Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Task planning in robotics: an empirical comparison of PDDL- and ASP-based systems

Yu-qian JIANG[1], Shi-qi ZHANG[‡2], Piyush KHANDELWAL[3], Peter STONE[1]

[1]*Department of Computer Science, The University of Texas at Austin, TX 78712, USA*

[2]*Department of Computer Science, The State University of New York at Binghamton, NY 13902, USA*

[3]*Cogitai, Inc., TX 78756, USA*

E-mail: jiangyuqian@utexas.edu; szhang@cs.binghamton.edu; piyushk@gmail.com; pstone@cs.utexas.edu

Received Aug. 29, 2018; Revision accepted Feb. 7, 2019; Crosschecked Mar. 14, 2019

Abstract: Robots need task planning algorithms to sequence actions toward accomplishing goals that are impossible through individual actions. Off-the-shelf task planners can be used by intelligent robotics practitioners to solve a variety of planning problems. However, many different planners exist, each with different strengths and weaknesses, and there are no general rules for which planner would be best to apply to a given problem. In this study, we empirically compare the performance of state-of-the-art planners that use either the planning domain description language (PDDL) or answer set programming (ASP) as the underlying action language. PDDL is designed for task planning, and PDDL-based planners are widely used for a variety of planning problems. ASP is designed for knowledge-intensive reasoning, but can also be used to solve task planning problems. Given domain encodings that are as similar as possible, we find that PDDL-based planners perform better on problems with longer solutions, and ASP-based planners are better on tasks with a large number of objects or tasks in which complex reasoning is required to reason about action preconditions and effects. The resulting analysis can inform selection among general-purpose planning systems for particular robot task planning domains.

Key words: Task planning; Robotics; Planning domain description language (PDDL); Answer set programming (ASP)

https://doi.org/10.1631/FITEE.1800514 **CLC number:** TP242

1 Introduction

In a general-purpose planning system, task planning problems are tackled by problem-independent solvers based on a description of the domain in a declarative language. Such planning systems are extremely useful in application domains where many different planning goals need to be accomplished, or the domain description evolves over time. For instance, in an application domain such as robotics, a mobile service robot may need to solve planning tasks such as collecting documents, making deliveries, or providing navigation assistance to visitors (Cambon et al., 2009; Erdem et al., 2012; Khandelwal et al., 2017). It is convenient to achieve all these tasks using knowledge declared in a single description of the domain, and general-purpose planning systems are well suited to the task.

To design a general-purpose planning system, a declarative language for formalizing the domain first needs to be selected, followed by the selection of a suitable solver which supports this language. Many different factors affect this selection process. Every language has its limitations in representing task planning problems, and given a particular language, specific language-dependent techniques may

‡ Corresponding author

ORCID: Yu-qian JIANG, http://orcid.org/0000-0003-0091-6871; Shi-qi ZHANG, http://orcid.org/0000-0003-4110-8213

need to be employed to succinctly formalize a particular planning problem. For instance, not all languages support default reasoning, which might bring difficulties in formalizing some planning problems. A solver is also typically tied to a particular language, but may not support all features in that language, requiring careful construction of the domain description using only supported features. Additionally, the properties of the domain can affect how quickly a given pair of language and solver can solve planning problems. For instance, some domains include many objects and their properties, which can be challenging to some planning systems. Finally, given a language, a solver and a planning problem, there can be many ways of formalizing the problem using the language. For these reasons, careful consideration needs to be given to the selection of language and solver.

This article aims to help in the language selection process, given a task planning problem at hand. Specifically, we compare two declarative languages: the planning domain definition language (PDDL) (McDermott et al., 1998), the most popular language in the planning community, and answer set programming (ASP) (Gelfond and Kahl, 2014; Lifschitz, 2008), a popular general knowledge representation and reasoning (KRR) language that has been recently used in a variety of task planning problems (Lifschitz, 2002; Yang et al., 2014; Erdem et al., 2016), including robotics (Erdem and Patoglu, 2018). PDDL was created for the explicit purpose of solving planning problems, whereas the development of ASP has focused on a broader set of reasoning tasks, such as inference and diagnosis, as well as planning.

The main contribution of this article is, within the context of robotics, a comparison of planning time between ASP-based and PDDL-based task planners when they are used to model the same domain. Evaluation is performed across three different benchmark problems. While it may be possible to construct ASP and PDDL planners specifically suited to these specific benchmarks, domain-independent solvers are compared in this article. Although planner performance can be sensitive to domain encoding, we take care, to the extent possible, to encode the domains similarly in each language. The benchmark problems consist of the blocks world and hiking problems from the International Planning Competition (IPC) (Coles et al., 2012), as well as a variant of the robot navigation problem (Yang et al., 2014; Zhang et al., 2015). The robot navigation problem typically requires more complex reasoning about action preconditions and effects than problems in the IPC, which focus on generating long plans efficiently. Various properties of the domain or task in these benchmarks are also varied during evaluation to analyze the effect on planning time. The goal of this article is to help a robot planning practitioner understand the effects of specific domain properties to aid the choice of language selection for general-purpose planning.

