**PRobPlan: A Framework of Integrating Probabilistic Planning Into ROS**

**DONGNING RAO¹, GUODONG HU¹, AND ZHIHUA JIANG²**

¹School of Computer, Guangdong University of Technology, Guangzhou 510006, China  
²Department of Computer Science, College of Information Science and Technology, Jinan University, Guangzhou 510632, China  

Corresponding author: Zhihua Jiang (tjiangzhh@jnu.edu.cn)

This work was supported by the projects of the Natural Science Foundation of Guangdong Province, China, under Grant 2016A030313084 and Grant 2016A030313700.

**ABSTRACT** The ROS (Robot Operating System) is a set of software libraries and tools used to build robotic systems. Recently, some researchers have been working on integrating planning systems into ROS. However, robotics domains are often probabilistically interesting. At present, only a few studies focused on integrating probabilistic planning into ROS. Moreover, they have limits on flexibility or scalability. Therefore, we propose a new framework, called **PRobPlan** (PRobabilistic Robotic Planning), to alleviate these problems. On one hand, our framework uses a series of generation programs to build a problem file instead of the knowledge base. This makes it more modifiable. On the other hand, our framework integrates a state-of-the-art planner, SOGBOFA, and enhances it with a new recommendation function. This makes it more scalable. We instantiate the proposed framework in a warehouse-robot domain where mobile robots are allocated tasks of fetching or packing goods. Both of the domain and problem instances are modeled with RDDL (Relational Dynamic Influence Diagram Language). Experimental results showed the effectiveness of the proposed framework. We also tested two top planners, SOGBOFA and PROST, for comparison. The enhanced version of SOGBOFA mildly outperformed its original version and vastly outperformed PROST. Our work promotes the integration of probabilistic planning with robotic systems.

**INDEX TERMS** Robot operating system (ROS), probabilistic planning, relational dynamic influence diagram language (RDDL).

**I. INTRODUCTION**

The ROS (Robot Operating System) [1], [2] is a flexible framework for writing robot softwares. It provides many efficiently implemented components and encourages collaborative robot software development. For example, the MoveIt! [3] is for robot motion planning and the Navigation [4] is for robot path planning. However, robot task planning is higher-level than robot motion or path planning [5], [6]. For instance, a robot can first receive a task (e.g., do something or go somewhere) and then manipulate its arms or follow a navigation route to finish the task. Therefore, it is promising to integrate global planners into ROS.

ROSPlan is a popular framework for integrating deterministic and contingent planning into ROS [7]. It includes two ROS nodes: KB (Knowledge Base) and PS (Planning System). The knowledge base is intended to collate the up-to-date model of the environment. The planning system acts as a wrapper for the planner, and also dispatches the plan. ROSPlan uses PDDL2.1 (Planning Domain Definition Language, version 2.1) [8] to describe planning domains and problem instances. Theoretically, any PDDL2.1 planner can be integrated into ROS. However, only considering deterministic planning is not enough because robotics domains are often probabilistically interesting [9]. For instance, the outcomes of actions can have a reasonable chance of failure. As mentioned in Canal et al. (2019) [10], one drawback of ROSPlan is that it has not used probabilistic planners within the framework.

Therefore, this paper works on integrating probabilistic planners into ROS. There are two existing studies most related to our work. First, Canal et al. (2019) [10] extended ROSPlan by handling RDDL (Relational Dynamic Influence Diagram Language) semantics. RDDL [11] is the official language to describe probabilistic planning domains used in IPPCs (International Probabilistic Planning Competitions).
However, their extension had to adhere to the existing KB interface to preserve compatibility with systems already using ROSPlan. This inevitably inherits the shortcomings of ROSPlan (e.g., does not support quantified or disjunctive conditions) and also endures the complexity of maintaining KBs (e.g., has to allow for the interchange of RDDL KBs and PDDL KBs). Second, Lacerda et al. (2019) proposed a probabilistic planning approach based on MDP (Markov Decision Process) models for service robot planning and execution [12]. But, their case study was very simple. There was only one mobile robot which served in the real environment. This was mainly because their MDP models were very complex and had a poor scalability.

To alleviate these problems, we propose a new framework, called PRobPlan (PRobabilistic Robot Planning). Similar to Canal et al. (2019), PRobPlan handles RDDL semantics, produces RDDL problem instances, and interfaces with any RDDL planner that can be used with RDDLSim (RDDL simulator)\(^1\) server used in IPPCs. However, the greatest difference between PRobPlan and Canal et al. (2019) is that PRobPlan is independent on ROSPlan. PRobPlan uses a suite of generation programs (written in Python) for deriving problem instances instead of ROSPlan KB. It is easier to modify programs than KB interfaces described with an ontology language. Besides, we propose a new recommendation function to improve embedded planners. This function recommends favored actions via taking advantage of RDDL-specific features.

For example, a gripper may fail to grasp an object because of air humidity or device malfunction. Also, multi robots can move in the warehouse simultaneously since they are autonomous. We formally described this domain with RDDL. We tested two state-of-the-art probabilistic planners, PROST (Probabilistic Planning Based on UCT) [14] and SOGBOFA (Symbolic Online Gradient Based Optimization for Factored Actions) [15], for comparison.\(^3\) Experimental results showed the effectiveness of PRobPlan. And an enhanced SOGBOFA mildly outperformed its original version and vastly outperformed PROST. We also ran this mission in the ROS simulator, Gazebo,\(^4\) to visualize plan generation and execution.\(^5\)

To sum up, the contributions of this paper are two folds:

- propose a new framework, PRobPlan, to integrate probabilistic planning into ROS;
- present a new recommendation function, which provides favored actions based on RDDL-specific features.

The rest of this paper is organized as follows. We introduce related work in Section II. We state the problem in Section III. Next, we give the overall architecture and the main algorithm of the proposed framework in Section IV. The example domain is introduced in detail in Section V. Experiments with analysis are shown in Section VI. Finally, we conclude along with future work in the last section.

II. RELATED WORK

A. ROS

ROS is a standard infrastructure for robot systems. Developers can use packages that abstract individual sensors or actuators, and messaging infrastructure on ROS.\(^6\) ROS relies on the concept of computational graphs [16] in Figure 2. A node is a process that performs computation. For example, movebase is a node that controls the robot navigation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and parameter server. Topics are named buses over which nodes exchange messages. Request-and-reply is done via a service. A providing ROS node offers a service under a string name, and a client calls the service by sending a request message and awaiting the reply message. A message is a simple data structure, comprising typed fields. A bag is a file format in ROS for storing ROS message data. ROS Master provides naming and registration services to the rest of nodes in ROS systems. It tracks publishers and subscribers to topics as well as services. A parameter server is a shared, multi-variate dictionary that is accessible via network APIs. Nodes use this server to store and retrieve parameters at runtime.

\(^1\)https://github.com/ssanner/rddlsim
\(^2\)http://pal-robotics.com/robots/tiago/
\(^3\)This was also for comparison between our framework and Canal et al. (2019). They only tested PROST in their experiment.
\(^4\)http://gazebosim.org/download
\(^5\)Our codes are open source at https://github.com/fip-lab/PRobPlan. The formal definition of the example domain is also provided. A recorded video which reveals the details of the simulation process is also available at the provided Github repository.
\(^6\)http://www.ros.org
ROS integrates a comprehensive set of libraries of open-source software components such as SLAM (Simultaneous Localization and Mapping), the MoveIt! and the Navigation. The MoveIt! is a library used for robotic arm manipulation. It provides us a plan that moves a robotic arm from one position to another and also takes charge of executing a plan. The kernel of the MoveIt! package is called move_group. It gets environment descriptions as configuration files. Then, it provides action services to user interfaces and calls controllers and sensors to realize these actions. The Navigation stack incorporates both global and local path planners to support ROS-enabled robot navigation. It uses sensor data as input. Then, the output including velocity commands is sent to a mobile base node. Its core component is called move_base, which links together planners to accomplish the navigation task.

