2020 IEEE Intelligent Transportation Systems Conference (ITSC)
Rhodes, Greece. September 20-23, 2020 (Virtual)

Explainability of Intelligent Transportation Systems using Knowledge Compilation: a Traffic Light Controller Case

Salomón Wollenstein-Betech[1], Christian Muise[2], Christos G. Cassandras[1],
Ioannis Ch. Paschalidis[1], and Yasaman Khazaeni[3]

***Abstract*— Usage of automated controllers which make decisions on an environment are widespread and are often based on black-box models. We use Knowledge Compilation theory to bring explainability to the controller's decision given the state of the system. For this, we use simulated historical state-action data as input and build a compact and structured representation which relates states with actions. We implement this method in a Traffic Light Control scenario where the controller selects the light cycle by observing the presence (or absence) of vehicles in different regions of the incoming roads.**

I. Introduction

Recent developments in computing power, algorithms and data handling have allowed for both accurate and complex automated decision-makers. These smart agents have been adopted widely in academia and industry to perform different tasks. With the same vigor as in other areas, these methods have been embraced to perform many tasks in the context of Smart Cities [1] and Intelligent Transportation Systems. Some examples include multi-agent traffic light controllers [2] and re-balancing of Mobility-on-Demand systems [3].

A typical model (e.g., deep reinforcement learning) uses high-dimensional inputs to provide powerful predictions or decisions to achieve the desired goals. Unfortunately, the trade-off between the complexity and the interpretation of the model often limits its adoption to many applications where stakeholders must trust and explain the decisions taken.

The importance of explainability does not rely purely on justifying an agent's decisions. Understanding more about its controller logic provides clarity over unknown vulnerabilities of the controller. Also, it helps in identifying and correcting errors (debugging), thus enhancing the controller [4].

With the success of non-interpretable methods (neural networks and boosting) the area of explainable AI (XAI) is now an exciting research topic, see Figure 1. The relevance of interpretability goes beyond the tech industry. For example, lawyers also benefit from understanding these models well when faced with claims about fairness of an agent's decision. As a response to this issue, the European Union has taken the lead by including a "right to an explanation" [5] stated in articles 12-15 of the recently passed *General Data Protection Regulation* (GDPR) law which pushes decision-maker architects to provide clear and concise statements of the logic involved.

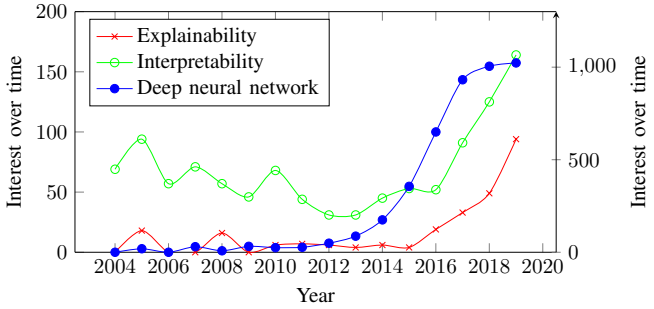


Fig. 1: Google Trend, read Explainability and Interpretability with left axis and Deep Neural Network with right axis

Within the transportation community, a tragic example is the *Death of Elaine Herzberg* in Tempe, Arizona [6]. This was the first recorded death provoked by a self-driving car. Many attribute the death to a mistake on the image classification method used by the vehicle. Nevertheless, we can't be sure of what exactly went wrong. This further encourages the interpretability of our intelligent transportation systems.

In addition to these examples, we observe a growing trend in the relevance of *explainability* (or *interpretability*[1]) in google searches and in academia. To give one example, there was a 300% growth in the number of papers presented at the *Intelligent Transportation Systems Conference* in 2019 relative to the previous year, that contained these words.

In this paper we present an interactive tool that helps bring interpretability to black-box controllers. We reason over the behavior of an agent by looking at historical data traces of state-action pairs without making assumptions on the system dynamics. Moreover, we assume the actions taken by the controller are fixed (i.e., a deterministic policy) but unknown.

Related work has tried to solve similar problems on imitation or apprenticeship learning [7], [8]. The idea is to learn a policy from state-action samples, so that when the automated controller takes an action, it closely resembles the behavior of the original decision-maker. To achieve this goal, these methodologies require learning the dynamics of the system typically modeled as a Markov Decision Process (MDP). Our methodology differs, since it does not aim to learn the system dynamics (transition probabilities) nor how to control the agent. Rather, our goal is to provide useful and fast knowledge about the behavior of the agent on the environment. Our tool allows answering questions that

Supported in part by NSF under grants ECCS-1509084, DMS-1664644, CNS-1645681, and IIS-1914792, by AFOSR under grant FA9550-19-1-0158, by ARPA-E's NEXTCAR program under grant DE-AR0000796, by the MathWorks, by the ONR under grant N00014-19-1-2571, and by the NIH under grant 1R01GM135930.

