaptation enables the successful utilization of the RNN-based algorithm within the Robotic Arm domain, overcoming the hindrance posed by non-differentiability.

Fuzzy Logic Transformations

To facilitate the transition from non-differentiable RDDL functions to differentiable forms, a weight parameter is introduced, multiplying the input of the fuzzy logic transformations (Zadeh 1988). This parameter allows for a controlled adjustment of the impact of each function, enabling a smooth transition from discrete to continuous representations. Some transformations examples are shown:

- Greater Than or Equal To ($\geq$): *sigmoid(a - b)*
- Greater Than ($>$): *sigmoid(a - b)*

Test Domain

To assess the impact of the weight parameter on the algorithm's performance, a simpler domain was constructed. This domain represents a bi-dimensional space of size 11x11, where the agent's objective is to navigate from the bottom left corner to the top right corner within the space. The RDDL code for the transition function for the position of the agent and the reward is the following:

```
x_position_a' = if((x_position_a +
    x_motion) > MAX_X | (x_position_a
     + x_motion < 0)) then
    x_position_a else x_position_a +
    x_motion;
y_position_a' = if((y_position_a +
    y_motion) > MAX_Y | (y_position_a
     + y_motion < 0)) then
    y_position_a else y_position_a +
    y_motion;
};

reward = (x_position_a' >=
    x_position_c ^ y_position_a' >=
    y_position_c);
```

By applying the fuzzy logic transformations to the state update functions and the reward function, we obtain the following continuous functions:

$$\begin{aligned}
&x'a(x_a, x_m, X\text{MAX}) = cx \cdot x_a + (1 - cx) \cdot (x_a + x_m)\\
&y'a(y_a, y_m, Y\text{MAX}) = cy \cdot y_a + (1 - cy) \cdot (y_a + y_m)\\
&r(x'_a, y'_a, x_c, y_c, x_m, y_m) = \text{sigmoid}(x'_a - x_c) +\\
&\text{sigmoid}(y'_a - y_c) - 0.1 \cdot (\text{sech}^2(-x_m) + \text{sech}^2(-y_m))
\end{aligned}$$

where y'_a is the next y position of the arm, y_a is the current y position, y_m is the motion in the y direction, Y_{MAX} is the maximum y coordinate that the arm can reach, x_c and y_c are the x and y positions of the target can, and cx and cy are given by

$$\begin{aligned}
cx &= \text{tconorm}\left(\text{sigmoid}\left((x_a + x_m) - X_{\text{MAX}}\right),\right.\\
&\quad \left.\text{sigmoid}\left(0 - (x_a + x_m)\right)\right)\\
cy &= \text{tconorm}\left(\text{sigmoid}\left((y_a + y_m) - Y_{\text{MAX}}\right),\right.\\
&\quad \left.\text{sigmoid}\left(0 - (y_a + y_m)\right)\right)
\end{aligned}$$

The weight parameter's influence on the continuous functions and their derivatives directly affects the algorithm's performance. Observations reveal that exceedingly high values of this parameter result in derivative functions with steep curves, rendering them unsuitable for Stochastic Gradient

Descent (SGD) optimization. As depicted in figure 2, such elevated parameter values yield derivatives that are impractical for effective optimization, posing challenges for the algorithm's convergence.

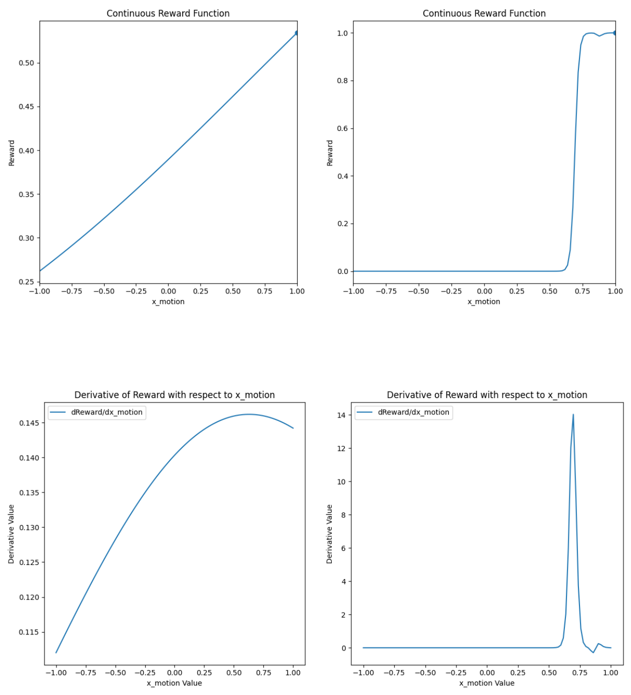

Figure 2: State functions and its derivatives

Experiments

In this section, we explore the application of the JaxPlan algorithm within the original arm domain. Our focus is on comprehensively analyzing the influence of various hyperparameters on the algorithm's performance. This involves meticulous tuning of weights, learning rates, epochs, and other crucial factors, providing insights into their impact on both effectiveness and efficiency.