B. PROBABILISTIC PLANNING

AI planning, also called automated planning, studies how to design information processing tools that give access to affordable and efficient planning resources [18]. At the beginning, the community focused on classical planning where there were eight assumptions to simplify problems. Soon, non-classical planning, such as probabilistic planning [19], [20], got great attention from researchers. Like other fields in AI, primary studies in automated planning have been focused on knowledge representation and problem solving.

For knowledge representation, formal planning description languages have been defined. PDDL is the most popular planning language for deterministic planning [21]. It was proposed for the first IPC (International Planning Competitions) in 1998. The probabilistic version of PDDL is called PPDDL (Probabilistic PDDL) [22]. However, PPDDL is inadequate for describing planning problems with high uncertainty and concurrence [11]. Therefore, RDDL was proposed. Its greatest feature is to describe domain dynamic with transitions functions (conditional probability functions or CPFs) instead of effect-based actions. RDDL has been the official language for all IPPCs from 2011 to now. Moreover, some researchers have proposed approaches for automatically acquiring RDDL domain descriptions [23].

For problem solving, many efficient and effective planning algorithms have been proposed. For deterministic planning, most approaches were based on heuristic searching [24] and SAT-based transformation [25]. While, for probabilistic planning, primary approaches were based on local searching and probabilistic reference. Existing RDDL planners use either iterative deepening searching, e.g., Glutton [26], or stochastic simulations, e.g., PROST [14]. Both of these two planners were top performers of the earliest IPPC, i.e., IPPC 2011. Recently, in IPPC 2018, almost all competitors were built upon the PROST framework, except for SOGBOFA [15], [27]. SOGBOFA is a planner which differs architecturally and algorithmically the most from PROST-based planners. It searches for the best action by calculating gradients on lifted computation graphs, which allows the planner to deal with large action spaces. Though PROST-DD [28] was the winner of IPPC 2018 at the official competition results, a subsequent report [29] showed that, based on updated planner versions after the competition, SOGBOFA turned to win PROST-DD in the benchmarks.

C. ROBOT TASK PLANNING

Researches on robot task planning have been active for many years. Tasks are similar to macro actions that treat several primitive actions as a compound to improve search efficiency [30]. Early studies often combined task planning with motion planning. A straightforward solution is the model-based reactivation method. For example, T-REX [31] is a timeline-based plan execution architecture. Based on reactors, it modifies state variables on different timelines. These reactors are built in physical components of robots. The system is synchronous and it enforces interdependencies between timelines. Complex hybrid systems use a deliberative high level, a reactive low level, and a synchronisation mechanism to mediate between the above two levels. For example, ROSco [32] and Smach [33] use Hierarchical Finite State Machines.

Recently, integrating AI planning into ROS has gained attention of researchers. In 2015, ROSPlan [7] was first proposed for this. It generated a sequence of task operations to achieve the goal when the initial state of the environment was known. A re-plan mechanism was used to deal with dynamic real-world problems. It was successfully deployed on planning inspection and valve-turning missions.
TABLE 1. Deterministic planning frameworks integrated into ROS.

| Framework | Has KB? | Language | Planner | Action | Domain | Problem | Application |
|-----------|---------|----------|---------|--------|--------|---------|-------------|
| ROSPlan   | YES     | PDDL2.1  | POPF    | Operator | Defined manually | Generated automatically | UAV          |
| SkiROS    | YES     | PDDL2.1  | Fast Downward | Skill | Inferred from skill set | Generated automatically | STAMINA UAV  |
| RHBP      | NO      | PDDL2.1  | Metric-FP | Behavior | Defined automatically | Generated automatically | UAV          |

for Autonomous Underwater Vehicles (AUVs) [5]. But, there are still some limitations in ROSPlan. First, it only uses deterministic planning in the framework. Second, it only supports PDDL2.1 which has an imperfect match with real applications [34]. Many realistic features, such as derived predicates in PDDL2.2 [35] or plan constraints in PDDL3.0 [36], have not involved in the framework yet. Last but not least, it uses a fixed KB to parse domains and generate problem files. As a result, once we apply the framework into a new domain, the KB interfaces have to be modified accordingly.

Other representative frameworks also include SkiROS [37] and RHBP (ROS Hybrid Behaviour Planner) [38]. SkiROS consists of three ROS nodes - task manager, skill manager and world model. Similar to ROSPlan, SkiROS uses the Web Ontology Language (OWL) standard for KB and the PDDL standard for the planner. A skill is an action with pre-conditions and post-conditions that can be concatenated to form a complete task. The planning domain can be automatically inferred by a reasoner at run-time from the available skill set. However, developing and maintaining a large skill set can be an overwhelming task. RHBP is a hybrid system with tight ROS integration. It takes advantage of both, reactive behavior approaches, being fast and more flexible, and symbolic planning, being more goal-directed. The target architecture includes a centralized meta-level symbolic planner that provides sub-goals for individual robots. Each robot has an own symbolic planner that influences its lower level behavior network. The experiment result showed the reduction of planning steps with comparison of pure behavior networks.

All of the three frameworks introduced above work on integrating deterministic planning into ROS. Table 1 gives more details for comparison. It should be mentioned that, RHBP automatically generates the required plan descriptions (domain and problem) for the PDDL-based planner via directly connecting input sensors to particular output actuators. Besides, the definition of actions is slightly different in each framework. In ROSPlan, an operator is like a macro action; in SkiROS, a skill is a group of primitives; in RHBP, a behavior is a reactive action in the behavior network.

Each framework tested different planners in its own case study. However, none of them provided the support for more updated PDDLs apart from PDDL2.1.

Next, we summarize existing studies which integrate probabilistic planning into ROS, along with comparison with our work. As shown in Table 2, on one hand, Lacerda et al. (2019) [12] is very different from the other two since it mainly relies on MDP models where state transitions are used to express the domain dynamic instead of explicit actions. On the other hand, Canal et al. (2019) [10] and PRobPlan (our work) share much on: supporting RDDL specification, calling RDDL planners, and generating problem instances automatically. However, the biggest difference between these two frameworks is whether to depend on ROSPlan or not. Canal et al. (2019) is an extension of ROSPlan, while our work breaks away from it. We will state the reason in the following paragraph. Besides, we adopt RDDL2.0, which contains more features such as action preconditions, than RDDL1.0. And we mainly use SOGBOFA, which conforms to our recommendation function most. Since all compared frameworks have to provide a manually-defined domain description, we omit the Domain column in Table 2. Further, it should be mentioned that, one distinct contribution of Lacerda et al. (2019) is to propose probabilistic verification techniques for generating optimal policies with formal performance guarantees.

Last, we explain the reason why not to follow ROSPlan in our framework. Canal et al. (2019) extended ROSPlan and adhered to the existing KB interfaces for two reasons: to preserve compatibility with systems already using ROSPlan, and to allow for the interchange of RDDL and PDDL KBs [10]. Therefore, the RDDL KB translates the RDDL domain and problems to PDDL-like structures. For doing this, action-fluents are mapped to PDDL operators, and state-action constraints are encoded as PDDL preconditions. Thus, a PDDL instance file can be generated from a RDDL instance and vice versa. Given that RDDL is more expressive than PDDL, this interchange between RDDL KBs and PDDL KBs will hinder the full provision of RDDL-specific functionality. Besides, developing and maintaining the KB interfaces is time-consuming and error-prone.
This section introduces the formal definition of probabilistic planning domains studied in this paper. An example domain will be given in Section V after the introduction of the overall architecture and algorithms of the proposed framework.