Research performed during an internship at the MIT-IBM Watson AI Lab.

[1]Dept. of Electrical and Computer Engineering, Division of Systems Engineering, Boston University, Boston, MA, USA. {salomonw, cgc, yannisp} @bu.edu

[2] School of Computing, Queen's University, Kingston, ON, Canada. christian.muise@queensu.ca

[3] IBM Research AI, Cambridge, MA, USA. yasaman.khazaeni@us.ibm.com

[1]In this paper we use these terms interchangeably

other approaches lack. For example, we can ask: *what is the probability of performing an action when a specific state variable is* `True`*?*.

We use the approach presented in [9]. The main idea is to use state-of-the-art knowledge compilation methods and to use disjunctive decomposable negation normal form (d-DNNF) as the key representation [10] of the controller's logical theory. This representation provides an interesting mix of compact representation, computationally efficient compilation, and expressive inference capabilities.

We bring this technology to the Transportation Community by analyzing a case study of the Traffic Light Control (TLC) problem. We chose to analyze this problem given the recent success of using Deep Reinforcement Learning (RL) to tackle the single-intersection TLC [11] and multi-intersection TLC [2], [12], [13]. However, despite Deep RL's great success on solving TLC, there is a limited amount of work attempting to explain the logic behind its trained controllers or predictors within the transportation domain. To the best of our knowledge, most attempts [14], [15] addressing the interpretability problem use Shapley Additive exPlanation (SHAP) methodology [16]. Our approach differs from SHAP in that we are, in a sense, constructing a hierarchical representation of interpretable insights rather than identifying the individual factors that contribute most to the output of the blackbox.

The rest of the paper is organized as follows. In Section II we present background information on Knowledge Compilation and the d-DNNF language which serves as the basis for the interpretability model stated in Section III. In Section IV we present the TLC model, the RL approach used to train the smart controller, as well as the experiments performed. In Section VI we present some examples of explanations provided by the interpretability tool. We conclude in Section VII with a summary and with future directions.

II. Knowledge Compilation

Background: The objective of Knowledge Compilation is to perform tractable operations of a complex logical theory. To achieve this, the technique builds structured representations of the logical theory in the form of a directed acyclic graph (DAG). The main idea is to compile off-line a complex and unorganized logical theory into a structured one, which is then used on-line to perform fast operations and reasoning.

In the framework developed by [9], the authors propose deterministic decomposable negation normal form (d-DNNF) as the target language to perform interpretation over a logical theory. The main characteristic of this language is that it allows to *condition* and to *count* the number of models in a propositional theory in polynomial time. These two operations are central in answering questions about an agent's behavior. Note that the off-line compilation to the d-DNNF language may be computationally expensive [10], however, this is just performed once. Let us now define properly the d-DNNF by introducing some of the languages used in Knowledge Compilation.

Languages:

1) Negation Normal Form (NNF): Most languages in Knowledge Compilation are subsets of the NNF language [10]. In this language, the only allowed Boolean operators are conjunctions ($\wedge$, `and`) and disjunctions ($\vee$, `or`). The negation operator ($\neg$, `not`) is directly applied to the Boolean variables. In practice, these languages by are represented with a Directed Acyclic Graph (DAG). The graph describing a NNF theory is composed by leaves taking positive or negative boolean variables and inner nodes that are either conjunction or disjunction, see Figure 2b as an example.

Formally, let Σ be a propositional theory (a DAG). Let C be any node in Σ and $\texttt{Vars}(C)$ be the set of variables appearing in the subgraph rooted at C.

2) Disjunctive Normal Form (DNF): This language is a subset of NNF and is formed by a disjunction of conjunctions (`or` of `and`'s), see Figure 2c as an example. Its DAG is *flat*, meaning that the distance from the root node to any leaf is 2. We represent the received data $\mathcal{D}$ using this language. For each state-action pair $\langle s_i, a_i \rangle$ we build a clause C_i. Then, we take the disjunction over all of the C_i's.

3) Conjunctive Normal Form (CNF): Similar to DNF, the CNF language is a conjunction of disjunctions (`and` of `or`'s) whose DAG is also *flat*. The intent of using this intermediate language between our data (DNF) and target (d-DNNF) is twofold. (1) Ease the compilation CNF→d-DNNF by using ready-to-use compilers such as DSHARP [17], c2d [18] or D4 [19], and (2) allow for encoding with different properties (see details on the different encoding flavors in [9]).