We hypothesize that current state-of-the-art PDDL-based planners perform better on tasks with long solutions, and ASP-based planners tend to perform better on shorter tasks with a large number of objects. The hypothesis is confirmed in all three benchmark domains. We also hypothesize that ASP-based planners outperform PDDL-based planners in domains where complex reasoning (specified in Section 3.1) is required to reason about action preconditions and effects. This is observed in a controlled experiment of the robot navigation and blocks world domains. To the best of our knowledge, this is the first work on empirical comparisons between PDDL- and ASP-based planning systems, and can serve as a useful reference to robot planning practitioners.

2 Background

Research in task planning dates back to one of the earliest research areas in artificial intelligence. Since the development of STRIPS (Fikes and Nilsson, 1971) (as part of the Shakey robot project), many languages have been developed for representing task planning domains. Such languages typically need to describe actions' preconditions and effects, and are commonly known as action languages. A summary of some early action languages is available (Gelfond and Lifschitz, 1998).

To plan for real-world problems (such as robot systems), action languages first need to be capable of formally representing complex planning domains. Some of the recent research in task planning is focused on developing languages that improve the representation capability of action languages (Giunchiglia et al., 2004; Lee et al., 2013; Babb and Lee, 2015). Planning using these action

languages usually requires a translation to more general knowledge representation and reasoning (KRR) languages such as ASP. A planning paradigm for ASP was proposed (Lifschitz, 2002), and has been used in many real-world applications (Chen et al., 2010; Erdem et al., 2012; Yang et al., 2014). In this article, we follow the same planning paradigm when encoding domains in ASP.

In parallel, another line of research in task planning aims at more efficient planning algorithms and their implementations. PDDL (McDermott et al., 1998) was developed as a common formalism with the goal of allowing more direct comparison of planning algorithms and implementations. Since then, many efficient search algorithms have been developed for task planning problems, such as fast-forward (Hoffmann, 2001) and fast-downward (Helmert, 2006). These algorithms have publicly available implementations including SAYPHI (de la Rosa et al., 2007), LAMA (Richter et al., 2011), and FDSS (Helmert et al., 2011).

PDDL requires axioms, in the form of logical formulas, for reasoning within a situation (whereas action descriptions are used for reasoning across successive situations). A fundamental difference between PDDL and ASP is on their (non)monotonicity property. The axiom-based reasoning in PDDL is monotonic in the sense of logical reasoning, meaning that previously achieved conclusions remain when new information becomes available. In contrast, ASP is nonmonotonic, so it allows removal of previously achieved conclusions given new information. The nonmonotonic property of ASP makes it useful in tasks that require default reasoning and reasoning about inertial facts. Existing research has studied translating PDDL programs to ASP (Gebser et al., 2011), and applying axioms extracted from PDDL programs to ASP-based planning (Miura and Fukunaga, 2017). In particular, robotics researchers have developed robot navigation algorithms that switch between ASP- and PDDL-based planning systems on mobile service robots (Lo et al., 2018). However, none of this research conducted empirical comparisons over the performances of the state-of-the-art PDDL- and ASP-based planning systems.

Action languages can be further categorized as action description languages and action query languages (Lifschitz, 1997). Action description languages focus on specifying the transition system. Given a transition system, action query languages are used for reasoning about the properties of trajectories, such as to reason about history, nondeterminism, or both for diagnosis purposes. From the perspective of design purposes, PDDL is an action description language, and ASP is an action query language, though their implementations oftentimes support both description and query functionalities. At the same time, action language systems are usually implemented by compilation. For instance, Coala (Gebser et al., 2010) is one such compilation system that provides compilation techniques for several action languages.

There is existing research on predicting planning time using features of domains and problems (Fawcett et al., 2014), or more generally on predicting time required to solve a problem (Leyton-Brown et al., 2002). These methods can be used to help a planning practitioner estimate the difficulty of a planning problem, after the planning language and system have been selected. In contrast, this work aims at analyzing what domain properties affect the performances of existing planning systems, and can serve as a reference on the selection of action languages used for encoding planning problems.

3 Domain formalization

In this section, we introduce the three benchmark domains, namely robot navigation, blocks world, and hiking, and formally describe each in both ASP and PDDL.

It may be possible that different styles of encoding result in different planning times. To ensure that the conclusion is general and fair, we select the benchmark domains from a variety of origins, and follow the encoding in the literature. Blocks world and hiking have been used in the IPC, and PDDL-based planners have been specifically designed to solve problems in the IPC. The PDDL versions are used as is, and translated into equivalent ASP code. Robot navigation is a domain used to demonstrate planning using an ASP description (Yang et al., 2014), and contains properties such as recursive action effects that are missing from IPC domains. We translate the complex reasoning rules of the robot navigation domain from ASP as PDDL axioms.

The languages themselves require certain differences in domain encodings, and direct comparisons

are complicated by the fact that typically the people doing the encoding have greater familiarity with one language or the other. We acknowledge that different encodings may be more suitable to each planner, but it is infeasible to control the encoding style in practice. In this work, we ensure the fairest comparison possible by enforcing in all translations that the ASP and PDDL versions for a given domain have exactly the same set of predicates and actions, along with the same preconditions and effects. Specifically, an action should be allowed to execute on the same set of states, and it should make the same change to the state, regardless of the language. Consequently, as we use optimal ASP and PDDL planners in the experiments, they generate identical plans, and only planning times need to be compared. A detailed explanation of how ASP can be used for planning is available in Lifschitz (2002).