Traditionally, a probabilistic planning domain can be modeled as a MDP problem.

**Definition 1:** A Markov Decision Process (MDP) is specified by a tuple \( \Sigma = (S, A, T, R, \gamma) \) [39]

1) \( S \) is a finite state space;
2) \( A \) is a finite action space;
3) \( T(s, a, s') = p(s'|s, a) \) defines the transition probabilities;
4) \( R : S \times A \rightarrow \Re \) is the immediate reward of taking action \( a \) in state \( s \);
5) \( \gamma \) is the discount factor.

The solution of MDP is a policy \( \pi : S \rightarrow A \) mapping from state \( s \) to action \( a \), indicating which action to choose at each state. Given a policy \( \pi \), the value function \( V^\pi : S \rightarrow \Re \) is the expected discounted total reward, where \( s_i \) is the \( i \)th state visited by following \( \pi \) and \( s_0 \) is the initial state. The action-value function \( Q^\pi : S \times A \rightarrow \Re \) is the expected discounted total reward when taking \( a \) at \( s \) and following \( \pi \) thereafter. MDPs with large state and action spaces are typically represented in factored form. The state space \( S \) is specified by a set of binary variables \( x_1, \ldots, x_m \) so that \( |S| = 2^m \). Similarly, \( A \) is specified by a set of binary variables \( y_1, \ldots, y_n \) so that \( |A| = 2^n \). Unlike a classical planning problem, a probabilistic planning problem often has not any predefined goal state. The agent’s objective is to select actions to maximize a long-term measure of total reward. So, such a problem is an finite horizon factored MDP problem.

However, it is not natural to describe planning domains as MDP models. Therefore, many specific planning definition languages have been proposed to write domain descriptions. RDDL [11] is a standard planning definition language for probabilistic planning. Its semantics is a ground DBN (Dynamic Bayes Net). The features of RDDL are as follows.

- Everything is a variable, including states, actions and observations.
- A variable is either a fluent or a non-fluent. The truth value of a fluent varies from state to state, while that of a non-fluent never changes.
- There are no explicit action effects in RDDL. The change of a state is described with conditional probabilistic functions.
- The reward is either a Bernoulli distribution or an arithmetic expression of quantifiers and aggregate functions.
- Constraints are very flexible and can be put on states or actions. They allow arithmetic or relational operations, implications, quantifiers, even functional calls and so on.

**Definition 2:** A RDDL domain is defined by a 6-tuple \( \Sigma = (\text{Req}, N, \text{CPF}, C, R, I) \) [11]

1) \( \text{Req} \) is a list of domain requirements. It specifies the features used in the domain.
2) \( N \) is a set of names, including types and variables.
3) \( \text{CPF} \) is the set of conditional probabilistic functions, which describes how the value of a state fluent changes given the current state and the action set.
4) \( C \) is the set of constraints. A constraint is a logical or relational expression which consists of fluent or non-fluent.
5) \( R \) is the reward function, which is the evaluation metric.
6) \( I \) is the instance block. It includes the initial state, the max number of concurrent actions, the horizon and the objects.

The above definition was given in the earliest version of RDDL (i.e., RDDL1.0), which was used in IPPC 2011 and IPPC 2014. Later, a new version, RDDL2.0, was proposed in IPPC 2018. On one hand, RDDL2.0 adds new features, including action preconditions, intermediate variables and finite-domain variables. Action preconditions are components of PDDL-specific actions. Intermediate variables are derived predicates in PDDL2.2 [35]. Finite-domain variables are enum-valued variables in SAS+ planning [40]. On the other hand, RDDL2.0 omits the max-nondef-actions section, where the largest number of applicable actions is specified. As a result, RDDL2.0 supports the absolute concurrency and poses a real challenge to the competitors. In fact, all the competitors in IPPC 2018 only used some subsets of it.\(^{14}\)

**IV. METHOD**

**A. ARCHITECTURE OF PRobPlan**

Figure 3 shows the overall architecture of PRobPlan. We will explain it in three distinct levels.

First, the robot level. In a multi-robot environment, robots can communicate with available message packages, e.g., multimaster_fkie. Under most circumstances, a multi-robot SLAM node will provide a global map including the locations of all robots along with environment information. The slam_gmapping and the multirobot_mapmerge can basically do this job, but more advanced tools might bring better maps [41], [42]. When the communication is done, a single robot SLAM can plan for itself. As a result, each robot only carries out relevant actions to itself.

Second, the node level. Services in ROS are provided as nodes. Environment information can be acquired from SLAM nodes. From these nodes, a robot is able to sense the current state and then transfer it to the planning node. Then, the planning node will call a RDDL planner. Most RDDL planners are Linux based. Therefore the planning node uses a system call and waits for the resulting plan, as previous studies [6], [7]. The final plan is a sequence of task operations which can be translated to ROS action messages. For example, a goto

\(^{13}\)https://ipc2018-probabilistic.bitbucket.io/domains

\(^{14}\)The eight domains that were used in IPPC 2018 existed in four different yet equivalent versions. But, the original RDDL model, potentially including intermediate variables and finite-domain variables, was not used by any competitor. All competitors used either an equivalent RDDL model without intermediate variables or an equivalent RDDL model without finite-domain variables and with reformulated action preconditions. More details can be found on the competition website: https://ipc2018-probabilistic.bitbucket.io
operation will be sent to ROS nodes on the navigation stack. Besides, other operations, which have been registered as services, can be handled as well.

Third, the service level. PReqPlan does not call ROS services directly. It takes full advantage of existing ROS components. For example, sensors are called by SLAM nodes; motion controllers are used by the MoveIt!; path navigation is controlled through the Navigation stack. In this way, various sensors\(^{15}\) and other faculties\(^{16}\) can be facilitated without being reproduced in our framework.

It is sure that our framework is tightly bound to the Navigation and the SLAM because they are the most basic components of ROS. Softwares in ROS are organized in packages. While we install ROS, many packages will be installed automatically, including the navigation package\(^{17}\) and the gmapping package.\(^{18}\) The navigation package takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base. The gmapping package provides laser-based SLAM as a ROS node called slam_gmapping.\(^{19}\)

### B. MAIN PROCEDURE

First of all, we illustrate the function of planning node in PReqPlan, as shown in Figure 4. It includes the following steps:

- **Preprocessing.** Use a suite of generation programs based on sensed data to generate a RDDL problem file.

- **Planning.** Call a RDDL planner for solving the generated planning problem. The solution plan is a series of decisions on how to allocate tasks to robots.

- **Executing.** Each robot executes an allocated task by calling ROS components or other registered services.

The three steps above will be carried out sequentially in each iteration, which forms a loop called re-planning. There are three reasons that the system may re-plan: an action fails; a robot malfunctions (such as has no power) or recovers (such as recharges); a task enters or ends.

