On Learning Sparse Boolean Formulae For Explaining AI Decisions

Susmit Jha*, Vasumathi Raman, Alessandro Pinto,
Tuhin Sahai, and Michael Francis

United Technologies Research Center, Berkeley
jha@csl.sri.com,{pintoa,sahait,francism}@utrc.utc.com

Abstract. In this paper, we consider the problem of learning Boolean formulae from examples obtained by actively querying an oracle that can label these examples as either positive or negative. This problem has received attention in both machine learning as well as formal methods communities, and it has been shown to have exponential worst-case complexity in the general case as well as for many restrictions. In this paper, we focus on learning *sparse* Boolean formulae which depend on only a small (but unknown) subset of the overall vocabulary of atomic propositions. We propose an efficient algorithm to learn these sparse Boolean formulae with a given confidence. This assumption of sparsity is motivated by the problem of mining explanations for decisions made by artificially intelligent (AI) algorithms, where the explanation of individual decisions may depend on a small but unknown subset of all the inputs to the algorithm. We demonstrate the use of our algorithm in automatically generating explanations of these decisions. These explanations will make intelligent systems more understandable and accountable to human users, facilitate easier audits and provide diagnostic information in the case of failure. The proposed approach treats the AI algorithm as a black-box oracle; hence, it is broadly applicable and agnostic to the specific AI algorithm. We illustrate the practical effectiveness of our approach on a diverse set of case studies.

1 Introduction

The rapid integration of robots and other intelligent agents into our industrial and social infrastructure has created an immediate need for establishing trust between these agents and their human users. The long-term acceptance of AI will depend critically on its ability to explain its actions, provide reasoning behind its decisions, and furnish diagnostic information in case of failures. This is particularly true for systems with close human-machine coordination such as self-driving cars, care-giving and surgical robots. Decision-making and planning algorithms central to the operation of these systems currently lack the ability to explain the choices and decisions that they make. It is important that intelligent

* The author is currently at SRI International.

agents become capable of responding to inquiries from human users. For example, when riding in an autonomous taxi, we might expect to query the AI driver using questions similar to those we would ask a human driver, such as "why did we not take the Bay Bridge", and receive a response such as "there is too much traffic on the bridge" or "there is an accident on the ramp leading to the bridge or in the middle lane of the bridge." These explanations are essentially propositional formulae formed by combining the user-observable system and the environment states using Boolean connectives.

Even though the decisions of intelligent agents are the consequence of algorithmic processing of perceived system and environment states [31, 25], the straight-forward approach of reviewing this processing is not practical. First, AI algorithms use internal states and intermediate variables to make decisions which may not be observable or interpretable by a typical user. For example, reviewing decisions made by the A* planning algorithm [21] could reveal that a particular state was never considered in the priority queue. But this is not human-interpretable, because a user may not be familiar with the details of how A* works. Second, the efficiency and effectiveness of many AI algorithms relies on their ability to intelligently search for optimal decisions without deducing information not needed to accomplish the task, but some user inquiries may require information that was not inferred during the original execution of the algorithm. Third, artificial intelligence is often a composition of numerous machine learning and decision-making algorithms, and explicitly modelling each one of these algorithms is not practical. Instead, we need a technique which can treat these algorithms as black-box oracles, and obtain explanations by observing their output on selected inputs. These observations motivate us to formulate the problem of generating explanations as an oracle-guided learning of Boolean formula where the AI algorithm is queried multiple times on carefully selected inputs to generate examples, which in turn are used to learn the explanation.

Given the observable system and environment states, S and E respectively, typical explanations depend on only a small subset of elements in the overall vocabulary $V = S \cup E$, that is, if the set of state variables on which the explanation ϕ depends is denoted by $support(\phi) \subseteq V$, then $|support(\phi)| << |V|$. This support or its exact size is not known a priori. Thus, the explanations are sparse formulae over the vocabulary V. The number of examples needed to learn a Boolean formula is exponential in the size of the vocabulary in the general case [20, 19, 8]. Motivated by the problem of learning explanations, we propose an efficient algorithm that exploits sparsity to efficiently learn *sparse* Boolean formula. Our approach builds on recent advances in oracle-guided inductive formal synthesis [17, 16, 18]. We make the following three contributions:

- We formulate the problem of finding explanations for decision-making AI algorithms as the problem of learning sparse Boolean formulae.
- We present an efficient algorithm to learn sparse Boolean formula where the size of required examples grows logarithmically (in contrast to exponentially in the general case) with the size of the overall vocabulary.
- We illustrate the effectiveness of our approach on a set of case-studies.

2 Motivating Example

We now describe a motivating example to illustrate the problem of providing human-interpretable explanations for the results of an AI algorithm. We consider the A* planning algorithm [21], which enjoys widespread use in path and motion planning due to its optimality and efficiency. Given a description of the state space and transitions between states as a weighted graph where weights are used to encode costs such as distance and time, A* starts from a specific node in the graph and constructs a tree of paths starting from that node, expanding paths in a best-first fashion until one of them reaches the predetermined goal node. At each iteration, A* determines which of its partial paths is most promising and should be expanded. This decision is based on the estimate of the cost-to-go to the goal node. We refer readers to [21] for a detailed description of A*. Typical implementations of A* use a priority queue to perform the repeated selection of intermediate nodes. The algorithm continues until some goal node has the minimum cost value in the queue, or until the queue is empty (in which case no plan exists). Figure 1 depicts the result of running A* on a 50×50 grid, where cells that form part of an obstacle are colored red. The input map (Figure 1 (a)) shows the obstacles and free space. A* is run to find a path from lower right corner to upper left corner. On the output map (Figure 1 (b)), cells on the returned optimal path are colored dark blue. Cells which ever entered A*'s priority queue are colored light cyan, and those that never entered the queue are colored yellow.