3.1 Robot navigation

The robot navigation domain differs from classical planning domains in the IPC in that complex reasoning needs to be performed to ensure that action preconditions are met, and that action effects are executed correctly. Specifically, this domain features action effects that require recursive reasoning to compute the final state of the world. In this domain, a mobile robot navigates an office floor which consists of a set of rooms that are connected to one another. Rooms can be connected to one another via doors, and closed doors need to be opened by the robot before it can pass. Alternatively, rooms can be directly connected to one another such that access is always possible from any location in one room to any location in the other.

The robot has the following perception and actuation modules available. Using a low-level controller, the robot can traverse to any room from its current location if its path is not blocked by a closed door, and this navigation can be encoded by a single high-level symbolic action “goto." Furthermore, the robot has some means of opening a closed door when the robot is next to the door, by either enlisting human aid or using a robot arm to open the door. On the perception side, the robot can sense its location, whether or not it is next to a door, and whether or not a door is open.

The domain knowledge can be formalized in ASP by statements defined using the following predicates:

hasdoor(R,D): This predicate specifies that room R has door D to move to an adjacent location. Statements expressed using hasdoor are specified during initialization, and do not change over time. The PDDL expression is (hasdoor ?r - room ?d - door).

connected(R1,R2): The connected predicate indicates that room R1 is directly connected to room R2 without a door. Similar to hasdoor, this predicate is used during initialization to describe directly connected locations. In PDDL, statements are described as (connected ?r1 - room ?r2 - room).

acc(R1,R2): acc specifies that room R1 is accessible from room R2 via a single navigation action executed by a low-level controller. Intuitively, any two rooms that are not separated by a closed door are accessible; i.e., the low-level controller can navigate from one room to another.

R1 is accessible from R2 if R1 is directly connected to R2, or R1 and R2 share the same door D which is open. Furthermore, acc is both commutative (i.e., R1 is accessible from room R2 if R2 is accessible from R1) and associative (i.e., if both R1 and R2 are accessible from R3, then they are accessible from one another). This associative property requires a recursive definition:

```
acc(R1,R2,n) :- connected(R1,R2).
acc(R1,R2,n) :- open(D,n),
            hasdoor(R1,D),
            hasdoor(R2,D).
acc(R1,R2,n) :- acc(R2,R1,n).
acc(R1,R2,n) :- acc(R1,R3,n),
            acc(R3,R2,n).
```

The recursive formulation of acc can be expressed in PDDL using the derived predicates:

```
(:derived (acc ?r1 - room ?r2 - room)
  (or (connected ?r1 ?r2)
     (exists (?d - door) (and (open ?d)
                        (hasdoor ?r1 ?d)
                        (hasdoor ?r2 ?d)))
     (acc ?r2 ?r1)
     (exists (?r3 - room) (and (acc ?r1 ?r3)
                         (acc ?r3 ?r2)))))
```

at(R,n): at is used to specify that the robot is at room R at timestep n (of the high-level plan). This predicate is inertial (i.e., the robot remains in room R if there is no evidence showing that it is not in room R anymore), and this property is specified as

```
at(R,n) :- at(R,n-1), not -at(R,n).
```

In PDDL, the predicate is expressed as (at ?r - room), and all predicates are inertial by default.

open(D,n): Door D is open at step n. door is inertial; i.e., the robot believes that a door will stay in the same state unless sensed differently. In ASP, the inertial property for this predicate is represented as

```
open(D,n) :- open(D,n-1), not -open(D,n).
```

In PDDL, open is expressed as (open ?d - door).

canopen(D,n): The robot can open door D if the robot is right next to the door. canopen is the action effect of approaching a door, and a precondition before the door can be opened. canopen is not inertial. In PDDL, the predicate is expressed as (canopen ?d - door).

visited(R,n): Once the robot visits a room, the visited fluent for that room remains true until the end of the planning process. visited is used to describe goal conditions. The persistence property is expressed in ASP as

```
visited(R,n) :- visited(R,n-1).
```

(visited ?r - room) expresses visited in PDDL.

There are three actions in the domain, goto, approach, and opendoor, with the following definitions:

goto(R2,n): This action specifies that the robot should navigate to room R2 at timestep n in the high-level plan. The precondition for this action is that the robot must be in a room R1 from which R2 is accessible. Once the robot goes to room R2, the goal condition visited is set to true for that room as well. The ASP rules defining the action preconditions and effects are as follows:

```
:- goto(R2,n), at(R1,n-1), not acc(R1,R2,n-1).
at(R2,n)     :- goto(R2,n).
-at(R1,n)    :- goto(R2,n), at(R1,n-1), R1 != R2.
visited(R2,n) :- goto(R2,n).
```

The same description in PDDL is expressed as follows:

```
(:action goto
 :parameters (?r2 - room)
 :precondition (exists (?r1 - room)
                 (and (at ?r1)
                 (acc ?r1 ?r2)))
 :effect (and (at ?r2)
          (forall (?r1 - room)
                (when (at ?r1)
                    (not (at ?r1))))
          (visited ?r2)
          (forall (?d1 - door)
                (not (canopen ?d1)))))
```

The action effects in PDDL have an additional statement to ASP to indicate that canopen is not inertial.