Algorithm 1 gives the main procedure of planning and execution in our framework. It is a wrapper for different services in ROS. It first initializes ROS nodes (line 1-2). Then, it acquires the initial state, the cost map, the goal specification and all constraints (line 3-4). Notice that, a tuple \(\langle x, y, \text{robot} \rangle\) indicates the position of a robot in the map, and a tuple \(\langle x, y, \text{task} \rangle\) indicates the position of a task. Next, a plan-and-execute loop is included (line 5-18). Because of the uncertainty, after executing a plan step, we should observe the concurrent environment, re-generate the problem file, and re-plan for the remaining tasks. Therefore, by calling the generate-problem subroutine, the framework gets a new RDDL problem file to express the

---

\(^{15}\)http://wiki.ros.org/Sensors
\(^{16}\)http://wiki.ros.org/Industrial/supported_hardware
\(^{17}\)http://wiki.ros.org/navigation
\(^{18}\)http://wiki.ros.org/gmapping
\(^{19}\)Indoor SLAM is often laser-based while outdoor SLAM can be based on a camera or GPS (global positioning system). If GPS is available, we can convert decimal GPS coordinates of waypoints to relative ROS position vectors, a 2D navigation goal in SLAM. Open source implementation for doing this can be found at https://gist.github.com/clungzta/c102e882588e220442cd94.
current state (line 6). The subroutine consists of a set of generation programs which produce different RDDL elements (see Algorithm 2 for the details). Along with the predefined domain file, the framework calls the RDDL planner to find a new plan (line 7). Next, we execute the first step of the resulting plan (line 8-17). Because of the concurrency, each plan step could be a set of tasks operated. If the operator is: a goto action, then we call the navigation stack; a do action, then we call the MoveIt!; other user-defined action, then we call some registered service. This loop will last until all tasks have been done or the given horizon (i.e., the maximal number of plan steps) is reached (line 18).

Instead of developing KB interfaces, we propose a set of generation programs to produce distinct RDDL components. Their functions are implied in their names. Algorithm 2 presents the detailed procedure of using these programs. First, the instance section is initialized, such as introducing domain name (line 1). Then, the dynamic system need to be addressed because of re-planning. Therefore, we need to produce correct object instances according to available robots and active tasks (line 2-3). Next, the positions of robots and the order of tasks are determined (line 4-5). The following steps are used to compute the cost of each operator, which is the negative part of the reward function (line 6-11). Similarly, the utility of each task is set, which is the positive part of the reward function (line 12-14). Finally, the horizon is reset (line 15).

The implementation of generation programs needs the interaction between the ROS master node and the planner node. All arguments of generating planning problems and running ROS are written into a configuration file. The master node and the planner node can access this file alternatively. First, the master node reads the file to get information of planning actions. These actions are mapped into ROS messages and sent to lower-level controllers. Then, when a check point arrives, the master node parses the returned messages, including the new locations of robots and the new list of tasks, and writes back into the configuration file. Also, it calculates new navigation costs since robot locations change. Last, the planner node reads the updated file and generates a new problem file. Therefore, the dynamic of a robot system can be captured continuously.

C. RECOMMENDATION FUNCTION
A recommendation function will provide favored actions when the planner fails to find out a reasonable action. The idea has a long history. For instance, PROST 2011 favored the action with the best expected outcome [14] and PROST-DD favored the action which is selected the most in the root node of the search [28]. For more information we recommend Keller and Helmert (2013) [43], as well as the PhD thesis of Keller [44] which introduced recommendation functions and contained a thorough theoretical and empirical evaluation of a multitude of algorithms.

We propose a new recommendation function to improve embedded planners in PRobPlan. The function is defined based on two considerations. On one hand, we favor an action which is sampled in the simulation for trial-based probabilistic algorithms. We can store all the information of a state trajectory, which includes the sampled action leading to the

---

**Algorithm 1** Main Procedure of PRobPlan

**Input:** initial state $I$, cost_map, goals $G$, constraints $C$

**Output:** a solution plan

1. Start ROS and initialize the robot environment
2. Subscribe ROS messages
3. Use cost_map and $I$ from multi-robot SLAM, e.g., mrpt_slam, to set $(x, y, robot)$ tuples
4. Use $G$ and $C$ to set $(x, y, task)$ tuples
5. repeat
6. Call generate_problem to generate problem file
7. Call a RDDL planner to generate plan $P$
8. Let $p = \text{top}(P)$
9. for each operation $o$ in $p$ do
   10. if $o$ is a goto action then
       11. Call ROS node based on the navigation stack
   12. else if $o$ is a do action then
       13. Call ROS node based on the MoveIt!
   14. else
       15. Call user-defined services
   16. end if
17. end for
18. until all tasks have been done or the given horizon is reached.

**Algorithm 2** The generate_problem Subroutine

**Input:** the sensed environment information

**Output:** a RDDL problem file

1. Initialize the instance block
2. Call generate_robot to generate robot instances according to available robots
3. Call generate_task to generate task instances according to active tasks
4. Use multi-robot SLAM to set the current positions of robots $(x, y, robot)$
5. Call generate_order to set the partial order of tasks
6. for each robot $r$ do
5. Call generate_do_cost to set the cost of executing do operator for $r$
8. end for
9. for each pair of locations $(l_1, l_2)$ do
10. Call generate_goto_cost to calculate the cost of moving between $l_1$ and $l_2$
11. end for
12. for each task $do$
13. Call generate_task_utility to set its utility
14. end for
15. Call generate_horizon to update the horizon
next state. If we really arrive the stored state, then the stored action should be applied with plausible reasoning. On the other hand, we favor an action group which includes most concurrent actions. Because of concurrency, these actions can be applied in the same time step. There are much more concurrent action groups in RDDL problems than non-RDDL problems. If we can use them frequently, the exploration of the action space could be sped up greatly. By the way, max-nondef-actions section in a domain description can be used to generate max-concurrent action groups. To sum up, we favor the first kind of actions because of the probabilistic nature of RDDL problems, and favor the second kind of actions because of the concurrent nature of RDDL problems.

In our framework, the recommendation function will work when the probe of a planner returns a no-op action. Such an action means doing nothing, which is a defaulted response to the time out. Sometimes, because of action explosion, a trial-based probabilistic algorithm fails to do a complete trial in the given time limit, thus throwing a no-op action. It is not beneficial for on-line planning. Therefore, we replace the no-op action with a favored action. Both of sampled actions and max-concurrent action groups can be collected into a set of favored actions. We can pick up the best one via some evaluation metric, or just choose one randomly for the consideration of efficiency. Some experimental results will be given in Section VI.

V. EXAMPLE DOMAIN
To evaluate the effectiveness of PRobPlan, we instantiate it in a warehouse-robot domain. An formal description of this example domain will be presented in this section. We use RDDL2.0 in this case study.

Example 1 (Robots in the Warehouse Domain): In a hypothetical warehouse, goods are placed on the shelves and robots need to move to the shelves and pick up desired goods. And there is a packing table near to the warehouse exit. Robots can drop down goods in the packing table and pack them into a box. Therefore, there are four kinds of actions for robots working in the warehouse. They are move, pick, drop and pack. Each kind of actions is described with a specific ability of robots. Besides, more details of this domain are listed as follows.

- Robots can execute tasks independently because there is no cooperation in this domain.
- The reward function is defined based on the trade-off between the utilities of all finished tasks and the costs of all executed actions.

Next, we give the formal description of the warehouse-robot domain. For the sake of space limitation, we only show key components of the domain description in this paper. A complete version can be found in the given github.

Example 2 (Requirements in the Warehouse-Robot Domain):

```plaintext
domain warehouse_robot_mdp {
requirements {
   reward-deterministic,
   precondition;
}
......
}
```

The requirement section indicates that this domain has a deterministic reward function and allows action-precondition specifications.

