Received July 16, 2017, accepted August 21, 2017, date of publication August 29, 2017, date of current version September 19, 2017.

*Digital Object Identifier 10.1109/ACCESS.2017.2744262*

# A Constraint-Programmed Planner for Deep Space Exploration Problems With Table Constraints

**XIAO JIANG AND RUI XU, (Member, IEEE)**
Key Laboratory of Autonomous Navigation and Control for Deep Space Exploration, Ministry of Industry and Information Technology, School of Aerospace Engineering, Beijing Institute of Technology, Beijing 100081, China

Corresponding author: Xiao Jiang (jiangxiaotwn@hotmail.com)

**ABSTRACT** In recent years because of the increasing number and types of scientific payloads on a probe, the constraints between payload and probe and the constraints among payloads have become increasingly complex. The technology of constraint processing has gradually become a focus of research in deep space planning. In this paper, we propose a constrained-programmed planner called DSPlan for deep space planning problems based on table constraints. We first propose a technique for automatically converting the planning domain definition language model used in planning into the form of table constraints. Following the common practice of coding a planning problem as a constraint satisfaction problem with multiple levels, we propose a dynamic constraint set and a corresponding mutex filtering algorithm to express the different combinations of constraints on varying levels that need to be satisfied. This new form of data structure uses explicit domain information to maintain the generalized arc consistency of table constraints. Empirical analyses demonstrate the efficiency of table constraints over the international planning competition problem and classic deep space instances in general arc consistency schema algorithms. Experimental results also prove that DSPlan and table constraints are highly promising general-purpose tools for deep space planning problems compared with other planners.

**INDEX TERMS** Planning, constraint satisfaction, table constraints, deep space exploration.

## I. INTRODUCTION

In the field of deep space exploration, traditional telemetric remote control is a major technical method of detector control. However, at certain key stages of exploration [1], such as the separation of the lander and detector (SLD) for Mars exploration, inevitable defects arise in telemetric remote control. First, because of the long distance between Earth and Mars, the time delay for data transmission is too long to satisfy the real-time requirements of separation; second, the traditional control strategy, in which a ground station sends commands to orbit, is a typical open-loop control method, meaning that even a slight error may cause data loss or even the failure of the entire task; and third, in the process of executing a command sequence, if unexpected hardware failure or an unexpected change in the environment occurs, the detector may abandon the original sequence and enter safe mode, waiting for the ground to produce a new command sequence. In this case, the SLD mission will miss its time window.

These challenges impose very harsh requirements for the traditional method of telemetric remote control [2]. In fact, NASA, the U.S. Department of Defense (DoD), and the European Space Agency (ESA) have already conducted extensive research in the field of autonomous planning and scheduling technology for deep space exploration [3], including the deep space exploration program in the U.S., space-based observation systems (the Hubble Space Telescope [4]), and the ESA's projects for on-board autonomy (the Proba series of satellites [5]–[8]). These autonomous planning and scheduling technologies have assisted in the establishment of autonomous control systems for deep space detectors, allowing these detectors to realize control without participation from a ground station. Using the knowledge available onboard and information obtained by sensors about the current state of the surrounding environment, the planner achieves its goal by determining a sensible motion execution sequence. This approach not only reduces the operating cost but also increases the reliability of the task.

With the increasingly long flight distances of recent deep space exploration missions, the payloads on probes must be increased in quantity and variety for scientific and economic reasons. The resulting complex constraints among payloads have given rise to new technical difficulties facing deep space planning technologies. For this reason, the constraint satisfaction technique has become a new hot topic of interest in the field of deep space planning.

The constraint satisfaction technique is a powerful paradigm for solving combinatorial problems because of its strong pruning ability and high processing efficiency. In 1996, Kautz and Selman first proposed the method of using the CSP formulation to solve planning problems [9]. They estimated the length of a planning problem during a preprocessing procedure and set this estimated value as a fixed bound, which was then used to convert the planning problem into an NP (Non-deterministic Polynomial) hard problem [10], [11]. The resulting problem could then be solved using the NP-complete CSP formalism [12]. The first constraint-programmed planner was CPLAN, developed in 1999 [13], with manually coded domain-dependent constraints. Since then, many constraint-programmed planners have been developed that transform various types of classic planners, such as HTN [14], Graphplan [15] and partial planners [16] into the CSP formalism. Pavel Surynek and Roman Barták encoded HTN Planning as a constraint-programmed planner using the Dynamic CSP technique. In their paper, they divided the variables into three classes, the first type encoding ground instances of primitive tasks as primitive task variables, the second type representing non-primitive tasks as non-primitive variables, and the third type treating all predicate symbols and steps as state variables. Then, these three types of variables could be used to specify constraints defining dynamic relations. GP-CSP [17] was the first GraphPlan-like constraint-programmed planner, which was proposed by Minh Binh Do and Subbarao Kambhampati. Because of the similarity of the frameworks between a constraint-programmed planner and GraphPlan, GP-CSP adopts a coding method in which the facts (or propositions) in GraphPlan are denoted by variables in CSP, whereas the actions that make the facts true are represented by values. Dynamic (or conditional) constraint satisfaction techniques are also used because the constraint-programmed planner contains a multi-layer framework. Experimental results proved that GP-CSP is superior to approaches based on both satisfiability (SAT) [18] and integer linear programming (ILP) [19].

The rich theoretical results discussed above have been widely used in deep space exploration [20]–[22]. The NASA deep space spacecraft DS-1 uses the Remote Agent (RA) autonomous control system [23], [24], which is the first software to be used as the autonomous closed-loop control software system of a spacecraft. Its planning and scheduling system, RAX-PS, uses the domain description language DDL [25] to describe the structure, functions, resources, and various types of constraints. The planning engine uses a variety of constraint propagation algorithms to generate the constraint network and to find and resolve conflicts until a solution is found. The satellites in the TechSat21 constellation and the EO1 spacecraft use the Autonomous Sciencecraft Experiment (ASE) software [26], whose decision-making module CASPER uses local constraint processing technology and can be used in continuous dynamic programming. ASPEN [27], [28] is used in combination with aerospace control, hardware models, scientific experiments, and operational procedures to automatically generate lower-level space manipulator sequences and is widely used by NASA in deep space missions, including Citizen Explorer, MARS-01, and DS-T. The EUROPA planning and scheduling system [29], [30] uses the framework of constraint interval description and considers the priority of observation tasks. The research discussed above has also focused on coding for the translation and extension of the CSP formalism to address the complex deep space environment, using approaches such as timelines [31], possibilities [32], and infinite data streams [33]. However, the domain-independent modeling method and planners that have been designed, such as ASPEN and EUROPA, become inefficient as the size of the problem increases with more complex constraints. Without specific guidance on model information, the standard CSP heuristics and search methods [34], [35] greatly restrict the efficiency of problem solving. By contrast, the domain-dependent approach has always shown impressive performance, as in [36], describing the CSP planner for the Express orbiter. However, such planners have the fatal flaw that they need to be coded manually and are difficult to transplant and maintain for later projects.

To take advantage of the efficiency of domain-dependent planners and overcome their defects, an automatic coding mode is needed instead of manual coding to allow the planner to cope with different planning domains. In this paper, we consider an automatic coding method for table constraints. Here, the word *table* has the same meaning as *extensional*, except that table constraints are usually non-binary. A table constraint is simply a list of the allowed combinations of values for a particular subset of variables and is particularly suitable for expressing relations between actions during planning. For example, an orbiter switches its direction to a new target domain, {Sun, Earth, Mars}. It is a simple matter to list all possible assignments, {<orbiter, Sun>, <orbiter, Earth>, <orbiter, Mars>}, in the form of a table constraint, but it would be an awkward task to use other types of constraints to express these relations.