Hyperparameters and Experimental Design

We detail the critical hyperparameters dictating the algorithm's behavior within the Task and Motion Planning (TAMP) domain and present the rationale behind their selection. The experimental design employed for evaluation is elucidated as follows:

Hyperparameters The hyperparameters under scrutiny are:

- Weight: Controls the steepness of fuzzy logic function transformations.
- Learning Rate: Influences convergence speed and stability.
- Epochs: Defines the number of iterations.

Experimental Design We meticulously designed systematic experiments for each combination of hyperparameter values. The selected ranges for hyperparameters are as follows:

- Weight: 0.005, 0.05, 0.1, 1.0, 10.0
- Learning Rates: 0.01, 0.1, 1.0
- Epochs: 100, 1000, 10000

These ranges were chosen based on relevance and effectiveness in similar scenarios. The algorithm was executed for each combination across four distinct instances of the arm domain. The discount factor was set to 0.99 and the rollout horizon to 200.

Arm Domain Instances

Four arm domain instances of varying complexity were selected to evaluate the algorithm's performance. In the following images, the black square represents the arm, the red ones the cans and the brown squares represent the shelves:

Instance 0 *Description:* Relatively simple scenario with three cans distributed across two shelves.

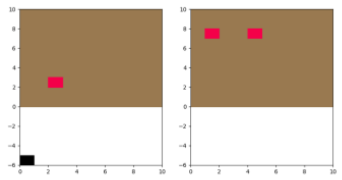

Figure 3: Visualization of instance 0

Instance 1 *Description:* Slightly more complex setup with five cans distributed across two shelves.

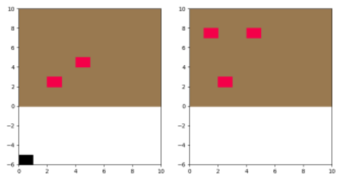

Figure 4: Visualization of instance 1

Instance 2 *Description:* Features two cans distributed on separate shelves, offering an interesting challenge due to symmetry.

Instance 3 *Description:* Higher complexity with four cans distributed across three shelves, requiring navigation through varying distances.

The evaluation across these instances with diverse hyperparameter settings provides comprehensive insights into the algorithm's behavior under varying conditions.

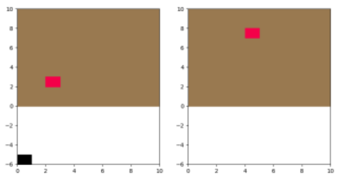

Figure 5: Visualization of instance 2

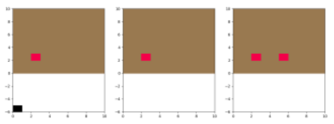

Figure 6: Visualization of instance 3

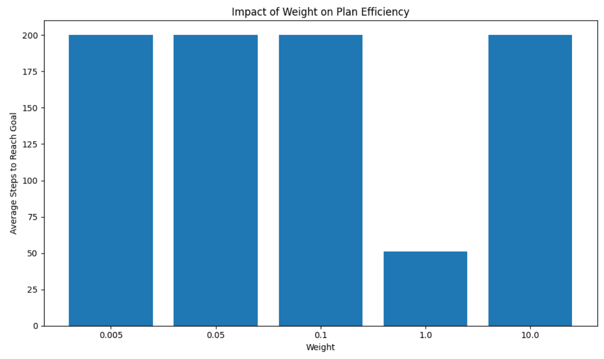

Figure 7: Weight parameter influence on performance

Results and Analysis

This section presents the outcomes of the algorithm's performance across diverse hyperparameter configurations in the original arm domain.

Hyperparameter Analysis Across our experiments, the weight hyperparameter emerged as the key influencer on the algorithm's performance within the arm domain. While other hyperparameters were explored extensively, their variations showed minimal impact on the algorithm's efficacy.

