Learning MDP-ProbLog Programs for Behavior Selection in Self-Driving Cars*

Alberto Reyes$^{1[0000-0002-8509-6974]}$, Héctor Avilés$^{2[0000-0001-5310-3474]}$, Marco Negrete$^{3[0000-0002-5468-2807]}$, Rubén Machucho$^{2[0000-0002-5731-6677]}$, Karelly Rivera$^{2[0000-0002-4749-0663]}$, Gloria I. de-la-Garza-Terán$^{2[0009-0004-2446-6435]}$

[1] National Institute of Electricity and Clean Energies, Morelos 62490, Mexico, `areyes@ineel.mx`

[2] Polytechnic University of Victoria, Cd. Victoria Tamaulipas 87138, Mexico, `{havilesa,rmachuchoc,1930435,2130071}@upv.edu.mx`

[3] Faculty of Engineering, National Autonomous University of Mexico, Mexico City 04510, Mexico, `marco.negrete@ingenieria.unam.edu`

Abstract. A two-stage scheme to learn MDP-ProbLog programs for self-driving cars is proposed. In a first stage, the transition and reward functions will be learned from simulated driving examples. Both functions will be described as an influence diagram (*ID*). In a second stage, the ID will be converted into a set of probabilistic clauses that will match the syntax of MDP-ProbLog. During this process, non-essential rules will be removed and redundant ones will be merged. The architecture of our self-driving car includes behavior selection, visual perception and control. This proposal is part of an ongoing research to evaluate the suitability of probabilistic logic to model safe autonomous decision-making in self-driving cars.

Keywords: Probabilistic logic, Factored Markov decision processes, self-driving cars.

1 Introduction

Self-driving cars promote potential positive effects for the mobility of humans and goods. An important capability of those type of vehicles is the autonomous selection of behaviors, that is responsible for deciding what reactive or short-term action (e.g., braking, accelerating, or stopping) is more appropriate in a current driving scenario to improve the safety of navigation.

We believe that probabilistic logic[1] is convenient for the selection of driving behaviors because: i) logical rules benefit the interpretation and explainability of decisions in comparison to pure numerical representations, and ii) probability theory is the most widely used framework for dealing with uncertainty. In

* This work was supported by UNAM-DGAPA under grant TA101222 and Consorcio de IA CIMAT-CONACYT

[1] We restrict our attention here to first-order logic clauses (rules and facts) extended with probability values.

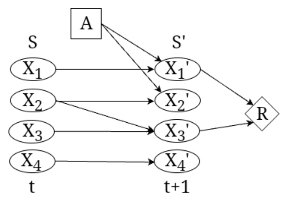

Fig. 1. An example of a influence diagram representing a transition function.

particular, Markov decision processes (*MDPs*) are gaining increasing attention for sequential decision-making in autonomous vehicles due to its capability to generate useful driving policies. Therefore, given that MDP-ProbLog does not include a learning model, we propose a two-stage scheme for learning MDP-ProbLog Programs (*MDP-PL*) [2] to generate action policies useful as a behavior selection scheme for self-driving cars. The numerical transition function and the reward function will be translated into a simpler propositional clause-based representation to construct the policy using MDP-ProbLog.

2 Factored Markov Decision Processes (*FMDPs*)

In Factored MDPs, states of the system are identified through the instances of a set of state random variables. This allows to reduce the number of model parameters in comparison to traditional MDPs, and to explore relationships among state variables. FMDPs require the definition of four main elements: i) the set of n state variables $\mathcal{X} = \{X_i\}_{i=1}^{n}$ along with the set $\mathbf{X}$ of all possible joint value assignments for all the variables in $\mathcal{X}$, ii) the set of possible actions $\mathcal{A}$ a decision maker can choose, iii) the transition function $p(\mathbf{x}'|\mathbf{x}, a)$ (that it is a shorthand notation for $p(X_1' = x_1', ..., X_n' = x_n'|X_1 = x_1, ..., X_n = x_n, a)$), where $\mathbf{x} \in \mathbf{X}$ represents a *pre-action* state, $\mathbf{x}' \in \mathbf{X}$ is a *post-action* state, and $a \in \mathcal{A}$, and iv) the reward model $R(\mathbf{x}, a)$. The transition function can be compactly represented via influence diagrams (*IDs*) [3] by taking advantage of conditional independence assumptions among random variables. A 4-variable ID representing a transition function is depicted in Fig. 1. IDs usually requires more compact conditional probability tables (*CPTs*) to represent probability distributions over post-action state variables, in comparison to a CPT of a transition function without conditional independence suppositions. As rewards are determined by the environment, the reward function $R(\mathbf{x}, a)$ could not be known beforehand by the decision maker, and it could be also learned.

3 Current progress on modeling driving decisions with MDP-PL

In this section, the behavior selection and the control and perception modules of the current self-driving architecture are briefly discussed. More details can be found in **[1]**.