Example 3 (Names in the Warehouse-Robot Domain):

```plaintext
domain warehouse_robot_mdp {
......
types {
   robot: object;
   ability: object;
   location: object;
   task: object;
};
pvariables {
   // state fluents
   robot-at(robot, location): { state-fluent,
                             bool, default = false };}
......
// action fluents
   goto(robot, location): { action-fluent,
                            bool, default = false };}
   do(robot, ability): { action-fluent, bool,
                         default = false };}
......
// non-fluent
   TASK(task, ability, location) :
      { non-fluent, bool, default = false };
   GOTO_COST(location, location): { non-fluent, real, default = 0 };
   DO_COST(robot, ability): { non-fluent, real, default = 0 };
   TASK_UTIL(task): { non-fluent, real, default = 100 };
   MOVE_PROB(robot): {non-fluent, real, default = 0.8};
   DO_PROB(robot): {non-fluent, real, default = 0.8};
......
// intermediate fluents
   move-success(robot): { \text{intern-fluent },
                         bool, level = 1 };}
   do-success(robot): { \text{intern-fluent},
                      bool, level = 1 };}
......
}
```

VOLUME 8, 2020 106523
The name section includes types definitions and predicates declarations. There are four types of objects: task, robot, ability, and location. It should be mentioned that, ability is used to describe the ability of a robot to execute a task. In the variables subsection, there are seven non-fluents, two inter-fluents, three state-fluents, and two action-fluents totally. For instance, GOTO_COST is a non-fluent which defines the navigation cost between two locations; do_success is an intermediate fluent which determines whether the do action is executed successful; robot-at is a state fluent which indicates the location of a robot; goto(robot, location) is an action fluent which expresses some robot’s navigation task. Particularly, the state fluent pre-task-finished(task, task) is used to define some order between tasks. For example, if a robot need to execute a do task in some designated location, it should be navigated to that place first.

Example 4 (CPF's in the Warehouse-Robot Domain):

```plaintext
domain warehouse_robot_mdp {
    ......
    cpf's {
        move-success(?r) = Bernoulli(MOVE_PROB(?r))
        do-success(?r) = Bernoulli(DO_PROB(?r))
        robot-at'(?r, ?l) = if (goto(?r, ?l) & move-success(?r)) then true
        else if (exists_{?l_nav: location} [goto(?r, ?l_nav) & move-success(?r)]) then false
        else robot-at(?r, ?l);

        done'(?g) = if (exists_{?r: robot, ?a: ability, ?l: location} // robot executes at most an action
            do(?r, ?a) & do-success(?r) & TASK(?g, ?a, ?l) & robot-at(?r, ?l))
        then true
        else done(?g);

        check-done'(?t) = if ( done(?t) ) then false
        else check-done(?t);

        pre-task-finished'(?t1, ?t2) = if ( pre-task-finished(?t1, ?t2) ) then true
        else if (exists_{?r: robot, ?a: ability, ?l: location} // robot executes at most a task
            do(?r, ?a) & do-success(?r) & TASK(?t1, ?a, ?l) & robot-at(?r, ?l))
        then true
        else false;
    }
}
```

CPF is the most important component in domain description. They can define transition dynamic with probabilistic functions. Suppose f is the state fluent defined by cpf. Then, f' is the new state fluent in the next state caused by cpf. Take the cpf for robot-at(?r, ?l) as example. If ?r is navigated successfully to ?l, then the value of robot-at'(?r, ?l) is true; otherwise, if ?r is navigated to other place ?l_nav, then the value of robot-at'(?r, ?l) is false. In addition to the two situations above, robot-at'(?r, ?l) keeps the value of robot-at(?r, ?l).

Example 5 (Constraints in the Warehouse-Robot Domain):

```plaintext
domain warehouse_robot_mdp {
    ......
    \text{action-preconditions} { 
        // robot executes at most an action
        forall_{?r: robot} { [sum_{?a: ability} [ do(?r, ?a)]) <= 1];
    }
    ......
    \text{state-invariants} { 
        // robot at exactly one location
        forall_{?r: robot} { [sum_{?l: location} [ robot-at(?r, ?l)]) == 1 ];
    }
}
```

The constraint block includes action preconditions and state invariants. In Example 5, the sample of action preconditions indicates that each robot can execute at most a do action at any time. This limits the applicability of multiple actions into the same robot. A state invariant is defined for state fluents, just like axioms in PDDL. The given example says that a robot must stay in only one location at any time.

Example 6 (Reward in the Warehouse-Robot Domain):

```plaintext
domain warehouse_robot_mdp {
    ......
    reward = 
        // 1: task utility
        [sum_{?t: task} [ [done(?t) & check-done(?t) ] * TASK_PRIORITY(?t) * TASK_UTILITY(?t) ] ];
        // 2: do cost
        - [sum_{?r: robot, ?a: ability} [ do-success(?r) & do(?r, ?a) ] * DO_COST(?r, ?a) ];
        // 3: goto cost
        - [sum_{?r: robot, ?l: location, ?l_nav: location} [ [ move-success(?r) & robot-at(?r, ?l) & goto(?r, ?l_nav) ] * GOTO_COST(?l, ?l_nav) ] ];
}
```

The reward function is defined as the distance between the accumulated utility of tasks which have been finished and the total cost of actions which have been executed successfully.

Example 7 (Instance in the Warehouse-Robot Domain):

```plaintext
instance warehouse_robot_mdp_inst_demo { 
    domain = warehouse_robot_mdp;
    objects { 
        robot: { tiago1, tiago2 }; 
        ability: { 
            a00_pick, // pick up goods
            a01_drop, // put down goods
            a02_pack // pack into the box 
        } 
    task: { 
        t00_pick_obj1, 
        t01_pick_obj2,
    }
}
```
Example 7. In the first step, the robot picks up the goods in the location block gives all reachable locations of robots in a map. The initial state of a problem instance, e.g., the original location of robot tiago1, must be done by a robot with the ability to navigate to the shelf where the goods obj1 are placed. The location block gives all reachable locations of robots in a map. The non-fluents block presents all state constants, including probabilities, costs, and utilities. For instance, the non-fluent TASK(t00_pick_obj1, a00_pick, l_t00) indicates that the task t00_pick_obj1 must be done by a robot with the ability a00_pick in the location l_t00. The initial state block shows the initial state of a problem instance, e.g., the original locations of robots. The last part sets the horizon and the discount factor used in the reward function.

Example 8 (Plan of a Warehouse-Robot Problem): A planning problem is an instance of the planning domain. For on-line probabilistic planning, a solution is a plan trace composed of sampled action steps. Each action step is a set of ground action fluents. Table 3 shows the first three steps of a plan trace which is a solution to the planning problem in Example 7. In the first step, the robot tiago1 is navigated to the shelf where the goods obj1 is placed and the robot tiago2 is navigated to the shelf where the goods obj2 is placed. In the second step, tiago1 picks up obj1 and tiago2 picks up obj2. In the third step, both of tiago1 and tiago2 are navigated to the packing table for the latter tasks. Also, Table 3 shows the mapping from a RDDL action to a ROS message. These ROS messages are divided into two categories: one is the messages sent to the MoveIt! and the other is the messages sent to the Navigation.