Learning Rate, Epochs, and Horizon: Despite altering these parameters within reasonable ranges, their variations did not notably influence the algorithm's performance. The diverse settings tested showed limited impact on plan efficiency and convergence, suggesting a relatively stable algorithm response.

Weight Parameter Significance: In stark contrast, the weight parameter significantly influenced the planning process. Varied weight values produced distinct outcomes, with a weight of 1.0 consistently yielding efficient plans across different hyperparameter configurations. Smaller or larger weights affected the steepness of fuzzy logic function transformations, altering the behavior and convergence of the algorithm noticeably.

This observation highlights the weight parameter's pivotal role in the algorithm's performance optimization, overshadowing the relatively negligible impact of other hyperparameters. This can be observed in figure 7 where each bar represents one value of the weight hyperparameter and the y-axis shows the average length plan obtained with that value (200 means no plan reaching the goal was found).

Experiments conclusion

The best model found during the experimentation was the following:
Weight: [1.0], **Learning Rate**: [0.1], **Epochs**: [10000], **Horizon**: [200], **Batch Size**: [128]

As depicted in Figure 8, the algorithm's performance with the best hyperparameters found exhibits variability across different instances within the TAMP domain. Each bar within the plot represents a distinct instance, while the y-axis signifies the frequency of successful completion for each instance during the experiments. Notably, the results highlight that instance 2 posed the least challenge, being solved more frequently compared to other instances. Conversely, instance 3 presented a significant challenge, as the algorithm failed to solve it within the experimentation framework.

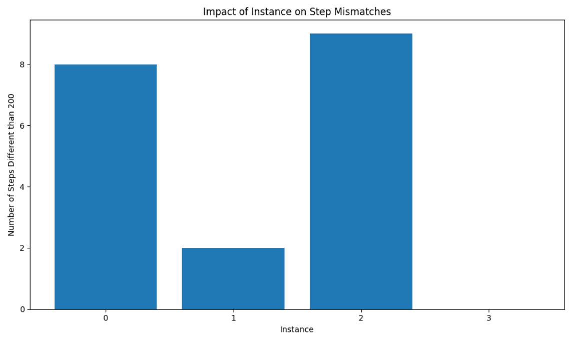

Figure 8: Instances results

These outcomes underscore the algorithm's capability to solve specific instances effectively while facing difficulties with more complex scenarios. While the algorithm demonstrated success in simpler configurations, it struggled or failed outright in more intricate instances. This emphasizes the nuanced nature of the TAMP domain, showcasing that while solvable, certain configurations pose considerable challenges for current planning algorithms. This insight contributes to understanding the algorithm's limitations and the complexities inherent in solving TAMP problems comprehensively.

The determination of this model was informed by a comprehensive evaluation of numerous experimental outcomes. However, an examination of the plans derived using this best model reveals a noteworthy issue – each state transition often involves the execution of multiple actions. This phenomenon initially posed a challenge due to the fact that the original domain formalization (RDDL code) did not account for this multi-action execution scenario. The underlying cause of this occurrence can be attributed to the nature of the planner algorithm and its gradient ascent-based approach.

Stochastic version results

We applied the algorithm, optimized for the deterministic domain, to its stochastic counterpart. Despite its success in solving the deterministic setup, the algorithm consistently failed to reach the goal state across all instances in the stochastic domain.

This limitation exposes challenges in adapting planning algorithms to uncertainties. It suggests a need for specialized optimization techniques tailored for complex probabilistic landscapes. While revealing the algorithm's shortcomings, these findings offer crucial insights for advancing planning algorithms in stochastic domains.

We believe that the reasons behind the failure of the algorithm are related to the algorithm itself not being able to find a solution and not to the formalization of the domain as it has been checked that the stochastic modeling was correctly made.

Related Work

The integration of Task and Motion Planning (TAMP) in robotics has been a subject of extensive research. The combination of high-level symbolic representations of tasks with low-level geometric constraints to generate feasible plans has been a notable line of research (Khodeir, Agro, and Shkurti 2023; Alami 2005; Hepburn and Montana 2022). These approaches aim to handle the complex continuous spaces inherent in TAMP domains, facilitating efficient exploration of feasible paths.

The JaxPlan algorithm has been used in hybrid nonlinear domains for planning (Beeson and Montana 2022). This algorithm leverages the power of recent advances in gradient descent with highly optimized toolkits like Tensorflow. It has shown promising results in terms of scalability and performance in various planning tasks.

