Integrating Reinforcement Learning and Declarative Programming to Learn Causal Laws in Dynamic Domains

Mohan Sridharan[1] and Sarah Rainge[2]

[1] Department of Electrical and Computer Engineering, The University of Auckland, NZ
[2] Department of Computer Science, Texas Tech University, USA
m.sridharan@auckland.ac.nz, sarah.rainge@ttu.edu

Abstract. Robots deployed to assist and collaborate with humans in complex domains need the ability to represent and reason with incomplete domain knowledge, and to learn from minimal feedback obtained from non-expert human participants. This paper presents an architecture that combines the complementary strengths of Reinforcement Learning (RL) and declarative programming to support such commonsense reasoning and incremental learning of the rules governing the domain dynamics. Answer Set Prolog (ASP), a declarative language, is used to represent domain knowledge. The robot's current beliefs, obtained by inference in the ASP program, are used to formulate the task of learning previously unknown domain rules as an RL problem. The learned rules are, in turn, encoded in the ASP program and used to plan action sequences for subsequent tasks. The architecture is illustrated and evaluated in the context of a simulated robot that plans action sequences to arrange tabletop objects in desired configurations.

1 Introduction

Robots deployed in assistive roles in complex domains such as healthcare and disaster rescue face some fundamental learning and representation challenges. For instance, it is difficult to equip a robot assisting caregivers in an elder care home with accurate (and complete) domain knowledge; some of the rules governing the domain dynamics (e.g., "pain medication cannot be stacked in the top shelf") may be unknown to the robot or may change over time. Furthermore, the robot has to reason with qualitative and quantitative descriptions of knowledge, and the human participants may not have the time and expertise to provide elaborate and accurate feedback.

As a step towards addressing these challenges, this paper presents an architecture that combines the complementary strengths of Reinforcement Learning (RL) and declarative programming to support commonsense reasoning and incremental discovery of (previously unknown) rules that govern the domain dynamics. Specifically, we use Answer Set Prolog (ASP), a declarative language, to represent domain knowledge in the form of objects, relations between objects, and any known rules governing the domain dynamics. Inference with this knowledge is used to obtain the components of an RL formulation of the task of incrementally discovering unknown domain rules. We illustrate this architecture in the context of a simulated robot that plans action sequences to arrange tabletop objects in desired configurations.

The remainder of the paper is organized as follows. Section 2 motivates the proposed architecture by briefly reviewing related work. Section 3 describes the proposed architecture and its individual components. The experimental setup and results in a simulated domain are described in Section 4, along with directions for future research, and Section 5 presents the conclusions.

2 Related Work

We motivate our architecture by reviewing a representative set of related work on: (a) learning from non-expert human feedback and environmental interactions; and (b) knowledge representation for such learning.

Reinforcement learning provides an elegant mathematical formulation for agents to learn from repeated interactions with the environment, selecting actions that maximize a numerical reward signal [15]. For an agent engaged in a sequential decision making task, and occasionally receiving reinforcement signals from a human trainer, it is challenging to learn the best possible action policy. Several RL-based algorithms have been developed to address this interactive shaping problem [10], e.g., the use of RL and animal training insights for clicker training to support interactive learning of synthetic characters [4], and for action and behavior learning on a four-legged robot [9]. Other researchers have used RL-based frameworks for interactive shaping [10,16]. For instance, the TAMER framework allows an agent to receive feedback about specific tasks from a human trainer fully aware of the agent's state and action capabilities [11]. These algorithms, however, do not consider human training in conjunction with feedback that agents can receive by interacting with the environment.

The feedback signals obtained from non-expert humans and the environment may differ in format and frequency; human feedback may also be a function of a set of (previous or future) states and actions. RL-based algorithms have been proposed to address this challenge. For instance, different linear functions have been considered for combining human reward and environmental reward signals in some benchmark simulated domains [12]. A bootstrap learning algorithm has also been developed to enable agents to incrementally and continuously estimate the relative importance of human feedback and environmental feedback in the Tetris domain and the multiagent Keepaway Soccer domain [1,14]. More recently, a policy shaping algorithm has been developed for including human feedback in interactive RL formulations of agent domains [8]. However, these algorithms do require complete knowledge of the domain and the rules that govern domain dynamics.

