Gradient-Based Mixed Planning with Discrete and Continuous Actions

Kebing Jin, Hankz Hankui Zhuo, Zhanhao Xiao, Hai Wan, Subbarao Kambhampati

School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
Department of Computer Science and Engineering, Arizona State University, US

Abstract
Dealing with planning problems with both discrete logical relations and continuous numeric changes in real-world dynamic environments is challenging. Existing numeric planning systems for the problem often discretize numeric variables or impose convex quadratic constraints on numeric variables, which harms the performance when solving the problem. In this paper, we propose a novel algorithm framework to solve the numeric planning problems mixed with discrete and continuous actions based on gradient descent. We cast the numeric planning with discrete and continuous actions as an optimization problem by integrating a heuristic function based on discrete effects. Specifically, we propose a gradient-based framework to simultaneously optimize continuous parameters and actions of candidate plans. The framework is combined with a heuristic module to estimate the best plan candidate to transit initial state to the goal based on relaxation. We repeatedly update numeric parameters and compute candidate plan until it converges to a valid plan to the planning problem. In the empirical study, we exhibit that our algorithm framework is both effective and efficient, especially when solving non-convex planning problems.

Keywords: AI Planning, Mixed Planning.

1. Introduction

Autonomous robots have become commonplace in commercial and industrial settings. For example, hospitals use autonomous mobile robots to move materials. Warehouses exploit mobile robotic systems to efficiently move materials from stocking shelves to order fulfillment zones. In scientific missions, Woods Hole Oceanographic Institution (WHOI) uses autonomous underwater vehicles (AUVs) to collect data of scientific interest. In these real-world applications, it is desirable to have autonomous
robots be capable of autonomously planning with optimization of numeric objectives related to metrics such as resources, time, and navigation distance, besides achieving desirable goals (e.g., in the form of propositions).

To handle numeric planning problems, there have been some approaches proposed to discretize numeric space and then use heuristic searching to approximate the optimization result, such as Metric-FF \[29\] and LPRPG \[10\]. Those numeric planners, however, do not address planning missions over long-term reasoning for autonomous robots because the size of discretization needs to be fixed in advance manually. It is hard to determine a proper size of discretization beforehand for various planning problems with respect to different environments. For example, in the ocean mission scenario as shown in Figure 1, a ship is equipped with an AUV (Autonomous Underwater Vehicle), i.e., the submarine in the figure, and an ROV (Remotely Operated Vehicle), i.e., the robot in the figure. The AUV aims to take images in region A, and the ROV aims to take samples in regions B and C. The planning mission is to make the three vehicles, i.e., the ship, AUV and ROV, reach the destination region (the blue area denoted by "destination region"), with avoiding obstacles (the black areas). Each action has three parameters, i.e., x-velocity $v_x$, y-velocity $v_y$ and duration $d$. In particular, if the ship deploys the ROV, which moves within a circle centered over the ship with
radius \( R \), the ship needs to stay at the deployment location until the ROV comes back again. We use red arrows to indicate the trajectory of the ship, green arrows to indicate the trajectory of the ROV, and blue arrows to indicate the trajectory of the AUV. An example plan generated by Metric-FF can be found from the red curve shown in Figure 1(a) (the corresponding action sequence is shown on the right). Note that we set all of the three parameters of actions to be 1, -1, or 0, owing to the requirement of fixed numeric effects of Metric-FF. For example, the ninth action “ROV-navigate(-1 -1 1)” of the plan indicates the x-axis of the ROV increases -1 (i.e., \(-1 \times 1 = -1\)) and the y-axis increases -1 (i.e., \(-1 \times 1 = -1\)). Note that the action model of “ROV-navigate” can be found from Figure 3 in Section 3. As shown in Figure 1(a), each movement of Metric-FF is fixed, which indicates they cannot make up a flexible plan. The desired solution is to dynamically adapt step lengths according to the positions and shapes of obstacles when generating the plan, as the one shown in Figure 1(b) (we will describe the plan shown in Figure 1(b) after we present our mxPlanner on Page 4).

There are also approaches that introduce control parameters into action models to handle numeric planning problems. For example, Kongming [42] was proposed to merge activity planning and trajectory optimization by introducing control parameters into effects of actions. It, however, requires fixed time discretization, which restricts its ability to scale to more complicated planning missions where short and long lived activities coexist. POPCORN [54] was built, using real number control parameters to allow action models with infinite domain parameters. It can, however, only be used in discrete numeric effects with constant rates of change, and only supports linear constraints. To relax the limitations of linear constraints, ScottyActivity [19], a hybrid activity and trajectory planner, was developed to effectively generate plans for autonomous robots over long-term reasoning. ScottyActivity supports convex state constraints and control parameters that are often used to model controllable change rates and robot velocities. With a continuous time formulation, ScottyActivity scales effectively to planning missions with long durations without discretizing control parameters, even in the presence of both short and long span activities. ScottyActivity utilizes convex optimization to choose continuous states, control parameters and times. Despite the success of ScottyActivity, it still requires the continuous search space to be convex, which in consequence does not allow “obstacles” in a navigation domain, which makes the search space be non-convex. In many real-world applications, however, the continuous search space is often non-convex. The problem will become more complicated when the non-convex space involves different variable types. It is undoubtedly challenging to solve planning missions that require both discrete action planning and non-convex continuous state searching.

Different from planners using optimization algorithms, Wu et al. [61] made use of gradient descent and Tensorflow, demonstrating the effectiveness of borrowing Recurrent Neural Networks (RNNs) on generating plans with continuous actions. The use of gradient descent avoids discretizing continuous numeric space and breaks the limitations of convex continuous numeric space and linear numeric effects. They, however, focus on problems mixed with discrete and continuous numeric changes rather than a mix of logical relations and numeric variables. For example, in the Navigation domain as shown in Figure 2, an agent starts from an initial location, aiming at reaching a goal destination \( g \) with the purpose of maximizing a reward function. The
domain only contains a moving action whose transition function is shown in Figure 2(b), where $d_t$ is the distance from location $s_t$ to deceleration zone $z$ in the middle of the map, $\lambda$ is a velocity reduction factor which is determined by $d_t$. The initial state $s_0$ is a two-dimensional location of the agent. The goal is to maximize the reward, i.e., to minimize the Manhattan distance from the agent to the goal location $g$, as shown in Figure 2(c). When the approach terminates, it returns a sequence of two-dimensional vector, each of which indicates a movement on x-axis and y-axis, as shown in Figure 2(d). As shown in the figure, the method proposed by Wu et al. does not involve any logical relations.

In this paper, we extend the scope of the previous numeric planning problem by simultaneously allowing three properties: non-convex continuous numeric space, propositional and numeric action preconditions and action effects, and more complicated action effects. We propose a novel approach, mxPlanner, which stands for gradient-based mixed Planner with discrete and continuous actions, to deal with non-convex numeric space and complex preconditions effects of actions. Specifically, we first build a heuristic module to estimate a best plan $\sigma$ to transit the initial state $s_0$ to the goal by relaxation. After that, we adopt a framework of RNN to propagate the states according to the plan $\sigma$. At the same time, we make use of a gradient-based method to optimize numeric parameters in the continuous preconditions and effects of the actions in $\sigma$. We repeat these two phases until our approach converges to a valid plan of the planning problem. Note that we do not only update the continuous numeric parameters of actions, but also modify the actions in $\sigma$ at the same time. Compared with the approach proposed by Wu et al., we adopt a framework of RNNs with a goal-directed heuristic module to tackle the challenges introduced by the mix of discrete and continuous actions. For example, in the AUV domain shown in Figure 1, another plan calculated by our mxPlanner can be found from the red curve in Figure 1(b) (the corresponding action sequence is shown on the right), whose length is much shorter than the one calculated by Metric-FF, since mxPlanner is more flexible than Metric-FF in avoid-
ing obstacles without fixing parameters (or length of each movement). For another example in the Navigation domain shown in Figure 2 differently, the description of mxPlanner contains two action models, as illustrated in Figure 2(e), including numeric parameters (e.g., “vel_\text{x}”), propositions (e.g., “(can-move)”), and preconditions, (e.g., “(= (car_\text{x\_y}) g)”)). As shown in Figure 2(f), the initial state is composed of numeric variables and propositions, and the goal is composed of a set of propositions. At last, mxPlanner computes a plan reaching goals, as shown in Figure 2(g), including action names and their parameters.

We summarize the contributions of the paper as follows:

- We extend previous numeric planning problems to simultaneously allow continuous numeric space to be non-convex, preconditions and effects of actions to be both propositional and numerical, and numerical effects to be non-linear.

- To handle mixed planning problems, we propose a novel approach, mxPlanner, which borrows the framework of recurrent neural networks with integration of symbolic heuristic searching in the framework.

- We empirically show that our mxPlanner outperforms existing planners in solving mixed planning problems with non-convex numeric space and complex preconditions and effects (involving propositions and numeric expressions). Besides, the experimental results show mxPlanner is also competitive when handling mixed planning problems with convex continuous numeric space.

In the remainder of the paper, we first introduce related works and a formal definition of our mixed planning problem. After that, we present our approach in detail and evaluate our approach by comparing with previous approaches to exhibit its superiority. Finally, we conclude the paper with future work.

2. Related Work

2.1. Planning with discrete and continuous actions

2.1.1. Numeric Planning

Automated planning aims to find a sequence of actions to complete a given task. There have been considerable advancements in the development to solve state-dependent goals planning tasks. A significant part of this progress comes from successful studies on classical planning composed of discrete actions, which are mostly based on heuristic forward search [28, 26, 23, 17].

However, real-world planning problems are often along with numeric variables, such as time, fuel, money and materials. Researchers extended the classical formulation to support numeric variables and metric minimizing to solve real-world missions. For example, Metric-FF [29] was built based on discretization and ignoring all effects that decrease the value of the affected variable. LPRPG [10] uses linear programming to calculate interval bounds from the constraints in the action preconditions and effects, as an adjunct to a relaxed planning graph. Ivankovic et al. [33] introduced an algorithm
to find an optimal plan under the classical, discrete action model with systems of constraints. Keyder et al. [37] proposed an approach to improve delete relaxation to compute upper and lower bounds by finding an informative set of conjunctions. Besides, in order to handle objective optimization, linear programming is an important technique, which can minimize or maximize a linear function when subjected to various constraints. For example, in [15] the authors proposed a model MILP to deal with optimization under an objective function with a variety of temporal flexibility criteria, such as makespan. ILP-PLAN [36] represents planning problems with resources or complex objective functions as Integer Linear Programming models. BBOP-LP [4] uses linear programming to encode a relaxed version of the partial satisfaction planning problem to obtain search heuristics.

Researchers also introduced control parameters to model realistic problems. For example, Pantke et al. [49] proposed a PDDL-based multi-agent planning system reasoning about the control parameters in the production control domain applications. Kongming [42] was proposed to capture the control parameters with hybrid flow graph which is capable of representing continuous trajectories in a discrete planning framework. However, Kongming is limited by its fixed time discretization. POPCORN [54] introduces continuous control parameters into the domain-independent planning applications to allow infinite continuous search space in actions. However, POPCORN can only be used in discrete numeric effects. ScottyActivity [19] was proposed to handle a mix of discrete and continuous actions by making use of convex optimization. However, ScottyActivity cannot deal with non-convex problems, e.g., with obstacles in planning problems, as done by our mxPlanner approach. ScottyPath [18] was proposed to handle domains with obstacles, it is based on ScottyActivity with constructing convex safe regions to avoid obstacles.

In general, these numeric planning approaches are limited by discretization or linear optimization methods, lacking the ability to handle complex planning problems, which contains continuous actions whose numeric effects are allowed to be linear and non-linear in convex or non-convex searching spaces.