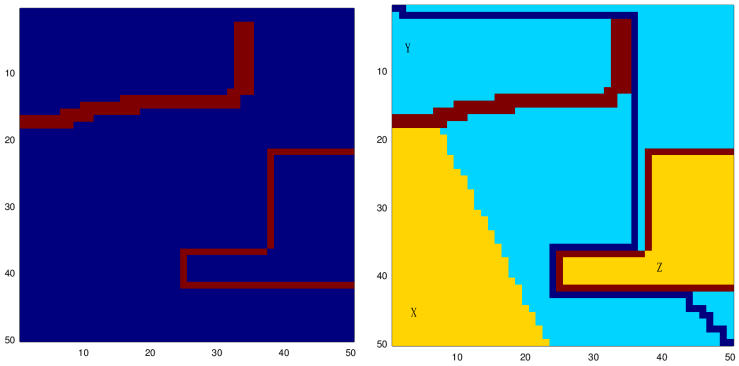

Fig. 1. (a) Input map to A* (b) Output showing final path and internal states of A*

Consider the three cells X,Y,Z marked in the output of A* in Figure 1 (b). An observer might want to enquire why points X, Y or Z were not selected for the optimal path generated by A*. Given the output and logged internal states of the A* algorithm, we know that Y was considered as a candidate cell and discarded due to non-optimal cost whereas X was never even considered as a candidate. But, this is not a useful explanation because a non-expert observing the behavior of a robot cannot be expected to understand the concept of a priority queue, or the details of how A* works. Looking at point Z, we notice that neither X nor

Z was ever inserted into the priority queue; hence, both were never considered as candidate cells on the optimal path. When responding to a user query about why X and Z were not selected in the optimal path, we cannot differentiate between the two even if all internal decisions and states of the A* algorithm were logged. So, we cannot provide the intuitively expected explanation that Z is not reachable due to certain obstacles, while X is reachable but has higher cost than the cells that were considered. This is an example of a scenario where providing explanation requires new information that the AI algorithm might not have deduced while solving the original decision making problem.

3 Problem Definition

The class of AI algorithms used in autonomous systems include path planning algorithms, discrete and continuous control, computer vision and image recognition algorithms. All of these algorithms would be rendered more useful by the ability to explain themselves. Our goal is to eventually develop an approach to generate explanations for the overall system, but we focus on individual components in this paper rather than the overall system. For example, the path planner for a self-driving car takes inputs from machine learning and sensor-fusion algorithms, which in turn receive data from camera, LIDAR and other sensors. The processed sensor data often has semantic meaning attached to it, such as detection of pedestrians on the road, presence of other cars, traffic distribution in a road network, and so on. Given this semantic information, the reason for a particular path being selected by the path planner is often not obvious: this is the sort of explanation we target to generate automatically.

A decision-making AI algorithm `Alg` can be modelled as a function that computes values of output variables `out` given input variables `in`, that is,

$$\texttt{Alg} : \texttt{in} \rightarrow \texttt{out}$$

The outputs are decision variables, while the inputs include environment and system states as observed by the system through the perception pipeline. While the decision and state variables can be continuous and real valued, the inquiries and explanations are framed using predicates over these variables, such as comparison of a variable to some threshold. Let the vocabulary of atomic predicates used in the inquiry from the user and the provided explanation from the system be denoted by $\mathcal{V}$. We can separate the vocabulary $\mathcal{V}$ into two subsets: $\mathcal{V}_Q$ used to formulate the user inquiry and $\mathcal{V}_R$ used to provide explanations.

$$\mathcal{V}_Q = \{q_1, q_2, \ldots q_m\}, \mathcal{V}_R = \{r_1, r_2, \ldots r_n\} \textit{ where } q_i, r_i : \texttt{in} \cup \texttt{out} \rightarrow \texttt{Bool}$$

Intuitively, $\mathcal{V}$ is the shared vocabulary that describes the interface of the AI algorithm and is understood by the human-user. For example, the inquiry vocabulary for a planning agent may include propositions denoting selection of a waypoint in the path, and the explanation vocabulary may include propositions denoting presence of obstacles on a map. An *inquiry* ϕ_Q from the user is an observation about the output (decision) of the algorithm, and can be formulated as a Boolean combination of predicates in the vocabulary $\mathcal{V}_Q$. Hence, we can

denote it as $\phi_Q(\mathcal{V}_Q)$ where the predicates in $\mathcal{V}_Q$ are over the set $\texttt{in} \cup \texttt{out}$, and the corresponding grammar is:

$$\phi_Q := \phi_Q \wedge \phi_Q \mid \phi_Q \vee \phi_Q \mid \neg\phi_Q \mid q_i \texttt{ where } q_i \in \mathcal{V}_Q$$

Similarly, the *response* $\phi_R(\mathcal{V}_R)$ is a Boolean combination of the predicates in the vocabulary $\mathcal{V}_R$ where the predicates in $\mathcal{V}_R$ are over the set $\texttt{in} \cup \texttt{out}$, and the corresponding grammar is:

$$\phi_R := \phi_R \wedge \phi_R \mid \phi_R \vee \phi_R \mid \neg\phi_R \mid r_i \texttt{ where } r_i \in \mathcal{V}_R$$

Definition 1. *Given an AI algorithm* `Alg` *and an inquiry* $\phi_Q(\mathcal{V}_Q)$, $\phi_R(\mathcal{V}_R)$ *is a* necessary and sufficient explanation *when* $\phi_R(\mathcal{V}_R) \iff \phi_Q(\mathcal{V}_Q)$ *where* $\mathcal{V}_R, \mathcal{V}_Q$ *are predicates over* $\texttt{in} \cup \texttt{out}$ *as explained earlier, and* $\texttt{out} = \texttt{Alg}(\texttt{in})$. $\phi_R(\mathcal{V}_R)$ *is a* sufficient explanation *when* $\phi_R(\mathcal{V}_R) \Rightarrow \phi_Q(\mathcal{V}_Q)$.