**TABLE 3. A fragment of a RDDL plan trace.**

| RDDL plan | ROS action message |
|-----------|-------------------|
| [1]: {goto(tiago1, l_t00), goto(tiago2, l_t01)} | move_base_mag/MoveBaseGoal.action |
| [2]: {do(tiago1, a00_pick), do(tiago2, a00_pick)} | moveit_msgs/MoveGroup.action |
| [3]: {goto(tiago1, l_t02), goto(tiago2, l_t03)} | move_base_mag/MoveBaseGoal.action |

**VI. EXPERIMENT AND ANALYSIS**

This section shows three parts of experimental results. The first part is to run PRobPlan in the example domain described in Section V. The second part is to simulate a warehouse mission in the ROS simulator, Gazebo. In the last part, more thorough experimental results of our recommendation function are provided. It is for future research on developing more domains to instantiate our framework.

**TABLE 4. Sizes of generated problem instances.**

| No. | #robot | #ability | #task | #s_ft | #a_ft |
|-----|--------|----------|-------|-------|-------|
| 1   | 2      | 2        | 4     | 36    | 16    |
| 2   | 2      | 3        | 8     | 100   | 26    |
| 3   | 5      | 2        | 10    | 195   | 85    |
| 4   | 5      | 3        | 10    | 195   | 90    |
| 5   | 5      | 3        | 15    | 355   | 115   |

A. EXPERIMENT ON THE WAREHOUSE DOMAIN

We tested our instantiated framework on five generated problem instances. The sizes of these instances are depicted in Table 4. The first column is the number of instances. The following columns are the number of robot instances, the number of robot abilities, the number of tasks, the number of ground state fluents (i.e., #s_ft), and the number of ground action fluents (i.e., #a_ft), in order. They lead to differences between the sizes of problem instances. As mentioned in the problem statement section, if there are m ground state fluents in the problem, then the size of the state space should be 2^m since all ground state fluents are binary. So is the size of the action space. Each instance in Table 4 has a large state (or action) space. For example, the smallest instance has a state space with size 2^{36} = 68719476736 and an action space with size 2^{16} = 65536.

We compared the performance of different RDDL planners embedded in our framework. We chose a random policy (i.e., randomly allocating tasks to robots) as the baseline. Table 5 showed experimental results. The first column is the number of problem instances. Each entry represents the average accumulated reward returned by the planner from 50 runs on each instance. In each run the planner was given 1 second per step. The horizon was set to be 10.

As depicted in Table 5, SOGBOFA outperformed PROST greatly. It solved all instances and provided a much better solution than the baseline for each instance. PROST only solved the smallest instance. In its instantiation phase,
PROST had to enumerate all applicable ground actions and this finally caused a running-time error due to out of memory. On the contrary, SOGBOFA did not instantiate the whole action space. In fact, previous studies [39] have shown that, when the problem size is increased even moderately, the performance of sample-based algorithms such as PROST will deteriorate rapidly. The above results not only showed the advantage of SOGBOFA over PROST in this example domain, and also implied the promising advantage of our framework over Canal et al. (2019). Their framework could not be used directly because of the lack of the written-manually KB for the warehouse domain.

To evaluate the proposed recommendation function, we implemented it in SOGBOFA, which led to an enhanced version called SOGBOFA+R. The implemented recommendation function favored both of sampled actions and max-concurrent action groups based on RDDL-specific features. It should be mentioned that: we did not do this in PROST, because it had implemented its own recommendation function. As depicted in Table 5, SOGBOFA+R outperformed SOGBOFA with an average 49.7% improvement in five instances. By the way, we also tested another RDDL planner called Gourmand [26], the runner-up of IPPC 2011. However, no instance was solved by it at all. Since this did not affect the main conclusion, we omitted the result of Gourmand in Table 5.

### B. EXPERIMENT ON THE SIMULATOR

We ran the problem instance depicted in Example 7 in Gazebo. The simulation is a continuous re-planning process depicted in Algorithm 1. In each iteration, the master node interacts with the planning node via reading or writing the configuration file. A re-plan will occur when a plan step is completely executed. Thus, ROS only executes the first step of the concurrent plan each time. If it contains a goto action, a "MoveBaseGoal.action" message will be sent. After the plan step is finished, the configuration file will be updated according to the simulated result. Thus, plan generation and execution is going on alternatively, until all tasks are completed.

We built a 15m*10m warehouse in Gazebo, shown in Figure 1. We used TIAGo robots. This kind of robots had mobility, perception, manipulation and human-robot interaction capabilities. The experimental setting was Intel(R) Core(TM) i5-9500 CPU @ 3.00GHz, 16G memory and a graphics card Radeon HD 8570. The simulation configuration used Ubuntu 16.04, ROS Kinetic and Gazebo 7. The embedded planner was SOGBOFA.

Figure 5 showed the simulation result. In the given mission, two goods were placed on the distinct shelf and warehouse robots were asked to fetch them and pack into a paper box. There were two TIAGo robots available, each waiting in a specific area at the beginning. When the simulation began, each robot was assigned a task to fetch a desired goods. First, the robot moved to the shelf and picked up the desired goods (A). Second, it moved to the packing table (B). Third, it dropped down the goods on the table (C). Finally, one of the robots packed the two goods into a paper box (D). The plan trace produced continuously by SOGBOFA was shown in Table 3.

In the simulation, we consider two kinds of factors which may affect the experimental results. One is the time efficiency. Since plan generation and execution is performed alternatively, the total running time is composed of the planning time and the simulation time. All the timing information can be exported by Gazebo automatically. For the given mission, the total running time was up to 300 seconds. Among that, the planning time was 67 seconds, 22% of the total time. The simulation time was heavily dependent on the hardware configuration. The faster the machine runs, the less the simulation time is. The other is the performance stability. Since SOGBOFA is a probabilistic planner, it may provide a different plan policy in multiple runs. However, because our mission was very simple, SOGBOFA always gave the optimal plan in each simulation.

### C. EXPERIMENT ON OTHER DOMAINS

Further, we showed some extra experimental results of the proposed recommendation function in some IPPC benchmarks. They are promising to be simulated through our framework in future. We implemented the recommendation function in a trial-based probabilistic algorithm, called UCT (Upper Confidence Bounds to Trees). It is a baseline but often performs poorly when solving large-action-space planning problems.

We generated max-concurrent action groups in the set of favored actions as follows. For IPPC 2011/2014 domains, we used max-ndef-actions to set the max size of action groups. However, for IPPC 2018 domains, we used the num-

---

20Because RDDL2.0 omits max-ndef-actions, we limit the size of maximal action groups not to exceed the number of robots for a specific problem.

21There was a 44.8% improvement on instance 1, 63.3% on instance 2, 45.7% on instance 3, 46.9% on instance 4, and 47.8% on instance 5. As a result, the average improvement on five instances was 49.7%.

22The Gourmand only supports the file format in PPDDL so that it need the rddlsim parser to transfer a RDDL file into a PPDDL file. But, all possible action combinations are also generated in the translation process, which quickly leads to a stack overflow. Its source code comes from: http://aiweb.cs.washington.edu/ai/planning/gourmand.html

23http://pal-robotics.com/robots/tiago/

---

### TABLE 5. Averaged rewards (the higher, the better) returned by tested planners.

| No. | Random | SOGBOFA | SOGBOFA+R | PROST |
|-----|--------|---------|-----------|-------|
| 1   | -91.00 | 116.49  | 168.25    | 193.28|
| 2   | -91.59 | 142.87  | 232.71    | -     |
| 3   | -271.90| 197.91  | 287.64    | -     |
| 4   | -272.90| 196.01  | 288.83    | -     |
| 5   | -259.69| 138.41  | 204.13    | -     |
FIGURE 5. Different phrases for finishing tasks in the simulation. (A) Two robots arrived the locations of shelves. (B) Two robots moved to the packing table, carrying desired goods. (C) Two robots dropped down the goods on the table. (D) One robot packed the goods into a box.