2.1.2. Hybrid Planning

Discretization is one of the viable approaches to solve hybrid planning problems [51, 52, 59]. However, discretization-based planners can’t scale to larger searching space. Indeed, there are planners that do not rely on discretization. For example, COLIN [12] presents a heuristic forward planner with delete relaxations approach to handle temporal planning with continuous time-dependent effects. It does not rely on discretization but linear programs. However, it only supports continuous time-dependent effects with “constant” rates of change. And it cannot deal with non-convex problems. POPF [11] and OPTIC [3] convert all numeric constraints to linear programming constraints, then they use forward searches combined with a Mixed Integer Programming (MIP) solver ensuring the temporal and numeric constraints of the problem are met. Although these successful approaches were proposed, they only can solve problems whose continuous actions with constant effects and they can’t handle real-world planning missions with non-linear numeric changes. Wu et al. [61] used a framework of RNNs and efficiency of Tensorflow to handle hybrid planning problems with non-linear numeric effects. However, their method cannot handle propositional
effects and preconditions. In other words, it cannot be used to solve a planning problem mixed with discrete logical relations and continuous numeric changes and with a specific propositional goal. SCIPPlan [55] plans in metric discrete time hybrid factored planning domains with instantaneous continuous actions with the purpose of handling discrete and continuous numeric effects and collision avoidance. To handle non-linear continuous effects, SMTPlan+ [7] makes use of polynomial process models which enable it to deal with non-linear polynomial changes. And OPTIC++ [14] reasons with non-linear domains by generating piecewise linear upper and lower bound approximations for non-linear functions. However, the continuous changes of variables are limited to be monotonic and OPTIC++ also cannot support non-convex domains.

In general, some of these state-of-the-art approaches make use of discretization, which limits their applications in real-world. Some of them can’t handle non-linear effects in non-convex problems due to their linear programming or convex optimization. mxPlanner avoids these problems by gradient descent.

2.1.3. Comparison with previous works

Compared with hybrid planners, mxPlanner is not only able to handle planning problems mixed with discrete and continuous behaviors, but also problems with more complicated effects, which are determined by multiple parameters. Compared with numeric planners handling problems with control parameters, mxPlanner is not only able to solve these problems as well as computing exact values of parameters, but also can handle the non-convex problems. Intuitively, we compare our approach against the previous approaches on the following features:

- **Non-discretization**: This feature indicates whether the corresponding approach is based on discretization or not.
- **Mixed effects**: This feature indicates whether the corresponding approach handles problems mixed with discrete logical relations and continuous numeric changes or not.
- **Non-linear effects**: This feature indicates whether the corresponding approach can handle continuous actions with non-linear effects or not.
- **Non-convexity**: This feature indicates whether the corresponding approach can handle non-convex continuous numeric space or not.

First, we compare the planners on the feature “non-discretization”. Discretization means that the planner only considers discretized actions and the solution space is only composed of fixed discrete numeric variables. In consequence, the planner cannot take into account the values that are between two neighbor discretized values. It will significantly reduce the completeness of the planner. Also, we compare them on the features about the application scopes of planners: “Mixed effects”, “Non-linear effect” and “Non-convexity”. The more features the planners possess, the larger the application scopes are. The results are shown in Table 1 where “Y” indicates the corresponding feature is true, while “N” indicates the corresponding feature is false. For example, the feature “Non-discretization” of mxPlanner is “Y”, indicating mxPlanner is not
based on discretization of the continuous searching space. From the table, we can see that only our mxPlanner approach allows all of the above-mentioned four features to be true.

Table 1: The comparison with previous approaches

| Approaches       | Non-discretization | Mixed effects | Non-linear effect | Non-convexity |
|------------------|--------------------|---------------|-------------------|---------------|
| Metric-FF [29]   | N                  | Y             | N                 | N             |
| LPRPG [10]       | N                  | Y             | N                 | N             |
| Kongming [42]    | N                  | Y             | N                 | Y             |
| UPMurphi [51]    | N                  | Y             | Y                 | N             |
| COLIN [12]       | Y                  | Y             | N                 | N             |
| POPF [11]        | Y                  | Y             | N                 | N             |
| OPTIC [8]        | Y                  | Y             | Y                 | N             |
| POPCORN [54]     | Y                  | Y             | N                 | N             |
| DiNo [52]        | N                  | Y             | Y                 | N             |
| ENHSP [56]       | N                  | Y             | Y                 | N             |
| SMTPlan+ [7]     | Y                  | Y             | Y                 | N             |
| Wu et al. [61]   | Y                  | N             | Y                 | Y             |
| ScottyActivity [19] | Y               | Y             | Y                 | N             |
| OPTIC++ [14]     | Y                  | Y             | Y                 | N             |
| mxPlanner        | Y                  | Y             | Y                 | Y             |

2.2. Trajectory Planning

To handle realistic planning missions, generating trajectories from initial states to goals is a fundamental task in automated planning research. Many approaches have been proposed extensively to handle real-world robot planning missions with complex dynamics for purposes of time and resource usage minimization. For example, Moon et al. [47] proposed a minimum time approach for obstacles avoidance by Non-Linear Trajectory Generation. Chen et al. [8] introduced a global optimization approach to handling kinematic and dynamic constraints based on hybrid genetic algorithms. Langeaal [41] proposed a tree-based algorithm to compute a feasible minimum energy trajectory from a start position to a distant goal by precomputing a set of branches from the space of allowable input. To discover an optimal collision-free trajectory, Gracia et al. [24] developed a supervisory loop to fulfill workspace constraints caused by robot mechanical limits, collision avoidance, and industrial security in robotic systems with geometric invariance and sliding mode related concepts. Evolutionary algorithms were proposed to solve trajectory planning problems [43, 1, 48, 64, 25, 22, 63, 45]. Despite the success of those approaches, they cannot solve trajectory planning problems involving both continuous and discrete actions, including numeric and logical preconditions and effects.
2.3. Recurrent Neural Networks

In the past few years, recurrent neural networks (RNNs) [32, 34] have been widely investigated in applications with sequential data, such as text, audio, and video. For example, Jorden networks [34] and Elman networks [16] were introduced for supervised learning on sequences. After that, some classical approaches for training RNNs are proposed, such as real-time recurrent learning (RTRL) [60], backpropagation Through Time (BPTT) [59]. However, long-term dependencies problems of RNNs often result in gradient vanishing and gradient explosion [27]. To improve that, researchers proposed many excellent approaches such as long short-term memory networks (LSTM) [27], clipping gradient [50], gated recurrent unit networks (GRU) [9], independent recurrent neural networks (IndRNN) [44]. The typical feature of the RNNs architecture is a cyclic connection, and it enables RNNs to update based on a past state and an input. Certainly, researchers also made use of RNNs in the field of planning, for example, Araujo et al. [2] proposed a three-layer partially recurrent neural network to perform trajectory planning. Wu et al. [61] used a framework of RNNs and efficiency of Tensorflow to handle hybrid planning problems with non-linear numeric effects. However, their method focuses on problems mixed with discrete and continuous numeric changes instead of a mix of discrete propositional relations and continuous numeric changes. To tackle these challenges, we also use a RNN framework and combine it with an algorithmic heuristic module to handle planning problems mixed with propositional logical relations and continuous numeric effects.

2.4. Planning Based on Deep learning

Planning based on deep learning methods also have recently gained considerable attention with aims at handling real-world planning problems and we only present some examples.

One of the most popular study is to solve the combined task and motion planning problem. Some approaches [53, 46, 38] were proposed making use of discretization for purpose of goals arriving or collision avoidance. However, discretization means these approaches need human computations in advance. HPN [35] is a hierarchical task planner which predict action consequences in continuous action and scene parameter spaces. Researchers also use location references for communicating geometric information to the task planner [6, 57]. Most of these approaches discussed so far have dealt almost exclusively with manipulation problems, facing a huge and complicated challenges.

Recently, applications in the other fields also made huge progress utilizing planning methods. For example, some text generation methods combining with planning to understand the relationship between events and infer a storyline. A plan-and-write framework [62] is proposed to create stories according to a explicit storyline composed of keywords generated from given titles. And Tambwekar et al. [58] introduced a deep reinforcement learning approach to controllable, automated story plot generation from a defined start state and a defined goal state. Deep learning planning methods are also used to learn models from raw observations to understand the transition between images. For example, Konidaris et al. [39] automatically abstracted state representations from raw observations, relied on a prespecified set of skills, and expressed them in
PDDL. Causal InfoGAN [40] uses Gumbel-Softmax to backprop through transitions of discrete binary states, and leveraging the structure of the binary states for planning.

3. Problem Definition

In this paper we aim to solve sequential planning problems with both discrete logical relations and continuous numeric changes. Specifically, we aim to consider the planning problem that has the following features:

- The continuous numeric space (which is composed of a set of continuous numeric variables in each state) is allowed to be non-convex; note that the objective based on the non-convex space is also non-convex.
- The effects of actions are allowed to be discrete (propositional operations), continuous (numeric changes) or both.
- The numeric effects of continuous actions are allowed to be non-linear, linear, or both.
- The numeric variables are allowed in preconditions of actions.

We call the above-mentioned planning problem as a mixed planning problem. Formally, we represent our mixed planning problem as a tuple $M = \langle S, s_0, g, A, B \rangle$, where $S$ is a set of states, each of which is composed of a set of propositions and assignments of continuous numeric variables in the prefix form (e.g., “(= $v$ 1)” indicates variable $v$ is assigned to be 1). We define a continuous numeric space by the set of numeric variables. Specifically, we use a vector $V$ to denote all of the numeric variables and use $V_i = \langle v_1^i, v_2^i, \ldots, v_K^i \rangle$ to denote their values in state $s_i$. $s_0 \in S$ is an initial state, and $g$ is a goal which is composed of a set of propositions.

$A$ is a set of action models, each of which is composed of a tuple $\langle a, pre(a), eff(a) \rangle$, where $a$ is an action name with zero or more parameters. An action is a grounding of an action model, i.e., every parameter in the action model is an object or a real number. Specifically, we use a vector $\Theta = \langle \Theta_0, \Theta_1, \ldots, \Theta_{N-1} \rangle$ to denote all numeric parameters occurring in $A$ of $N$ steps, where $N$ is the maximal length of the potential solution plan $\sigma$. $\Theta_i = \langle \theta_1^i, \theta_2^i, \ldots, \theta_T^i \rangle$ is all of the numeric parameters of actions in $A$ in step $i$. $T$ is the number of different numeric parameters of all actions in $A$. Note that the parameters of actions are different from the set of variables in states. For example, “ROV-navigate(?$v_x$,?$v_y$,?d)” indicates a movement of ROV, and each movement is defined by three numeric parameters. $pre(a)$ is a set of preconditions, each of which is either a proposition or a numeric constraint. For example, “($\geq ?x ?l$)” is a numeric constraint indicating variable $?x$ should not be less than $?l$. Each numeric precondition can also be considered as a real-number interval. The precondition set can be divided into two sets: propositional preconditions, denoted by $pre^p(a)$, and numeric preconditions $pre^n(a)$. $eff(a)$ is a set of effects, each of which is either a literal (i.e., a propositional or its negation), or a numeric updating expression. For example, “(increase location-x (+ ?$v_x$ ?d))” means to increase the value of “location-x” by $?v_x \times ?d$. We call the sets of positive literals and negative literals in the effect set $eff(a)$ as positive effects and
negative effects, denoted by $\text{eff}^+(a)$ and $\text{eff}^-(a)$, respectively. Also, we call the set of numeric updating expressions as numeric effects, denoted by $\text{eff}_n(a)$. An action model or action is called continuous if it has numeric updating effects; otherwise it is called discrete. An action is applicable in a state if its precondition is satisfied by the state.