If the algorithm $\texttt{out} = \texttt{Alg}(\texttt{in})$ could be modelled explicitly in appropriate logic, then the above definition could be used to generate explanations for a given inquiry using techniques such as satisfiability solving. However, such an explicit modelling of these algorithms is currently outside the scope of existing logical deduction frameworks, and is impractical for large and complicated AI systems even from the standpoint of the associated modelling effort. The AI algorithm `Alg` is available as an executable function; hence, it can be used as an oracle that can provide an outputs for any given input. This motivates oracle-guided learning of the explanation from examples using the notion of confidence associated with it.

Definition 2. *Given an AI algorithm* `Alg` *and an inquiry* $\phi_Q(\mathcal{V}_Q)$, $\phi_R(\mathcal{V}_R)$ *is a* necessary and sufficient explanation *with confidence* κ *when* $Pr(\phi_R(\mathcal{V}_R) \iff \phi_Q(\mathcal{V}_Q)) \geq \kappa$ *where* $\mathcal{V}_R, \mathcal{V}_Q$ *are predicates over* $\texttt{in} \cup \texttt{out}$ *as explained earlier,* $\texttt{out} = \texttt{Alg}(\texttt{in})$ *and* $0 \leq \kappa \leq 1$. $\phi_R(\mathcal{V}_R)$ *is a* sufficient explanation *with confidence* κ *when* $Pr(\phi_R(\mathcal{V}_R) \Rightarrow \phi_Q(\mathcal{V}_Q)) \geq \kappa$.

The oracle used to learn the explanation is implemented using the AI algorithm. It runs the AI algorithm on a given input in_i to generate the decision output out_i, and then marks the input as a positive example if $\phi_Q(out_i)$ is true, that is, the inquiry property holds on the output. It marks the input as a negative example if $\phi_Q(out_i)$ is not true. We call this an *introspection oracle*, and it marks each input as either positive or negative.

Definition 3. *An* introspection oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ *for a given algorithm* `Alg` *and inquiry* ϕ_Q *takes an input* in_i *and maps it to a positive or negative label, that is,* $\mathcal{O}_{\phi_Q,\texttt{Alg}} : \texttt{in} \rightarrow \{\oplus, \ominus\}$.

$\mathcal{O}_{\phi_Q,\texttt{Alg}}(in_i) = \oplus$ *if* $\phi_Q(\mathcal{V}_Q(out_i))$ *and* $\mathcal{O}_{\phi_Q,\texttt{Alg}}(in_i) = \ominus$ *if* $\neg\phi_Q(\mathcal{V}_Q(out_i))$, *where* $out_i = \texttt{Alg}(in_i)$, *and* $\mathcal{V}_Q(out_i)$ *is the evaluation of the predicates in* $\mathcal{V}_Q$ *on* out_i

We now formally define the problem of learning Boolean formula with specified confidence κ given an oracle to label examples.

Definition 4. *The problem of oracle-guided learning of Boolean formula from examples is to identify (with confidence κ) the target Boolean function ϕ over a set of atomic propositions $\mathcal{V}$ by querying an oracle $\mathcal{O}$ that labels each input in_i (which is an assignment to all variables in $\mathcal{V}$) as positive or negative $\{\oplus, \ominus\}$ depending on whether $\phi(in_i)$ holds or not, respectively.*

We make the following observations which relates the problem of finding explanations for decisions made by AI algorithms to the problem of learning Boolean formula.

Observation 1 *The problem of generating explanation ϕ_R for the AI algorithm* `Alg` *and an inquiry ϕ_Q is equivalent to the problem of oracle-guided learning of Boolean formula using oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ as described in Definition 4.*

$\phi[r_i]$ denotes the restriction of the Boolean formula ϕ by setting r_i to `true` in ϕ and $\phi[\overline{r_i}]$ denotes the restriction of ϕ by setting r_i to `false`. A predicate r_i is in the support of the Boolean formula ϕ, that is, $r_i \in \texttt{support}(\phi)$ if and only if $\phi[r_i] \neq \phi[\overline{r_i}]$.

Observation 2 *The explanation ϕ_R over a vocabulary of atoms $\mathcal{V}_R$ for the AI algorithm* `Alg` *and a user inquiry ϕ_Q is a sparse Boolean formula, that is, $|\texttt{support}(\phi_R)| << |\mathcal{V}_R|$.*

These observations motivate the following problem definition for learning sparse Boolean formula.

Definition 5. *Boolean function ϕ is called k-sparse if $|\texttt{support}(\phi_R)| \leq k$. The problem of oracle-guided learning of k-sparse Boolean formula from examples is to identify (with confidence κ) the target k-sparse Boolean function ϕ over a set of atomic propositions $\mathcal{V}$ by querying an oracle $\mathcal{O}$ that labels each input in_i (which is an assignment to all variables in $\mathcal{V}$) as positive or negative $\{\oplus, \ominus\}$ depending on whether $\phi(in_i)$ holds or not, respectively.*

Further, the explanation of decisions made by an AI algorithm can be generated by solving the problem of oracle-guided learning of k-sparse Boolean formula. In the following section, we present a novel approach to efficiently solve this problem.

4 Learning Explanations as Sparse Boolean Formula

Our proposed approach to solve the k-sparse Boolean formula learning problem has two steps:

1. In the first step, we find the support of the explanation, that is, $\texttt{support}(\phi_R) \subseteq \mathcal{V}_R$. This is accomplished using a novel approach which requires a small number of runs (logarithmic in $|\mathcal{V}_R|$) of the AI algorithm `Alg`.