4) Deterministic Decomposable Negation Normal Form (d-DNNF): This language is a subset of the NNF in which the properties of *decomposability* and *determinism* hold. The objective of having these two properties is to perform fast (polynomial time) operations of model counting and conditioning over the logical theory. The definition of these properties (as in [10]) are:

Definition 1 (Decomposability)**.** A NNF satisfies the decomposability property if for any conjunction C, the conjuncts of C do not share any variable. In other words, if $C_1, ..., C_n$ are children of an `and` node C, then $\texttt{Vars}(C_i) \cap \texttt{Vars}(C_j) = \emptyset$ for $i \neq j$.

Definition 2 (Determinism)**.** A NNF satisfies the determinism property if for any disjunction C, every pair of disjuncts of C are logically contradictory. That is, if $C_1, ..., C_n$ are children of an `or` node C, then $\forall i, j \in [1, ..., n]$ where $i \neq j$, $C_i \wedge C_j = \texttt{False}$.

A. Logical operations in d-DNNF form

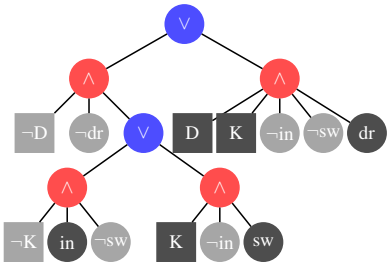
1) Model Counting: Without loss of generality, it is possible to count the number of models of a d-DNNF in polynomial time. Consider a logical theory Σ and replace its `or` nodes by *additions* and its `and` nodes by *products*. Then, assign to each leaf the value of 1. Note that an additional easy-to-comply property of *smoothness* is needed to obtain a normalized count.

Definition 3 (Smoothness)**.** A NNF satisfies the smoothness property if for each disjunction C, each disjunct of C mentions the same variable. That is, $C_1, ..., C_n$ are children of an `or` node C, then $\texttt{Vars}(C_i) = \texttt{Vars}(C_j)\ \forall i, j \in [1, ..., n]$ where $i \neq j$.

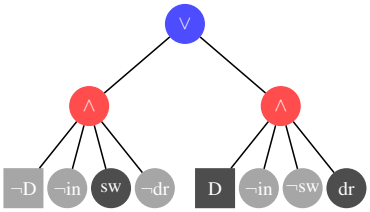
2) Conditioning: Let Σ be a d-DNNF and consider the problem of conditioning on variable x, i.e., we would like $\Sigma | x = \texttt{True}$. Then, we can replace x by `True` and $\neg x$ by `False` and propagate this information throughout the DAG

Drive mode on	**K**ey inside car	Action
True	True	**dr**ive
False	True	**sw**itch_to_drive_mode
False	False	**in**sert_key

(a) Data



(b) Compiled d-DNNF



(c) Conditioned d-DNNF on K

Fig. 2: Example compilation of agent behaviour. Let *D*, *K*, *dr*, *sw*, *in* be `Drive_mode_on`, `Key_inside_car`, `drive`, `switch_to_drive_mode`, and `insert_key` respectively. Figure 2a denotes the input $\mathcal{D}$ to the model, which we read every row as a conjunction. Figure 2b is the generated d-DNNF representation of the logical theory, which is compiled following the interpretability model depicted in Figure 3. To read the d-DNNF DAG, we suggest starting from top to bottom, and look at the left-most element on after a disjunction. This depicts the determinism property since the children of the `or` node are logically inconsistent. Lastly, Figure 2c is the conditioned version of the d-DNNF when we set K to be active. One can observe that to get the conditioned tree we can prune the left branch of the lower `or` node in Figure 2b.

using standard logical rules. For example, if $\Sigma = (x \vee y) \wedge (\neg x \vee z)$, then, $\Sigma|x = (\texttt{True} \vee y) \wedge (\texttt{False} \vee z)$ which simplifies to z. Through the combination of conditioning and counting, the likelihood of a particular variable can be computed using $P(x = \texttt{True}) = \frac{count(\Sigma|x)}{count(\Sigma)}$.

III. Interpretability Model

Recall that our goal is to reason over an agent's (traffic light) decisions and to answer questions about its underlying logic by analyzing samples of data. Given that we are using a model based on logical theories, the domain is restricted to discrete representations of states and actions. Nevertheless, one might consider using discretization to parse continuous to discrete domains.