The integration of reinforcement learning with planning techniques in robotics has also been a significant area of research (Ibarz et al. 2021). Reinforcement learning has been recognized as a successful method for performing low-level robot control under noisy conditions (Eppe, Nguyen, and Wermter 2019). The integration of action planning with model-free reinforcement learning has been explored, with reward sparsity serving as a bridge between the high-level and low-level state and action spaces.

In conclusion, while there has been considerable research in the areas of TAMP (Zhang et al. 2022), robotic arm operation, the use of the JaxPlan algorithm, and the integration of reinforcement learning with planning techniques, our study introduces a novel TAMP domain implemented in RDDL. This domain, centered around a robotic arm operating within a shelf environment, integrates reinforcement learning with the JaxPlan algorithm, offering valuable contributions to autonomous robotics.

Conclusions

In conclusion, this study represents a substantial stride in the realm of autonomous robotics by introducing a novel Task and Motion Planning (TAMP) domain in the Relational Dynamic Influence Diagram Language (RDDL). The creation of this domain serves as a pivotal contribution, offering a comprehensive framework for modeling both discrete and continuous actions within uncertain and dynamic environments.

The experiments conducted within this domain underscore the challenges of planning and decision-making, revealing nuances in algorithmic performance across varied hyperparameter settings. Furthermore, the insights gained from the application of planning algorithms within this domain illuminate the intricate interplay between uncertainties, planning methodologies, and the efficacy of autonomous systems.

The inability of established planning algorithms to seamlessly handle uncertainties, particularly in stochastic variants of the domain, emphasizes the need for further advancements in planning techniques. This emphasizes the critical role played by the newly introduced TAMP domain in guiding future research endeavors towards enhancing planning algorithms capable of effectively addressing uncertainties in real-world robotic applications.

By providing a structured platform for simulating complex robotic scenarios, the RDDL-based TAMP domain stands as a cornerstone for advancing planning and decision-making strategies in autonomous robotics. Its utility in evaluating and fine-tuning algorithms within dynamic and uncertain environments serves as an invaluable asset for researchers and practitioners alike, fostering progress in the pursuit of agile and adaptive robotic systems.

Acknowledgements

We extend our thanks to Peter Stone, from the University of Texas at Austin for his initial contributions to this project and for providing valuable feedback along the way. His insights and guidance have been greatly appreciated during the development process.

References

Alami, R. 2005. A Robot Task Planner that Merges Symbolic and Geometric Reasoning. *Springer Tracts in Advanced Robotics.*

Beeson, A.; and Montana, G. 2022. Improving TD3-BC: Relaxed Policy Constraint for Offline Learning and Stable Online Fine-Tuning. arXiv:2211.11802.

Bueno, T. P.; de Barros, L. N.; Mauá, D. D.; and Sanner, S. 2019. Deep reactive policies for planning in stochastic nonlinear domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7530–7537.

Eppe, M.; Nguyen, P. D. H.; and Wermter, S. 2019. From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving. arXiv:1905.09683.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems.*

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2020. PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning. arXiv:1802.08705.

Gimelfarb, M.; Taitler, A.; and Sanner, S. 2024. JaxPlan and GurobiPlan: Optimization Baselines for Replanning in Discrete and Mixed Discrete and Continuous Probabilistic Domains. In *34th International Conference on Automated Planning and Scheduling*.

Hepburn, C. A.; and Montana, G. 2022. Integrating Symbolic and Geometric Planning for Mobile Manipulation. *arXiv preprint arXiv:2211.11603*.

Ibarz, J.; Tan, J.; Finn, C.; Kalakrishnan, M.; Pastor, P.; and Levine, S. 2021. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4–5): 698–721.

Khodeir, M.; Agro, B.; and Shkurti, F. 2023. Learning to Search in Task and Motion Planning With Streams. *IEEE Robotics and Automation Letters*, 8(4): 1983–1990.

Medsker, L. R.; and Jain, L. 2001. Recurrent neural networks. *Design and Applications*, 5(64-67): 2.

Sanner, S. 2010. Relational Dynamic Influence Diagram Language (RDDL): Language Description. Http://users.cecs.anu.edu.au/ ssanner/IPPC$_2011/RDDL.pdf$.

Zadeh, L. 1988. Fuzzy logic. *Computer*, 21(4): 83–93.

Zhang, X.; Zhu, Y.; Ding, Y.; Zhu, Y.; Stone, P.; and Zhang, S. 2022. Visually Grounded Task and Motion Planning for Mobile Manipulation. arXiv:2202.10667.