To build a domain-dependent constraint-programmed planner, we use table constraints to express the action relations in the planning domain. The most important thing is to identify all possible assignments in the constraints, specifically, the variables and values. First, we analyze the domain file for a planning problem and abstract several characteristic atoms from the predicates as the variables. Then, for each action in the planning problem, the corresponding variables will be selected according to the action's parameters. Finally, every table constraint will be instantiated with the possible assignments according to the information contained in the planning problem files. After the model has been specified, we propose a method of constructing a dynamic constraint set, which is used to solve the problem that these table constraints (converted from the actions of the planning problem) cannot be satisfied simultaneously. Based on this dynamic constraint set, a planning-domain-dependent filtering algorithm is proposed. In this algorithm, we adopt the unique data structure of table constraints, as described above, to prune the redundant assignments in the list.

Experimental results from previous IPC planning problems and several classic instances of deep space exploration are used in the evaluation of the proposed algorithm for table constraints and demonstrate which domain is suitable for the algorithm and that the time spent for coding the actions into the table constraints is a worthy trade-off for the improvement in efficiency. We also compare our approach with a specialized planner for deep space exploration. The results demonstrate that DSPlan is much faster on most problematic instances; however, since our method treats the planning domain in a limited manner, there are many aspects of the planner that need to be improved.

The remainder of this paper is structured as follows: Section Two introduces background on the planning process, the CSP formulation, and table constraints; Section Three introduces the technique for automatically converting the actions in a planning problem into the form of table constraints; Section Four proposes the corresponding dynamic constraint set and filtering algorithm; and the experimental results and conclusions are presented in Section Five.

## II. BACKGROUND

Planning is a human decision-making process that seeks to achieve a given outcome using a set of predictable operations in sequence [37]. An Artificial Intelligence (AI) planning problem is defined by a triplet consisting of an initial state, a goal state, and a set of possible actions. An action modifies the current state and can only be applied if certain conditions are met. The purpose of planning is to organize a sequence of actions to cause a transition from the initial state (also denoted by $S_0$) to the goal state ($S_w$), which includes every element of the goal set $G$. This action sequence is called the solution to the planning problem. To manage the complexity of the real world, in AI planning, the information described is restricted, and much unnecessary detail is abstracted. In this paper, we consider classical planning problems that include only deterministic actions and assume complete information about the planning states.

*Definition 1 (AI Planning):* A planning problem is defined by a triplet $\Pi = \langle A, I, G\rangle$ consisting of the following:

- A set of initial states $I = \{i_1, i_2, \cdots, i_l\}$.
- A set of actions $A = \{a_1, a_2, \cdots, a_m\}$ such that each action is a tuple $\langle pre\,(a)\,, eff\,(a)\rangle$, where $pre(a)$ is the set of preconditions for action $a$ and $eff\,(a)$ is the set of effects of action $a$.
- A set of goals $G = \{g_1, g_2, \cdots, g_t\}$ that need to be satisfied.

Therefore, the state that results from executing action $a$ in state $s$ can be expressed as $Result(s, a) = (s - del(a)) \cup add(a)$. Finally, the goal $G$ is a set of planning states satisfying a propositional property specifying the final states of the plan. Therefore, a plan $p$ is a finite sequence of actions $\langle a_0, a_1, \ldots, a_n\rangle$ such that the execution of $p$ yields a state $s \in G$. A PDDL model consists of two types of files for describing a problem, i.e., the domain file and the problem file. The former gives the domain description of the problem, such as the types, predicates, and behavioral actions involved in the model; the latter simply defines the problem to be solved in terms of the objects used, the initial states, and the final goal states.

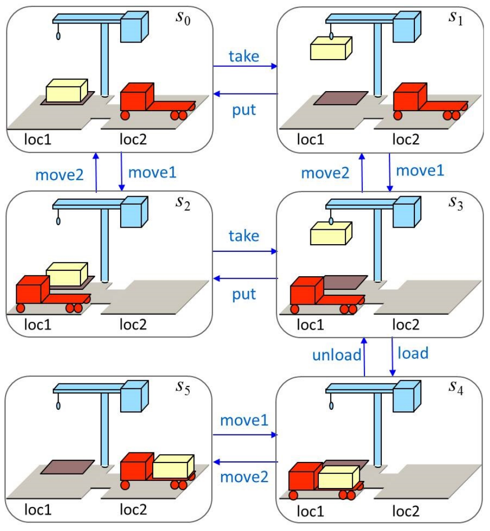

**FIGURE 1. Dock Worker Robots planning problem.**

The example shown in Figure 1 is a classic planning problem from previous instances of the International Planning Competition (IPC) [38], [39]. The Dock Worker Robots (DWR) problem [40] is a state transition system involving goods on a pile, a truck that can carry the goods and move them from one location to another, and a crane that can pick up and put down the goods. As listed in Figure 1, the set of states is $\{s_0, s_1, s_2, s_3, s_4, s_5\}$, and the set of actions is $\{take, put, load, unload, move\}$. The state of the world will change according to the different actions executed, as the example shows.

When planning is begun, a planner attempts to choose actions and construct an action sequence to change the current state of the problem until the goal state is achieved. If a partial action sequence proves inconsistent, the planner will backtrack to the last choice and change it to another action. The key to accelerating the search procedure is to find the correct order of actions while reducing or even preventing backtracking. In the DWR problem, we assume that the starting state is $s_0$: the goods are at loc1, and the truck is at loc2.

The goal state is $s_5$: the goods are at loc2, loaded on the truck. For the planning procedure introduced above, we need to construct a sequence of actions to cause the world state to transition from $s_0$ to $s_5$. In state $s_0$, the candidate action set is $\{move, take\}$, and each action will transform the starting state into a different state. Thus, we iterate the action selection procedure until the state $s_5$ is reached (Figure 2).

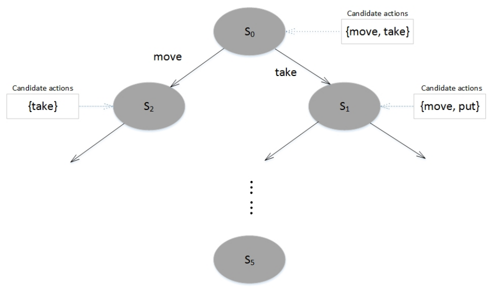

**FIGURE 2. Planning procedure for the DWR problem.**

For the action selection stage, heuristics have been shown to be an effective method [41]–[44]. An action selection heuristic is a method of ranking a set of actions in the order of their relative desirability. Usually, such a heuristic is modeled as a function $h$, which can be used to compute a numeric evaluation $h(a_i)$ for each candidate action in the action set, with a convention that says that the preferred action $a_i \in A$ is the action for which $h(a_i)$ is smallest, i.e., $Select(A) = argmin\,\{h(a_i) \mid a_i \in A\}$.

Many heuristic approaches to planning have been proposed in recent years. Shivashankar *et al.* [45] proposed Hierarchical Goal Network (HGN) Planning, which is a new hierarchical planning formalism for developing the Hierarchically Optimal Goal Decomposition Planner (HOpGDP), an HGN planning algorithm that computes hierarchically optimal plans. In the cited paper, HOpGDP is guided by hHL, a new HGN planning heuristic that extends existing admissible landmark-based heuristics from classical planning to compute admissible cost estimates for HGN planning problems. In the work of Erdem *et al.* [46], as in the work of Burgard *et al.* [47], a task planner that is based on explicit causal reasoning is augmented with the ability to check for the existence of paths for a robot. Garrett *et al.* [48] showed how the heuristic ideas underlying one of the most successful symbolic planners, the FastForward (FF) planner [49], can be extended to motion planning and efficiently computed. They used a multi-query roadmap structure that can be conditionalized to model different placements of movable objects.