2. In the second step, we find the Boolean combination of the atoms in $\mathcal{V}_{\phi_R}$ which forms the explanation ϕ_R. This is accomplished by distinguishing input guided learning of propositional logic formula which we have earlier used for the synthesis of programs [16].

Before delving into details of the above two steps, we introduce additional relevant notations. Recall that the vocabulary of explanation is $\mathcal{V}_R = \{r_1, r_2, \ldots, r_n\}$. Given any two inputs in_1 and in_2, we define the *difference* between them as follows.

$$\texttt{diff}(in_1, in_2) = \{i \mid r_i(in_1) \neq r_i(in_2)\}.$$

Next, we define a *distance* metric d on inputs as the size of the difference set, that is,

$$d(in_1, in_2) = |\texttt{diff}(in_1, in_2)|$$

Intuitively, $d(in_1, in_2)$ is the Hamming distance between the n-length vectors that record the evaluation of the atomic predicates r_i in $\mathcal{V}_R$. We say that two inputs in_1, in_2 are *neighbours* if and only if $d(in_1, in_2) = 1$. We also define a partial order $\preceq$ on inputs as follows:

$$in_1 \preceq in_2 \textit{ iff } r_i(in_1) \Rightarrow r_i(in_2) \textit{ for all } 1 \leq i \leq n$$

Given an input in and a set $J \subseteq \{1, 2, \ldots, n\}$, a *random J-preserving mutation* of in, denoted $\texttt{mutset}(in, J)$, is defined as:

$$\texttt{mutset}(in, J) = \{in' | in' \in \texttt{in} \textit{ and } r_j(in') = r_j(in) \textit{ for all } j \in J\}$$

Finding the support: We begin with two random inputs in_1, in_2 on which the oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ returns different labels, say it returns positive on in_1 and negative on in_2 without loss of generality. Finding such in_1, in_2 can be done by sampling the inputs and querying the oracle until two inputs disagree on the outputs. The more samples we find without getting a pair that disagree on the label, the more likely it is that the Boolean formula being used by the oracle to label inputs is a constant (either `true` or `false`). We later formalize this as a probabilistic confidence. Given the inputs in_1, in_2 , we find $J = \texttt{diff}(in_1, in_2) = \{i_1, i_2, \ldots, i_l\}$ on which the inputs differ with respect to the vocabulary $\mathcal{V}_R = \{r_1, r_2, \ldots, r_n\}$. We partition J into two subsets $J_1 = \{i_1, i_2, \ldots, i_{\lfloor l/2 \rfloor}\}$ and $J_2 = \{i_{\lfloor l/2 \rfloor+1}, i_{\lfloor l/2 \rfloor+2}, \ldots, i_l\}$. The two sets J_1 and J_2 differ in size by at most 1. The set of inputs that are halfway between the two inputs w.r.t the Hamming distance metric d defined earlier is given by the set $\texttt{bisect}(in_1, in_2)$ defined as:

$$\texttt{bisect}(in_1, in_2) = \{in' | \forall j \in J_1\ r_j(in')\ \texttt{iff}\ r_j(in_1), \forall j \in J_2\ r_j(in')\ \texttt{iff}\ r_j(in_2)\}$$

Satisfiability solvers can be used to generate an input in' from $\texttt{bisect}(in_1, in_2)$. The oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ is run on in' to produce the corresponding label. This label will match either the label for the input in_1 or that of the input in_2. We discard the input whose label matches in' to produce the next pair of inputs, that is,

$$\texttt{introspect}(in_1, in_2) = \begin{cases} (in_1, in') & \textit{if } \mathcal{O}_{\phi_Q,\texttt{Alg}}(in') \neq \mathcal{O}_{\phi_Q,\texttt{Alg}}(in_2) \\ (in', in_2) & \textit{if } \mathcal{O}_{\phi_Q,\texttt{Alg}}(in') \neq \mathcal{O}_{\phi_Q,\texttt{Alg}}(in_1) \end{cases}$$
$$\textit{where } in' \in \texttt{bisect}(in_1, in_2)$$

Starting from an initial pair of inputs on which $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ produces different labels, we repeat the above process, considering a new pair of inputs at each iteration until we have two inputs in_1, in_2 that are neighbours, with $\texttt{diff}(in_1, in_2) = \{j\}$. Hence, $r_j \in \mathcal{V}_R$ is in the support of the explanation ϕ_R. We add this to the set of variables $\mathcal{V}_{\phi_R}$. We repeat the above process to find the next variable to add to the support set. For example, consider a 2-sparse Boolean formula $x_1 \vee x_2$ over the vocabulary set x_1, x_2, x_3, x_4, x_5. Given two random samples $(\texttt{T}, \texttt{F}, \texttt{T}, \texttt{F}, \texttt{F})$ and $(\texttt{F}, \texttt{F}, \texttt{F}, \texttt{T}, \texttt{T})$ - the first is labelled positive by oracle $\mathcal{O}$ and the second is negative. The `diff` set is $\{1, 3, 4, 5\}$ and the `bisect` produces a new example $(\texttt{T}, \texttt{F}, \texttt{T}, \texttt{T}, \texttt{T})$ which is labelled positive. So, the next pair is $(\texttt{T}, \texttt{F}, \texttt{T}, \texttt{T}, \texttt{T})$ and $(\texttt{F}, \texttt{F}, \texttt{F}, \texttt{T}, \texttt{T})$. The `bisect` now produces new example $(\texttt{T}, \texttt{F}, \texttt{F}, \texttt{T}, \texttt{T})$ which is labelled positive. Now, the `diff` set is a singleton set $\{1\}$. So, x_1 is in the support set of ϕ_R. This is repeated to find the full support $\{x_1, x_2\}$. The efficiency of the introspection process to obtain each variable is summarized in Lemma 1.