Declarative languages provide appealing knowledge representation and commonsense reasoning capabilities that have been used for simulated and physical robots deployed in assistive roles in complex domains [6,18]. Algorithms have been developed to generalize from a limited number of samples by using knowledge representation in RL frameworks, e.g., relational RL incorporates a relational learner in a traditional RL algorithm [5]. However, the agent still needs to be provided accurate and elaborate knowledge about domain objects and rules. The architecture described in this paper is a step towards enabling robots to incrementally discover the rules governing the domain dynamics by integrating the commonsense reasoning capabilities of declarative programming with the incremental learning capabilities of RL.

3 Problem Formulation

This section describes our architecture for incrementally learning the rules governing the domain dynamics. As an illustrative example used throughout this paper, we consider scenes with a simulated robot in a tabletop domain with blocks characterized by three properties (color, shape, and size). The robot's objective is to plan an action sequence to achieve the desired arrangement (i.e., configuration) of blocks in collaboration with human participants (if available). The perception and actuation challenges are abstracted away to focus on the representation and learning challenges.

3.1 Architecture Overview

Figure 1 shows the proposed architecture. The robot is initially provided some domain knowledge in the form of objects (and their properties), relations between the objects, and some rules governing the domain dynamics. This domain knowledge is encoded in the ASP knowledge base (KB). Incremental learning of the (previously unknown) rules is formulated as an RL problem, and the current beliefs encoded in KB are used to define the components of the RL formulation that supports the use of high-level feedback (e.g., positive or negative reinforcement) that can be provided even by non-expert humans. This formulation and the action policy computed by RL are used to discover previously unknown rules governing the domain dynamics. These rules are encoded in the ASP KB and used for planning action sequences for subsequent tasks. Although the architecture's components are described below for the tabletop domain, it is applicable to other human-robot collaboration domains.

3.2 ASP Knowledge Base

ASP is a declarative language that can represent recursive definitions, defaults, causal relations, special forms of self-reference, and other language constructs that occur frequently in non-mathematical domains, and are difficult to express in classical logic formalisms [3]. ASP is based on the stable model semantics of logic programs; it builds on the research in non-monotonic logics and disjunctive databases [7].

The syntax, semantics and representation of the transition diagram of our illustrative domain are described in an *action language* AL [7]. Action languages are formal models of parts of natural language used for describing transition diagrams. AL has a sorted signature containing three *sorts*: *statics*, *fluents* and *actions*. Statics are domain properties whose truth values cannot be changed by actions, while fluents are properties whose truth values are changed by actions. Actions are defined as a set of elementary actions that can be executed in parallel. A domain property p or its negation $\neg p$ is a domain literal. AL allows three types of statements:

$$\begin{array}{ll} a \ \textbf{causes} \ l_{in} \ \textbf{if} \ p_0, \ldots, p_m & \text{(Causal law)} \\ l \ \textbf{if} \ p_0, \ldots, p_m & \text{(State constraint)} \\ \textbf{impossible} \ a_0, \ldots, a_k \ \textbf{if} \ p_0, \ldots, p_m & \text{(Executability condition)} \end{array}$$

where a is an action, l is a literal, l_{in} is an inertial fluent literal, and $p_0, \ldots, p_m$ are domain literals. The causal law states that action a causes inertial fluent literal l_{in} if the literals $p_0, \ldots, p_m$ hold true. A collection of statements of AL forms a system description.

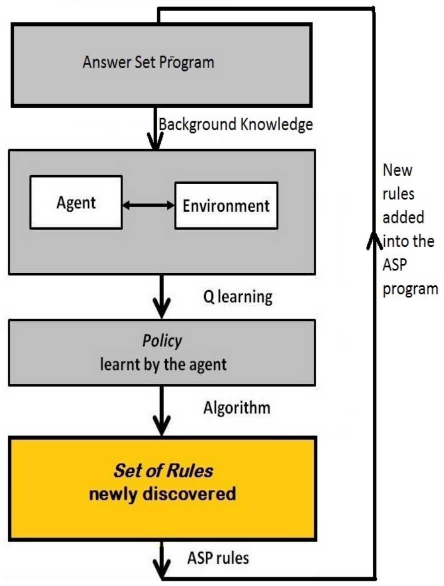

Fig. 1. Proposed Architecture: the closed loop of knowledge representation, reasoning and reinforcement learning enables discovery of new rules and their use in subsequent tasks