TABLE 6. Action space sizes of IPPC 2011/2014 benchmarks. Each domain has 10 instances. Instances 11-15 are intentionally generated by ourselves.

| Domain          | Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----------------|--------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| sysadmin-2011   | #action| 11 | 11 | 21 | 21 | 31 | 31 | 41 | 41 | 51 | 51 | 211 | 466 | 821 | 10701 | 20876 |
|                 | #max   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 2   | 2   | 2   | 3   | 3   |
| game-of-life-2011| #action| 10 | 10 | 17 | 17 | 26 | 26 | 26 | 26 | 31 | 137 | 326 | 466 | 2626 | 4526 |
|                 | #max   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 2   | 2   | 2   | 3   | 3   |
| wildfire-2014   | #action| 19 | 19 | 33 | 33 | 51 | 51 | 61 | 61 | 73 | 73 | 121 | 9871 | 12881 | 16291 | 20101 |
|                 | #max   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 2   | 2   | 2   | 2   | 2   |
| academic-2014   | #action| 11 | 56 | 16 | 121 | 21 | 211 | 26 | 326 | 31 | 466 | 1351 | 6196 | 21700 | 31931 | 66712 |
|                 | #max   | 1  | 2  | 1  | 2  | 1  | 2  | 1  | 2  | 1  | 2  | 3   | 4   | 5   | 4   | 4   |

number of robots for each specific problem, just as we did in the warehouse domain. It was observed that a robot can do at most one action at any time in most benchmarks. Particularly, we replaced the large set of instantiated actions with a small set of favored actions. This generated an enhanced version of UCT, called UCT+R. The same thing was also done to the random-policy baseline used in the first part of this section, generating an enhanced version, Random+R.

We ran these four algorithms: UCT, UCT+R, Random, and Random+R, in six IPPC benchmark domains. They were Game-of-life-2011, Sysadmin-2011, Academic-2014, Wildfire-2014, Academic-2018 and Wildlife-2018. Among them, Academic-2018 was the domain with the largest number of applicable actions,\(^{24}\) which poses a real challenge to the competitors [29]. All instances in these domains were tested. Their action-space sizes were presented in Table 6 and Table 7. In Table 6, \#action meant the

\(^{24}\) Academic-2018 is equivalent to Academic-2014 apart from some minor changes that became necessary due to altered competition rules. However, if we compare the largest instances, that number grew from 466 in 2014 to more than 1011 in 2018, and the median over the instances increased from 43 to 1862.
TABLE 7. Action space sizes of IPPC 2018 benchmarks. Each domain has 20 instances.

| Domain     | Number | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  |
|------------|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| academic-2018 | #action | 16  | 121 | 25  | 28  | 326 | 31  | 821 | 497 | 45  | 1954|
| wildlife-2018 | #action | 4   | 4   | 5   | 25  | 6   | 36  | 7   | 49  | 8   | 64  |

| Domain     | Number | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  |
|------------|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| academic-2018 | #action | 32568 | 2081 | 1771 | 134138 | 6787 | 147537 | 20811575 | 540423 | NaN | NaN |
| wildlife-2018 | #action | 9   | 81  | 100 | 1000 | 144 | 1728 | 2744 | 2744 | 65536 | 65536 |

FIGURE 6. Experimental Results in six IPPC benchmarks.

total number of ground action instances and #max meant max-nondef-actions. Problem instances (from 1 to 10) were original in IPPCs. However, problem instances (from 11 to 15) were intentionally generated by ourselves to increase action spaces dramatically. It should be mentioned that we did not change the nature of the domain but simply added objects. All instances in Table 7 were original without any addition. We conducted 50 runs for each instance and averaged the returned rewards. The horizon was set to be 40. Each time step was given 2 seconds. The RAM limitation was 2.00GB. All algorithms run in the rddlsim.25

The result of each tested domain was shown in Figure 6, respectively. Both UCT-R and Random-R performed better than their original versions. This showed the effectiveness 25https://github.com/ssanner/rddlsim
of the recommendation function. Especially in Sysadmin2011 and Academic-2014, the enhanced algorithms had obvious advantages over the originals, such as for intentionally generated instances (from 11 to 15). Also, we compared the above results with the official results\(^\text{26}\) of the PROST baseline and SOGBOFA in Academic-2018 and Wildlife-2018. As depicted in Figure 7, both two enhanced algorithms outperformed the PROST baseline and SOGBOFA. Particularly, for instance 12-20 in Wildlife-2018, our enhanced algorithms achieved high rewards, while the two competitors got a zero reward.

### VII. CONCLUSION

We proposed a framework called PRobPlan for embedding probabilistic planning into ROS in this paper. PRobPlan can be provided as a ROS node. It automatically generates problem files in the robot environment and calls top RDDL planners, such as SOGBOFA, to solve generated problem instances. Furthermore, we presented a new recommendation function to enhance the performance of embedded planners. This framework showed its effectiveness in our case study.

Our work provides a new way to combine planning technologies and robot systems. However, there is still much work in future. First, we will introduce more ROS components into our framework, such as Pocket Sphinx\(^\text{27}\) for voice recognition. Therefore, a task can be generated via the voice input. Also, we may consider the multi-master configuration\(^\text{28}\) when it becomes mature in future. Second, we will consider regional planning according to the layout of the warehouse. That is, a planner is only responsible for a specific region. This will increase the efficiency of plan generation and execution. Last but not least, we will instantiate the proposed framework into more planning domains. It is promising to take advantage of the existing IPPC benchmarks for this.

### REFERENCES

1. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: An open-source robot operating system,” in Proc. ICRA Workshop Open Source Softw., vol. 3, 2009, p. 5.
2. A. Koubâa, Robot Operating System (ROS): The Complete Reference, vol. 1. Cham, Switzerland: Springer, 2016.
3. S. Chitta, I. Sucan, and S. Cousins, “MoveIt! [ROS topics],” IEEE Robot. Autom. Mag., vol. 19, no. 1, pp. 18–19, Mar. 2012.
4. R. L. Guimarães, A. S. de Oliveira, J. A. Fabro, T. Becker, and V. A. Brenner, “ROS navigation: Concepts and tutorial,” in Robot Operating System (ROS), vol. 925. Cham, Switzerland: Springer, Feb. 2016, pp. 121–160.
5. M. Cashmore, M. Fox, D. Long, D. Magazzini, and B. Ridder, “Opportunistic planning in autonomous underwater missions,” IEEE Trans. Autom. Sci. Eng., vol. 15, no. 2, pp. 519–530, Apr. 2018.
6. M. Crosby, R. P. A. Petrick, C. Toscano, R. C. Dias, F. Rovida, and V. Krüger, “Integrating mission, logistics, and task planning for skill-based robot control in industrial kitting applications,” in Proc. PlanSIG, 2017, pp. 135–174.
7. M. Cashmore, M. Fox, D. Long, D. Magazzini, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, “Rosplan: Planning in the robot operating system,” in Proc. ICAPS, 2015, pp. 333–341.
8. M. Fox and D. Long, “PDDL2.1: An extension to PDDL for expressing temporal planning domains,” J. Artif. Intell. Res., vol. 20, pp. 61–124, Dec. 2003.
9. I. Little and S. Thiebaux, “Probabilistic planning vs. replanning,” in Proc. ICAPS Workshop IPC, Past, Present Future, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.144.1163
10. G. Canal, M. Cashmore, S. Krivic, G. Alenyà, D. Magazzini, and C. Torras, “Probabilistic planning for robotics with ROSPlan,” in Proc. 20th Annu. Conf. Towards Auto. Robotic Syst. (TAROS), London, U.K., Jul. 2019, pp. 236–250.
11. S. Sanner, “Relational dynamic influence diagram language (RDDL): Language description,” Austral. Nat. Univ., Tech. Rep., 2010, p. 32. [Online]. Available: http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf
12. B. Lacerda, F. Faruq, D. Parker, and N. Hawes, “Probabilistic planning with formal performance guarantees for mobile service robots,” Int. J. Robot. Res., vol. 38, no. 9, pp. 1098–1123, Aug. 2019.
13. DHL, March. Robotics in Logistics: A DPDHL Perspective on Implications and Use Cases for the Logistics Industry. Troisdorf, Germany: DHL, 2016.
14. T. Keller and P. Eyerich, “PROST: Probabilistic planning based on UCT,” in Proc. ICAPS, 2012, pp. 119–127.
15. H. Cui, T. Keller, and R. Khardon, “Stochastic planning with lifted symbolic trajectory optimization,” in Proc. AAAI, 2019, pp. 119–127.
16. L. Joseph, Mastering ROS for Robotics Programming. Birmingham, U.K.: Packt, 2015.
17. H. Darweesh, E. Takeuchi, K. Takeda, Y. Ninomiya, A. Suiji, L. Y. Morales, N. Akai, T. Tomizawa, and S. Kato, “Open source integrated planner for autonomous navigation in highly dynamic environments,” J. Robot. Mechatronics, vol. 29, no. 4, pp. 668–684, Aug. 2017.
18. M. Ghallab, D. Nau, and P. Traverso, Automated Planning Acting. Cambridge, U.K.: Cambridge Univ. Press, 2016.