$\mathcal{B}$ is an interval to constrain numeric parameters, which is defined by $\mathcal{B} = [\underline{\beta}, \overline{\beta}]$. We use two vectors $\underline{\beta} = \langle \underline{\beta}_1, \underline{\beta}_2, \ldots, \underline{\beta}_T \rangle$ and $\overline{\beta} = \langle \overline{\beta}_1, \overline{\beta}_2, \ldots, \overline{\beta}_T \rangle$ to denote lower bounds and upper bounds of parameters, respectively. For each parameter $\theta^j \in \Theta$, we assume it is constrained by a lower bound $\underline{\beta}_j$ and an upper bound $\overline{\beta}_j$. It is noted that if a parameter is not limited by any upper bounds, we let $\overline{\beta}_j = \infty$. Similarly, if a parameter is not limited by any lower bounds, we let $\underline{\beta}_j = -\infty$. Different from previous approaches with control parameters, $\text{mxPlanner}$ is not just based on searching graphs constructed by the bounds of control parameters. $\text{mxPlanner}$ uses gradient descent to accomplish a finer searching. Such that, $\text{mxPlanner}$ can handle numeric parameters without being constrained by bounds.

Besides, we use $\psi(a)$ to denote the cost of action $a$. In this paper, we focus on benchmarks about path planning, so we define the cost of an action as the distance yielded by it. We define the cost of a plan $\sigma$ as the sum of the cost of all actions in $\sigma$, i.e., $C(\sigma) = \sum_{a_i \in \sigma} \psi(a_i)$.

Given a mixed planning problem $\mathcal{M}$, we aim at computing a plan $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$, which achieves $g$ from $s_0$ with the minimal cost. Different from classical planning problems, in this paper, we require to not only find appropriate actions but also determine their continuous numeric parameters to minimize the cost.

In this paper, we extend the syntax of PDDL 2.1 [20] to express our mixed planning problem. Specifically, we use “event” to define events as PDDL+ [21] does, events are instantaneous actions which happen the instant their preconditions are met. Events in our paper are similar to the format of action models. Once the preconditions of an event are satisfied, it happens immediately and its effects turn to be true. Similar to numeric planning with control parameters, we use “parameters-bound” to define bounds of numeric parameters. Bounds of parameters restrict different parameters ranging within the bounds. It is noted that $\text{mxPlanner}$ makes no use of the bounds during the gradient descent to compute numeric parameters. Hence, the bounds can be infinite.

Compared with PDDL 2.1 and PDDL+, (1) $\text{mxPlanner}$ is able to handle more complex mathematical functions (such as $x^n$, $\sqrt{x}$, $\sin x$). Besides, the effects of $\text{mxPlanner}$ can not only be extended functions based on duration, but also functions decided by multiple numeric parameters. They cannot be syntactically accepted in PDDL+ or PDDL 2.1. (2) In PDDL+ and PDDL 2.1, actions only accept duration as a continuous parameter, while in our problem definition, numeric parameters can be accepted by actions. It significantly expands the searching space and makes the problem more complicated, as the effects can be decided by any numeric parameters rather than numeric variables.

Intuitively, we show an example of a mixed planning problem in Figure 3. In Figure 3(a), “deploy-ROV()” is a discrete action model which does not have numeric effects, while “ROV-navigate(?u_x ?u_y ?d)” is a continuous action which has numeric effects such as “(increase location-x_ROV(* ?u_x ?d))”. Figure 3(b) defines bounds of three
Figure 3: An example of the mixed planning problem on ocean mission.

Figure 4: The Gradient-based framework of mxPlanner, which follows the RNN framework to perform forward simulation.

In this section, we address our mxPlanner approach in detail. We show a framework of our mxPlanner in Figure 4 in the form of unfolded RNN Cells. Each RNN Cell represents a step in a plan, which is composed of a heuristic module, a transition module, and a loss module. The heuristic module is an algorithm to output an action, the transition module is used to update a state according to action models, and the loss module is composed of functions to calculate losses. It is notable that the heuristic module and the transition module are not implemented by neural networks. We just...
“borrow” the framework of RNN. We assume the number of RNN cells, $N$, is sufficiently large to compute a plan. The overall procedure of computing a plan is as shown below:

1. We first randomly initialize numeric parameters $\Theta$ of actions which will be optimized by the subsequent steps.

2. The heuristic module takes as input a state $s_i$, numeric parameters $\Theta_i$, and the goal $g$, and outputs an action $a_i$, upper bound vector $U_i$ and lower bound vector $L_i$ of numeric variable values. The upper bound vector and lower bound vector are two real number vectors with the size of the number of numeric variables. These two bound vectors designate the value ranges of all numeric variables in the next state.

3. The transition module takes as input a state $s_i$, action $a_i$, and numeric parameters of actions $\Theta_i$ in $a_i$, and outputs the next state $s_{i+1}$.

4. The loss module takes as input an action $a_i$, upper bound vector $U_i$, lower bound $L_i$ and state $s_{i+1}$, and outputs the loss of the step $L_i$. We calculate the total loss $L_{all}$ by accumulating losses from all of the RNN Cells.

5. We inversely optimise numeric parameters in each step $\Theta_0, \Theta_1, \ldots, \Theta_{N-1}$ by minimizing $L_{all}$. It is noted that, after updating numeric parameters, the transferred states at each step are also updated in the next iteration, along with the numeric parameters updated. Hence, the actions computed by the heuristic module may be different in the following iterations. With the values of numeric parameters updated, the heuristic module repeatedly estimates actions based on current updated states until a plan with the minimal cost is achieved.

6. We repeat the second to the fifth steps until we find a valid plan $\sigma$. Note that actions in $\sigma$ are not fixed and they can be changed during iterations. That is, we not only update continuous parameters, but also update actions at the same time.

In the following subsections, we will introduce in detail the heuristic module, the transition module, the loss module and the inverse optimization procedure.

### 4.1. Heuristic Module

The heuristic module aims to find an appropriate action towards the goal and to predict the value range of numeric variables in the next state. We build the heuristic module based on the relaxed planning graph [5] and the interval-based relaxation [56]. Relaxed planning graphs are widely applied in heuristics searching planning approaches and the interval-based relaxation is used to capture the numeric effects of actions. We first give some simple backgrounds about relaxed planning graphs and interval-based relaxation.

Relaxed planning graph is built form a state $s$ only containing propositions. It contains actions and propositions that are possibly reachable from the state $s$. In other words, the graph doesn’t include unreachable actions or propositions. A relaxed planning graph is composed of interlaced nodes at state-level and nodes at action-levels. Nodes at state-level are propositions possibly being true. Nodes at action-level are actions that might be possible to be executed. Two nodes are connected by an edge, indicating preconditions or effects. A relaxed planning graph is first started from a
level of state nodes, each node is a proposition in state $s$. Then a level of action nodes follows, each node is an action whose preconditions are satisfied in $s$. If a node at the first state-level is a precondition of a node at the first action-level, we link them by an edge. Then we add all effects of actions at the first action-level into $s$, and regard the updated propositions as the nodes at the second state-level nodes. If a state node at the second state-level is an effect of an action at the first action-level. We connect the state node and action node by an edge. We repeat the procedure until all goal propositions are in state nodes at the same state-level.

Interval-based relaxation is an extension of the principle of monotonic relaxation to numeric planning. In the interval-based relaxation, each numeric variable is defined by an interval, representing the set of values that the variable can have. The interval can be extended following a series of rules, which do not only formulate simple binary operations, but also complex mathematical functions.

![Figure 5: The procedure of the heuristic module.](image)

Formally, given the current state $s_i$ and goal $g$, the heuristic module $H$ selects an appropriate action $a_i$ which leads to a state $s_{i+1}$ with the minimal heuristic value, an upper bound vector $U_i$ and a lower bound vector $L_i$ which make up the intervals that all numeric variables in $s_{i+1}$ should satisfy. As shown in Figure 5 we first introduce a set of relaxed actions and update state $s$ to the next relaxed state $s^+_{i+1}$ with an action $a$ (Boxes 1 and 2 in Figure 5). Then we construct a relaxed planning graph and compute intervals based on the graph (Boxes 3 and 4 in Figure 5). After that, we revise the graph by pruning and repeatedly extending actions (Boxes 5, 6 and 7 in Figure 5). When all relaxed actions in the graph are valid, we compute a heuristic value for action $a$, an upper bound and a lower bound (Boxes 8 and 9 in Figure 5).

For a state $s_i$ and one of its applicable actions $a$ whose numeric parameters are given by $\Theta_i$, we define the next relaxed state $s^+_{i+1}$ by adding the positive effect of $a$ and updating numeric variables with numeric effects $\text{eff}^+(a)$ computed by $\Theta_i$. Then we introduce a set of relaxed actions, denoted by $A^+$, which are obtained from actions by removing the negative effects and numeric preconditions and effects. Following Graphplan [5], based on the relaxed action set $A^+$, we construct a relaxed planning graph from $s^+_{i+1}$ to $g$, denoted by $\Lambda(s^+_{i+1}, g)$. As the relaxed action set does not contain any numeric preconditions and effects, we need to extend the planning graph. For a planning graph $\Lambda(s^+_{i+1}, g)$ which has $h$-level, we use $A_h(s^+_{i+1}, g)$ and $P_h(s^+_{i+1}, g)$ to denote the sets of actions and propositions in the $n$-th level at $\Lambda(s^+_{i+1}, g)$. Specifically,
we conduct the following steps to extend $\text{\Lambda}(s_{i+1}^+, g)$:

- Following the interval-based relaxation \cite{56}, we restrict the value of each numeric variable in a state within a real number interval and extend the interval according to the numeric effects of actions and the parameter bounds $B$. We use $I_n(s_{i+1}^+, g)$ to denote the intervals in the $n$-th level. Noted that $B$ are used to estimate the number of actions which are possibly to be executed, rather than divided the continuous searching space to construct graphs.

- We then remove the redundant actions $a'$ from $\text{\Lambda}(s_{i+1}^+, g)$. We consider an action $a' \in A_{n}(s_{i+1}^+, g)$ is redundant, if its positive effects do not contain any goal and any proposition that belongs to the preconditions $\text{pre}^p(a')$ of actions in $A_{n}(s_{i+1}^+, g)$ with $n > m$. We repeat this step until there is no such redundant action in $\text{\Lambda}(s_{i+1}^+, g)$.

- It is possible that there exists an action $b$ whose numeric preconditions are not satisfied. That is, there exists a numeric variable $v$ related to a numeric precondition of $b$ in $A_{n}(s_{i+1}^+, g)$ that is disjoint with $I_n(s_{i+1}^+, g)$. Then we add all relaxed actions $A^+(v)$ in $A^+$, which are applicable to $P_{n-1}(s_{i+1}^+, g)$ and include a numeric effect related to $v$, into $A_{n-1}(s_{i}^+, g)$. After that, we update $P_n(s_{i}^+, g)$ and $I_n(s_{i}^+, g)$. If the numeric precondition about $v$ of $b$ is not satisfied, we repeatedly add $A^+(v)$ into $A_{n-1}(s_{i}^+, g)$, until it is satisfied. As each interval is expanded monotonically, the numeric precondition finally will be satisfied. We repeat this step until all numeric preconditions of all actions in $\text{\Lambda}(s_{i+1}^+, g)$ are satisfied.

Intuitively, the actions in the extended relaxed planning graph can make up a valid plan. Thus, we consider the number of actions in $\text{\Lambda}(s_{i+1}^+, g)$ as the heuristic value of the relaxed state $s_{i+1}^+$. For a current state $s_i$, from all actions applicable in $s_i$, we select an action $a_i$ which leads to the relaxed state $s_{i+1}^+$ with the least heuristic value.