Let the set describing the state variables to be $\mathcal{F}$, the state-space (environment) be $\mathcal{S} \in 2^{\mathcal{F}}$ and a particular state to be $s \in \mathcal{S}$. We use $\mathcal{A}$ to represent the action space and $a \in \mathcal{A}$ to specify an action. Furthermore, let a state-action observation be a tuple $\langle s, a \rangle$ and our data be $\mathcal{D} = \{\langle s_1, a_1 \rangle, ..., \langle s_m, a_m \rangle\}$. The problem we face is then to succinctly represent the mapping $\mathcal{P} : \mathcal{S} \rightarrow \mathcal{A}$. One intuitive way of solving the mapping problem would be to use a large table of state-actions. Unfortunately, this approach becomes quickly intractable due to the *curse of dimensionality*. To overcome this computational burden, we use the model proposed by [9] based on Knowledge Compilation.

The processing framework in [9] uses the sampled data of state-action tuples as clauses in disjunctive normal form (DNF). Then, it compiles this theory to various flavors of conjunctive normal form (CNF), see [9] for different encodings and properties. Once the theory is in CNF form, it compiles it using any off-the-shelf compilers [17], [19] to produce a d-DNNF, which has the properties of *decomposability* and *determinism*. Then, easy operations of model counting and conditioning generate probabilistic inference responses on the behavior of the system. See Figure 3 as a summary of the approach. As an example, consider the data received on Figure 2a, its corresponding d-DNNF representation on Figure 2b, and its conditioned response on Figure 2c.

IV. Traffic Light Control

The Traffic Light Control (TLC) problem consists of dynamically adjusting a road's green and red light cycle to maximize the traffic flow trough an intersection. In the past, the problem has been tackled using estimates of traffic

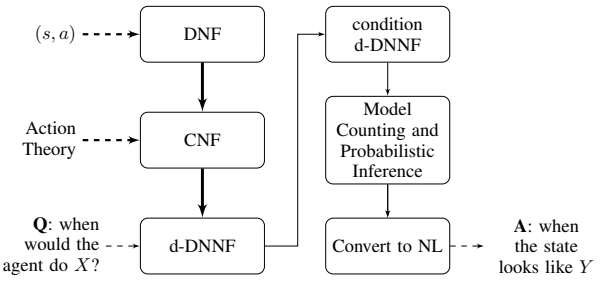


Fig. 3: Flowchart of the logical model. Dotted arrows indicate inputs to the model whereas solid arrows indicate processing steps. Bold arrows indicate off-line (pre-processing) tasks.

flows on each road for a specific time of the day [20], [21]. Today, with the ability to gather and communicate information in real time, traffic-responsive techniques are taking the lead on controlling these systems. See examples using Infinitesimal Perturbation Analysis [22], SCOOT [23] and many Reinforcement Learning variations for single [11]–[13], and multi-intersection [2], [24] control. The use of RL agents for traffic light control is motivated by the fact that agents can self-train. If these are trained correctly and for long enough samples, we can expect them to adapt to different situations including road accidents, weather and other variables. In particular, Deep RL differentiates itself by its ability to handle high-dimensional inputs. The RL agent learns to maximize a *reward* function by observing the system state and by training a model that relate an output by using a complicated function of input variables.

Given that our interpretability model needs both states and actions to be discrete, we would like to choose a TLC formulation that meets these requirements and facilitates the analysis. Hence, we consider a single intersection traffic light control scenario. We build a simulation model using SUMO [25] consisting of a cross intersection with 4 incoming and outgoing lanes. Each incoming road to the intersection is set to be 750 meters long. We divide every road on the network into n *movements*. These *movements* have predefined routes for all the cars flowing through a particular lane. In our case, let the left-most lane of every incoming edge be a movement (left turn) and the other lanes be another movement (keep straight), see Figure 4 as a reference. During the simulation, the agent (traffic light controller) samples the environment and receives a state s_t and a reward r_t at time t. According to this observation the agent chooses its next action a_t. At the same time the agent learns about the consequence of having

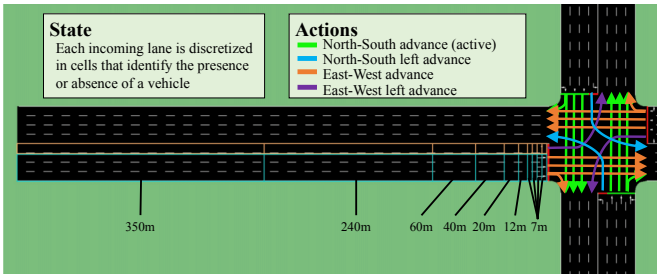


Fig. 4: Traffic Light environment.

chosen its previous action and updates its decision policy.