Constraint programming (CP) is a powerful paradigm for solving combinatorial problems [37]. CP originated as a multidisciplinary research area that incorporates techniques and concepts from many other areas, among which AI, computer science, databases, programming languages, and operations research play an important role. Constraint programming is currently applied with success in many domains, such as scheduling, planning, vehicle routing [50], configuration [51], networking [52], and bioinformatics [53]. A constraint satisfaction problem and table constraint can be defined as follows:

*Definition 2 (CSP):* A constraint satisfaction problem consists of the following:

- A set of variables $X = \{x_1, x_2, \cdots, x_n\}$.
- A set of domains $D = \{D_1, D_2, \cdots, D_n\}$ such that, for each variable $x_i$, there is a domain $D_i$.
- A set of constraints $C = \{c_1, c_2, \cdots, c_k\}$ such that each constraint defines a predicate that is a relation over a particular subset of $X$.

*Definition 3 (Table constraint):* A table constraint consists of the following:

- An ordered subset of variables, denoted by $scp(c_i)$.
- A subset of Cartesian products that specify the allowed combinations of values for the variables in $scp(c_i)$.

Figure 3 shows a typical constraint satisfaction problem, with the variables and values shown in Table 1. The constraint of this CSP is that two adjacent variables cannot be assigned the same value. For example, if variable A is assigned the value *green*, then its adjacent variables D and B can only be assigned the value *red*.(Figure 4)

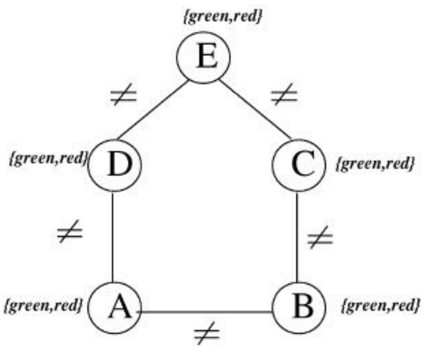

**FIGURE 3. An example of a typical CSP.**

**TABLE 1. Variables and values for the example CSP.**

| Variable | Value |
|---|---|
| A | green, red |
| B | green, red |
| C | green, red |
| D | green, red |
| E | green, red |

When a CSP solver starts, it attempts to iteratively extend a consistent partial assignment of the variables in the problem. This procedure will continue until a consistent assignment of all variables is made. If a partial assignment proves inconsistent, in other words, if the algorithm reaches a dead end, backtracking will be performed. The important aspects of this procedure are the order in which the variables are selected, the order in which the values are assigned to the variables, the method by which the assigned variables are propagated

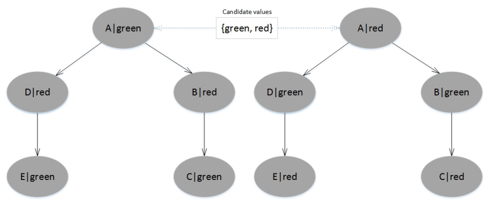

**FIGURE 4.** Two different results for the example CSP.

through the constraints, and the means by which the search procedure backtracks.

Usually, variable selection is the first part of a CSP algorithm, and the ordering of the selection can be either static or dynamic. In the former, the order is selected before the search begins, whereas in the latter, decisions are made based on the current states of the variables. An example of a computationally inexpensive variable ordering heuristic is the Minimum Remaining Values (MRV) heuristic [54], or the first-fail-variable-picked-first method. In this approach, the variable with the lowest number of remaining values is chosen because this variable can soonest be identified as being consistent or not. Consequently, the branching factor is minimized for the longest possible time. The MRV heuristic has been shown to work well on a large number of CSPs [55].

After a variable is chosen to be instantiated, the next step is to assign a value to that variable. A good choice of value will reduce the amount of backtracking required. It should be noted that if the correct choice of value is made at each point in the search, it is possible to obtain a solution with no backtracking. In contrast to the strategy for variable ordering, the strategy for value selection is to choose the value that is most likely to succeed because failure would cause the search to backtrack. An example of a value ordering heuristic is the min-conflict heuristic [56]–[58]. In this method, the values are ordered based on the number of conflicts with the unassigned variables in which they are involved.

Once a variable has been selected and a value assigned, constraint inference can be applied based on this latest assignment. There are many propagation methods available, including forward checking and k-consistency strategies [59]. One commonly used technique is arc consistency [60]–[62]. Here, the algorithm guarantees that any allowable value in a variable's domain is consistent with a permitted value in the domain of any other single variable.

If an inconsistency is found in the propagation step, it is necessary to backtrack. A simple backtracking technique steps back to the last-made assignment and attempts an alternative value from that variable's domain. More advanced methods can pinpoint the variable responsible for the failure and will backtrack accordingly. These include back marking [63], back jumping [64], and conflict-directed back jumping (CBJ) [65], [66].

Among the various differences between an AI planning problem and a CSP, one key distinction is that, in an AI planning problem, the plan length is unknown before the search begins, whereas the size of a CSP is static. A typical solution to this incompatibility is to set a fixed bound $k$ on the horizon of an AI problem and translate the problem into a CSP with $k$ layers. If a solution is found, it is extracted; otherwise, a new horizon bound $k + 1$ is attempted. In this paper, we use $k$ steps to denote this bound. Generally speaking, the number of steps is positively correlated with the plan size.

Another key issue is that the action constraints of an AI planning problem are different from the traditional constraints in a CSP. For example, the instance shown in Figure 1 is a typical CSP. When a solution to the CSP has been found, all constraints must be satisfied (in the example above, the constraint is that no two adjacent elements may be the same). However, the situation is different in a constraint-programmed planner with action constraints. For a simple data download model, there are two actions, *download* and $pointing - transition$, which have already been coded as action constraints. The *download* constraint requires that the probe hold its direction relative to the ground station, whereas the $pointing - transition$ constraint requires the probe to change its direction. Obviously, these two constraints cannot be satisfied simultaneously. In the next section, a dynamic set of constraints is proposed to solve this problem.

## III. AUTOMATICALLY CONVERTING FROM A PDDL MODEL TO A CSP

Following common practice in many planning approaches, we consider a bounded planning problem; i.e., we restrict our target to finding a plan with a length of at most $t$, where $t$ is an integer that is given a priori. In the following, we explain how the planning domain is encoded into a CSP with a horizon length parameter $t \geq 0$. Every variable $var$ in the CSP is converted into a time-tagged form expressed as $var@T$. Correspondingly, for every action $a = \langle pre, eff \rangle$, we have the following formulas for $t \in \{0, \ldots, T - 1\}$:

$$a@t \rightarrow pre@t \tag{1}$$

for expressing the preconditions for an action and

$$a@t \rightarrow eff@(t + 1) \tag{2}$$

for expressing the effects of an action. If the value of a state variable $var$ changes, we have the following frame axioms:

$$\begin{aligned}(var@t \wedge \neg var@(t + 1))& \\ \rightarrow (a_1 \wedge pre(a_1)) \vee \ldots \vee (a_n \wedge pre(a_n))&\end{aligned} \tag{3}$$

From the above formulas, we can see that the actions are specific local relations of variables between two adjacent levels with domain information. Thus, they can be seen as domain-dependent constraints, unlike general CSP constraints. Here, we consider the *download* action constraint as an example. The *download* action constraint describes a situation in which, before the data are downloaded, the memory of the probe stores the data, the observer of interest holds its direction with respect to the ground station, and the ground station has not yet received any of the data. After the data are downloaded, the data in memory are cleared, the observer continues holding its direction with respect to the ground station, and the ground station has received the data. The variables involved in this constraint and their values are as follows:

$$download(mem_t(1), observer_t(Earth), groundstation_t(0), mem_{t+1}(1), observer_{t+1}(Earth), groundstation_{t+1}(1)) \tag{4}$$

The problem space for a complex deep space probe always consists of many actions constructing the relations among different payloads. These actions can be helpful for guiding efficient variable or value selection; however, coding this domain-dependent information by hand is an extremely tedious task. For this reason, we consider how to automatically encode actions as constraints. In the above example, we can extract the features of the action constraint based on the following three considerations.