Lemma 1. *The introspective search for each new variable $r_j \in \mathcal{V}_{\phi_R}$ takes at most $O(\ln n)$ queries to $\mathcal{O}_{\phi_Q,\texttt{Alg}}$.*

Proof. The size of the difference set $J = \texttt{diff}(in_1, in_2)$ for any inputs in_1, in_2 is at most n for a vocabulary ϕ_R of size n. The i-th call to `introspect` reduces the size of the difference set as follows: $|J(i)| \leq |J(i-1)|/2 + 1$. Thus, the number of calls to `introspect` before the difference set is singleton and the two inputs are neighbours, obtained by solving the above recurrence equation, is $O(\ln n)$.

This introspective search for variables in the support set $\mathcal{V}_{\phi_R}$ is repeated till we cannot find a pair of inputs in_1, in_2 on which the oracle produces different outputs. We check this condition probabilistically using Lemma 2.

Lemma 2. *If m random samples $in_1, in_2, \ldots, in_m$ from $\texttt{mutset}(in, J)$ produce the same output as input 'in' for the oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ where ϕ_R is k-sparse, then the probability that all mutations $in' \in \texttt{mutset}(in, J)$ produce the same output is at least κ, where $m = 2^k \ln(1/(1-\kappa))$.*

Proof. If all the mutations $in' \in \texttt{mutset}(in, J)$ do not produce the same output, then the probability of the Oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ differing from the output of in for any random sample in' is at least $1/2^k$ since the size of the set $\texttt{mutset}(in, J)$ is at most $s = 2^k$. So,

$$\begin{aligned} &(1 - 1/s)^m \geq 1 - \kappa \quad \Leftarrow e^{(-1/s)m} \geq 1 - \kappa \text{ (since } 1 - x \leq e^{-x}) \\ \Leftrightarrow\ &(-1/s)m \geq \ln(1 - \kappa) \Leftrightarrow m \leq s\ \ln(1/(1-\kappa)) \end{aligned}$$

We can now define $\texttt{sample}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in, J, \kappa)$ that samples $m = 2^k \ln(1/(1-\kappa))$ inputs from the set $\texttt{mutset}(in, J)$ and generates two inputs on which the oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ disagrees and produces different outputs. If it cannot find such a pair of inputs, it returns $\perp$. The overall algorithm for finding the support of the explanations ϕ_R with probability κ is presented in Algorithm 1.1 using the oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$. It is a recursive algorithm which is initially called with a randomly generated input in and an empty set J. Notice that the support of a sufficient

explanation can be found by making the recursive call on only one of the two inputs, that is, $\texttt{getSupport}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in_1, J, \kappa)$ or $\texttt{getSupport}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in_2, J, \kappa)$ instead of both.

Algorithm 1.1 Introspective computation of $\mathcal{V}_{\phi_R}$: $\texttt{getSupport}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in, J, \kappa)$

```
if sample(O_{φ_Q,Alg}, in, J, κ) = ⊥ then
   return {} // The J-restricted Boolean formula is constant with probability κ.
else
   (in_1, in_2) ⇐ sample(O_{φ_Q,Alg}, in, J, κ)
   while |diff(in_1, in_2)| ≠ 1 do
      in_1, in_2 ⇐ introspect(in_1, in_2)
   r_i is the singleton element in diff(in_1, in_2), J ⇐ J ∪ {i}
   return {r_i} ∪ getSupport(O_{φ_Q,Alg}, in_1, J, κ) ∪ getSupport(O_{φ_Q,Alg}, in_2, J, κ)
```

Theorem 1. *The introspective computation of the support set $\mathcal{V}_{\phi_R}$ of variables of the k-sparse Boolean formula ϕ_R defined over the vocabulary of size n using at most $O(2^k \ \ln(n/(1-\kappa)))$ examples.*

Proof. Each variable in $\mathcal{V}_{\phi_R}$ can be found using an introspective search that needs at most $O(\ln n)$ examples according to Lemma 1. So, the `while` loop in Algorithm 1.1 makes at most $O(\ln n)$ queries. In Lemma 2, we showed that the maximum number of examples needed for `sample` is $O(2^k \ \ln(1/(1-\kappa))$. The recursion is repeated at most $O(2^k)$ times. Thus, the overall algorithms needs at most $O(2^{2k} \ (\ln(1/(1-\kappa)) + \ln n))$, that is, $O(2^{2k} \ \ln(n/(1-\kappa)))$ examples.

Learning Boolean formula ϕ_R: Learning a Boolean formula that forms the explanation ϕ_R for the given query ϕ_Q is relatively straight-forward once the variables $\mathcal{V}_{\phi_R}$ which form the support of the Boolean formula have been identified. Efficient techniques have been developed to solve this problem in the context of program synthesis, and we adopt a technique based on the use of distinguishing inputs proposed by us in [16]. The algorithm starts with a single random input in_1. The oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ is queried with the example and it is marked positive or negative depending on the label returned by the oracle. A candidate explanation ϕ_R^c is generated which is consistent with the positive and negative examples seen so far. Then, the algorithm tries to find an alternative consistent explanation ϕ_R^a. If such an alternate explanation ϕ_R^a cannot be found, the algorithm terminates with ϕ_R^c as the final explanation. If ϕ_R^a is found, we find an input which distinguishes ϕ_R^c and ϕ_R^a and query the oracle with this new input in order to mark it as positive or negative. This refutes one of the two explanation formulae R^c and R^a. We keep repeating the process until we converge to a single Boolean formula. Algorithm 1.2 summarizes this learning procedure.