State: Following the model in [11] we let the state of the system be the collection of variables describing the presence or absence of vehicles on movement cells. We divide each incoming road into movements and we assume the total number of movements approaching the intersection is m. In our scenario $m = 8$ as we divided each road in 2 (vehicles turning left or keep straight) and we have 4 incoming roads. Then, we divide each of these movements $i = 1, .., m$ into b_i cells, which might differ in size (see Figure 4). The choice of the length of a cell is not trivial. If cells are too long we have lower granularity, in contrast, when cells are too short it brings higher computational complexity which requires longer training times. The state is then $s_t = \{x_{ij}(t) \mid i = 1, ..., m;\ j = 1, .., b_i\}$ where the variable $x_{ij}(t)$ is equal to 1 when there is at least one vehicle present on cell j of movement i at time t, and 0 otherwise. Then the number of possible states in the TLC system is $|\mathcal{S}| = 2^{\sum_{i=1}^{m} b_i}$.

Note that our interpretability tool is not limited to this particular state-space. However, this representation meets our need for discrete state variables. Some other approaches include variables such as the relative velocity between vehicles [26], current traffic light phase [27], among others.

Actions: We consider a single agent (the traffic light controller) which can choose between four possible actions. Each of these actions corresponds to a given configuration of red and green lights. The possible light phases are: North-South (NS), North-South left-most lane (NSL), East-West (EW), and East-West left-most lane (EWL). Hence $\mathcal{A} = \{\text{NS}, \text{NSL}, \text{EW}, \text{EWL}\}$. Once an action is selected, the traffic light will maintain this light phase for a fixed amount of time. We chose this length to be 10 seconds. Note, that the controller might choose the same action if for example, there is a lot of vehicles flowing on a particular direction. This does not impose an upper bound on the time a specific phase is active. In contrast, if the new selected action is different from the previous action, a 4 seconds yellow phase is initiated before starting the new phase. This delay allows drivers to anticipate and prepare before reaching the intersection. Additionally, it also benefits the model by preventing switching too often between light cycles.

Reward Function: In a RL setting, the reward function is the feedback the agent receives from the environment after performing an action. This feedback helps the agent improve its model of the environment in order to make better decisions in the future. In the setting of TLC, the objective is to maximize the traffic flow through the intersection over time. Hence, the reward is usually a measure of a vehicle's delay, queue lengths, waiting times or overall throughput which serve as proxies for traffic flow maximization. In this paper we use the reward function r_t proposed in [11].

Let $w_\theta(i, t)$ be the cumulative time over which a vehicle i has had speed smaller than θ up to time t. Then, assuming n cars have arrived to the environment before time t, the total cumulative waiting time at t is $W_t(\theta) = \sum_{i=1}^{n} w_\theta(i, t)$. We define a reward function r_t such that a positive value encourages an action and a negative value discourages it. Hence, a bad action can be represented by the increase in the cumulative waiting time when compared with the previous agent step (decision time). Let's assume that the agent would make decisions on the times defined by the sequence $\{t_1, t_2, ...\}$. Then the reward function at decision time t_i is $r_{t_i}(\theta) = W_{t_{i-1}}(\theta) - W_{t_i}(\theta)$.

Learning Process:

1) Model: We use Deep Q-Learning as our learning algorithm. This technique combines Q-Learning and Deep Neural Networks. Q-learning is a basic form of Reinforcement Learning which uses Q-values to iteratively improve an agent's decision. These values are a learned metric of how good it is to take a particular action given a specific state and are formally expressed by

$$Q(s_{t_i}, a_{t_i}) \leftarrow Q(s_{t_i}, a_{t_i}) + \alpha(r_{t_{i+1}}(\theta) + \gamma \max_{a \in \mathcal{A}} Q(s_{t_{i+1}}, a) - Q(s_{t_i}, a_{t_i}))$$

where α is the learning rate, and $\gamma \in [0, 1]$ is the discount factor used to leverage the importance of future rewards compared to the immediate one.

Often, computing all possible combinations of states and actions is intractable due to the curse of dimensionality which results in not having Q-values for some state-action pairs. To overcome this limitation, we estimate the Q-learning function using a deep neural network (DNN). We use the DNN architecture as in [11]. This DNN is fully-connected and is composed by an input layer of $\sum_{i=1}^{m} b_i$ (the size of the state of the system) and 5 hidden layers of 400 neurons each using a rectified linear unit (ReLU) function. The output layer contains 4 neurons with a linear activation function representing the value of an action given a particular state.

2) Training: We use *Experience Replay* [28] as our training method. This approach uses *batch* learning instead of adjusting an agent's policy at every decision. In other words, rather than updating the policy at every step, the agent uses all gathered information to update only at pre-defined moments. We call every training cycle an *episode* and define E as the total number of *episodes* used to train an RL agent.