1) The variables in the action constraint appear in pairs, with a symmetric structure, and are divided into two adjacent time steps.
2) Some variables change in value in the latter time step; we call these variables *changeable variables* of this constraint. Some variables maintain their values between the former and latter time steps; we call these variables *unchangeable variables* of this constraint.
3) Apparently, the variables in time step $t$ represent the preconditions for planning, whereas the variables in time step $t+1$ represent the effects. We can naturally adopt the paradigm of conditional CSP (or dynamic CSP) to present the action constraint as follows:

$$download(mem_t(1), observer_t(Earth), groundstation_t(0) \Rightarrow mem_{t+1}(1), observer_{t+1}(Earth), groundstation_{t+1}(1)) \tag{5}$$

The variables that represent the preconditions for planning are also called *driving variables*, and the variables that represent the effects are called *response variables*.

To transform a planning problem into a CSP, we need to be able to automatically code the domain-dependent conditional action constraints. Here, we adopt table constraints for convenience because action constraints can be expressed in a universal tabular form. An action constraint written as a table constraint consists of two types of components: variables and feasible assignments. The variables can be extracted from the domain file of the planning problem, and the feasible assignments can be extracted from the problem file. In the following subsections, we shall explain the translation process using a classic planning problem known as the gripper problem as an example. The gripper problem is described as follows. In the initial state, several balls are placed randomly in several rooms. The planning goal is to move these balls to the desired location using robots. Usually, each robot has two grippers with which to hold and transport a ball.

## A. TYPES AND FILES

A planning model consists of two kinds of files. One is the domain file, which includes the domain name, the types of variables and predicates, and all relations between variables (which are also called actions). The other is the problem file, which instantiates all elements in the domain file and specifies the initial and goal states. For example, the domain file for the gripper problem includes the variable type, which has a special syntax for declaring parameter types. The following shows the typing declaration for the gripper problem.

```
(:type room - location
       gripper ball robby - object)
```

The domain file also includes the following predicate:

```
(:predicates (at-robby ?room)
             (at ?ball ?position)
             (free ?gripper)
             (carry ?ball ?gripper))
```

Finally, the domain file defines actions, including *move*, *pick*, and *drop*.

In the problem file, the variables will be specified as follows:

```
(:location
     room1 room2 room3 room4 room5
 objects
     left right - gripper
     ball1 ball2 ball3 ball4 ball5
     robby
```

Similarly, the initial/goal states are also specified in the problem file.

These model files provide sufficient information with which to build a table constraint for each action. For example, we can store the variable information $room = \{room1, room2, room3, room4, room5\}$ for later use.

**TABLE 2.** Variables and values for the example table constraint.

| ?x | robot1 | robot2 |
|---|---|---|
| ?r | room1 | room1 |
|  | room2 | room2 |

## B. PREDICATES

In PDDL, preconditions and effects are expressed as logical expressions of predicates. Most predicates are meaningful for convenience in understanding a statement, e.g., $(at?x?r)$ or $(locked?x)$. However, for table constraints, these different logical relations can be simplified as enumerated feasible variable-value relations. For example, the predicate $(at?x?r)$ can be expressed as shown in Table 2. for $x = \{robot1, robot2\}$ and $r = \{room1, room2\}$. For a

**TABLE 3. Variables for action "pick".**

| Variables | $ball_t$ | $gripper_t$ | $room_t$ | $state_t$ | $position_t$ | $robby_t$ | $ball_{t+1}$ | $gripper_{t+1}$ | $room_{t+1}$ | $state_{t+1}$ | $position_{t+1}$ | $robby_{t+1}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Values | | | | | | | | | | | | |

**TABLE 4. Simplified variables for action "pick".**

| Variables | $ball$ | $gripper$ | $robby$ | $room$ | $position_t$ | $state_t$ | $position_{t+1}$ | $state_{t+1}$ |
|---|---|---|---|---|---|---|---|---|
| Values | | | | | | | | |

**TABLE 5. Table constraint for action "pick".**

| Variables | $ball$ | $gripper$ | $robby$ | $room$ | $position_t$ | $state_t$ | $position_{t+1}$ | $state_{t+1}$ |
|---|---|---|---|---|---|---|---|---|
| Values | ball1 | left | robot | room1 | room1 | 0 | left | 1 |
| | ball1 | right | robot | room1 | room1 | 0 | right | 1 |
| | ball2 | left | robot | room1 | room1 | 0 | left | 1 |
| | ball2 | right | robot | room1 | room1 | 0 | right | 1 |
| | ball1 | left | robot | room2 | room2 | 0 | left | 1 |
| | ball1 | right | robot | room2 | room2 | 0 | right | 1 |
| | ball2 | left | robot | room2 | room2 | 0 | left | 1 |
| | ball2 | right | robot | room2 | room2 | 0 | right | 1 |
| ... | | | | | | | | |

different logical relation such as ($move ?x ?p$), the form of the constraint is still the same.

For a consistent structure, we convert all monadic predicates into binary predicates. For example, the predicate ($locked ?x$) will be expressed as ($islocked ?x ?state$). Here, the type of $?state$ can be Boolean or enumeration. Notably, for most binary predicates of the form $pre < ?x, ?y >$ that appear in planning problems, the atom $y$ is used to constrain or indicate a certain property of $x$. For each predicate in the domain file, we select atoms as the candidate variables and record them as a set named $candiVar[]$.

## C. ACTIONS

In a PDDL domain description, actions are defined to specify the behavioral aspects of the model in terms of preconditions and effects. When the object types and predicates are known, the translation of actions into a CSP is straightforward. Let us consider the *pick* action in the domain of the gripper problem as an example.

First, we check the predicates of the action to obtain the variables for the table constraint.

```
(:action pick
    :parameters (?ball ?room ?gripper ?robby)
    :precondition (and (at ?ball ?room)
                  (at-robby ?robby ?room)
                  (isfree ?gripper ?state))
    :effect      (and (carry ?ball ?gripper)
                  (not (at ?ball ?room))
                  (not (isfree ?gripper ?state))))
```

In the *pick* action, the variables include $\{?ball, ?room, ?gripper, ?robby, ?state, ?position\}$. As mentioned before, an action connects two adjacent levels that serve as preconditions and effects; therefore, the table constraint is extracted as shown in Table 3.

Then, we examine the preconditions for and effects of the action following the three steps listed below:

1) Record the value type for each variable.
2) Record the variation of the variables between the two levels.
3) Select the unchanged variables and merge.

For the example shown in Table 3, the unchanged variables are *ball*, *gripper*, *room*, and *robby*. Therefore, the table constraint can be further simplified as shown in Table 4 to save storage space.

## D. PROBLEM FILE

In PDDL, the specific conditions of a planning problem are captured by the problem file. For the example above, the specific conditions are as follows:

```
(at-robby room1)
    (free left)
    (free right)
    (at ball1 room1)
    (at ball2 room1)
    (at ball3 room1)
    (at ball4 room1)
    (at ball5 room1)
```

From this problem information and the relations in Table 4, we can fill out the incomplete table as in Table 5, thereby completing the conversion of the action constraint into the form of a table constraint.