An example is shown in Figure 6 where Figure 6(a) enumerates a simple domain containing three action models and an instance. Numeric parameter “distance” is limited by a bound. In step 1 of Figure 6(b), we first ignore numeric variables, effects and preconditions, and we construct a relaxed planning graph $\text{\Lambda}(s_{i+1}^+, g)$ based on Graphplan. The propositions at the first and second state-level are denoted by $P_0(s_{i+1}^+, g)$ and $P_1(s_{i+1}^+, g)$, which are framed by red squares. The actions in the first action-level are denoted by $A_0(s_{i+1}^+, g)$, which are framed by a green square. As shown in step 2, we rewrite variables by intervals and extend the intervals following the Interval-based relaxation based on $\text{\Lambda}(s_{i+1}^+, g)$. Similarly, we denote them by $I_0(s_{i+1}^+, g)$ and $I_1(s_{i+1}^+, g)$ and use blue squares to frame them. In step 3 of Figure 6(b) we remove the redundant actions from $\text{\Lambda}(s_{i+1}^+, g)$ whose effects are not required in goal or any actions at following action-levels. At last, in step 4 of Figure 6(b) we extend the graph by adding actions into $\text{\Lambda}(s_{i+1}^+, g)$ until all numeric preconditions of actions are satisfied. And the heuristic value is the number of actions in $\text{\Lambda}(s_{i+1}^+, g)$, which is 3 for the example.

To guide the computation of parameters, we also estimate an upper bound vector $U_i = \langle u_1^+, u_2^+, ..., u_K^+ \rangle$ and a lower bound vector $L_i = \langle l_1^-, l_2^-, ..., l_K^- \rangle$ of numeric variables at the next state $s_{i+1}$ updated by action $a_i$ we have chosen. Generally, the actions
occurring in the smaller levels of the planning graph are more probably to be selected in the beginning of a valid plan than those actions in the bigger levels. Thus, we should guarantee the numeric preconditions of such actions in the smaller levels to be satisfied in priority. For every numeric variable \( v^k \) in \( V \), we compute its upper bound \( u^k_i \) and lower bound \( l^k_i \) in \( s_{i+1} \) as follows:

- Given an extended relaxed planning graph \( \Lambda(s_{i+1} \cdot g) \), we find the actions whose preconditions relate to \( v^k \) and which occur at the smallest level, denoted by \( A(v^k) \).

- Suppose \( \hat{a} \in A(v^k) \) has a numeric precondition \( [\hat{l}, \hat{u}] \) related to numeric variable \( v^k \). For the current state \( s_i \), we denote the distance to the numeric precondition of \( \hat{a} \) as \( |v^k_i - \frac{\hat{l} + \hat{u}}{2}| \), where \( v^k_i \) is the value of \( v^k \) in \( s_i \). For every numeric variable \( v^k \), we choose the action \( \hat{a} \) with the smallest distance w.r.t. \( v^k \), and set its bounds in the next state \( s_{i+1} \) by the numeric precondition of \( \hat{a} \), i.e., \( l^k_i = \hat{l} \) and \( u^k_i = \hat{u} \).

### 4.2. Transition Module

The transition module essentially is a transition function \( \gamma \) that transforms \( s_i \) to \( s_{i+1} \) according to an action \( a_i \) with its numeric parameters given by \( \Theta_i \). The transition function is designed according to action models, which are taken as input, including logical updating and numeric changes. Compared with the other planners about updating variables directly, we first compute numeric effects which is defined by \( \Theta_i \) and \( a_i \), then we update variables following the action model. Note that only the numeric
parameters related to action $a_i$ are used. Formally, the transition module is defined as follows:

$$s_{i+1} = \gamma(s_i, a_i; \Theta_i)$$  \hfill (1)

In the example of Figure 3, a ship navigates according to the parameters “vel_x”, “vel_y”, and “duration”. An effect of \texttt{ROV-navigate(?v_x ?v_y ?d)} is \texttt{“(increase location-x (\ast ?v_x \cdot ?d)” indicating that next x-axis of the robot is the product of ?v_x and ?d plus current x-axis. Assuming the axis of robot location-x = 0, and numeric parameters $\Theta_i = \langle 2, -2, 1 \rangle$, the x-axis of ROV will be updated into 2 after \texttt{“ROV-navigate(2 -2 1)”}.

4.3. Inverse Optimization of Parameters

To calculate an optimal plan, we repeatedly update action parameters $\Theta$ via gradient descent. In our framework, the model of the RNN cells is composed by the heuristic and transition modules, which is assumed to be fixed when updating $\Theta$. We view the parameters $\Theta$ as the input of the RNN framework at each step, and name the procedure of updating the input as an Inverse optimization, which is different from RNN approaches that aim at learning parameters of models. Similar work on inverse optimization has been demonstrated effective by Wu et al. [61], which made use of the framework of RNNs and calculated continuous action sequences via optimizing the input. Compared to their work, our problem is more challenging since we need to consider logical relations and symbolic heuristic information when computing solution plans. while they focus on discrete and continuous numeric change and cannot handle propositions in our problem.

To do the inverse optimization, we design a novel objective function with three different losses, as shown in Equation (2):

$$L_i = w_1 L_{b_i} + w_2 L_{o_i} + w_3 \psi(a_i)$$  \hfill (2)

where $w_1$, $w_2$ and $w_3$ are hyperparameters. The three losses are shown as follows:

- $L_{b_i}$ is a loss to let next state satisfy numeric bounds computed by heuristic module, defined by Equation (3):

$$L_{b_i} = \|\text{ReLU}(V_{i+1} - U_i)\|_2 + \|\text{ReLU}(L_i - V_{i+1})\|_2$$  \hfill (3)

where $\text{ReLU}(x) = \max(0, x)$, $V_{i+1}$ denotes values of numeric variables in state $s_{i+1}$. In heuristic module, $L_i$ and $U_i$ were calculated to require variables $V_{i+1}$ satisfy $L_i \leq V_{i+1} \leq U_i$. Once a numeric variable $v_{i+1}^k (1 \leq k \leq K)$ in state $s_{i+1}$ exceeds its upper bound $U_i^k$, a loss is generated. The case for lower bounds is similar.

- $L_{o_i}$ is a loss for capturing the scenario of avoiding obstacles, which is defined by:

$$L_{o_i} = \sum_{\alpha=1}^{M} m_\alpha \|y_\alpha - p_{i+1}\|_2$$  \hfill (4)
where $m_\alpha$ is:

$$m_\alpha = \begin{cases} 
1, & \text{if } p_ip_{i+1} \cap O_\alpha \neq \emptyset \\
0, & \text{otherwise}
\end{cases} \quad (5)$$

where $p_i$ is a position of the agent in the state $s_i$, $p_{i+1}$ is the next position of the agent in state $s_{i+1}$, $p_ip_{i+1}$ indicates a line between $p_i$ and $p_{i+1}$, and $y_\alpha'$ is a selected target position that guides the agent to avoid obstacle $O_\alpha$ ($\alpha = 1, \ldots, M$). To avoid $O_\alpha$, we define a convex cone whose vertex is $p_i$, covering $O_\alpha$. We use $Y_\alpha$ to denote the intersection of vertices in obstacle $O_\alpha$ and the convex cone. Let $y_\alpha \in Y_\alpha$ be the closest vertex to next position $p_{i+1}$ of the robot in next state $s_{i+1}$, i.e., $y_\alpha = \arg \min_{y \in Y_\alpha} ||p_{i+1} - y||^2$. We define $y_\alpha'$ as a position that satisfies $||y_\alpha - y_\alpha'||^2 = \varepsilon$ where $\varepsilon$ is a small positive real number and $y_\alpha'$ is not in the convex cone. An example is shown in Figure 7. Intuitively, when the robot tries to go through the obstacle, $L_{o_\alpha}$ aims to guide its destination $p_{i+1}$ getting close to $y_\alpha'$ for getting rid of $O_\alpha$.

![Figure 7: The dark gray pentagon ABCDE is an obstacle $O_\alpha$, $p_i$ is the current position and $p_{i+1}$ is next position. The light gray area is the convex cone which intersects with $O_\alpha$ at A and D. $y_\alpha = D$ is closer to $p_{i+1}$. $D'$ is a position outside $O_\alpha$ that is $\varepsilon$ away from $D$ and can be seen as a candidate of $y_\alpha'$.](image)

- $\psi(a_i)$ is the cost of $a_i$, which we view as the navigating distance in this paper. For example, the effects of “ROV-navigate(?v_x ?v_y ?d)” are “(increase location-x (* ?v_x ?d))”, “(increase location-y (* ?v_y ?d))” and “(increase total-time ?d)”, so its cost is $\sqrt{(?d?v_x)^2 + (?d?v_y)^2}$.

We define the accumulated loss $L_{all}$ as the sum of instantaneous losses until the goal is achieved, i.e.,

$$L_{all} = \sum_{i=0}^{\mu-1} L_i, \quad \text{s.t. } a_\mu = end \quad (6)$$

where $\mu$ is the final step where the goal is achieved and $a_\mu$ is a terminator indicating the goal is arrived.

We then compute the partial derivatives of the accumulated loss based on Equation 7, as shown below:
\[
\frac{\partial L_{\text{all}}}{\partial \Theta_i} = \sum_{\lambda=1}^{\mu-1} \frac{\partial L_{\text{all}}}{\partial L_\lambda} \frac{\partial L_\lambda}{\partial \Theta_i} \\
= \sum_{\lambda=1}^{\mu-1} \left( \frac{\partial L_{\text{all}}}{\partial L_\lambda} \frac{\partial L_\lambda}{\partial \Theta_i} \prod_{k=i+1}^{\lambda} \frac{\partial N_{k+1}}{\partial N_k} \right) \\
= \frac{\partial N_{i+1}}{\partial \Theta_i} \left( \sum_{\lambda=i}^{\mu-1} \left( \frac{\partial L_{\text{all}}}{\partial L_\lambda} \frac{\partial L_{\text{all}}}{\partial L_{\beta_\lambda}} \frac{\partial L_{\beta_\lambda}}{\partial \Theta_i} \prod_{k=i+1}^{\lambda} \frac{\partial N_{k+1}}{\partial N_k} \right) \\
+ \frac{\partial L_{\text{all}}}{\partial L_{\alpha_\lambda}} \frac{\partial L_{\text{all}}}{\partial \Theta_i} \prod_{k=i+1}^{\lambda} \frac{\partial N_{k+1}}{\partial N_k} \right) \\
(7)
\]

The back propagation procedure is shown in Figure 8, where the gradient flow propagates across time steps. Intuitively, the gradient of \( \Theta_i \) is determined by the loss from \( L_i \) to \( L_{\mu-1} \). The gradient of the numeric parameters irrelated to \( a_i \) is zero. By gradient descent, the total cost as a loss is minimized.

Figure 8: The procedure of back propagation in mxPlanner.

The parameters \( \Theta_i \) in each step are updated by the gradient from the total loss, as shown in Equation (8):

\[
\Theta_i = \Theta_i - \omega \frac{\partial L_{\text{all}}}{\partial \Theta_i},
\]

where \( \omega \) is a learning rate. We set \( \omega = 0.001 \) in this paper.