In order to face the *exploration-exploitation* trade-off, we use an ϵ-greedy method with a linear exploration strategy. Let $e \in \{1, 2, ..., E\}$ indicate the current episode index, then the ϵ parameter at e is expressed by $\epsilon_e = 1 - e/E$. This method gives more weight to exploring at the beginning of the training phase, but as learning occurs, the agent exploits and reinforces its learned knowledge about the system.

V. Experiments

We perform experiments consisting on using the TLC model presented in Section IV to train an efficient black-box agent to control a traffic light. Once the agent is trained, we compute multiple traces of state-action pairs which serve as the input data $\mathcal{D}$ for the interpretability model explained

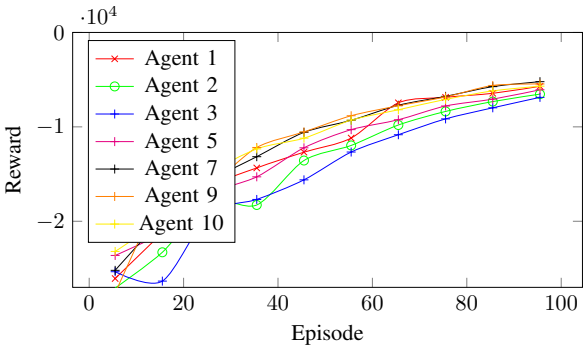


Fig. 5: Training different controllers. The agent number corresponds to the number of observable cells from the intersection on each movement.

in Section III. Then, we interact with the interpretabilty tool to reason over the underlying logic of the RL controller. To train the agent we use the simulation environment created in [11]. We provide a diverse set of traffic flows aiming to learn real world scenarios. Moreover, we randomly increase and decrease the traffic generation on each of the incoming roads to create combinations of high and low traffic intensities on the roads. Each simulation (or episode) consists of $5,400$ seconds equivalent to 1.5 hours.

We consider the scenario with 8 movements and divide each of these into 10 cells (i.e., $n_i = 10$ for all i) as in Figure 4. Based on this state-space representation we trained 7 different RL controllers aimed at assessing their performance and reasoning over their underlying behavior. The difference between these agents is their observation ability. For each agent $l = 1, ..., 10$ we define their state-space to be the first l cells of a movement i, formally, $s_t(l) = \{x_{ij}(t) \mid i = 1, ..., m;\ j = 1, .., l\}$. For example, agent 2 will be trained on a state-space based on $x_{ij}(t)$ for $j = 1, 2$ exclusively. To assess the performance of each agent we learn over 100 episodes ($E = 100$) and compare the reward trend while learning, see Figure 5. An interesting observation about this training process is that Agent 2 and 3 have worse performance than Agent 1 even though they observe everything 1 does and more. We believe this behavior happens because the controller has difficulties differentiating between presence on a cell due to congestion or due to a passing vehicle. However, once the agent has more information further down the road, as Agent 10 does, it is easier to differentiate the cause of the presence of a vehicle in a cell.

VI. TLC Interpretability Results

After training the controller, we ran 100 simulations for each agent $l = 1, ..., 10$ to create historical data $\mathcal{D}_l$. We used $\mathcal{D}_l$ as the input of the interpretability model to interact it with the representation of the logical theory of the traffic light. The results obtained matched our intuition on how a TLC should operate. Take for example the first row in Table I. In this case Agent 1 observes the first cell of each movement. We asked the interpretability model to give us the likelihood of an action when a vehicle is present on the first cell of the east-straight movement, unknown situation on the east-left movement, and no vehicles on any other movement. The result shows that the agent will choose either EW or EWL cycle with 50% chance, given those condition. In Fig. 6, we show the conditioned d-DNNF DAG for this query.

TABLE I: All the queries assume unknown state variables unless specified in the table. Notation: **R-GM_N** stands for incoming **R**oad (N/S/E/W), **G**reen phase, **M**ovement (**0**=straight/**1**=left), and cell position **N**.