Theorem 2. *The overall algorithm to generate k-sparse explanation ϕ_R for a given query ϕ_Q takes $O(2^k \ \ln(n/(1-\kappa)))$ queries to the oracle, that is, the number of examples needed to learn the Boolean formula grows logarithmically with the size of the vocabulary n.*

Algorithm 1.2 Learning ϕ_R given the vocabulary $\mathcal{V}_{\phi_R}$ and oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$

Randomly sample an input in_0
if $\mathcal{O}_{\phi_Q,\texttt{Alg}}(in_0) = \oplus$ **then**
  $E^+ \Leftarrow E^+ \cup \{in_0\}$
else
  $E^- \Leftarrow E^- \cup \{in_0\}$
ϕ_R^c = Boolean formula consistent with E^+, E^-
while Alternative ϕ_R^a consistent with E^+, E^- exists **do**
  Generate distinguishing input in that satisfies $(\phi_R^c \wedge \neg\phi_R^a) \vee (\phi_R^a \wedge \neg\phi_R^c)$
  if $\mathcal{O}_{\phi_Q,\texttt{Alg}}(in) = \oplus$ **then**
    $E^+ \Leftarrow E^+ \cup \{in\}$
  else
    $E^- \Leftarrow E^- \cup \{in\}$
  ϕ_R^c = Boolean formula consistent with E^+, E^-
return ϕ_R^c

Proof. The first-step to compute the support set $\mathcal{V}_{\phi_R}$ of the explanation ϕ_R takes $O(2^k \ \ln(n/(1-\kappa)))$ queries and after that, the learning of explanation ϕ_R takes $O(2^k)$ queries. So, the total number of queries needed is $O(2^k \ \ln(n/(1-\kappa)))$.

Thus, our algorithm adopts a binary search like procedure using the Hamming distance metric d to find the support of the Boolean formula over a vocabulary of size n using a number of examples that grow logarithmically in n. After the support has been found, learning the Boolean formula can be accomplished using the formal synthesis based approach that depends only on the size of the support set and not on the vocabulary size n. Algorithms that do not exploit sparsity have been previously shown to need examples that grow exponentially in n [20, 19] in contrast to the logarithmic dependence on n of the algorithm proposed here. The proposed algorithm is very effective for sparse Boolean formula, that is, $k << n$, which is often the case with explanations.

5 Experiments

We begin by describing the results on the motivating example of A* presented in Section 2. The vocabulary is $\mathcal{V}_Q = \{\texttt{on}_{ij}$ for each cell i, j in the grid $\}$ where $\texttt{on}_{ij}$ denotes the decision that i, j-th cell was selected to be on the final path, and $\neg\texttt{on}_{ij}$ denotes the decision that the i, j-th cell was not selected to be on the final path. The vocabulary $\mathcal{V}_R = \{\texttt{obst}_{ij}\}$ for each cell i, j in the grid where $\texttt{obst}_{ij}$ denotes that the cell i, j has an obstacle and $\neg\texttt{obst}_{ij}$ denote that the cell i, j is free. The explanation query is: "Why were no points in $25 \leq i \leq 50, j = 40$ (around z) not considered on the generated path?" The inquiry framed using $\mathcal{V}_Q$ is $\bigwedge_{25\leq i\leq 50} \neg(on_{i,40})$. A sufficient explanation for this inquiry is $\texttt{obst}_{42,32} \wedge \texttt{obst}_{37,32}$ with κ set to 0.9. This is obtained in 2 minutes 4 seconds (48 examples). The second query is for the area around x: $\bigwedge_{0\leq i\leq 20} \neg(on_{i,44})$ and the sufficient explanation obtained is $\texttt{obst}_{2,17} \wedge \texttt{obst}_{2,18}$ in 2 minutes 44

seconds (57 examples). The third query for area around y is $\bigwedge_{0 \leq i \leq 5} \neg(on_{i,5})$ and the corresponding explanation is $\texttt{obst}_{4,17} \wedge \texttt{obst}_{4,18}$ which was obtained in 1 minutes 48 seconds (45 examples). Given the 177 obstacles, a naive approach of enumerating all possible explanations would require 1.9×10^{153} runs of A* which is clearly infeasible in each of these three cases. Even if we assumed that the number of explanations is 2 (but did not know which two variables are in the support set), there are more than $15,000$ cases to be considered.

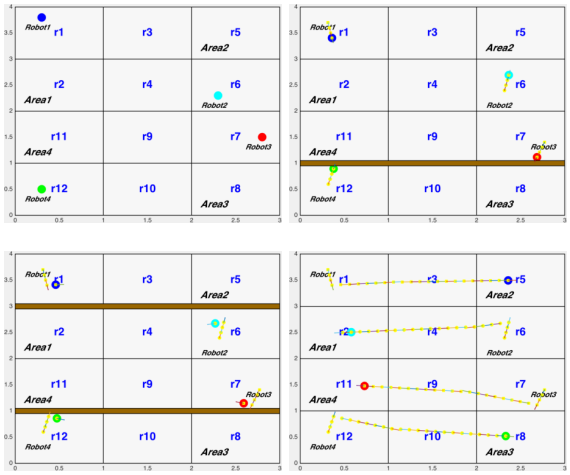

Fig. 2. Execution of reactive strategy for particular sequence of door closings. Each Robot i is initially assigned to goal Area i, but they can swap if needed to achieve the global goal (each marked Area must eventually get one robot). Brown lines indicate closed doors preventing the robots' motion. Time steps depicted are 0, 3, 4 and 24.