approach(D,n): This action specifies that the robot should approach door D. The action is executable only when the robot is in a room that has door D. After executing this action, the robot can open door D. In ASP, this action is expressed as

```
:- approach(D,n), at(R1,n-1), not hasdoor(R1,D).
canopen(D,n) :- approach(D,n).
```

In PDDL, this action is expressed as

```
(:action approach
 :parameters (?d - door)
 :precondition (exists (?r1 - room)
                 (and (at ?r1)
                     (hasdoor ?r1 ?d)))
 :effect (and (canopen ?d)
          (forall (?d1 - door)
                (when (not (= ?d1 ?d))
                    (not (canopen ?d1))))))
```

The action effects in PDDL express that door d can be opened, and that no other doors in the domain can be opened without approaching them first.

opendoor(D,n): This action allows the robot to open door D if canopen(D) is true. Opening an open door does not change the state of the world. This action is represented in ASP as

```
:- opendoor(D,n), not canopen(D,n-1).
open(D,n) :- opendoor(D,n).
```

In PDDL, this action is represented as

```
(:action opendoor
 :parameters (?d - door)
 :precondition (canopen ?d)
 :effect (and (open ?d)
          (forall (?d1 - door)
                (not (canopen ?d1)))))
```

In all three actions, a specific action effect in PDDL describes the non-inertial property of canopen. This effect is not required in ASP because the canopen is not inertial.

The goal in the robot navigation domain is to visit a randomly selected set of rooms. To visit a room, the robot needs to recursively reason about which rooms are accessible from one another. If a door in its path is closed, the robot needs to explicitly approach the door and execute an opendoor action. Whenever opendoor is executed, the direct accessibility of rooms changes, and this action effect needs to be computed recursively. This recursive property of the domain differentiates it from traditional IPC planning domains such as blocks world and hiking.

3.2 Blocks world

In the blocks world domain, the goal is to move a set of stackable blocks from one configuration to another using a robot hand. We use the official domain definition in IPC-2011 as the PDDL domain description, and translate the domain to ASP. The full description of the IPC domain is available online (All links have anonymized for the review process) in PDDL (http://pastebin.com/raw/b07aMTJB) and ASP (http://pastebin.com/raw/SAqM3xbF). We describe only the translation of the pick-up action to ASP as an illustrative example; the other actions (put-down, stack, and unstack) follow similarly. pick-up allows a robot to pickup a block that has no blocks underneath it (designated by ontable). Furthermore, it should have no blocks stacked on top of it (designated by clear). Finally, a block can be picked up only if the robot hand is empty (designated by handempty). The effect of the action is that the robot hand is holding the block, and all preconditions for the action become false. This action description translated to ASP looks as follows:

```
:- pickup(B,n), not clear(B,n-1).
:- pickup(B,n), not ontable(B,n-1).
:- pickup(B,n), not handempty(n-1).
-ontable(B,n) :- pickup(B,n).
-clear(B,n)  :- pickup(B,n).
-handempty(n) :- pickup(B,n).
holding(B,n)  :- pickup(B,n).
```

The first three statements specify the same action preconditions specified in the PDDL description, and the last four statements specify the same action effects as specified in the PDDL description.

The extended version of the blocks world domain introduces a recursively defined predicate: above (Thiébaux et al., 2005). The PDDL definition is as follows:

```
(:derived (above ?x ?y)
   (or (on ?x ?y)
      (exists (?z) (and (on ?x ?z) (above ?z ?y)))))
```

The predicate is defined in ASP as

```
above(X,Y,n) :- on(X,Y,n).
above(X,Y,n) :- on(X,Z,n), above(Z,Y,n).
```

The complete domain descriptions in PDDL (http://pastebin.com/raw/FwZgGmZf) and ASP (http://pastebin.com/raw/9D2PuNze) are online. We use both the original and the extended versions of the blocks world domain in experiments.

3.3 Hiking

We select the hiking domain, new in IPC-2014, as our third benchmark domain. The hiking domain features negative preconditions. We use the official PDDL formalization in IPC, and an equivalent definition in ASP for this study.

In short, the purpose of this domain is to arrange activities for a number of couples, so each couple can hike to their destination with a tent ready. A hiking problem specifies connections between places, and initial locations of couples, cars, and tents. A typical plan transports and sets up tents at the destination, drives each couple to the starting point of their hike, and then has them walk together along the hike. The complete description of the hiking domain is available online for both PDDL (http://pastebin.com/raw/v3wkv57W) and ASP (http://pastebin.com/raw/Dw1BwG0Z).

4 Experiments

The experiments in this section are designed to compare planning times when the domains and problems formalized in the previous section are optimally solved using state-of-the-art solvers. Our encoding strategy and the award-winning optimal planning systems ensure that all planners generate the same plan given the same pair of domain and problem.

We select FastDownward (Helmert, 2006) with the setting FDSS-1 (Helmert et al., 2011), which had the highest score in the sequential optimization track at IPC 2011. Note that the later versions of the track do not take into account or announce the time needed to solve each problem. Instead, a hard time constraint (e.g., 30 min) is given to all planners. As a result, most PDDL planners always use the maximum time allowed to avoid reporting suboptimal or incorrect solutions (which is greatly penalized), and are therefore unfit for a comparison of planning times. The robot navigation domain and the extended blocks world domain require derived predicates. Since none of the planners in the optimization track support derived predicates, we use FastDownward with the setting LAMA-2011 (Richter et al., 2011), the winner of Sequential Satisficing Track in 2011. We use version 4.5.4 of Clingo (Gebser et al., 2014) in incremental mode as the ASP solver. Clingo is an answer set solving system that integrates Clasp,

the winner of the fifth Answer Set Programming Competition in 2015 (Calimeri et al., 2016).