\(^{26}\)https://ipc2018-probabilistic.bitbucket.io/#results

\(^{27}\)http://wiki.ros.org/pocketsphinx

\(^{28}\)https://github.com/fkie/multimaster_fkie
[19] A. Cimatti, M. Roveri, and P. Traverso, “Automatic OBDD-based generation of universal plans in non-deterministic domains,” in *Proc. AAAI/IAAI*, 1998, pp. 875–881.

[20] N. Kushmerick, S. Hanks, and D. S. Weld, “An algorithm for probabilistic planning,” *Artif. Intell.*, vol. 76, nos. 1–2, pp. 239–286, Jul. 1995.

[21] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, “PPDDL: The planning domain definition language,” Yale Center Comput. Vis. Control, New Haven, CT, USA, Tech. Rep. CVC TR-98-003/DCA TR-1165, 1998.

[22] H. L. Younes and M. L. Littman, “PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects,” Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-04-162, 2004, vol. 2, p. 99.

[23] D. Rao and Z. Jiang, “Learning planning domain descriptions in RDDL,” *Int. J. Artif. Intell. Tools*, vol. 24, no. 3, Jun. 2015, Art. no. 1550002.

[24] J. Hoffmann and B. Nebel, “The FF planning system: Fast plan generation through heuristic search,” *J. Artif. Intell. Res.*, vol. 14, pp. 253–302, May 2001.

[25] Z. Xing, Y. Chen, and W. Zhang, “Maxplan: Optimal planning by decomposed satisfiability and backward reduction,” in *Proc. ICAPS*, 2006, pp. 53–56.

[26] A. Kolobov, P. Dai, M. Mausam, and D. S. Weld, “Reverse iterative deepening for finite-horizon MDPs with large branching factors,” in *Proc. ICAPS*, 2012, pp. 146–154.

[27] H. Cui and R. Khardon, “The SOGBOFA system in IPC 2018: Lifted BP for conformance approximation of stochastic planning,” in *Proc. IPC-6 Planner Abstr.*, 2018, pp. 1–6.

[28] F. Geißer and D. Speck, “Prost-DD: Utilizing symbolic classical planning in THTS,” in *Proc. IPC-6 Planner Abstr.*, 2018, pp. 13–16.

[29] F. Geißer, D. Speck, and T. Keller, “An analysis of the probabilistic track of the IPC 2018,” in *Proc. ICAPS Workshop*, 2019, pp. 27–55.

[30] B. Schäpers, T. Niemueler, G. Lakemeyer, M. Gebser, and T. Schaub, “ASP-based time-bounded planning for logistics robots,” in *Proc. 28th Int. Conf. Automated Planning Scheduling (ICAPS)*, Delft, The Netherlands, Jun. 2018, pp. 509–517.

[31] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen, “A deliberative architecture for AUV control,” in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2008, pp. 1049–1054.

[32] H. Nguyen, M. Ciocarlie, K. Hsiao, and C. C. Kemp, “ROS commander (ROSCO): Behavior creation for home robots,” in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 467–474.

[33] J. Bohren and S. Cousins, “The SMACH high-level executive [ROS News],” *IEEE Robot. Autom. Mag.*, vol. 17, no. 4, pp. 18–20, Dec. 2010.

[34] M. S. Boddy, “Imperfect match: PDDL 2.1 and real applications,” *J. Artif. Intell. Res.*, vol. 20, pp. 133–137, Dec. 2003.

[35] S. Edelkamp and J. Hoffmann, “PDDL2.2: The language for the classical part of the fourth international planning competition,” Univ. Freiburg, Freiburg im Breisgau, Germany, Tech. Rep. 195, 2004.

[36] A. Gerevini and D. Long, “Plan constraints and preferences in PDDL3—the language of the fifth international planning competition,” Dept. Electron. Automat., Univ. Brescia, Brescia, Italy, Tech. Rep. 2005-08-47, 2005.

[37] F. Rovida, M. Crosby, D. Holz, A. S. Polydoros, B. Großmann, R. P. A. Petrick, and V. Krüger, “SkiiROS—a skill-based robot control platform on top of ROS,” in *Robot Operating System (ROS): The Complete Reference* (Studies in Computational Intelligence), vol. 707, A. Koubaa, Ed. Cham, Switzerland: Springer, 2017, pp. 121–160.

[38] C.-E. Hrabia, S. Wypler, and S. Albayrak, “Towards goal-driven behaviour control of multi-robot systems,” in *Proc. 3rd Int. Conf. Control, Autom. Robot. (ICCAR)*, Apr. 2017, pp. 166–173.

[39] H. Cui, R. Khardon, A. Fern, and P. Tadepalli, “Factored MCTS for large scale stochastic planning,” in *Proc. AAAI*, 2015, pp. 1–7.

[40] R. Huang, Y. Chen, and W. Zhang, “SAS*: Planning as satisfiability,” *J. Artif. Intell. Res.*, vol. 43, pp. 293–328, Mar. 2012.

[41] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, Feb. 2007.

[42] K.-C. Ma, Z. Ma, L. Liu, and G. S. Sukhatme, “Multi-robot informative and adaptive planning for persistent environmental monitoring,” in *Distributed Autonomous Robotic Systems*. Cham, Switzerland: Springer, 2018, pp. 285–298.

[43] T. Keller and M. Helmert, “Trial-based heuristic tree search for finite horizon MDPs,” in *Proc. 23rd Int. Conf. Automated Planning Scheduling (ICAPS)*, Rome, Italy, Jun. 2013, pp. 135–143.

[44] T. Keller, “Anytime optimal MDP planning with trial-based heuristic tree search,” Ph.D. dissertation, Dept. Math. Comput. Sci., Univ. Freiburg, Freiburg im Breisgau, Germany, 2015.