Explaining reactive strategy [27] : We also applied our approach to a reactive switching protocol for multi-robot systems generated according to the approach described in [27]. The task involves 4 robots operating in the workspace depicted in Figure 2. In the beginning, each robot is assigned the corresponding area to surveil (i.e. Robot i is assigned to Area i). Starting from their initial positions, they must reach this region. However, in response to the opening and closing of doors in the environment at each time step, they are allowed to swap goals. As can be seen from the Figure 2, robots 1 and 2 swap goals because the top door closes, and robots 3 and 4 swap goals because the bottom door is closed. They stand by these decisions even though the doors later reopen. The simulation takes 24 time steps for all the robots to reach their final goals. The vocabulary is $\mathcal{V}_Q = \{\texttt{final}_{ij}$ for each robot i and area $j\}$, where $\texttt{final}_{ij}$ denotes that robot i ended up in area j. The vocabulary $\mathcal{V}_R = \{\texttt{door}_{\texttt{top},t}, \texttt{door}_{\texttt{bot},t}, \texttt{door}_{\texttt{left},t}, \texttt{door}_{\texttt{right},t}\}$, where $\texttt{door}_{\texttt{top},t}$ denotes that the door between the top and middle row of areas is closed at time t, $\texttt{door}_{\texttt{left},t}$ denotes that the door between the left and middle column of areas is closed at time t, etc. We pose the query,"Why did Robot 1 end up in Area 2?", i.e. $\texttt{final}_{12}$. Starting with the original input sequence and one in which no door-related events occur, the generated

explanation is $\texttt{door}_{\texttt{bot},3}$, which is obtained in 0.76 seconds, and 7 introspective runs of the protocol on mutated inputs (door activity sequences). The second query was, "Why did Robot 3 not end up in Area 3?", or $\neg\texttt{final}_{33}$. This took 0.61 seconds and 6 runs to generate ", $\texttt{door}_{\texttt{top},4}$. Given that there are 4 doors and 24 time steps, a naive approach of enumerating all possible explanations would require $(2^4)^{24} = 7.9 \times 10^{28}$ runs of the reactive protocol.

Explaining classification error in MNIST [22] : MNIST database of scanned images of digits is a common benchmark used in literature to evaluate image classification techniques. MNIST images were obtained by normalization of original images into greyscale 28×28 pixel image. We consider a k-NN classifier for k=9 as the machine learning technique. Some of the test images are incorrectly identified by this technique and we show one of these images in Figure 3 where 4 is misidentified as 9. We deploy our technique to find explanations for this error. The k-NN classifier uses voting among the k-nearest neighbours to label test data. We show the nearest neighbour with label '9' to the misclassified image in the figure below. This image of 4 had 6 neighbours which were labelled '9'. The oracle for generating explanations works as follows: If the number of neighbours of the image labelled '9' decreases from 6 (even if the final label from the k-NN classifier does not change), the oracle marks the image as positive, and negative, otherwise. The vocabulary of explanation is formed by 4×4 pixel blocks (similar to superpixels in [30]) being marked completely dark or clear (this corresponds to predicate abstraction of greyscale pixels). The set of atomic propositions in the support of the explanation is illustrated in the third figure by manually picking assignment values to support variables for purpose of illustration. The last two figures show images which are filtered by two conjunctions in the generated explanation. The generation of the explanation took 3 minutes 48 seconds and required 58 examples where we initialized the algorithm with the images of 4 and 9 in the figure below.

Fig. 3. Left to right: Misclassified image of '4', closest image of '9', changing all pixels corresponding to support of explanations, changing pixels for one of the sufficient explanation, changing pixels for another sufficient explanation

6 Related Work

Our approach relies on learning logical explanations in the form of sparse Boolean formula from examples that are obtained by carefully selected introspective simulations of the decision-making algorithm. The area of active learning Boolean formula from positive and negative examples has been studied in literature [19, 1] in both exact and probably approximately correct (PAC) setting. Exact learning Boolean formula [20, 3] requires a number of examples exponential in the size of the vocabulary. Under the PAC setting, learning is guaranteed to find an approximately correct concept given enough independent samples [2, 24, 26]. It is known that k-clause conjunctive normal form Boolean formula are not PAC learnable with polynomial sample-size, even though monomials and disjunctive normal form representations are PAC learnable [26, 8]. Changing the representation from CNF to DNF form can lead to exponential blow-up. In contrast, we consider only sparse Boolean formula and our goal is to learn the exact Boolean formula with probabilistic confidence, and not its approximation. Efficient learning techniques exist for particular classes of Boolean formulae such as monotonic and read-one formulae [12, 15], but explanations do not always take these restricted forms, and hence, our focus on sparse Boolean formulae is better suited for this context.

Another related research area is the newly emerged field of formal synthesis, which combines induction and deduction for automatic synthesis of systems from logical or black-box oracle specifications [16, 17]. Unlike active learning, formal synthesis is also concerned with defining techniques for the generation of interesting examples and not just its inductive generalization, much like our approach. While existing formal synthesis techniques have considered completion of templates by inferring parameters [4, 29, 33], composition of component Boolean functions or uplifting to bitvector form [16, 13, 36, 7], inferring transducers and finite state-machines [6, 5, 11], and synthesis of invariants [34, 32], our work is the first to consider sparsity as a structural assumption for learning Boolean formulae.

The need for explanations of AI decisions to increase trust of decision-making systems has been noted in the literature [23]. Specific approaches have been introduced to discover explanations in specific domains such as MDPs[9], HTNs[14] and Bayesian networks[37]. Explanation of failure in robotic systems by detecting problems in the temporal logic specification using formal requirement analysis was shown to be practically useful in [28]. Inductive logic programming [10] has also been used to model domain-specific explanation generation rules. In contrast, we propose a domain-independent approach to generate explanations by treating the decision-making AI algorithm as an oracle. Domain-independent approaches have also been proposed in the AI literature for detecting sensitive input components that determine the decision in a classification problem [35, 30]. While these approaches work in a quantitative setting, such as measuring sensitivity from the gradient of a neural network classifier's ouput, our approach is restricted to the discrete, qualitative setting. Further, we not only detect sensitive inputs (support of Boolean formulae) but also generate the explanation.