Planner performances are evaluated on a general-purpose high throughput computing (HTC) cluster that is operated by the Department of Computer Science at the University of Texas at Austin. We filter out machines with less than 8 GB memory in the experiments, resulting in more than 10 machines with different hardware configurations (e.g., memory ranging from 8 GB to more than 500 GB). All data points are averaged across at least 10 trials to reduce noise, and standard deviations are reported. Given the wide range of hardware configurations and statistical analysis, we aim to conclude with observations that are generally valid and useful to practitioners.

During evaluations, various domain characteristics are also changed to measure the difference in performance of different planning paradigms. Note that we are generally more interested in comparing sensitivities of planning systems (instead of comparing individual data points) given different domain characteristics. The sensitivity that can be reflected by the "trend" of a series of data points is typically more robust to implementation details of planning systems such as programming languages and compilers. The goal of these evaluations is to test the hypothesis that PDDL-based approaches work better in situations where the generated plans have many steps, and ASP-based approaches work better in situations where the domain is large or substantial reasoning is required at every step of the plan to compute the world state.

Robot navigation: Results from the robot navigation domain are shown in Fig. 1. Two different versions of the domain have been created. The small domain has 10 rooms, and each of them is connected via a central corridor. Four rooms are connected via doors, and the rest are directly connected. In contrast, the large domain has 15 rooms where six rooms are connected via doors to the corridor.

The robot is initially located in the corridor, and as its goal, it needs to visit a number of rooms that are randomly selected in each trial. Since rooms that are connected by doors take more steps to visit, we increase the number of trials (to 50), on which we compute the average planning time for each data point. Regardless, the number of rooms in the goal is positively correlated with the plan length. The planning time for each planner is represented on the y axis (log scale, same for all following figures).

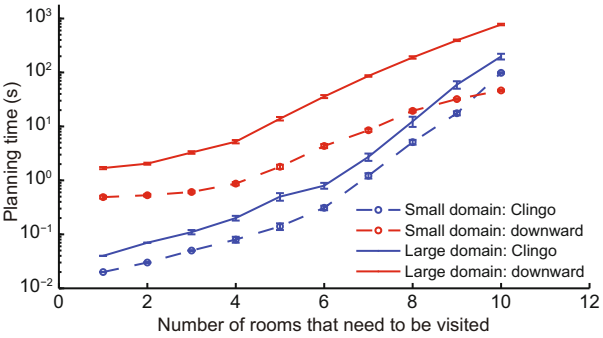

Fig. 1 Robot navigation: small domain, 10 rooms; large domain, 15 rooms. Forty percent of rooms are connected via doors, and the rest are directly accessible from the corridor

We can observe that the red curves have smaller slopes, since they intersect (or will do so) with the blue curves. This confirms our hypothesis that PDDL planners are better at solving planning problems which require a large number of steps. We can also observe that the gap between red curves is larger than the gap between blue curves, even on the logarithmic scale. This observation again supports the hypothesis that ASP-based planning is less sensitive to object scaling.

Furthermore, ASP-based planning is much faster than PDDL-based planning when the number of rooms is smaller than eight, and finishes within a reasonable amount of time. This is especially useful in domains such as robotics where fast real-time operation is necessary. This better performance is probably a consequence of recursive action effects embedded in the domain. We further verify this observation in a controlled experiment using the blocks world domain.

Blocks world: Fig. 2 reports results from the regular blocks world domain. Similarly, two versions of the domain are evaluated: a small domain with 15 blocks in the environment, and a large domain with 60 blocks in the environment. Initially, all blocks are unstacked and on the table (so as to control the optimal plan length). The goal of the planners is to generate a plan that builds a single stack of a specified height with randomly specified blocks. The plan length is proportional to the height of the stack, and is represented as the x axis in Fig. 2. All results have been averaged across 10 trials.

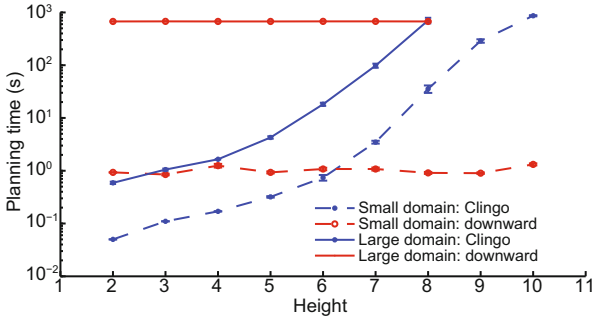

Fig. 2 Blocks world: small domain, 15 blocks; large domain, 60 blocks. We use a timeout of 1800 s (same as IPC). The graph plots only configurations where all trials of both planners finished before timeout (same for all following experiments). References to color refer to the online version of this figure