4.4. Overview of mxPlanner

An overview of mxPlanner is shown in Algorithm 1. We first initialize numeric parameters \( \Theta \) of \( N \) steps and build a heuristic module and a transition module to iteratively estimate action \( a_i \) (Line 5 of Algorithm 1) and update \( s_{i+1} \) (Line 6) to attain a candidate plan \( \sigma \). After that we update the parameters \( \Theta \) by optimizing \( L_{\text{all}} \) (Line 9 and 10). We repeat the above procedure until the stop requirement is satisfied (Line 2) and output the solution plan \( \sigma \).
Algorithm 1 \texttt{mxPlanner}

\textbf{input:} $M = \langle A, S, s_0, g, B \rangle$.  
\textbf{output:} $\sigma$.

\begin{enumerate}
\item initialize numeric parameters in $N$ steps $\Theta = \langle \Theta_0, \ldots, \Theta_{N-1} \rangle$ randomly;
\item while $\mathcal{L}_{\text{stop}} \neq 0$ (Equation (10)) do
\item \hspace{0.5cm} $\sigma = \langle \rangle$;
\item \hspace{0.5cm} while $i = 0, \ldots, N-1$ do
\item \hspace{1cm} predict $a_i, U_i, L_i$ with $A, s_i, g, \Theta_i$;
\item \hspace{1cm} update $s_{i+1}$ with $a_i, \Theta_i$ and $s_i$ (Equation (1));
\item \hspace{1cm} $\sigma = [\sigma|a_i]$;
\item \hspace{0.5cm} end while
\item \hspace{0.5cm} calculate accumulated loss $\mathcal{L}_{\text{all}}$ (Equation (6));
\item \hspace{0.5cm} update $\Theta$ by $\Theta_i = \Theta_i - \omega \frac{\partial \mathcal{L}_{\text{all}}}{\partial \Theta_i}$ (Equation (8));
\item \hspace{0.5cm} update $\Theta$ by $\Theta = \max(\min(\Theta, B), B)$ (Equation (9));
\item end while
\item return $\sigma$;
\end{enumerate}

Particularly, to ensure the numeric parameters fall within the bounds $B = [\underline{B}, \overline{B}]$, we limit numeric parameters by Equation (9).

$$\Theta = \max(\min(\Theta, B), B).$$

To guarantee the solution plan $\sigma$ is executable and can achieve the goal by avoiding all obstacles, we define $\mathcal{L}_{\text{stop}}$ in Equation (10). When $\mathcal{L}_{\text{stop}} = 0$, a solution plan is found.

$$\mathcal{L}_{\text{stop}} = \begin{cases} 
\sum_{i=0}^{\mu-1} \mathcal{L}_{o_i}, & \text{if } g \subseteq s_\mu \text{ and } \sigma \text{ is executable} \\
\infty, & \text{otherwise}
\end{cases}$$

\texttt{mxPlanner} has a soundness property as follows:

\textbf{Theorem 1.} The action sequence computed by \texttt{mxPlanner} is a valid plan for the planning problem.

\textbf{Proof:} According to Algorithm 1, \texttt{mxPlanner} outputs a plan $\sigma$ when the loss $\mathcal{L}_{\text{stop}} = 0$. As shown in Equation (10), we get $\mathcal{L}_{\text{stop}} = 0$ if and only if plan $\sigma$ is executable, the accumulated sum of $\mathcal{L}_o$ is 0, and $g \subseteq s_\mu$. An executable plan $\sigma$ indicates all preconditions of actions in $\sigma$ are satisfied and the states are updated correctly. We get $\sum_{i=0}^{\mu-1} \mathcal{L}_{o_i} = 0$ if and only if all goals are achieved after executing the $\mu$th action in $\sigma$ and sum of $\mathcal{L}_o$ of each step is 0. We get $g \subseteq s_\mu$ when the targeted propositions are involved in $s_\mu$ and the numeric variables in $s_\mu$ fall within the required intervals. In other words, all obstacles are avoided when reaching the goal, and the plan is executable, i.e., preconditions of actions are satisfied at the states where they are executed. Thus, the output action sequence is a solution plan for the problem.
5. Experiments

In this section, we evaluate our mxPlanner approach in three domains: AUV \(^1\), Taxi \(^2\), and Rover \(^3\), which were also used in \([19, 54, 31]\). Since there is no planner that can directly handle our mixed planning problems, we compare mxPlanner with three state-of-the-art planners with modifications, Metric-FF \([29]\), POPCORN \([54]\), and ScottyActivity \([19]\) with different settings.

(1) For convex domains, we compare mxPlanner against Metric-FF, POPCORN, and ScottyActivity. We run ScottyActivity and POPCORN with the Enforced Hill-Climbing \([30]\). Since the original domains include non-linear continuous effects which POPCORN is unable to handle, we adapt POPCORN by the following two steps:

- We assign the duration of actions as a fixed value, which makes each movement of x-axis and y-axis be a product of a fixed duration and another numeric parameter, respectively, i.e., x-velocity and y-velocity. Such a modification makes the effect become linear, which can be handled by POPCORN.

- Based on the modified domains, we configure POPCORN with the best-first searching strategy and then use CPLEX\([13]\) to optimize the plans, aiming at minimizing our evaluation criterion by updating numeric parameters.

We denote the adapted POPCORN as POPCORN\(^+\). We also adapt Metric-FF, which is shown in the following.

(2) For non-convex domains, we also compare mxPlanner against Metric-FF with numeric effects discretized. We do not compare with POPCORN and ScottyActivity in non-convex domains, it is difficult to adapt these two planners to handle non-convex continuous numeric space, as they are optimization algorithms focusing on convex continuous numeric space. To let Metric-FF to handle our problem, we adapt Metric-FF by the following steps:

- As in this paper, we only evaluate approaches on path planning domains, we discretize continuous actions by stipulating each movement to have a fixed step length.

- We introduce eight movement actions for eight angles to update positions, i.e., two horizontal movements, two vertical movements, and four diagonal movements. Each action updates the position by the length of a step.

- As each movement is linear, we can decide whether a movement crashes an obstacle by checking whether the line between the current position to the stop position crosses the region of the obstacle. Then we enumerate all lines with a fixed length that do not crash with any obstacle and write them into a PDDL file.

---

\(^1\) It is from ScottyActivity \([19]\)
\(^2\) http://agents.fel.cvut.cz/codmap/
\(^3\) https://ipc02.icaps-conference.org/
We denote the adapted Metric-FF as Metric-FF+. We run Metric-FF [29] with all six canonical configurations (i.e., Standard-FF, best-first search, best-first search with helpful actions pruning, Weighted A*, A* epsilon, Enforced Hill-Climbing with helpful actions pruning then A* epsilon) and choose the best solution as the final plan.

Finally, we evaluate how the hyperparameters influence the performance and iterations of mxPlanner. Note that we do not compare mxPlanner to Wu et al. [61], as their approach cannot handle mixed planning problems with propositional effects and preconditions.

5.1. Benchmarks

For each domain, we randomly generate 15 planning instances with a 150 × 150 sized map. For each instance, we randomly generate one to five objective regions which do not overlap each other. These objective regions are squares ranging from 5 × 5 to 20 × 20. To generate non-convex domains, we randomly place obstacles in the map which makes the continuous searching space be non-convex. The obstacles are also randomly generated in the form of rectangle regions ranging from 5 × 5 to 25 × 25. Note that obstacles do not overlap each other either.

5.1.1. The AUV Domain

In the AUV domain, an AUV (automated underwater vehicle) aims to reach each objective region and to take samples without touching obstacles. We modify the AUV domain used in ScottyActivity [19]: the effect of the action “glide” is obtained by the product of the velocity in the x-axis or y-axis (“vel_x” or “vel_y”) and its execution time “duration”. We add an event to describe the situation that the AUV collides with an obstacle. Especially, we denote a region in the form of \( R_L \), where \( R \) is a symbol indicating to a region and \( L \) is a set of vertices. The whole description of the AUV domain is shown in Figure 9.

Figure 9 also includes a planning instance in the AUV domain. In this instance, an AUV “v0” aims to take samples in five regions with avoiding obstacle “O_1”. Specifically, “\((\text{Obstacle } O_1)\)” denotes an obstacle region “O_1” that consists of four vertices (40,30), (50,30), (50,40), and (50,40).

5.1.2. The Taxi Domain

In the Taxi domain, an agent drives a taxi aiming at picking up passengers and carrying them to their personal destinations. The total description of the Taxi domain is shown in Figure 10. This domain includes three action models, “pick-up”, “get-off”, and “glide” with three numeric parameters, “vel_x”, “vel_y” and “duration”. A passenger can be picked up by a taxi when they are at the same region. Every taxi also needs avoid the obstacles during its movement.

Figure 10 also contains a planning instance in the Taxi domain. In this instance, a taxi “car0” needs to pick up two passengers “p1”, “p2” and carries them to region “C”.

5.1.3. The Rover domain

In the Rover domain, a rover navigates on a planet surface, aiming at taking samples of soil and rock, taking images and communicating them back to the lander. The partial
Figure 9: The AUV domain and one of its planning instances.

description of the Rover domain is shown in Figure 11. It includes a continuous action model “navigate” with numeric parameters (“vel_x”, “vel_y” and “duration”) and eight discrete action models such as “sample-soil”, “communicate-rock-data” and so on.

Figure 11 also shows a planning instance in the Rover domain. In this instance, a rover “rover0” aims to obtain samples at “waypoint0” and “waypoint1” and an image at “waypoint2”.

5.1.4. Comparison of three domains

Indeed, the three domains aim to solve a path with obstacle avoided, they are actually differ from each other. In Table 2 we compare the three domains with respect to the following features: (1) the number of object types; (2) the number of predicates; (3) the number of action models.
The AUV domain is a domain containing two action models and two types of objects. One action model contains numeric preconditions, which requires numeric variables to fall within a region. The other one includes non-linear effects, moving an agent to the next position. Compared with the AUV domain, the Taxi domain may have a solution with round paths, due to the randomness of passengers and their destinations. A round path means that a taxi first reaches region “B” from region “A” for picking up a passenger, then goes back region “A” for having him off. It in fact significantly increases the difficulties of solving problems. For the Rover domain, it is more complicated than the other domains, as it includes more object types, predicates and action models. As the samples are required to returned back to the lander, the Rover domain...
also may have solutions with round paths. Although these three domains in a way belong to path planning domains, they have various properties and difficulties.

5.1.5. Evaluation Criterion

In this paper, we consider the navigation distance as the optimization objective for three domains. So, we only define the cost of moving action models, such as “glide”, which updates the location of the agent by effects: “increase x-location (* ?vₓ ?d)” and “increase y-location (* ?vᵧ ?d)”. Given a plan \( \sigma = \langle a₀, a₁, \ldots, aₙ₋₁ \rangle \), we its total cost \( C(\sigma) \) as our evaluation criterion:

\[
C(\sigma) = \sum_{i \in \sigma} \psi(aᵢ) = \sum_{aᵢ \in \sigma} \sqrt{(?vₓ?d)^2 + (?vᵧ?d)^2}
\]
It is notable that we can also define the makespan or the plan length as the optimization objective function by redefining the cost of actions.

5.2. Experimental Results

We evaluate these approaches with the purpose of minimizing the total distance. We use “mx”, “MFF”, “SA”, and “POP” to denote mxPlanner, Metric-FF+, Scotty-Activity and POPCORN+. We use a subscript to denote the bound of the numeric parameters except for “duration”. For example, “mx10” and “SA10” indicate that all numeric parameters in mxPlanner and ScottyActivity are restricted within [-10, 10], respectively. Hereafter we will use $\delta$ to denote this subscript, i.e., the bound of parameters. Different from mxPlanner and ScottyActivity, Metric-FF+ and POPCORN+ can only handle discrete parameters. As we mentioned above, we assign the numeric parameters with values in advance in order to make action effects discrete. Then we use $\delta$ to denote the value of numeric parameters in Metric-FF. For example, “MFF10” indicates that all numeric parameters of Metric-FF+ are 10 (e.g., “vel_x” = 10). Specially, we restrict the parameter “duration” of mxPlanner, ScottyActivity as $0 \leq \text{duration} \leq 1$ and set “duration” of Metric-FF+ and POPCORN+ as $\text{duration} = 1$.