### E. GENERAL CONSTRAINTS

In addition to planning-domain-dependent constraints, as considered above, DSPlan also includes some general constraints that are commonly used in domain-independent planners. These constraints are directly coded in the planner, and we briefly introduce the general constraints included in DSPlan as follows:

1) Shortest-path constraints are constraints that restrict redundant action sequences that achieve the same result. For example, an action $pointing-transition(a, b)$ will result in an observer on the probe changing its direction from $a$ to $b$, and an action sequence of $pointingtransition(a, c)$, $pointingtransition(c, d)$, and $pointingtransition(d, b)$ achieves the same result. With shortest-path constraints, such redundant action sequences can be avoided.
2) Symmetric value constraints are constraints that break symmetries in the values that can be assigned to variables. For example, suppose that there are two observers on a probe and two targets that need to be imaged. The observer variables and their domains are often symmetric so that, when there is an assignment for one observer, there can be an equivalent assignment for the other observer that is simply swapped. Symmetric value constraints have been found to be very important for reducing the symmetric structure of the domains that we have explored.
3) No-op constraints are inspired by the concept of no-op actions in GraphPlan. Similar to no-op actions, no-op constraints transform the value of a variable $V_t$ to $V_{t+1}$ if no other constraint ties this variable or changes its value. These constraints have been found to be important for constraint handling and propagation.

## IV. MUTEX FILTERING ALGORITHM

After converting the actions into table constraints, we need to address the problem of action constraints that cannot be satisfied simultaneously. Here, we construct a dynamic set of action constraints with multiple layers. The rules for constructing this set are as follows:

1) The layer number of a constraint is equal to the selected variable's time step, which means that the layer number starts from one.
2) Action constraints in the same layer cannot be mutually exclusive.
3) Newly added constraints cannot change the values of other assigned variables; otherwise, other constraints would no longer be satisfied.
4) Action constraints in the same layer can be satisfied simultaneously, which denotes the concurrency of these actions.

Based on the four rules above, the pseudo code for this construction process is shown in algorithm 1.

**Algorithm 1** The Algorithm for Constructing the Dynamic Set

```
Input: current_variable
Output: consistent set of action constraints
1  for every constraint involved current_variable do
2      if layer(current_variable.TimeStep) == Null then
3          $C_i.layer = current\_variable.TimeStep$;
4      end
5      else
6          if isconsistent(layer(current_variable.TimeStep), $C_i$) then
7              $C_i.layer = current\_variable.TimeStep$;
8          end
9          return 0;
10     end
11 end
```

Layer $k$ is used to denote the $k$-th layer of the constraints. Because a variable always has the same data structure at different time steps, in the third phase of the heuristic, a driving variable will always be selected before its response variable. Therefore, the layer number of a constraint is always equal to the time step of the driving variable, thus avoiding confusion in the constraint hierarchy.

In the construction process, the function *isconsistent*() is the core step in determining whether an action constraint is added to the set or dropped. To make this determination, the first step is to check whether the constraint is consistent with other constraints in the same layer. Clearly, if two action constraints have the same driving variable but different values, they must be mutually exclusive. Next, if they have the same value, we list the four different situations in Table 6.

When two driving variables have the same value, we check whether they are a changeable variable. As situation 1 shows, if a variable in $C_1$ is changeable while that variable in $C_2$ is unchangeable, their response variables will obtain different values, which makes $C_1$ and $C_2$ inconsistent.

We can see that only scenario 3 is consistent. Finally, we must ensure that newly added constraints cannot change the values of other assigned variables; otherwise, other constraints would no longer be satisfied. Thus, the response variables of the checked constraint must be unassigned. The corresponding pseudo code is shown in algorithm 2.

Once the action constraints have been addressed as above, the constraint process can be applied using the general constraints and the selected action constraints. The pseudo code for the planner is shown in algorithm 3.

Algorithm 3 starts with a *currentlevel* equal to one. In the algorithm, the key functions are in steps 4, 5, and 8. Once the next variable has been chosen, a value in this variable's domain will be selected, and the algorithm will check whether

**TABLE 6. Consistent situations between constraints.**

| Situation | $C_1$ | $C_2$ | consistent |
|---|---|---|---|
| 1 | changeable | unchangeable | NO |
| 2 | changeable | changeable | NO |
| 3 | unchangeable | unchangeable | YES |
| 4 | unchangeable | changeable | NO |

**Algorithm 2** Function iscnsistent()

```
Input: $C_i$, layer(n)
Output: consistent or not
1  for every constraint in layer(n) do
2      for every driving variable in $C_i$ do
3          if $var \in C_j$ then
4              if var.value in $C_i$ != var.value in $C_j$ then
5                  return 0;
6              end
7              else
8                  if situation 3 in table 3 no longer
                   satisfied then
9                      return 0;
10                 end
11                 else
12                     if response variable of var has been
                       assigned then
13                         return 0;
14                     end
15                 end
16             end
17         end
18     end
19     return 1;
20 end
```

**Algorithm 3** Complete Algorithm Process

```
Input: CSP(the problem information), current_level
       (started from one)
Output: solution
1  if current_level==1 then
2      obtain constraint number of each variable involved;
3  end
4  current_variable ← VarSelection();
5  construct_dynamic_set(current_variable);
6  for every value in the domain of the current_variable do
7      assignValue(current_variable);
8      if ConstraintCheck(CSP,current_variable)==1
       then
9          DSPlan(CSP,solution,current_level + 1);
10     end
11     else
12         restore();
13     end
14 end
```

the consistency of the general constraints and dynamic action constraints can be satisfied for this value. In the function *ConstraintCheck*(), the GAC algorithm [67] is adopted. If the assignment is consistent, algorithm 3 will continue to *currentlevel* + 1; otherwise, algorithm 3 will return to the current variable, look for another value for this variable, or even backtrack.

## V. SIMULATION EXPERIMENT

The developed solver was written in C/C++ and is called DSPlan. In the experimental section, we demonstrate that DSPlan is suitable for deep space exploration modelčwhich is characterized by more loads and complex load constraints. To evaluate the performance of the proposed modeling method and algorithm, we performed experimental comparisons with the GP-CSP and Europa planners. GP-CSP is a GraphPlan-like constraint-programmed planner, and Europa is a partial order planner that has been widely used in the aerospace field since the Deep Space 1 (DS1) mission. All the compared planners are domain-independent planners. In the comparison experiments, these planners were used to test the domains from previous IPCs, and three deep space exploration missions were also proposed to check the performance of DSPlan in real problems. The experiments were performed on a system with an Intel i5-2430 2.4 GHz CPU and 4 GB of RAM.

### A. IPC DOMAINS

The tested IPC domains include: blockworld, gripper, and logistic. These three domains separately contain different numbers of actions. For example, the blockworld domain is described as moving blocks to their desired state. Accordingly, there is only one action, *move*, in its domain file. Gripper is an instance of the automated planning of a robot with two gripper arms to move balls among different rooms, and there are three actions in the gripper's domain. Logistic, which has six actions, is the most complex domain compared to blockworld and gripper. In logistic, a driver must manage to move a package to another location. He or she needs to *load* the package on the truck, *drive* to the airport, *unload* the package, *load* the package on the airplane, and finally *fly* to the desired location. This domain is very similar to deep space exploration problems. Figure 5 shows the experimental result for blockworld and the runtime for generating the table constraints.

In the figure, we can see that, although the coding time for table constraints did not take too much of the total running time of DSPlan, the efficiency was much worse than EUROPA and GP-CSP. This is because the blockworld domain is so simple that it only contains one action. Coding it as table constraints did not obtain sufficient domain information to compensate for the time cost. The efficiency of DSPlan improves with the increases in the problem's complexity, as we can see in the planning result in Figure 6 for the gripper domain.