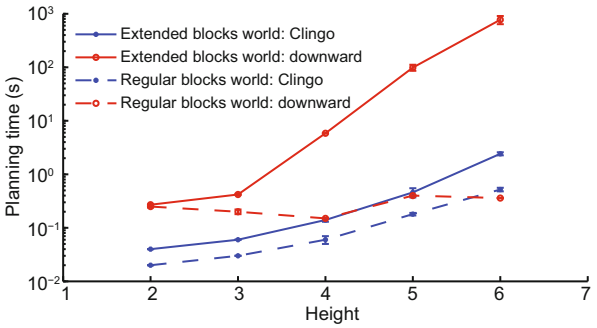

Fig. 3 Blocks world: regular domain vs. extended domain. References to color refer to the online version of this figure

The two red lines are far apart and almost horizontal; i.e., the PDDL planner slows down significantly in the larger domain, but planning time does not depend on the plan length. The planning time of Clingo grows as the plan length increases, but the difference between different sized domains is smaller than that in PDDL. These observations support our hypotheses that PDDL-based planning is fast at producing long plans, while ASP-based planning is faster in large domains where smaller plans are necessary.

Fig. 3 compares the performance of each planner between the regular blocks world domain and the extended version. In this experiment, the domain has 10 blocks. The task is the same as before: building a tower of various heights from unstacked blocks. The difference of the extended domain is that the goal condition is expressed with the predicate above, where the planners reason about recursive effects. Similarly, the results are averaged across 10 trials.

Hiking: In Fig. 3, there is a small difference in ASP solving time, but a significant change in the performance of the PDDL planner. In contrast to Fig. 2, we can observe that complex reasoning affects PDDL planning time in its slope with respect to plan length, whereas object scaling shifts the curve up. The observation confirms that ASP-based planning is better for domains that involve complex reasoning (reasoning about recursive action effects in this case).

Figs. 4 and 5 show results from the hiking domain. In each graph, the x axis is the total number of trips made by all couples, and the y axis is the planning time in seconds. Fig. 5 shows the performance of each planner for two sizes of the domain. Although we can make similar observations that the PDDL planner becomes faster than the ASP planner at higher plan lengths but slower with more objects, the evidence is weaker than the results above. Since the hiking domain has four types of objects that affect planning in different ways, the domain size has four dimensions. Fig. 4 shows a more controlled set of experiments in which the larger domain increases only one type of objects. Objects are added in a way that does not affect the plan length or the number of optimal plans at each point of the x axis. In all graphs the slope of blue curves is larger than the slope of red curves. So, the hypothesis about the plan length holds for all object types.

When adding cars and tents to the domain, from Figs. 4a and 4b, we can observe that the ASP solver is less sensitive than the PDDL planner, but there is no significant difference in the case of couples and places (Figs. 4c and 4d). We find that couples and places are more heavily used as action parameters than two other types of objects. Actions such as drive_passenger and walk_together even take two parameters of each. Therefore, increasing the number of these objects complicates the grounding (Plan generation in ASP is a two-step process that includes grounding and solving, where the grounding step outputs a variable-free representation) of ASP problems. We also observe that most of the increased planning time of Clingo is in grounding. For instance, when the number of couples increases from 5 to 10 (Fig. 4c), average solving time for one trip stays below 0.8 s, while average grounding time grows from 16.0 s to 354.2 s. Based on these two observations, we hypothesize that if the domain has actions that

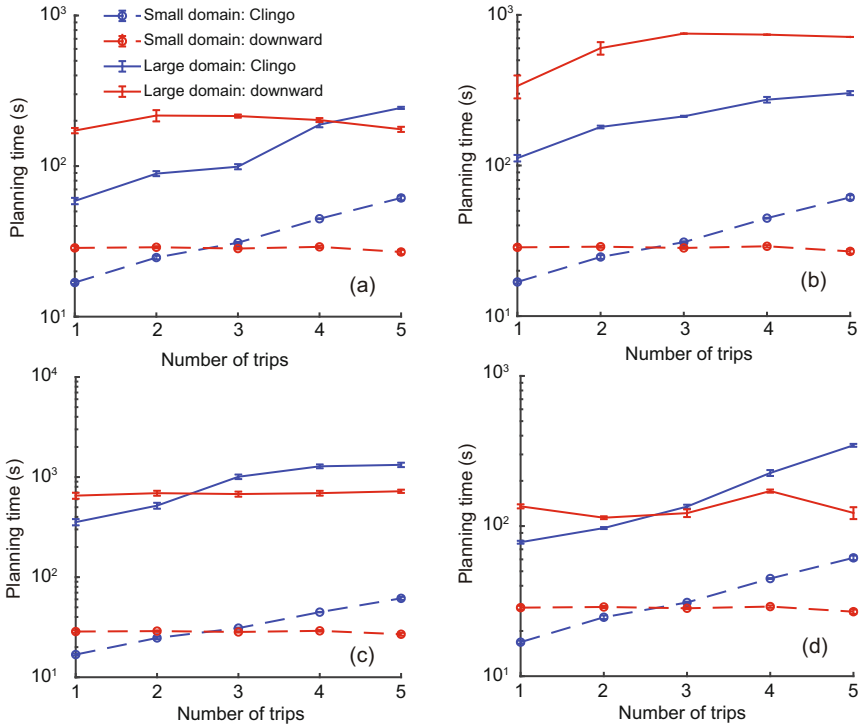

Fig. 4 Varying individual domain attributes in the hiking domain: (a) small–3 cars, large–10 cars; (b) small–3 tents, large–10 tents; (c) small–5 couples, large–10 couples; (d) small–4 places, large–8 places. References to color refer to the online version of this figure

check or change the state of many objects, Clingo's advantage at planning in large domains can be canceled out by the extra grounding time. A further analysis of how parameter type and the number of trips affect grounding time is left for future research.