Query	Agent	Action Likelihood			
		NS	NSL	EW	EWL
Vehicle in E-G0_0-7; E-G1_0-7 unknown; No vehicle in the rest	1	0.0%	0.0%	50.0%	50.0%
No conditioning	7	33.6%	13.7%	37.3%	15.4%
Vehicle present in E-G0_0-7	7	21.4%	11.7%	58.2%	8.0%

More interestingly, on the second row of Table I, we asked for the likelihood of an action without conditioning on anything. As expected, the time of EW vs. EWL (and NS vs. NSL) was not uniform. This is because in the simulation, we consider higher vehicle flows going straight versus turning left (3 straight lanes versus 1 turning left). Hence, if the RL controller tries to minimize the overall delay, we expect it to give preference to the straight trajectories over turning trajectories. From a debugging point of view we are satisfied with this result as it matches our intuition. The last row considers the presence of a car in the east-straight lane in position 1 for Agent 7, and all the other state variables are unspecified. Recall that Agent 7 observes for each road the first seven cells starting from the intersection. We expect this query to give us information about the marginal gain that the action EW receives when the controller observes presence of a vehicle on the first cell (0-7 meters from the traffic light). Intuitively, we expect the likelihood of action EW to be greater than the likelihood of other actions, which is exactly what we observe in the results.

Another type of query we can ask our tool is to understand the environment conditional on an action being active. Take for example the question asked to Agent 1: *What is the likelihood of a state variable when the controller decides on action NS?* The result of this query is shown in Figure 7. As we anticipate, we see that it is very likely to have vehicles in the north and south straight movements, whereas it is less likely to observe vehicles on the east, west or left movements. In summary, the results obtained by the interpretability model match our intuition. The technique naturally lends itself to debugging black-box controllers, and ultimately aids in understanding the operation of the controller.

VII. Conclusions

We use Knowledge Compilation techniques to reason over the behavior of black-box controllers. We present an example of this tool on a traffic light control setting where the controller uses a Deep Neural Network to decide its actions given a state. In this setting, the objective of the agent is to dynamically select the best light phase given a state composed by the presence (or absence) of vehicles in different cells of a road network. Once the black-box controller is trained, it is used to control the system. We then sample historical state-action pairs and use them as an input to the interpretability model. Then, we use the interactive tool which allows us to reason over the environment and the agent's decisions. It is worth pointing out that this general framework allows to reason over any type of decision-maker (including humans) and it is not reserved for a particular technique such as RL or any application such as TLC. For all the queries we performed, the interpretation given

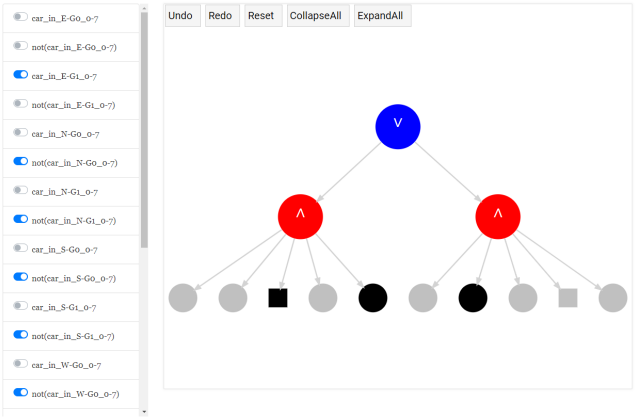


Fig. 6: Screenshot of the controller logic of Agent 1 in the interactive tool. On the left hand, the user can toggle (on/off) to condition on a particular state variable or action. The controller is conditioned as stated in the first line of Table I.

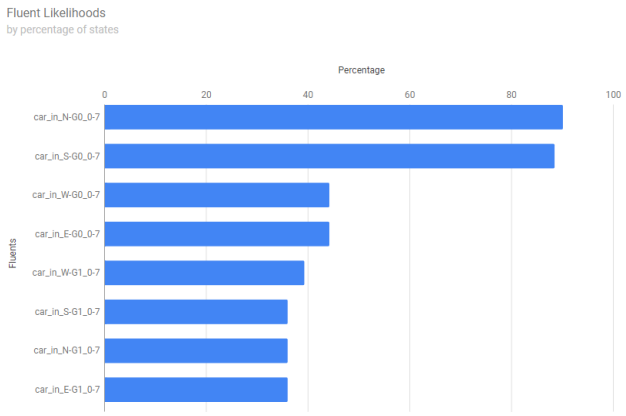


Fig. 7: Environment likelihood of Agent 1 when it performs action NS. I.e., How does the state look like when Agent 1 performs action NS?.

by the tool about the traffic light controller matches our intuition on the decisions we expect the agent must take given that information. Also, it provides a very nice platform for human-in-the-loop interaction with the system and we see its potential to be used for debugging purposes.

Future Work: We identify two interesting and important areas for future work. First, we would like to relax the assumption on policy determinism to allow for non-deterministic policies. This implementation will require weighted model counting which is a studied method within the knowledge compilation community. Second, we would like to have this interpretability tool available for richer settings including continuous time action and state spaces. This will require to have a pre-processing phase in which the discretization of the domain occurs. This task is very complex as it requires an optimization on the thresholds of the state and action variables such that the interpretation of the controller is maximized.