The domain representation consists of a system description $\mathscr{D}$ and history $\mathscr{H}$. $\mathscr{D}$ consists of a sorted signature and axioms used to describe the transition diagram τ; $\mathscr{H}$ stores the history of actions executed and observations received. The sorted signature is a tuple that defines the names of objects, functions, and predicates available for use in the domain. The sorts of the tabletop domain include: *block*, *location*, *color*, *shape*, and *size*. The domain's fluent: $on(block, location)$, defined in terms of the arguments, states that a specific block is at a specific location; this is an inertial fluent that obeys the laws of inertia. There are some defined fluents for block properties: $blockColor(block, color)$, $blockShape(block, color)$ and $blockSize(block, color)$. The action $put(block, location)$ puts a block at a specific location (*table* or another block). The dynamics are defined in terms of causal laws such as:

$$put(b_1, loc_1) \;\; \textbf{causes} \;\; on(b_1, loc_1)$$

state constraints such as:

$$\neg on(b_1, loc_1) \;\; \textbf{if} \;\; on(b_1, loc_2),\; loc_1 \neq loc_2$$

and executability constraints such as:

$$\textbf{impossible} \;\; put(b_1, loc_1) \;\; \textbf{if} \;\; on(b_2, loc_1),\; b_2 \neq b_1$$

The domain representation ($\mathscr{D}$, $\mathscr{H}$) is translated into an ASP program Π, i.e., a collection of statements describing domain objects and relations between them. Π consists of the causal laws of $\mathscr{D}$, inertia axioms, closed world assumption for defined fluents, reality checks, and records of observations amd actions from $\mathscr{H}$. The ground literals in an *answer set* obtained by solving Π represent the beliefs of an agent associated with program Π. Program consequences are statements that are true in all such belief sets. Tasks such as planning and diagnostics in the example domain can be reduced to computing answer sets of the corresponding ASP program and extracting the sequence of actions to be executed at specific time steps. For more details about the translation (from AL to an ASP program) and planning using ASP, please see [7].

It is difficult to encode all the domain knowledge in the ASP KB. However, if the KB is incomplete, the corresponding plans may not succeed. Consider the scene in which three blocks of the same size: *red square* (b_1), *red triangle* (b_2), and *blue rectangle* (b_3), are on the table, and the objective is to stack the blocks. One valid plan is:

$$put(b_3,b_2), \quad put(b_1,b_3)$$

The robot should (theoretically) be able to use this plan to stack the blocks. However, execution of this plan results in failure because unknown to the robot, a block cannot be placed on top of a triangle in this domain. Our architecture includes an RL component to incrementally discover such rules.

3.3 RL Formulation and Rule Learning

A reinforcement learning problem is represented by the tuple $\langle S, A, T, R\rangle$, whose entries correspond to: a set of states, a set of actions, an unknown state transition function ($T : S \times A \times S' \rightarrow [0,1]$) and an unknown real-valued reward function ($R : S \times A \times S' \rightarrow \Re$). The objective is to find a policy $\pi^* : S \rightarrow A$ that maximizes the cumulative expected reward over a planning horizon. For the tabletop domain:

- States are the different configurations of blocks on the table. Constraints in the ASP KB eliminate impossible states. The desired configuration is the goal state.
- Actions move a block between locations, resulting in a state transition. Constraints in the ASP KB eliminate impossible actions and state transitions.
- The state transition and reward function are unknown to the robot; they are designed in the simulator to mimic the robot's interaction with the real world.
- The reward function provides a large utility (i.e., positive value) for achieving the desired configuration of blocks; a large negative utility is provided to actions that do not produce the desired effect.

For interactions in the real world, rewards will be assigned based on the robot's observations of action outcomes and the feedback provided by humans (when available).