Experiments of mxPlanner are conducted on a machine with an i7-7700 CPU and a memory of 16GB. Experiments of Metric-FF+, POPCORN, and ScottyActivity are conducted on a machine with Ubuntu 16.04 on a 50GB of memory. We set the cutoff time as 36000 seconds.

Next, we evaluate our approach with respect to the following aspects:

- **Costs with parameter bounds in convex problems:** We evaluate the performance of mxPlanner in problems without obstacles compared with Metric-FF+, ScottyActivity and POPCORN+.

- **Costs with parameter bounds in non-convex problems:** We evaluate Metric-FF+ and mxPlanner with varying $\delta$ in the instances including obstacles.

- **Costs without parameter bounds in non-convex problems:** Note that it is not necessary for our approach mxPlanner to indicate numeric parameter bounds. We use $\mathcal{R}$ as a subscript to denote the case that parameters are unconstrained. Then we also test mxPlanner on the problems without restriction of parameter bounds.

- **Costs with the number of obstacles increasing:** We also evaluate the performance of mxPlanner and Metric-FF+ on the problems with different number of obstacles.

- **Costs with different hyperparameters of mxPlanner:** We evaluate the optimization performance of mxPlanner by varying the values of its three hyperparameters, i.e., $w_1$, $w_2$, and $w_3$, in the loss function $L_{all}$ (Equation (2)).

- **Iterations with different hyperparameters of mxPlanner:** We also evaluate the running iterations of mxPlanner by varying the values of its three hyperparameters.
In a nutshell, we compare \texttt{mxPlanner} with Metric-FF$^+$, POPCORN$^+$, and Scotty-Activity in three domains. Notably, \texttt{mxPlanner} combines gradient descent with heuristic searching, consequently \texttt{mxPlanner} is able to handle more complicated problems, such as those problems containing obstacles. Obstacles make partial values of some numeric variables unavailable. Therefore, the continuous numeric space becomes non-convex. As for planners based on optimization algorithms, such as Scotty-Activity, it is hard for them to directly solve problems with a non-convex numeric space. Hence, we test \texttt{mxPlanner} by only comparing with Metric-FF$^+$. Besides, different from the approaches based on searching in bounded graphs, \texttt{mxPlanner} is a gradient-based approach, which thus has the ability to handle numeric parameters without bounds. At last, we evaluate \texttt{mxPlanner} by varying its hyperparameters on planning performance and searching iterations.

5.2.1. Costs with parameter bounds in convex problems

We first compare \texttt{mxPlanner} with Metric-FF$^+$, POPCORN$^+$, and ScottyActivity. As shown in Table 3, we count the solved instances computed by four planners with different parameter bounds. From Table 3 we can see that \texttt{mxPlanner} and ScottyActivity with different parameter bounds succeed in computing valid plans for all instances. Unfortunately, Metric-FF$^+$ and POPCORN$^+$ fail to solve all instances. The failure of Metric-FF$^+$ is ascribed to the overlarge state space due to appropriate discretization. Metric-FF$^+$ successfully computes plans for all instances when the parameters are set as 10, but it fails on some instances when the parameters are 20. In some domains, POPCORN$^+$ fails to calculate plans within the cut-off time. The reason is that POPCORN$^+$ is a forward-searching heuristics planner, it builds up all linear constraints in terms of the numeric parameters. However, the small bounds of the numeric parameters make the problems be too difficult to be solved, because POPCORN$^+$ needs to extend a larger graph when the bounds are smaller. Hence, in the AUV domain and Taxi domain, POP$_{20}$ solves more instances than POP$_{10}$. On the other hand, bigger bounds do not always mean to be better, because the bigger bounds of parameters are, the more possible values of numeric parameters are. The enlarging of the possible value scope of numeric parameters will essentially increase the difficulty of problem solving.

|     | mx$_{10}$ | MFF$_{10}$ | POP$_{10}$ | SA$_{10}$ | mx$_{20}$ | MFF$_{20}$ | POP$_{20}$ | SA$_{20}$ |
|-----|-----------|------------|------------|-----------|-----------|------------|------------|-----------|
| AUV | 15        | 15         | 14         | 15        | 15        | 14         | 13         | 15        |
| Taxi| 15        | 15         | 7          | 15        | 15        | 13         | 11         | 15        |
| Rover| 15       | 15         | 15         | 15        | 15        | 12         | 15         | 15        |

Table 3: The count of solved problems in the AUV, Taxi, Rover domain.

Next, in Figure 12 we use histograms to show the average plan costs of \texttt{mxPlanner}, Metric-FF$^+$, POPCORN$^+$ and ScottyActivity with different parameter bounds. We only considers those instances that are solved by four planners with both $\delta = 10$ and $\delta = 20$. More specifically, in the AUV domain, the Taxi domain and the Rover domain, we only consider 13, 6, 12 instances respectively. The results show that \texttt{mxPlanner} has the top performance. Especially, \texttt{mxPlanner} performs similarly with ScottyActivity in AUV and Rover, whose instances are simpler than those of Taxi. The us-
age of convex optimization makes ScottyActivity compute plans with high quality in some simple instances. However, its plan quality significantly depends on the heuristic searching procedure on the plan skeleton before invoking a convex optimization solver. It also explains why ScottyActivity performs much worse than mxPlanner in Taxi. Also, in Taxi, the average plan costs of computed by $\text{SA}_{10}$ and $\text{POP}_{10}$ are significantly higher than those of plans w.r.t. $\delta = 20$. The reason is that some instances in Taxi contain round trips, which means that the car needs to go forth and back. When the problems contain multiple persons waiting for being moved, $\text{SA}_{10}$ and $\text{POP}_{10}$ compute plans with repeated loops, resulting in a larger cost. The average plan costs of Metric-FF$^+$ are the largest in most cases of three domains. Because the discretization of Metric-FF$^+$ lets agents move more compared with the planners using an optimization algorithm. Also, $\text{MFF}_{10}$ performs better than $\text{MFF}_{20}$ in all domains, because a large step may lead to a detour from the path.

5.2.2. Costs with bound of parameters in problems with obstacles

We also evaluate mxPlanner on more complicated planning problems with obstacles, comparing with Metric-FF$^+$. The results are shown in Table 4. From the table, we observe that mxPlanner solves all instances successfully, while Metric-FF$^+$ fails on several instances within the cutoff time. For the instances solved by two planners, mxPlanner also outperforms Metric-FF$^+$. This is because in Metric-FF$^+$ numeric effects are discretized and fixed on all instances, which leads it impossible to dynamically adapt step lengths to search a plan. In contrary, mxPlanner computes numeric parameters for each step, which avoids the unnecessary paths due to fixing the step length. The results reflect that mxPlanner is able to deal with complex changes, and compute plans with lower cost, making no use of discretization.

As shown in Table 4, compared with Metric-FF$^+$, the performance of mxPlanner is less influenced by the parameter bound. In contrary, the larger the parameters are, the fewer problems are solved by Metric-FF$^+$. It is because that when the parameters are larger, the step length becomes longer, leading it more possible to cross an objective region instead of entering it. In other words, for discretized planners, it is more likely to enter an objective region with a smaller step length. On the other hand, it becomes increasingly difficult to solve such a searching problem when the parameter bound gets
smaller. Compared with discretized planner, mxPlanner is not limited with fixed parameters, neither heuristic searching nor updating parameters.

From Table 4, we can also see that the average of plan costs in Rover domain is less than that in domains AUV and Taxi. This is because all three domains require arriving at some regions and executing actions to accomplish goals. The regions in the Rover domain are more concentrated than those in domains AUV and Taxi, resulting in the average of plan costs in the Rover domain is a little lower than both AUV and Taxi.

### 5.2.3. Costs without parameter bounds in non-convex problems

| AUV domain | Taxi domain | Rover domain |
|------------|-------------|--------------|
| mF² | mF₁₀ | mF₂₀ | mk₈ | mF² | mF₁₀ | mF₂₀ | mk₈ | mF² | mF₁₀ | mF₂₀ | mk₈ |
| 1 | 122.85 | 124.57 | 124.83 | 107.15 | 165.42 | 164.85 | 153.14 | 143.51 | 40.28 | 54.14 | \ | 35.68 |
| 2 | 142.31 | 158.28 | 156.57 | 128.78 | 295.97 | 154.85 | 136.57 | 111.19 | 125.62 | 134.85 | 136.57 | \ | 112.36 |
| 3 | 130.08 | 144.14 | 148.28 | 110.70 | 169.82 | 221.42 | 122.89 | 47.60 | 78.28 | 68.28 | 44.11 | \ | \ |
| 4 | 206.68 | 381.13 | 442.84 | 189.91 | 333.30 | 275.56 | 261.42 | 147.47 | 140.37 | 130.71 | 136.57 | 91.19 | \ |
| 5 | 143.60 | 136.57 | 136.57 | 127.58 | 385.42 | 213.14 | 139.28 | 84.08 | 93.52 | 88.28 | 73.94 | \ | 53.36 |
| 6 | 103.05 | 106.57 | 116.57 | 105.29 | 538.76 | 241.42 | 132.26 | 156.65 | 120.71 | 108.28 | 79.82 | \ | \ |
| 7 | 194.85 | 194.85 | 216.57 | 175.81 | 518.22 | 287.99 | 329.71 | 95.40 | 108.28 | 95.40 | 70.92 | \ | \ |
| 8 | 266.05 | 208.99 | 216.57 | 169.41 | 417.02 | 266.27 | 221.42 | 135.84 | 81.05 | 82.43 | 78.25 | \ | \ |
| 9 | 176.91 | 245.56 | \ | 155.62 | 840.57 | 293.85 | 102.03 | 139.97 | 148.99 | 103.92 | \ | \ |
| 10 | 201.62 | 397.99 | 386.27 | 155.14 | 537.53 | 414.56 | 110.73 | 195.97 | 251.42 | 118.95 | \ | \ |
| 11 | 391.47 | 567.77 | 477.99 | 239.68 | 135.89 | 98.28 | 88.28 | 244.68 | 217.28 | 143.21 | \ | \ |
| 12 | 405.33 | 766.69 | 642.84 | 270.09 | 339.57 | 363.85 | 146.51 | 99.05 | 104.14 | 100.00 | 59.28 | \ | \ |
| 13 | 98.14 | 92.43 | 108.28 | 85.25 | 225.54 | 122.45 | 140.96 | 96.68 | 122.02 | 102.43 | 148.28 | 79.42 | \ |
| 14 | 110.63 | 114.14 | 148.28 | 101.53 | 524.56 | 378.77 | 321.42 | 126.37 | 174.85 | \ | 100.65 | \ | \ |
| 15 | 166.14 | 180.71 | 228.28 | 153.99 | 128.85 | 156.57 | 140.27 | 218.59 | 134.85 | 184.85 | 106.48 | \ | \ |
| A | 151.96 | 254.82 | 157.32 | 253.63 | 134.21 | 243.58 | 142.20 | 201.33 | 86.38 | 122.04 | 83.20 | 99.71 | \ | \ |

Table 4: Costs of instances with obstacles in the AUV domain, the Taxi domain, and the Rover domain with different parameter bounds. Each domain contains 15 instances. Each problem contains less than five obstacles. A in the table indicates the average cost of the solved problems.

“\( mk₈ \)” indicates \( mxPlanner \) with no parameter bounds except for “duration”. The columns of “MFF” are the same as Table 4.

Table 5 shows the results of \( mxPlanner \) on the instances with obstacles without parameter bounds, i.e. the lower bounds and upper bounds of parameters are from \(-\infty\) to \(+\infty\).
to $\infty$. As shown in the table, without parameter bounds, $\text{mxPlanner}$ still solves all instances and outperforms Metric-FF$^+$. It also shows that $\text{mxPlanner}$ is able to solve mixed planning problems without prior knowledge on discretizing parameters.