REFERENCES

[1] C. G. Cassandras, "Smart cities as cyber-physical social systems," *Engineering*, vol. 2, no. 2, pp. 156–158, 2016.

[2] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): methodology and large-scale application on downtown toronto," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1140–1150, 2013.

[3] J. Wen, J. Zhao, and P. Jaillet, "Rebalancing shared mobility-on-demand systems: A reinforcement learning approach," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 220–225.

[4] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," *IEEE Access*, vol. 6, pp. 52 138–52 160, 2018.

[5] B. Goodman and S. Flaxman, "European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation"," *AI Magazine*, vol. 38, no. 3, pp. 50–57, 2017.

[6] T. Griggs and D. Wakabayashi, "How a self-driving uber killed a pedestrian in arizona," *NY Times*, Mar 2018. [Online]. Available: https://www.nytimes.com/interactive/2018/03/20/us/self-driving-uber-pedestrian-killed.html

[7] M. K. Hanawal, H. Liu, H. Zhu, and I. C. Paschalidis, "Learning policies for markov decision processes from data," *IEEE Transactions on Automatic Control*, vol. 64, no. 6, pp. 2298–2309, June 2019.

[8] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[9] C. Muise, S. Wollenstein-Betech, S. Booth, J. Shah, and Y. Khazaeni, "Modeling blackbox agent behaviour via knowledge compilation," in *The AAAI 2020 Workshop on Plan, Activity, and Intent Recognition*, 2020.

[10] A. Darwiche and P. Marquis, "A knowledge compilation map," *J. Artif. Intell. Res.*, vol. 17, pp. 229–264, 2002.

[11] A. Vidali, L. Crociani, G. Vizzari, and S. Bandini, "A deep reinforcement learning approach to adaptive traffic lights management," in *Workshop From Objects to Agents (WOA)*, 2019.

[12] L. Prashanth and S. Bhatnagar, "Reinforcement learning with average cost for adaptive control of traffic lights at intersections," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2011, pp. 1640–1645.

[13] S. M. A. Shabestary and B. Abdulhai, "Deep learning vs. discrete reinforcement learning for adaptive traffic signal control," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 286–293.

[14] S. G. Rizzo, G. Vantini, and S. Chawla, "Reinforcement learning with explainability for traffic signal control," in *2019 IEEE Intelligent Transportation Systems Conference*. IEEE, 2019, pp. 3567–3572.

[15] A. Barredo-Arrieta, I. Laña, and J. Del Ser, "What lies beneath: A note on the explainability of black-box machine learning models for road traffic forecasting," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2232–2237.

[16] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in neural information processing systems*, 2017, pp. 4765–4774.

[17] C. Muise, S. A. McIlraith, J. C. Beck, and E. Hsu, "DSHARP: Fast d-DNNF Compilation with sharpSAT," in *Canadian Conference on Artificial Intelligence*, 2012.

[18] A. Darwiche, "New advances in compiling CNF to decomposable negation normal form," in *Proceedings of the 16th European Conference on Artificial Intelligence*, 2004, pp. 318–322.

[19] J. Lagniez and P. Marquis, "An Improved Decision-DNNF Compiler," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 2017, pp. 667–673.

[20] D. I. Robertson, "Transyt: a traffic network study tool," 1969.

[21] J. D. Little, M. D. Kelson, and N. H. Gartner, "Maxband: A versatile program for setting signals on arteries and triangular networks," 1981.

[22] J. L. Fleck, C. G. Cassandras, and Y. Geng, "Adaptive quasi-dynamic traffic light control," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 3, pp. 830–842, 2015.

[23] P. Hunt, D. Robertson, R. Bretherton, and R. Winton, "Scoot-a traffic responsive method of coordinating signals," Tech. Rep., 1981.

[24] M. Abdoos, N. Mozayani, and A. L. Bazzan, "Traffic light control in non-stationary environments based on multi agent q-learning," in *2011 14th International IEEE conference on intelligent transportation systems (ITSC)*. IEEE, 2011, pp. 1580–1585.

[25] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *2018 IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[26] J. Gao, Y. Shen, J. Liu, M. Ito, and N. Shiratori, "Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network," *arXiv preprint arXiv:1705.02755*, 2017.

[27] K.-L. A. Yau, J. Qadir, H. L. Khoo, M. H. Ling, and P. Komisarczuk, "A survey on reinforcement learning models and algorithms for traffic signal control," *ACM Computing Surveys*, vol. 50, pp. 1–38, 2017.

[28] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.