Algorithm 1 summarizes the steps to discover new rules; it is based on the observation that actions that have much lower utility than other actions did not provide desired outcomes, and thus should not (or cannot) be performed. The algorithm takes as input the domain knowledge encoded in the ASP KB. To generate additional samples, new scenes may be created by randomly changing the property values of blocks in the scene. This optional step may be omitted to limit exploration and/or for scenes with a small number of blocks. For each scene, the robot generates components of the RL

Algorithm 1. Algorithm to discover domain rules.

```
Input: Domain knowledge in ASP KB; N=1.
Output: ASP KB with newly discovered rules.
1  Generate scenes (and initial and goal states) by randomly changing the property values of
       blocks in the scene; N= no. of scenes. // optional step
2  for $i \in [1,N]$ do
3      Determine components of RL formulation from ASP KB.
4      Learn policy $\pi_i$ using Q-learning [15,17].
5      Create table $Tb_i$ that stores the relative (numerical) utility of combinations of
       property values of the blocks.
6      for each state $s \in S$ do
7          Identify the best action ($a_{best}$) and worst action ($a_{worst}$) by selecting the highest
           and lowest (Q)values of actions corresponding to $s$.
8          Identify property values of the blocks involved in the execution of $a_{best}$ and
           $a_{worst}$.
9          Increment the utility of entries (i.e., rows) in $Tb_i$ that correspond to the property
           values identified for $a_{best}$.
10         Decrement the utility of entries in $Tb_i$ that correspond to the property values
           identified for $a_{worst}$.
11     end
12 end
13 $Tb_{total} = \sum_i Tb_i$.
14 Convert rows in $Tb_{total}$ with larger negative utilities than other rows to ASP rules.
15 Merge with existing ASP rules and generalize.
16 return ASP KB with new rules.
```

problem and learns a policy to achieve the desired configuration using Q-learning; this learning may be achieved using a combination of simulation and real-world trials. A table is created whose rows correspond to combinations of property values of blocks that can be involved in a *put* action. For each state in the set of states, the best action and the worst action are selected based on the computed policy, and the property values of blocks corresponding to these actions are identified. Entries in the table corresponding to the property values identified for action a_{best} (a_{worst}) have their relative utilities incremented (decremented). If multiple tables were created, they are summed up and the entries in the resultant table corresponding to large negative utilities are considered to represent actions that should not occur. The corresponding rules are encoded and merged with existing rules in the ASP KB, and used in subsequent planning tasks.

4 Experimental Setup and Results

In the tabletop domain, the robot's objective is to stack blocks with different properties in desired configurations (Section 3). We use the knowledge representation language SPARC to write the ASP programs [2]; it expands CR-Prolog that includes consistency restoring rules in ASP [7], and uses DLV [13] to obtain answer sets.

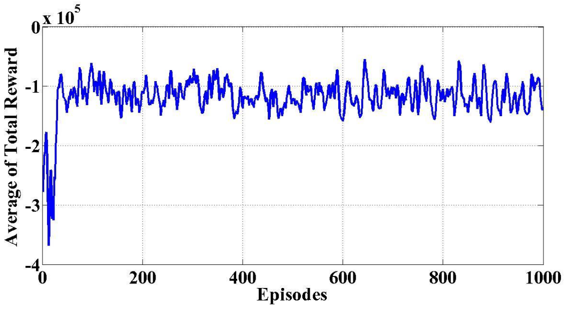

Fig. 2. Learning curve for three blocks with different shape (*square*, *rectangle*, *triangle*) but same color and size

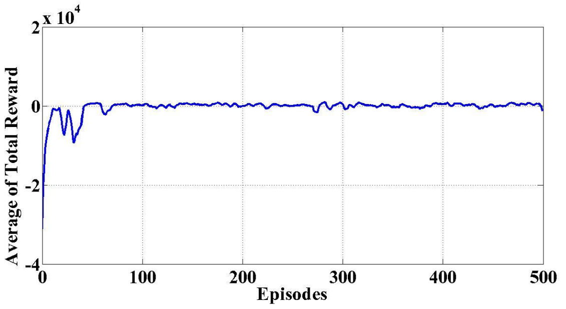

Fig. 3. Learning curve for three blocks with same shape (*triangle*), color, and size

4.1 Learning Curves

For Q-learning, we experimentally chose a learning rate $\alpha = 0.01$ and a discount factor $\gamma = 0.8$; these parameters influence the extent to which previously unseen regions of the state-action space are explored. The corresponding learning curves, convergence rates, and the average rewards are different based on the property values (and the number) of the blocks. Figure 2 and Figure 3 show learning curves obtained for two different scenes with blocks of the same size: (1) three blocks with different shapes but same color; (2) three triangles of the same color; the objective is to stack the blocks on top of each other. The curves are different because each initial and desired configuration of blocks determine the possible state transitions, actions and thus the rewards obtained during the learning phase. In all scenes, the policies are learned incrementally and efficiently.