**TABLE 7.** Subsystems and activity of detector.

| Subsystem | Activity |
|---|---|
| AttitudeSys | Pointing |
| | Turning |
| ThrustSys | ThrustPrepare |
| | Thrusting |
| | ThrustClose |
| | ThrustOpen |
| ComsystoHome | ComhomeOpen |
| | PhoningHome |
| | ComhomeClose |
| ComsysLander | ComlanderOpen |
| | PhoningLander |
| | ComlanderClose |
| SunArray | Stay |
| | ToAxis |
| | ToZero |
| DetachMechanism | Cutwire |
| | WireSeparating |
| | Unlock |
| | Deblock |
| GNCSys | NoneAdjust |
| | BeforeAdjust |
| | AfterAdjust |
| Observer | CamOpen |
| | CamClose |
| | Photograph |
| | Focusing |

**TABLE 8.** Events for orbit adjustment mission.

| Number | Event |
|---|---|
| 1 | The detector enters the orbit adjustment planning state |
| 2 | Close the loads |
| 3 | Adjusting measurement and control mode |
| 4 | Locking mechanism of solar wing drive |
| 5 | Attitude maneuver,build up ignition attitude |
| 6 | Orbit adjustment |
| 7 | Adjusting measurement and control mode |

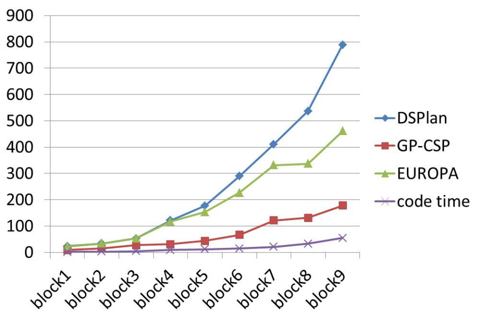

**FIGURE 5.** Planning results of blockworld.

There are three gripper actions. Although the coding time has increased, the total running time decreases, and DSPlan is faster than the other two planners in some of the instances.

**TABLE 9.** Events for SLD mission.

| Number | Event |
|---|---|
| 1 | The detector enters the SLD planning state |
| 2 | Adjusting measurement and control mode |
| 3 | Locking mechanism of solar wing drive |
| 4 | Adjusting GNC mode |
| 5 | Attitude maneuver, build up ignition attitude |
| 6 | Inertia orientation, hold ignition attitude |
| 7 | Descending |
| 8 | Descending to the separation point |
| 9 | Cable power off |
| 10 | Cable cutting |
| 11 | Lander unlock and separation |
| 12 | Attitude maneuver, build up ignition attitude |
| 13 | Inertia orientation, hold ignition attitude |
| 14 | Ascending |
| 15 | Ascending to target orbit |
| 16 | Attitude maneuver, build up ignition attitude |
| 17 | Adjusting solar wing |
| 18 | Adjusting GNC mode |
| 19 | Adjusting measurement and control mode |

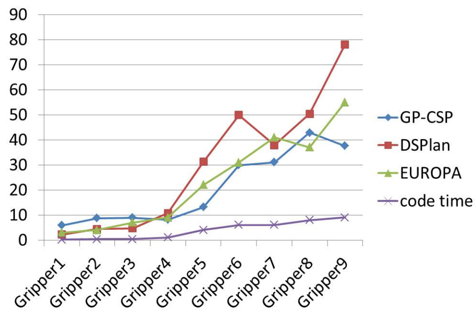

**FIGURE 6.** Planning results of gripper.

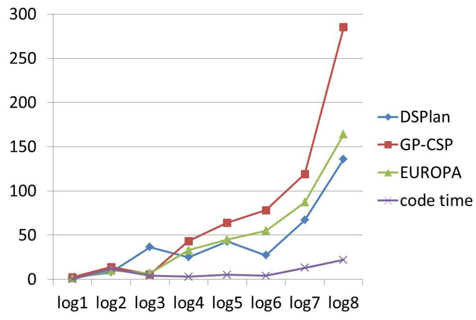

**FIGURE 7.** Planning results of logistic.

Figure 7 further proves that, with more actions in the domain, the DSPlan has a greater advantage compared to the domain-independent planner.

### *B. DEEP SPACE EXPLORATION INSTANCES*

In this section, we tested DSPlan with three deep space exploration instances: an SLD mission, orbit adjustment mission, and observation mission. The domain file for a deep space

**TABLE 10.** Ten instances of the observation mission.

| Instance | Object/num | Observer/num | Mem/num | Warmer/num | Groundstation/num |
|---|---|---|---|---|---|
| 1 | 2 | 2 | 2 | 2 | 2 |
| 2 | 3 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 | 2 |
| 4 | 5 | 3 | 3 | 3 | 2 |
| 5 | 12 | 3 | 3 | 3 | 2 |
| 6 | 12 | 6 | 6 | 3 | 3 |
| 8 | 12 | 6 | 3 | 6 | 3 |
| 9 | 5 | 3 | 3 | 3 | 4 |
| 10 | 12 | 6 | 6 | 6 | 4 |

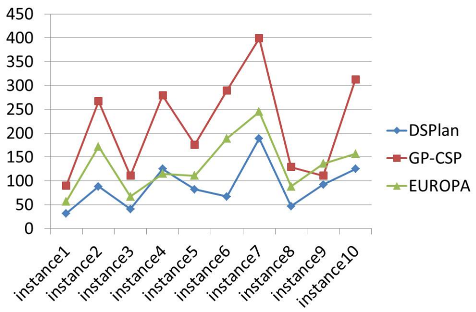

**FIGURE 8.** Planning result for orbit adjustment mission.

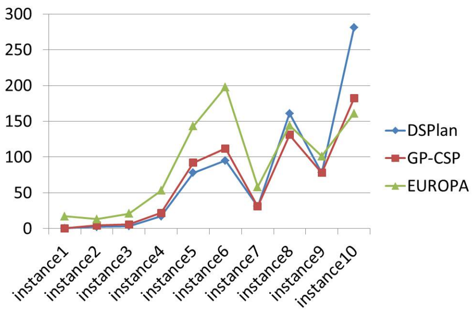

**FIGURE 10.** Planing result for observation mission.

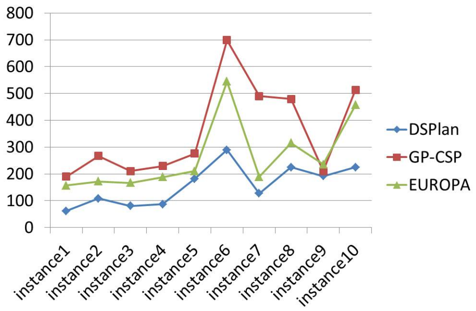

**FIGURE 9.** Planning result for an SLD mission.

detector is built in Table 7. The model for the probe consists of 8 subsystems and 26 actions. Part of the systems and actions are only used for particular missions. For example, subsystem DetachMechanism is only used for an SLD mission.

An orbit adjustment mission's purpose is to figure out the actions before and after the orbit adjustment event, as listed in Table 8.

We adjusted the initial status and performed 10 tests respectively. The results are shown in Figure 8. The advantage of a domain-dependent planner becomes more obvious with domain-specific action constraints and a means of specific processing to help guide the search. In addition, it is especially suitable for deep space exploration planning, which has complex structures as well as numerous constraints.

Table 9 shows the events in an SLD mission, and the planning results are shown in Figure 9.