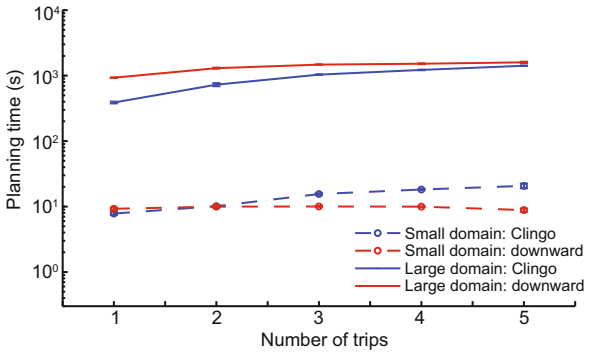

Fig. 5 Hiking: small domain vs. large domain. References to color refer to the online version of this figure

Remark: All of the above results support our hypothesis that PDDL-based approaches work better in situations where the generated plans have many steps, and ASP-based approaches work better in situations where the domain is large or substantial reasoning is required at every step of the plan to compute the world state.

Given the wide range of hardware configurations of machines in the HTC cluster and the low standard deviation values reported in the results, we can see that machines of different configurations did not cause significant differences in planning time, which indicates that the trends are consistent across different types of computing platforms.

5 Conclusions and discussions

In this article, we empirically compared ASP- and PDDL-based task planners using three robotic benchmark domains. PDDL is the dominant action language in the task planning community; ASP is widely used for knowledge representation and reasoning, and can be used for task planning. The analysis in this article demonstrates that PDDL-based planners perform better when tasks require long solutions. However, ASP-based task planners are less

susceptible to an increase in the number of domain objects, as long as the number of objects does not explode the number of grounded actions. Finally, in domains requiring complex reasoning such as the robot navigation domain, ASP-based planners can be considerably faster than PDDL-based planners for shorter plans. Such observations can serve as a useful reference to task planning practitioners in the process of action language selection.

This article, by no means, aims to provide a list of the best planners given a pair of planning domain and planning problem, which is infeasible in practice. From a high-level perspective, the observations shared in this article are intended to contribute to the community's understanding of how properties of planning domains and problems affect the performance of different planning systems. Based on one's knowledge and intuition about properties of the planning problems at hand, a practitioner can then make a more informed choice of the planning system.

In this work, we selected the Clingo system for evaluating the ASP-based planning formalism, and the FastDownward planning systems of FDSS-1 and LAMA-2011 for the PDDL-based formalism. These award-winning optimal systems represent the state-of-the-art algorithms and implementations of the two planning paradigms. As the first work on empirical comparisons of PDDL- and ASP-based task planners, we focus on a clear presentation of the methodology and results. We carefully selected the three domains that are as distinct as possible for a representative comparison. We believe that the conclusions hold in most cases. In the future, we will conduct further evaluations over the two planning formalisms and their implementations on other task planning problems using other more extreme computing platforms (such as low-end onboard computers on mobile robots). The experiments in this article were conducted using an ASP-based planner and a PDDL-based planner that are meant to be representative of their respective classes. While we believe that our results and observations will generalize to other such planners, we acknowledge that there is no way to establish the generalization conclusively without empirical comparison with many other planners, which is beyond the scope of this article.

This article does not include formal analysis or comparisons between PDDL- and ASP-based paradigms. One of the main reasons is that the original definition of the PDDL language includes only its syntax, and its semantics is not discussed. As a result, different PDDL solvers have different "interpretations" of PDDL programs, although they all aim at producing optimal solutions. Another direction for future work is to look into the semantics of PDDL (McDermott, 2003; Thiébaux et al., 2005) and conduct formal analysis between the two planning paradigms.

Acknowledgements

A portion of this work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-1637736, IIS-1651089, IIS-1724157), ONR (N00014-18-2243), FLI (RFP2-000), Intel, Raytheon, and Lockheed Martin. Peter STONE serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

Babb J, Lee J, 2015. Action language BC+: preliminary report. Proc 29th AAAI Conf on Artificial Intelligence, p.1424-1430.

Calimeri F, Gebser M, Maratea M, et al., 2016. Design and results of the fifth answer set programming competition. *Artif Intell*, 231:151-181. https://doi.org/10.1016/j.artint.2015.09.008

Cambon S, Alami R, Gravot F, 2009. A hybrid approach to intricate motion, manipulation and task planning. *Int J Robot Res*, 28(1):104-126. https://doi.org/10.1177/0278364908097884

Chen XP, Ji JM, Jiang JQ, et al., 2010. Developing high-level cognitive functions for service robots. Proc 9th Int Conf on Autonomous Agents and Multiagent Systems, p.989-996.