4.2 Discovering New Rules

Once one or more policies are obtained, a table is constructed; each row of the table represents possible property values for two blocks involved in a *put* action. Table 1 shows an illustrative example of such a table, using a policy generated for the example in Section 3.2 with three blocks: *red square* (b_1), *red triangle* (b_2), and *blue rectangle* (b_3). The column of relative utilities ("Utility" in Table 1) is initialized to contain zeros.

The subsequent steps in Algorithm 1 are executed to update values in the table. For instance, for the best action in a specific state (a_{best}), the corresponding property values are identified and the utilities of the appropriate rows in the table are incremented.

Table 1. Table of relative utilities for a domain with three blocks of the same size (red square, red triangle and blue rectangle). The numbers in the last column represent the relative utility of placing a block of a specific shape (column 1) on another specific-shaped block (column 2) when they are of the same color or different color.

*Shape*1	*Shape*2	*Color*	*Utility*
Square	Square	Different	0
Square	Triangle	Different	0
Square	Rectangle	Different	5
Triangle	Square	Different	0
Triangle	Triangle	Different	0
Triangle	Rectangle	Different	10
Rectangle	Square	Different	-5
Rectangle	Triangle	Different	-20
Rectangle	Rectangle	Different	0
Square	Square	Same	0
Square	Triangle	Same	-10
Square	Rectangle	Same	0
Triangle	Square	Same	10
Triangle	Triangle	Same	0
Triangle	Rectangle	Same	0
Rectangle	Square	Same	0
Rectangle	Triangle	Same	0
Rectangle	Rectangle	Same	0

In Table 1, we observe that the largest negative utility corresponds to an action that would place a rectangle on top of a triangle of a different color. The next lowest utility corresponds to placing a square on top of a triangle with same color. The first observation can be translated into an ASP rule:

$$\begin{aligned}\neg occurs(put(b_1,b_2)) :- & blockShape(b_1,rectangle), \quad blockShape(b_2,triangle).\\ & blockColor(b_1,C_1), \quad blockColor(b_1,C_2), \quad C_1 != C_2.\end{aligned}$$

Each such rule is merged with existing rules by matching common predicates and generalizing across different groundings of a predicate, and the ASP KB is revised. Over multiple experimental trials, the robot may determine, for instance, that *no block can be placed on a triangle*:

$$\neg occurs(put(b_1,b_2)) :- blockShape(b_2,triangle).$$

For the task of stacking the three blocks (in Section 3.2), the revised ASP program produces the new plan:

$$put(b_3,b_1), \quad put(b_2,b_3)$$

Unlike the previous attempt (in Section 3.2) before using RL to discover rules, executing this action sequence results in the robot successfully stacking the three blocks.

4.3 Discussion and Future Work

We have evaluated our architecture on scenes with different number of blocks that have different properties and property values. The experiments indicate that the robot is able to incrementally and efficiently identify rules governing the domain dynamics. As the robot acquires more (accurate) domain knowledge, the ability to use the corresponding plans to complete the assigned tasks increases (and approaches 100%). Although the architecture is demonstrated in a simplistic domain in this paper, it addresses key knowledge representation and learning challenges in robotics. The architecture is thus applicable to other domains in which robots collaborate with non-expert humans.