However, the proposed method is not suitable for problems with increasing size because the method to generate table constraints is a combinatorial problem. With larger-sized problems, the spatial complexity of the problem will increase dramatically. This can be seen in an observation mission (Table 10). The purpose of this mission is to adjust the warmers and attitude to observe the targets, then download the phonos to the ground stations. As we increase the number of targets, observers, warmers, and ground stations, the efficiency of planning decreases. For example, in instance 10 in Figure 10, the time spent in the DSPlan has gone far beyond EUROPA and GP-CSP. Fortunately, the number of each payload on the detector is relatively small, and the number is already fixed before the task begins.

## VI. CONCLUSION

Domain-dependent planners are usually highly efficient, unlike domain-independent planners, and this is especially important in deep space exploration because of the stringent requirements for fast planning for such detectors. In this paper, we designed a constraint-programmed planner and coded the actions in the planning problem as domain-dependent constraints. To overcome the defects of domain-dependent planners, we proposed a technique for automatically converting the PDDL model used in planning into the form of table constraints. Experimental results proved that the time cost for conversion is a worthy trade-off for the gain in efficiency.

Based on the formulated table constraints, we also proposed a dynamic constraint set and a corresponding mutex filtering algorithm to address the problem of action constraints that cannot be satisfied simultaneously. Experimental results suggest that the more constraints there are in a planning problem, the more obvious are the advantages of DSPlan because it can obtain more domain information to help to prune the relevant branches and guide the search.

However, there are still some deficiencies in the proposed planner because we have designed it based only on classical domains and, consequently, some numerical constraints are not included. In future research, we will incorporate time constraints and resource constraints into the probe model to achieve a stronger connection with engineering practice.

## REFERENCES

[1] H. Price, R. Manning, and E. Sklyanskiy, "A high-heritage blunt-body entry, descent, and landing concept for human mars exploration," in *Proc. 54th AIAA Aerosp. Sci. Meet.*, 2015, pp. 1–5.

[2] P. Cui, X. U. Rui, S. Zhu, and F. Zhao, "State of the art and development trends of on-board autonomy technology for deep space explorer," *Acta Aeron. Et Astronautica Sinica*, vol. 35, no. 1, pp. 13–28, 2014.

[3] E. C. Ezell and L. N. Ezell, *On Mars: Exploration of the Red Planet, 1958-1978–The NASA History*. North Chelm, MA, USA: Courier Corp., 2013.

[4] D. Rubin *et al.*, "A calibration of NICMOS camera 2 for low count rates," *Astronomical J.*, vol. 149, no. 5, p. 15, 2015.

[5] W. Davies and P. North, "Estimating aod using a quad-modal size distribution," in *Proc. EGU Gen. Assembly Conf.*, vol. 15. 2013, paper EGU2013-3163.

[6] S. Santandrea *et al.*, "Proba2: Mission and spacecraft overview," *Solar Phys.*, vol. 286, no. 1, pp. 5–19, 2013.

[7] T. V. Peters, J. Branco, D. Escorial, L. T. Castellani, and A. Cropp, "Mission analysis for PROBA-3 nominal operations," *Acta Astron.*, vol. 102, pp. 296–310, Oct. 2014.

[8] X. Zhao, H. Yang, H. R. Karimi, and Y. Zhu, "Adaptive neural control of MIMO nonstrict-feedback nonlinear systems with time delay," *IEEE Trans. Cybern.*, vol. 46, no. 6, pp. 1337–1349, Jun. 2016.

[9] H. Kautz, D. McAllester, and B. Selman, "Encoding plans in propositional logic," *Proc. KR*, vol. 96, pp. 374–384, Nov. 1996.

[10] W. Carnielli and M. Matulovic, "Non-deterministic semantics in polynomial format," *Electron. Notes Theor. Comput. Sci.*, vol. 305, pp. 19–34, Jul. 2014.

[11] X. Zhao, P. Shi, and X. Zheng, "Fuzzy adaptive control design and discretization for a class of nonlinear uncertain systems," *IEEE Trans. Cybern.*, vol. 46, no. 6, pp. 1476–1483, Jun. 2015.

[12] K. Ghédira, *Constraint Satisfaction Problems: CSP Formalisms and Techniques*. Hoboken, NJ, USA: Wiley, 2013.

[13] P. van Beek and X. Chen, "CPLAN: A constraint programming approach to planning," in *Proc. AAAI/IAAI*, 1999, pp. 585–590.

[14] L. A. Castillo, J. Fernández-Olivares, O. Garcia-Perez, and F. Palao, "Efficiently handling temporal knowledge in an HTN planner," in *Proc. ICAPS*, Jun. 2006, pp. 63–72.

[15] I. Little and S. Thiebaux, "Concurrent probabilistic planning in the graphplan framework," in *Proc. ICAPS*, 2006, pp. 263–273.

[16] M. Kapadia, J. Falk, F. Zünd, M. Marti, R. W. Sumner, and M. Gross, "Computer-assisted authoring of interactive narratives," in *Proc. I3D*, 2015, pp. 85–92.

[17] M. B. Do and S. Kambhampati, "Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP," *Artif. Intell.*, vol. 132, no. 2, pp. 151–182, 2001.

[18] N. Creignou and D. L. Berre, *Theory and Applications of Satisfiability Testing—SAT* (Lecture Notes in Computer Science). Paris, France: Springer, 2016.

[19] P. Q. Pan, *Integer Linear Programming (ILP)*. Berlin, Germany: Springer, 2014.

[20] B. Zhang, L. Tang, J. DeCastro, M. Roemer, and K. Goebel, "Autonomous vehicle battery state-of-charge prognostics enhanced mission planning," *Int. J. Progn. Health Manag*, vol. 5, no. 2, pp. 1–12, 2014.

[21] S. Chien *et al.*, "A demonstration of robust planning and scheduling in the TechSat-21 autonomous sciencecraft constellation," *Ear Nose Throat J.*, vol. 86, no. 8, pp. 506–511, 2014.

[22] S. Yin, H. Yang, and O. Kaynak, "Sliding mode observer-based FTC for Markovian jump systems with actuator and sensor faults," *IEEE Trans. Autom. Control*, vol. 62, no. 7, pp. 3551–3558, Jul. 2017.

[23] N. Muscettola, P. Nayak, B. Pell, and B. C. Williams, "Remote agent: To boldly go where no AI system has gone before," *Artif. Intell.*, vol. 103, nos. 1–2, pp. 5–47, 1998.

[24] S. Yin, H. Gao, J. Qiu, and O. Kaynak, "Descriptor reduced-order sliding mode observers design for switched systems with sensor and actuator faults," *Automatica*, vol. 76, pp. 282–292, Feb. 2017.

[25] B. Pell *et al.*, "A remote agent prototype for spacecraft autonomy," *Proc. SPIE*, vol. 2810, pp. 74–90, Oct. 1996.

[26] B. Cichy *et al.*, "Validating the autonomous EO1 science agent," in *Proc. Int. Workshop Planning Schedule Space*, Darmstadt, Germany, Jun. 2004, 2010.

[27] R. Knight, C. Chouinard, G. Jones, and D. Tran, "Leveraging multiple artificial intelligence techniques to improve the responsiveness in operations planning: Aspen for orbital express," *AI Mag.*, vol. 35, no. 4, pp. 26–36, 2014.

[28] S. Yin, H. Gao, J. Qiu, and O. Kaynak, "Fault detection for nonlinear process with deterministic disturbances: A just-in-time learning based data driven method," *IEEE Trans. Cybern.*, to be published.

[29] J. Faublee, "Planning and scheduling for fleets of earth observing satellites," in *Proc. 6th Int. Symp. Artif. Intell. Robot. Autom. Space*, 2001, p. 307.