Coles A, Coles A, Olaya AG, et al., 2012. A survey of the seventh international planning competition. *AI Mag*, 33(1):83-88. https://doi.org/10.1609/aimag.v33i1.2392

de la Rosa T, Olaya AG, Borrajo D, 2007. Using cases utility for heuristic planning improvement. Int Conf on Case-Based Reasoning, p.137-148. https://doi.org/10.1007/978-3-540-74141-1_10

Erdem E, Patoglu V, 2018. Applications of ASP in robotics. *KI-Künstl Intell*, 32(2-3):143-149. https://doi.org/10.1007/s13218-018-0544-x

Erdem E, Aker E, Patoglu V, 2012. Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. *Intell Ser Robot*, 5(4):275-291. https://doi.org/10.1007/s11370-012-0119-x

Erdem E, Gelfond M, Leone N, 2016. Applications of answer set programming. *AI Mag*, 37(3):53-58. https://doi.org/10.1609/aimag.v37i3.2678

Fawcett C, Vallati M, Hutter F, et al., 2014. Improved features for runtime prediction of domain-independent

planners. Proc 24th Int Conf on Automated Planning and Scheduling, p.355-359.

Fikes RE, Nilsson NJ, 1971. Strips: a new approach to the application of theorem proving to problem solving. *Artif Intell*, 2(3-4):189-208. https://doi.org/10.1016/0004-3702(71)90010-5

Gebser M, Grote T, Schaub T, 2010. Coala: a compiler from action languages to ASP. European Workshop on Logics in Artificial Intelligence, p.360-364. https://doi.org/10.1007/978-3-642-15675-5_32

Gebser M, Kaminski R, Knecht M, et al., 2011. plasp: a prototype for PDDL-based planning in ASP. In: Delgrande JP, Faber W (Eds.), Logic Programming and Nonmonotonic Reasoning. Springer, Berlin, p.358-363. https://doi.org/10.1007/978-3-642-20895-9_41

Gebser M, Kaminski R, Kaufmann B, et al., 2014. Clingo=ASP+control: preliminary report. http://arxiv.org/abs/1405.3694

Gelfond M, Kahl Y, 2014. Knowledge Representation, Reasoning, and the Design of Intelligent Agents the Answer-Set Programming Approach. Cambridge University Press, Cambridge.

Gelfond M, Lifschitz V, 1998. Action languages. *Electron Trans Artif Intell*, 3(6):195-210.

Giunchiglia E, Lee J, Lifschitz V, et al., 2004. Nonmonotonic causal theories. *Artif Intell*, 153(1-2):49-104. https://doi.org/10.1016/j.artint.2002.12.001

Helmert M, 2006. The fast downward planning system. *J Artif Intell Res*, 26:191-246. https://doi.org/10.1613/jair.1705

Helmert M, Röger G, Karpas E, 2011. Fast downward stone soup: a baseline for building planner portfolios. Int Conf on Automated Planning and Scheduling Workshop on Planning and Learning, p.28-35.

Hoffmann J, 2001. FF: the fast-forward planning system. *AI Mag*, 22(3):57-62.

Khandelwal P, Zhang SQ, Sinapov J, et al., 2017. BWIBots: a platform for bridging the gap between AI and human–robot interaction research. *Int J Robot Res*, 36(5-7):635-659. https://doi.org/10.1177/0278364916688949

Lee J, Lifschitz V, Yang F, 2013. Action language BC: preliminary report. Proc 23rd Int Joint Conf on Artificial Intelligence, p.983-989.

Leyton-Brown K, Nudelman E, Shoham Y, 2002. Learning the empirical hardness of optimization problems: the case of combinatorial auctions. Int Conf on Principles and Practice of Constraint Programming, p.556-572. https://doi.org/10.1007/3-540-46135-3_37

Lifschitz V, 1997. Two components of an action language. *Ann Math Artif Intell*, 21(2-4):305-320. https://doi.org/10.1023/A:1018973620715

Lifschitz V, 2002. Answer set programming and plan generation. *Artif Intell*, 138(1-2):39-54. https://doi.org/10.1016/S0004-3702(02)00186-8

Lifschitz V, 2008. What is answer set programming? Proc 23rd National Conf on Artificial Intelligence, p.1594-1597.

Lo SY, Zhang S, Stone P, 2018. PETLON: planning efficiently for task-level-optimal navigation. Proc 17th Conf on Autonomous Agents and Multiagent Systems, p.220-228.

McDermott D, 2003. The formal semantics of processes in PDDL. Proc ICAPS Workshop on PDDL, p.101-155.

McDermott D, Ghallab M, Howe A, et al., 1998. PDDL—the planning domain definition language. http://www.cs.yale.edu/homes/dvm/

Miura S, Fukunaga A, 2017. Automatic extraction of axioms for planning. Proc 27th Int Conf on Automated Planning and Scheduling, p.218-227.

Richter S, Westphal M, Helmert M, 2011. Lama 2008 and 2011. Int Planning Competition, p.117-124.

Thiébaux S, Hoffmann J, Nebel B, 2005. In defense of PDDL axioms. *Artif Intell*, 168(1-2):38-69. https://doi.org/10.1016/j.artint.2005.05.004

Yang F, Khandelwal P, Leonetti M, et al., 2014. Planning in answer set programming while learning action costs for mobile robots. AAAI Spring Symp on Knowledge Representation and Reasoning in Robotics, p.71-78.

Zhang S, Yang F, Khandelwal P, et al., 2015. Mobile robot planning using action language BC with an abstraction hierarchy. Proc 13th Int Conf on Logic Programming and Nonmonotonic Reasoning, p.502-516. https://doi.org/10.1007/978-3-319-23264-5_42