This architecture opens up multiple directions for future research. It is non-trivial to generalize from the discovered rules and revise existing rules in the KB. Although the domain considered in this paper simplifies this problem, it is an interesting topic for further investigation. Another direction for future research is to explore the architecture's extension to support formulations similar to relational reinforcement learning (RRL). However, unlike existing RRL formulations, the use of ASP will support commonsense reasoning while the robot uses abstractions across different states and actions to make learning computationally tractable. It may also be possible to use the diagnostics capabilities of ASP to focus on a specific subset of the state-action space (in the RL formulation) in response to the failure of specific steps in the plan. Finally, the current implementation abstracts away the perception and actuation challenges in robotics. For physical robots in real world application domains, we are investigating the architecture's extension to partially observable states and non-deterministic action outcomes, using probabilistic belief states in the RL formulation.

5 Conclusions

This paper described an architecture that integrates the complementary strengths of RL and declarative programming to support knowledge representation, commonsense reasoning, and incremental discovery of unknown rules governing the domain dynamics. The domain knowledge encoded in the ASP KB is used to formulate the incremental discovery of domain rules as an RL problem. The action policies obtained by RL are used to discover rules that are, in turn, encoded in the ASP KB and used to plan action sequences for subsequent tasks. This architecture is thus a significant step towards the long-term objective of designing robots that can collaborate with and assist non-expert humans in real world application domains.

Acknowledgments. The authors thank Michael Gelfond for his feedback on the architecture described in this paper. This research was supported in part by the U.S. Office of Naval Research (ONR) Science of Autonomy Award N00014-13-1-0766. Opinions and conclusions are those of the authors and do not necessarily reflect the views of the ONR.

References

1. Aerolla, M.: Incorporating Human and Environmental Feedback for Robust Performance in Agent Domains. Master's thesis, Department of Computer Science, Texas Tech University (May 2011)
2. Balai, E., Gelfond, M., Zhang, Y.: Towards Answer Set Programming with Sorts. In: Cabalar, P., Son, T.C. (eds.) LPNMR 2013. LNCS, vol. 8148, pp. 135–147. Springer, Heidelberg (2013)
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
4. Blumberg, B., Downie, M., Ivanov, Y., Berlin, M., Johnson, M.P., Tomlinson, B.: Integrated Learning for Interactive Synthetic Characters. In: International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), pp. 417–426 (2002)
5. Dzeroski, S., Raedt, L.D., Driessens, K.: Relational Reinforcement Learning. Machine Learning 43, 7–52 (2001)
6. Erdem, E., Aker, E., Patoglu, V.: Answer Set Programming for Collaborative Housekeeping Robotics: Representation, Reasoning, and Execution. Intelligent Service Robotics 5(4), 275–291 (2012)
7. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning and the Design of Intelligent Agents. Cambridge University Press (2014)
8. Griffith, S., Subramanian, K., Scholz, J., Isbell, C., Thomaz, A.: Policy Shaping: Integrating Human Feedback with Reinforcement Learning. In: International Conference on Neural Information Processing Systems, Lake Tahoe, USA (2013)
9. Kaplan, F., Oudeyer, P.-Y., Kubinyi, E., Miklosi, A.: Robotic Clicker Training. Robotics and Autonomous Systems 38 (2002)
10. Knox, W.B., Fasel, I., Stone Design, P.: principles for creating human-shapable agents. In: AAAI Spring 2009 Symposium on Agents that Learn from Human Teachers (2009)
11. Knox, W.B., Stone, P.: Tamer: Training an Agent Manually via Evaluative Reinforcement. In: International Conference on Development and Learning, ICDL (2008)
12. Knox, W.B., Stone, P.: Combining Manual Feedback with Subsequent MDP Reward Signals for Reinforcement Learning. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS (2010)
13. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (2006)
14. Sridharan, M.: Augmented Reinforcement Learning for Interaction with Non-Expert Humans in Agent Domains. In: International Conference on Machine Learning Applications, ICMLA (December 2011)
15. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
16. Thomaz, A., Breazeal, C.: Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance. In: National Conference on Artificial Intelligence, AAAI (2006)
17. Watkins, C., Dayan, P.: Q-learning. Machine Learning 8, 279–292 (1992)
18. Zhang, S., Sridharan, M., Gelfond, M., Wyatt, J.: Integrating Probabilistic Graphical Models and Declarative Programming for Knowledge Representation and Reasoning in Robotics. In: Planning and Robotics (PlanRob) Workshop at ICAPS, Portsmouth, USA (2014)