3.1 Behavior selection module

In the selected problem domain, there are one self-driving car and four vehicles traveling on a one-way street with two lanes. Once the self-driving car moves, accordingly to the existence of other cars nearby, it has to decide one of three actions: a) cruise motion, b) overtaking or c) keep distance. The self-driving starts its movement on the right lane. The perceptual system (described in Section 3.2) reports constantly if there are other cars in predefined positions on the right and left lane. Those positions are labeled as North, North-West, West and South-West. Each one of these labels are associated with a Boolean state variable, and hence, the occupancy of the spaces around the self-driving car defines each one of the $2^4 = 16$ possible states of the system (Fig. 2).

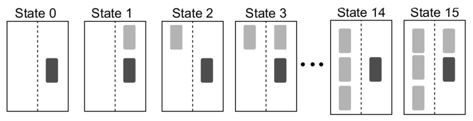

Fig. 2. States of our driving environment. The dark grey rectangle represents the self-driving car and the light grey rectangles are the nearby vehicles.

An early version of an MDP-PL was designed and implemented manually and coded in MDP-ProbLog. The reward function is based on (independent) additive utilities assigned to actions (without regarding on the state in which the actions are performed) and to state variables (and hence, it adds the utility to subsets of states that share the same value of the state variable). However, deterministic reward functions can be defined by including rules that assign utilities to specific state-action pairs. The source code is available in: https://github.com/hector-aviles/CaDis_Workshop. A number of 320 decision trials were performed, from which the car selected the right action in 98.75% of cases.

3.2 Perception and control modules

The perception module is divided in lane detection and lane tracking. Lane detection is based on regions of interest, the Canny edge detection and the Hough Transform to find the right and left limits of the lane, using an RGB camera. Steering is calculated based on the error between the expected and observed lines. The position of other vehicles and the estimation of their speed can be summarized in the following steps: i) a 3D Lidar sensor generates a point cloud that is filtered by distance and height of the vehicles nearby, ii) filtered points are clustered by K-means, and iii) the velocity of the vehicles is estimated by Kalman Filters. The source code of the architecture can be downloaded from: https://github.com/hector-aviles/CodigosEIR22-23.

4 Learning MDP-ProbLog programs

To learn an MDP-PL, data are collected from random driving actions performed by the self-driving car on the environment described in section 3.1. This data will be partitioned and sequentially registered in ordered 4-tuples $d_t(\mathbf{x}, a, \mathbf{x}', r)$ indexed in time $t \in \{1, ..., T\}$, such that $T \in \mathbb{N}$, $\mathbf{x} \in \mathbf{X}$ is the current observed state, $a \in \mathcal{A}$ is the current performed action, $\mathbf{x}' \in \mathbf{X}$ is the resulting state and r is a numerical reward value assigned to $\mathbf{x}$ and a. It's been considered to record from the simulator: timestamps, relative positions of the detected cars and velocities, action selected, and if its execution resulted successful or not. The reward r can be a positive quantity if the state-action pair does not lead to an accident or a negative value if a car crash takes place.

In the first learning stage, the K2 algorithm will be used to learn the transition function $p(\mathbf{x}'|\mathbf{x}, a)$. The reward function will be obtained by using J48 to generate a reward decision tree as detailed in [5]. In the second learning stage, the ID will be converted into an MDP-PL. These probabilistic clauses could be further simplified as referred in [4].

5 Conclusions and future work

A two-stage scheme to learn MDP-ProbLog programs to select driving behaviors in self-driving cars was proposed. The plan is to learn a dynamic Bayes net (factored transition function) and the reward function. We believe that this approach will help select safe driving actions, improve the understandability of the model, and discover causal relationships among variables that represent the entities on the road and the driving decisions to perform safe manouvers.

References

1. Avilés, H., Negrete, M., Machucho, R., Rivera, K., Trejo, D., Vargas, H.: Probabilistic logic markov decision processes for modeling driving behaviors in self-driving cars. In: Advances in Artificial Intelligence–IBERAMIA 2022: 17th Ibero-American Conference on AI, Cartagena de Indias, Colombia, November 23–25, 2022, Proceedings. pp. 366–377. Springer (2023)
2. Bueno, T.P., Mauá, D.D., De Barros, L.N., Cozman, F.G.: Markov decision processes specified by probabilistic logic programming: representation and solution. In: 2016 5th Brazilian Conference on Intelligent Systems (BRACIS). pp. 337–342. IEEE (2016)
3. Darwiche, A., M., G.: Action networks: A framework for reasoning about actions and change under understanding. In: Proc. of the Tenth Conf. on Uncertainty in AI, UAI-94. pp. 136–144. Seattle, WA, USA (1994)
4. Muggleton, S.H.: Duce, an oracle-based approach to constructive induction. In: IJCAI. vol. 87, pp. 274–281. Citeseer (1987)
5. Reyes, A., Ibarguengoytia, P.H., Santamaría, G.: SPI: A software tool for planning under uncertainty based on learning factored Markov Decision Processes. In: Advances in Soft Computing. Lecture Notes in Computer Science Series, vol. 11835. Springer (2019)