[30] S. Bernardini and D. E. Smith, "Developing domain-independent search control for europa2," in *Proc. Workshop Heuristics Domain*, 2008.

[31] G. Verfaillie and C. Pralet, "A timeline, event, and constraint-based modeling framework for planning and scheduling problems," in *Proc. Knowl. Eng. Planning Scheduling*, 2013, p. 61.

[32] D. Achlioptas, S. Hamed Hassani, N. Macris, and R. Urbanke, "Bounds for random constraint satisfaction problems via spatial coupling," in *Proc. 27th ACM-SIAM Symp. Discrete Algorithms*, 2016, pp. 469–479.

[33] M. Bodirsky. (2012). "Complexity classification in infinite-domain constraint satisfaction." [Online]. Available: https://arxiv.org/abs/1201.0856

[34] C. D. Rosin. (2014). "Unweighted stochastic local search can be effective for random CSP benchmarks." [Online]. Available: https://arxiv.org/abs/1411.7480

[35] K. Stergiou, "Restricted path consistency revisited," in *Principles and Practice of Constraint Programming*. Berlin, Germany: Springer, 2015, pp. 419–428.

[36] M. Kolombo and R. Barták, *A Constraint-Based Planner for Mars Express Orbiter*. Berlin, Germany: Springer, 2014.

[37] E. P. K. Tsang, *Foundations of Constraint Satisfaction*. London, U.K.: DBLP, 1993, pp. 188–224.

[38] M. Vallati, L. Chrpa, M. Grzes, T. L. Mccluskey, M. Roberts, and S. Sanner, "The 2014 international planning competition: Progress and trends," *AI Mag.*, vol. 36, no. 3, pp. 90–98, 2015.

[39] A. Coles, "A survey of the seventh international planning competition," *Ai Mag.*, vol. 33, no. 1, pp. 83–88, 2012.

[40] R. W. Butler, R. I. Siminiceanu, and C. A. Munoz, "The ANMLite language and logic for specifying planning problems," NASA Langley Res. Center, Hampton, VA, USA, Tech. Rep. NASA/TM-2007-215088, 2008.

[41] E. Karpas and C. Domshlak, "Cost-optimal planning with landmarks," in *Proc. IJCAI*, Jul. 2009, pp. 1728–1733.

[42] P. Haslum, "Admissible heuristics for optimal planning," in *Proc. AIPS*, 2010, pp. 140–149.

[43] B. Bonet and M. Helmert, "Strengthening landmark heuristics via hitting sets," in *Proc. Conf. ECAI*, 2010, pp. 329–334.

[44] E. Keyder, S. Richter, and M. Helmert, "Sound and complete landmarks for and/or graphs," in *Proc. Ecai*, 2010, pp. 335–340.

[45] V. Shivashankar, R. Alford, M. Roberts, and D. W. Aha, "Cost-optimal algorithms for planning with procedural control knowledge," in *Proc. IOS*, 2016, pp. 1702–1703.

[46] E. Erdem, K. Haspalamutgil, C. Palaz, T. Uras, and V. Patoglu, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *Proc. ICRA*, May 2011, pp. 4575–4581.

[47] W. Burgard, C. Stachniss, G. Grisetti, and B. Steder, "A comparison of slam algorithms based on a graph of relations," in *Proc. Int. Conf. Intell. Robots Syst. (IROS*, 2009, pp. 2089–2095.

[48] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, *FFRob: An Efficient Heuristic for Task and Motion Planning*. New York, NY, USA: Springer, 2015, pp. 179–195.

[49] J. Hoffmann, "FF: The fast-forward planning system," *AI Mag.*, vol. 22, no. 3, pp. 57–62, 2014.

[50] C. Ratke, H. Hoffmann, and H. Nehring, "Routing of vehicles using CSP: Case sudy," in *Proc. 9th Int. Conf. Complex, Intell. Soft. Intensive Syst.*, 2015, pp. 250–253.

[51] D. J. Geschwender, R. J. Woodward, and B. Y. Choueiry, "Characterizing performance of consistency algorithms by algorithm configuration of random CSP generators," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 4162–4163.

[52] P. Antonino, A. Sampaio, and J. Woodcock, *A Refinement Based Strategy for Local Deadlock Analysis of Networks of CSP Processes*. New York, NY, USA: Springer, 2014.

[53] M. Sanchez, S. D. Givry, and T. Schiex, "Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques," in *Proc. Conf. Artif. Intell. Res. Develop.*, 2007, p. 917.

[54] I. P. Gent, E. Macintyre, P. Prosser, and T. Walsh, "The constrainedness of search," in *Proc. AAAI*, vol. 1. 2002, pp. 246–252.

[55] R. Dechter and I. Meiri, "Experimental evaluation of preprocessing algorithms for constraint satisfaction problems," *Artif. Intell.*, vol. 68, no. 2, pp. 211–241, 1994.

[56] R. Barták, "Constraint processing," *Acta Autom. Sinica*, vol. 17, no. 2, p. 687, 2007.

[57] H. Li, X. Yang, and Y. Ouyang, "MCHRC: Min-conflict heuristic based Web services chain reconfiguration approach," in *Proc. Int. Conf. Comput. Intell. Softw. Eng.*, 2009, pp. 1–4.

[58] H. Li and X. Yang, "A min-conflict heuristic-based Web service chain reconfiguration approach," *Intell. Inf. Manage.*, vol. 2, no. 10, pp. 597–607, 2010.

[59] E. C. Freuder, "Synthesizing constraint expressions," *Commun. ACM*, vol. 21, no. 11, pp. 958–966, 1978.

[60] A. K. Mackworth, "Consistency in network of relations," *Artif. Intell.*, vol. 8, no. 1, pp. 99–118, 1977.

[61] N. Manthey, P. Steinke, and T. Philipp, "A more compact translation of pseudo-Boolean constraints into CNF such that generalized arc consistency is maintained," in *Proc. Adv. Artif. Intell.*, 2014, pp. 123–134.

[62] M. Siala, E. Hebrard, and M.-J. Huguet, "An optimal arc consistency algorithm for a particular case of sequence constraint," *Constraints*, vol. 19, no. 1, pp. 30–56, 2014.

[63] Y. Jiang, T. Richards, and B. Richards, "Nogood backmarking with min-conflict repair in constraint satisfaction and optimization," in *Proc. Int. Workshop Principles Pract. Constraint Program.*, 1994, pp. 21–39.

[64] R. Dechter and D. Frost, "Backjump-based backtracking for constraint satisfaction problems," *Artif. Intell.*, vol. 136, no. 2, pp. 147–188, 2002.

[65] P. Prosser, "Hybrid algorithms for the constraint satisfaction problem," *Comput. Intell.*, vol. 9, no. 3, pp. 268–299, 1993.

[66] X. Chen, V. Beek, and Peter, "Conflict-directed backjumping revisited," *J. Artif. Intell. Res.*, vol. 14, no. 1, pp. 53–81, 2011.

[67] K. C. K. Cheng and R. H. C. Yap, "Maintaining generalized arc consistency on ad-hoc *n*-ary Boolean constraints," in *Proc. ECAI*, 2006, pp. 78–82.

**XIAO JIANG** born in 1986. He is currently pursuing the Ph.D. degree in aerospace engineering with the Beijing Institute of Technology, China. His research interests are deep space probe autonomous operation and autonomous mission planning.

**RUI XU** received the Ph.D. degree from the Harbin Institute of Technology. He is currently a Doctoral Supervisor and an Associate Professor of aerospace engineering with the Beijing Institute of Technology. His work has been published in a number of different journals. His research interests are deep space probe autonomous operation, autonomous mission planning, and autonomous navigation.