To evaluate how the step length influences Metric-FF$^+$, we also show the results of Metric-FF$^+$ with different step lengths. As we mentioned before, a larger step length is more likely to result in crossing obstacles or more cost. Hence, we also apply Metric-FF$^+$ with step length being one and two. However, we do not show the results w.r.t. one, because the discretized problems contain too many propositions and Metric-FF$^+$ fails to solve them. In the AUV domain, Metric-FF$^+$ with a smaller step length generally performs better than a larger step length. Especially, for the 6-th instance, MFF$_2$ even has the top performance, as the instance has a more direct solution plan that can be found more simply. However, for other two domains, MFF$_2$ performs worse, compared against the larger step lengths. It is because that Metric-FF$^+$ has to face larger and larger searching space when step lengths get smaller increasingly, and it is getting more difficult for discretized planners to solve them. Different from Metric-FF$^+$, $\text{mxPlanner}$ does not rely on discretization and it dynamically searches step lengths and step angles. All the results demonstrate that $\text{mxPlanner}$ computes solution plans with lower costs making no use of prior knowledge on parameter discretization.

Compared to the discrete planner Metric-FF$^+$, which must be fed with fixed numeric parameters, $\text{mxPlanner}$ not only can plan without restriction of step length, but also can compute plans with lower costs.

5.2.4. Costs with the number of obstacles increasing

Figure 13 shows the average cost of 10 instances with $\delta = 10$ in the AUV domain. For different obstacle numbers, we generate afresh 10 instances with keeping the target regions. Surprisingly, the increasing number of obstacles does not lead to higher costs. Counter-intuitively, for the both planers, the average cost w.r.t. three obstacles is a little higher than that of the problems with four obstacles. The reason is that in some instances with three obstacles, the places of obstacles make a detour from the path, which leads a higher cost.

![Figure 13: Average costs of ten instances with increasing numbers of obstacles in the AUV domain.](image-url)
Compared to Metric-FF, the trend of average costs computed by \textit{mxPlanner} grows more slowly. It shows that our approach is less influenced by the number of obstacles.

### 5.2.5. Costs with different hyperparameters

Next, we investigate the influence of different hyperparameters (i.e., $w_1$, $w_2$, $w_3$ in Equation (2)) on the performance of \textit{mxPlanner}. We first randomly select 10 instances in the AUV domain. We show the average cost of these instances by \textit{mxPlanner} under different settings of hyperparameters and $\delta = 10$.

![Figure 14: Average costs of 10 instances in the AUV domain with different hyperparameters.](image)

(a) $w_1$ varied, $w_2 = w_3 = 1$.  
(b) $w_2$ varied, $w_1 = w_3 = 1$.  
(c) $w_3$ varied, $w_1 = w_2 = 1$.

As shown in Figure 14, we record the average cost of plans computed by \textit{mxPlanner} under different hyperparameters. As illustrated in Figure 14(a), the weight $w_1$ has a negligible effect on the planning performance. Because $w_1$ is the weight related to $L_b$ which guides variables to satisfy numeric preconditions and to achieve goals, it does not affect the cost directly. Figure 14(b) demonstrates that the cost tends to decrease when $w_2$ gets larger. It is because that when $w_2$ increases, the propagated gradient about crashing an obstacle becomes larger. It further leads \textit{mxPlanner} to prioritize avoiding obstacles above meeting numeric preconditions. In consequence, it is more possible to find a detour path. Figure 14(c) shows a negative correlation between costs and the weight $w_3$, the weight of the loss about plan costs. Intuitively, the bigger weight of cost is, the bigger the propagated gradient about cost is.

We also give a more intuitive exhibition about how hyperparameters influence plan costs in a heatmap, as shown in Figure 15. The weight $w_2$ is set to 1. The x-axis is the ratio of $w_3$ to $w_2$ and the y-axis is the ratio of $w_1$ to $w_2$, which both range from 0.01 to 100. The darker the area is, the smaller the average cost is. Obviously, when $w_1 : w_2 : w_3 = 1 : 1 : 100$, \textit{mxPlanner} has the best performance and has an average cost of 130.77.

From the heatmap, we can observe that from right to left, the cost tends to decrease. That is, the weight $w_3$ is getting larger.

### 5.2.6. Iterations with different hyperparameters

Next, we test the computing iterations of \textit{mxPlanner} with different hyperparameters. Here we use the same benchmarks in Section 5.2.5.
Figure 15: The heatmap of the cost with different hyperparameters.

Figure 16 demonstrates the average iterations of \texttt{mxPlanner} with different setting of hyperparameters. Generally, \texttt{mxPlanner} takes fewer iterations to find a plan with $w_2$ growing. In contrary, with $w_1$ and $w_3$ increasing, the iteration tends to increase.

(a) $w_3$ varied, $w_2 = w_3 = 1$.

(b) $w_2$ varied, $w_1 = w_3 = 1$.

(c) $w_3$ varied, $w_1 = w_2 = 1$.

Figure 16: Average computing iterations of 10 instances in the AUV domain with different hyperparameters.

Similarly, we also use a heatmap to exhibit the influences of the hyperparameters on the running iterations of \texttt{mxPlanner}, as shown Figure 17. Different from the average costs, when $w_1$ is small, the iterations also become small. When $w_1 : w_2 : w_3 = 0.01 : 1 : 1$, \texttt{mxPlanner} takes the least iterations to solve problems. It is because that $w_1$ relates to the gradient of $L_b$ which guides to find a relaxed plan. A small value
of $w_1$ means to be a modest modification of the relaxed plan. In other words, when the relaxed plan is similar with a valid plan, mxPlanner is able to find a valid plan quickly.

Figure 17: The heatmap of average computing iterations of 10 instances with different hyperparameters.

5.2.7. Running time:

Unfortunately, mxPlanner is superior to the other approaches but its running time generally exceeds theirs. The running time of mxPlanner varies significantly, depending on the complexity of planning problems. More specifically, its running time ranges from 43 seconds to 7892 seconds for AUV, from 36 seconds to 15269 seconds for Taxi, and from 25 seconds to 23647 seconds for Rover, respectively. Compared with ScottyActivity, mxPlanner is an iteration-based planning approach which needs to find a relaxed plan and to minimize the loss in each iteration. ScottyActivity invokes an off-the-shelf solver of convex optimization which is efficient in problem solving. Its efficiency helps ScottyActivity to find a valid plan quickly. Also, in each iteration, mxPlanner calculates heuristic value and it determines whether the agent collides with each obstacle or not. Different from Metric-FF, POPCORN, and ScottyActivity, mxPlanner spends more time in precisely avoiding obstacles by using gradient descent to iteratively minimize the loss. Its running time is generally higher than other planners that divide the searching space into subspaces before planning, which helps reduce the planning time. We will seek more efficient algorithms or revise codes for greater efficiency in the future.
5.2.8. Analysis

Next, we will analyze the merits and demerits of mxPlanner and the other approaches, with detailed instances illustrated.

- **Comparison with Metric-FF**: mxPlanner adjusts the step lengths and angles in each step dynamically by searching appropriate numeric parameters to make up a flexible plan, instead of a fixed step length decided by prior knowledge, as done by Metric-FF. Figure 18 shows two valid plans computed by MFF$_{10}$ (a) and mx$_{10}$ (b) respectively for the same instance with obstacles in the AUV domain. As the parameters are fixed as 10, each movement of Metric-FF has the identical step length, which finally leads a higher plan cost. While from Figure 18(b), it is not difficult to observe that in the trajectory computed by mxPlanner each step has a different length.

![Figure 18: The trajectories of an instance in the AUV domain computed by Metric-FF and mxPlanner respectively with parameter bounds are 10. Each bold red dot is the stop position of each movement. Blue areas are target regions and black areas are obstacles.](image)

- **Comparison with ScottyActivity**: ScottyActivity optimizes a computed plan, which may lead it to fall into a local optimum. Differently, mxPlanner searches candidate plans and optimizes parameters simultaneously, which, to a great extent, reduces the probability of local optimum. Figure 19 shows two valid plans computed by ScottyActivity (a) and mxPlanner (b) respectively for the same obstacle-free instance in the AUV domain. We can see that the trajectory of ScottyActivity makes a detour and is significantly longer than that of mxPlanner. Notably, in Figure 19(a), movements each are too close, making the trajectory look like a bold line.

- **Challenging instances**: The Taxi domain contains a more complicated problem setting which may include round paths, compared with the other two domains. As shown in Figure 20, the plan trajectory goes back and forth on the map. In particular, the step lengths in the beginning of the trajectory are longer than the following steps. The reason is that, according to gradient backpropagation defined in Equation (7), parameters in a step are affected by the instantaneous
loss about the action sequence from this step to the last step. In other words, parameters in the latter steps are less influenced by the instantaneous loss.

- **Failures to the best:** In some cases prior knowledge on parameter discretization indeed helps computing better solution plans than \texttt{mxPlanner}, since the heuristic module in \texttt{mxPlanner} may fall into a local optimum. Meanwhile, since \texttt{mxPlanner} tries to satisfy the nearest numeric preconditions of candidate actions during the heuristic searching procedure, \texttt{mxPlanner} does not guarantee plan optimality.
6. Conclusion

In this paper, we introduce mixed planning problems, which are extended from numeric planning problems with control parameters. In order to handle mixed planning problems, we present \textit{mxPlanner}, which is based on gradient descent by borrowing the framework of RNN combined with heuristic searching. We evaluate \textit{mxPlanner} in three domains and the experimental results show the superiority of \textit{mxPlanner} on plan quality, especially in obstacles avoidance problems compared against state-of-the-art approaches. Also, we evaluate the influence of the hyperparameters on \textit{mxPlanner}.

Compared with the previous works, the advances introduced by our planner are shown as follows. First of all, the combination of heuristic searching and gradient-based framework gives \textit{mxPlanner} the ability to handle mixed planning problems without discretization. Especially, \textit{mxPlanner} performs well when handling mixed planning problems with non-convex continuous numeric space, i.e., containing obstacles. Second, \textit{mxPlanner} is also competitive in planning problems without obstacles compared with ScottyActivity, a state-of-the-art planner on planning missions with convex continuous numeric space.

Finally, we present several directions to improve our approach. First, \textit{mxPlanner} spends a lot on gradient descent. Also, deciding whether a trajectory go through an obstacle is also time-consuming. It is promising to create a more efficient approach of parameter updating. Second, plans computed by \textit{mxPlanner} may not be optimal, the reason is that \textit{mxPlanner} always attempts to satisfy the nearest numeric preconditions of candidate actions in heuristic module. Last but not the least, it is also interesting to investigate more efficient approaches to improve heuristic searching.

References

[1] Alvarez, A., Caiti, A., Onken, R., 2004. Evolutionary path planning for autonomous underwater vehicles in a variable ocean. IEEE Journal of Oceanic Engineering 29 (2), 418–429.

[2] Araújo, A. F. R., Jr., H. D., 1998. A partially recurrent neural network to perform trajectory planning, inverse kinematics, and inverse dynamics. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, SMC 1998, Hyatt Regency La Jolla, San Diego, California, USA, October 11-14,1998. pp. 1784–1789.