7 Conclusion and Future Work

We proposed a novel algorithm that uses a binary-search like approach to first find the support of any sparse Boolean formula followed by a formal synthesis approach to learn the target formula from examples. We demonstrate how this method can be used to learn Boolean formulae corresponding to the explanation of decisions made by an AI algorithm. This capability of self-explanation would make AI agents more human-interpretable and decrease the barriers towards their adoption in safety-critical applications of autonomy. We identify two dimensions along which our work can be extended. First, our approach currently uses a predicate abstraction to Boolean variables for learning explanations. We plan to extend our technique to a richer logical language such as signal temporal logic for explanations involving real values. Second, we need to extend our approach to infer multiple valid explanations in response to an inquiry. This work is a first step towards using formal methods, particularly, formal synthesis to aid artificial intelligence by automatically generating explanations of decisions made by AI algorithms.

References

1. A. Abouzied, D. Angluin, C. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *ACM symposium on Principles of database systems*, pages 49–60. ACM, 2013.
2. D. Angluin. Computational learning theory: survey and selected bibliography. In *ACM symposium on Theory of computing*, pages 351–369. ACM, 1992.
3. D. Angluin and M. Kharitonov. When won't membership queries help? In *ACM symposium on Theory of computing*, pages 444–454. ACM, 1991.
4. B. Bittner, M. Bozzano, A. Cimatti, M. Gario, and A. Griggio. Towards pareto-optimal parameter synthesis for monotonie cost functions. In *FMCAD*, pages 23–30, Oct 2014.
5. B. Boigelot and P. Godefroid. Automatic synthesis of specifications from the dynamic observation of reactive programs. In *TACAS*, pages 321–333, 1997.
6. M. Botinčan and D. Babić. Sigma*: Symbolic learning of input-output specifications. In *POPL*, pages 443–456, 2013.
7. B. Cook, D. Kroening, P. Rümmer, and C. M. Wintersteiger. Ranking function synthesis for bit-vector relations. *FMSD*, 43(1):93–120, 2013.
8. A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82(3):247 – 261, 1989.
9. F. Elizalde, E. Sucar, J. Noguez, and A. Reyes. *Generating Explanations Based on Markov Decision Processes*, pages 51–62. 2009.
10. C. Feng and S. Muggleton. Towards inductive generalisation in higher order logic. In *9th International Workshop on Machine learning*, pages 154–162, 2014.
11. P. Godefroid and A. Taly. Automated synthesis of symbolic instruction encodings from i/o samples. *SIGPLAN Not.*, 47(6):441–452, June 2012.
12. J. Goldsmith, R. H. Sloan, B. Szörényi, and G. Turán. Theory revision with queries: Horn, read-once, and parity formulas. *Artificial Intelligence*, 156(2):139–176, 2004.

13. A. Gurfinkel, A. Belov, and J. Marques-Silva. *Synthesizing Safe Bit-Precise Invariants*, pages 93–108. 2014.
14. M. Harbers, J.-J. Meyer, and K. van den Bosch. Explaining simulations through self explaining agents. *Journal of Artificial Societies and Social Simulation*, 2010.
15. L. Hellerstein and R. A. Servedio. On pac learning algorithms for rich boolean function classes. *Theoretical Computer Science*, 384(1):66–76, 2007.
16. S. Jha, S. Gulwani, S. A. Seshia, and A. Tiwari. Oracle-guided component-based program synthesis. In *ICSE*, pages 215–224. IEEE, 2010.
17. S. Jha and S. A. Seshia. A theory of formal synthesis via inductive learning. *Acta Informatica*, pages 1–34, 2017.
18. S. Jha, S. A. Seshia, and A. Tiwari. Synthesis of optimal switching logic for hybrid systems. In *EMSOFT*, pages 107–116. ACM, 2011.
19. M. Kearns, M. Li, and L. Valiant. Learning boolean formulas. *J. ACM*, 41(6):1298–1328, Nov. 1994.
20. M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.
21. S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.
22. Y. Lecun and C. Cortes. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.
23. J. Lee and N. Moray. Trust, control strategies and allocation of function in human-machine systems. *Ergonomics*, 35(10):1243–1270, 1992.
24. Y. Mansour. Learning boolean functions via the fourier transform. In *Theoretical advances in neural computation and learning*, pages 391–424. 1994.
25. D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
26. L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Journal of the ACM (JACM)*, 35(4):965–984, 1988.
27. V. Raman. Reactive switching protocols for multi-robot high-level tasks. In *IEEE/RSJ*, pages 336–341, 2014.
28. V. Raman, C. Lignos, C. Finucane, K. C. T. Lee, M. P. Marcus, and H. Kress-Gazit. Sorry Dave, I'm Afraid I can't do that: Explaining unachievable robot tasks using natural language. In *Robotics: Science and Systems*, 2013.
29. A. Reynolds, M. Deters, V. Kuncak, C. Tinelli, and C. Barrett. *Counterexample-Guided Quantifier Instantiation for Synthesis in SMT*, pages 198–216. 2015.
30. M. T. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?": Explaining the predictions of any classifier. In *KDD*, pages 1135–1144, 2016.
31. J. Russell and R. Cohn. *OODA Loop*. Book on Demand, 2012.
32. S. Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In *HSCC*, pages 221–230, 2010.
33. S. Sankaranarayanan, C. Miller, R. Raghunathan, H. Ravanbakhsh, and G. Fainekos. A model-based approach to synthesizing insulin infusion pump usage parameters for diabetic patients. In *Annual Allerton Conference on Communication, Control, and Computing*, pages 1610–1617, Oct 2012.
34. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constructing invariants for hybrid systems. *FMSD*, 32(1):25–55, 2008.
35. E. Štrumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *KIS*, 41(3):647–665, 2014.
36. C. Urban, A. Gurfinkel, and T. Kahsai. *Synthesizing Ranking Functions from Bits and Pieces*, pages 54–70. 2016.
37. C. Yuan, H. Lim, and T.-C. Lu. Most relevant explanation in bayesian networks. *J. Artif. Intell. Res.(JAIR)*, 42:309–352, 2011.