[3] Benton, J., Coles, A. J., Coles, A., 2012. Temporal planning with preferences and time-dependent continuous costs. In: Mccluskey, L., Williams, B. C., Silva, J. R., Bonet, B. (Eds.), Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012. AAAI. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4699
[4] Benton, J., van den Briel, M., Kambhampati, S., 2007. A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. In: Boddy, M. S., Fox, M., Thiébaux, S. (Eds.), Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007. AAAI, pp. 34–41. URL http://www.aaai.org/Library/ICAPS/2007/icaps07-005.php

[5] Blum, A., Furst, M. L., 1997. Fast planning through planning graph analysis. Artif. Intell. 90 (1-2), 281–300. URL https://doi.org/10.1016/S0004-3702(96)00047-1

[6] Cambon, S., Alami, R., Gravot, F., 2009. A hybrid approach to intricate motion, manipulation and task planning. Int. J. Robotics Res. 28 (1), 104–126. URL https://doi.org/10.1177/0278364908097884

[7] Cashmore, M., Fox, M., Long, D., Magazzeni, D., 2016. A compilation of the full PDDL+ language into SMT. In: Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016. pp. 79–87.

[8] Chen, C.-T., Liao, T.-T., 02 2011. A hybrid strategy for the time- and energy-efficient trajectory planning of parallel platform manipulators. Robotics and Computer-integrated Manufacturing - ROBOT COMPUT-INTEGR MANUF 27, 72–81.

[9] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL. pp. 1724–1734.

[10] Coles, A., Fox, M., Long, D., Smith, A., 2008. A hybrid relaxed planning graph’lp heuristic for numeric planning domains. In: Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008. pp. 52–59.

[11] Coles, A. J., Coles, A., Fox, M., Long, D., 2010. Forward-chaining partial-order planning. In: Brafman, R. I., Geffner, H., Hoffmann, J., Kautz, H. A. (Eds.), Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010. AAAI, pp. 42–49. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1421

[12] Coles, A. J., Coles, A., Fox, M., Long, D., 2012. COLIN: planning with continuous linear numeric change. J. Artif. Intell. Res. 44, 1–96.

37
[13] Cplex, I. I., 2009. V12. 1: User’s manual for cplex. International Business Machines Corporation 46 (53), 157.

[14] Denenberg, E., Coles, A. J., 2019. Mixed discrete continuous non-linear planning through piecewise linear approximation. In: ICAPS. pp. 137–145.

[15] Do, M. B., Kambhampati, S., 2003. Improving temporal flexibility of position constrained metric temporal plans. In: ICAPS. pp. 42–51.

[16] Elman, J. L., 1990. Finding structure in time. Cognitive Science 14 (2), 179–211. URL https://doi.org/10.1207/s15516709cog1402_1

[17] Eyerich, P., Mattmüller, R., Röger, G., 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In: Gerevini, A., Howe, A. E., Cesta, A., Refanidis, I. (Eds.), Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009. AAAI. URL http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/742

[18] Fernández-González, E., 2018. Generative multi-robot task and motion planning over long horizons. Ph.D. thesis, Massachusetts Institute of Technology.

[19] Fernández-González, E., Williams, B. C., Karpas, E., 2018. Scottyactivity: Mixed discrete-continuous planning with convex optimization. J. Artif. Intell. Res. 62, 579–664.

[20] Fox, M., Long, D., 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. Journal of artificial intelligence research 20, 61–124.

[21] Fox, M., Long, D., 2006. Modelling mixed discrete-continuous domains for planning. J. Artif. Intell. Res. 27, 235–297.

[22] Fu, Y., Ding, M., Zhou, C., 2011. Phase angle-encoded and quantum-behaved particle swarm optimization applied to three-dimensional route planning for uav. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 42 (2), 511–526.

[23] Gerevini, A., Serina, I., 2002. LPG: A planner based on local search for planning graphs with action costs. In: Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France. pp. 13–22.

[24] Gracia, L., Sala, A., Garelli, F., 2012. A supervisory loop approach to fulfill workspace constraints in redundant robots. Robotics and Autonomous Systems 60 (1), 1 – 15.

[25] Guo, S., Gao, B., 2009. Path-planning optimization of underwater microrobots in 3-d space by pso approach. In: 2009 IEEE International Conference on Robotics and Biomimetics (ROBIO). IEEE, pp. 1655–1620.
[26] Helmert, M., 2006. The fast downward planning system. J. Artif. Intell. Res. 26, 191–246. 
URL https://doi.org/10.1613/jair.1705

[27] Hochreiter, S., 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6 (2), 107–116. 
URL https://doi.org/10.1142/S0218488598000094

[28] Hoffmann, J., 2001. FF: the fast-forward planning system. AI Magazine 22 (3), 57–62.

[29] Hoffmann, J., 2003. The metric-ff planning system: Translating “ignoring delete lists” to numeric state variables. J. Artif. Intell. Res. 20, 291–341.

[30] Hoffmann, J., Nebel, B., 2001. The FF planning system: Fast plan generation through heuristic search. J. Artif. Intell. Res. 14, 253–302. 
URL https://doi.org/10.1613/jair.855

[31] Höller, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H., Pellier, D., Alford, R., 2020. Hddl: An extension to pddl for expressing hierarchical planning problems. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020). AAAI Press, pp. 9883–9891.

[32] Hopfield, J. J., 1982. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences 79 (8), 2554–2558. 
URL https://www.pnas.org/content/79/8/2554

[33] Ivankovic, F., Haslum, P., Thiébaux, S., Shivashankar, V., Nau, D. S., 2014. Optimal planning with global numerical state constraints. In: Chien, S. A., Do, M. B., Fern, A., Ruml, W. (Eds.), Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014. AAAI. 
URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7941

[34] Jordan, M. I., 1997. Serial order: A parallel distributed processing approach. In: Advances in psychology. Vol. 121. Elsevier, pp. 471–495.

[35] Kaelbling, L. P., Lozano-Pérez, T., 2011. Hierarchical task and motion planning in the now. In: IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011. IEEE, pp. 1470–1477. 
URL https://doi.org/10.1109/ICRA.2011.5980391

[36] Kautz, H., Walser, J., 12 2000. Integer optimization models of ai planning problems. The Knowledge Engineering Review 15.
[37] Keyder, E. R., Hoffmann, J., Haslum, P., 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. J. Artif. Intell. Res. 50, 487–533. URL https://doi.org/10.1613/jair.4277

[38] Kingston, Z. K., Wells, A. M., Moll, M., Kavraki, L. E., 2020. Informing multi-modal planning with synergistic discrete leads. In: 2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020. IEEE, pp. 3199–3205. URL https://doi.org/10.1109/ICRA40945.2020.9197545

[39] Konidaris, G., Kaelbling, L. P., Lozano-Pérez, T., 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. J. Artif. Intell. Res. 61, 215–289. URL https://doi.org/10.1613/jair.5575

[40] Kurutach, T., Tamar, A., Yang, G., Russell, S. J., Abbeel, P., 2018. Learning plannable representations with causal infogan. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (Eds.), NIPS. Curran Associates, Inc., pp. 8747–8758.

[41] Langelaan, J. W., 2008. Tree-based trajectory planning to exploit atmospheric energy. In: American Control Conference. pp. 2328–2333.

[42] Li, H. X., Williams, B. C., 2008. Generative planning for hybrid systems based on flow tubes. In: Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008. pp. 206–213.

[43] Li, J., Deng, G., Luo, C., Lin, Q., Yan, Q., Ming, Z., 2016. A hybrid path planning method in unmanned air/ground vehicle (uav/ugv) cooperative systems. IEEE Transactions on Vehicular Technology 65 (12), 9585–9596.

[44] Li, S., Li, W., Cook, C., Zhu, C., Gao, Y., 2018. Independently recurrent neural network (indrnn): Building a longer and deeper RNN. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. pp. 5457–5466.

[45] Liu, L.-Q., Yu, F., Dai, Y.-T., 2008. Path planning of underwater vehicle in 3d space based on ant colony algorithm. Journal of System Simulation 20 (14), 3712–3716.

[46] Lozano-Pérez, T., Kaelbling, L. P., 2014. A constraint-based method for solving sequential manipulation planning problems. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014. IEEE, pp. 3684–3691. URL https://doi.org/10.1109/IROS.2014.6943079
[47] Moon, J., Prasad, J., 08 2011. Minimum-time approach to obstacle avoidance constrained by envelope protection for autonomous uavs. Mechatronics 21, 861–875.

[48] Nikolos, I. K., Valavanis, K. P., Tsourveloudis, N. C., Kostaras, A. N., 2003. Evolutionary algorithm based offline/online path planner for uav navigation. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 33 (6), 898–912.

[49] Pantke, F., Edelkamp, S., Herzog, O., 2014. Planning with numeric key performance indicators over dynamic organizations of intelligent agents. In: Müller, J. P., Weyrich, M., Bazzan, A. L. C. (Eds.), Multiagent System Technologies - 12th German Conference, MATES 2014, Stuttgart, Germany, September 23-25, 2014. Proceedings. Vol. 8732 of Lecture Notes in Computer Science. Springer, pp. 138–155. URL https://doi.org/10.1007/978-3-319-11584-9_10

[50] Pascanu, R., Mikolov, T., Bengio, Y., 2013. On the difficulty of training recurrent neural networks. In: ICML. pp. 1310–1318.

[51] Penna, G. D., Magazzeni, D., Mercorio, F., 2012. A universal planning system for hybrid domains. Appl. Intell. 36 (4), 932–959. URL https://doi.org/10.1007/s10489-011-0306-z

[52] Piotrowski, W. M., Fox, M., Long, D., Magazzeni, D., Mercorio, F., 2016. Heuristic planning for hybrid systems. In: Schuurmans, D., Wellman, M. P. (Eds.), Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. AAAI Press, pp. 4254–4255. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12394

[53] Plaku, E., Hager, G. D., 2010. Sampling-based motion and symbolic action planning with geometric and differential constraints. In: IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010. IEEE, pp. 5002–5008. URL https://doi.org/10.1109/ROBOT.2010.5509563

[54] Savas, E., Fox, M., Long, D., Magazzeni, D., 2016. Planning using actions with control parameters. In: ECAI. pp. 1185–1193.

[55] Say, B., Sanner, S., 2019. Metric hybrid factored planning in nonlinear domains with constraint generation. In: Rousseau, L., Stergiou, K. (Eds.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019. Proceedings. Vol. 11494 of Lecture Notes in Computer Science. Springer, pp. 502–518. URL https://doi.org/10.1007/978-3-030-19212-9_33
Scala, E., Haslum, P., Thiébaux, S., Ramírez, M., 2016. Interval-based relaxation for general numeric planning. In: ECAI. pp. 655–663.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S. J., Abbeel, P., 2014. Combined task and motion planning through an extensible planner-independent interface layer. In: 2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014. IEEE, pp. 639–646. URL https://doi.org/10.1109/ICRA.2014.6906922

Tambwekar, P., Dhuliawala, M., Mehta, A., Martin, L. J., Harrison, B., Riedl, M. O., 2018. Controllable neural story generation via reinforcement learning. CoRR abs/1809.10736. URL http://arxiv.org/abs/1809.10736

Werbos, P. J., 1990. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78 (10), 1550–1560.

Williams, R. J., Zipser, D., 1989. A learning algorithm for continually running fully recurrent neural networks. Neural Computation 1 (2), 270–280. URL https://doi.org/10.1162/neco.1989.1.2.270

Wu, G., Say, B., Sanner, S., 2017. Scalable planning with tensorflow for hybrid nonlinear domains. In: NIPS. pp. 6273–6283.

Yao, L., Peng, N., Weischedel, R. M., Knight, K., Zhao, D., Yan, R., 2019. Plan-and-write: Towards better automatic storytelling. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. AAAI Press, pp. 7378–7385. URL https://doi.org/10.1609/aaai.v33i01.33017378

Zhang, C.-B., Gong, Y.-J., Li, J.-J., Lin, Y., 2014. Automatic path planning for autonomous underwater vehicles based on an adaptive differential evolution. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. ACM, pp. 89–96.

Zheng, C., Li, L., Xu, F., Sun, F., Ding, M., 2005. Evolutionary route planner for unmanned air vehicles. IEEE Transactions on robotics 21 (4), 609–620.