Journal of Automated Reasoning
https://doi.org/10.1007/s10817-018-9499-8

Explaining AI Decisions Using Efficient Methods for Learning Sparse Boolean Formulae

Susmit Jha[1] · Tuhin Sahai[2] · Vasumathi Raman[2] · Alessandro Pinto[2] · Michael Francis[2]

Received: 11 November 2018 / Accepted: 17 November 2018

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 two algorithms—first, based on binary search in the Hamming space, and the second, based on random walk on the Boolean hypercube, 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 these algorithms 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 show that the number of examples needed for both proposed algorithms only grows logarithmically with the size of the vocabulary of atomic propositions. We illustrate the practical effectiveness of our approach on a diverse set of case studies.

Keywords Explainable AI · Boolean formula learning · Machine learning · Formal methods · Interpretable AI · Sparse learning

✉ Susmit Jha
jha@csl.sri.com

1 SRI International, Menlo Park, USA

2 United Technologies Research Center, Berkeley, USA

Published online: 29 November 2018

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 ability of human-understandable self-explanation is fundamental to establishing such trust. 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, and 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. This is particularly problematic when the results returned by these algorithms are counter-intuitive. In such cases, it is particularly important that intelligent agents become capable of responding to inquiries from human users. The resulting explanations must be human-understandable, and hence must use terms in a pre-defined, user-specified vocabulary. 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 formulae in propositional logic formed by combining the atomic propositions corresponding to the user-observable system and the environment states using Boolean connectives.

Robots rely on a perception pipeline to observe the environment, orient themselves, decide what action to take, and then act accordingly. This repeated sequence of observe-orient-decide-act is also called OODA loop [31,39] in literature on autonomous and intelligent systems. Even though the decisions of intelligent agents are the consequence of algorithmic processing of perceived system and environment states, the straight-forward approach of reviewing this processing is not practical. There are three key reasons for this. 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 [27] 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. For example, a state may never be included in the queue of a heuristic search algorithm like A* because either it is unreachable or it has very high cost. Thus, the ability to explain why this state is not on the computed path will require additional effort. Third, artificial intelligence is often a composition of numerous machine learning and decision-making algorithms, and explicitly modeling 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 $|support(\phi)| << |V|$. Thus, the explanations are sparse formulae

over the vocabulary V. But this support set $support(\phi)$ or its size is not known a priori. The number of examples needed to learn a Boolean formula is exponential in the size of the vocabulary in the general case [12,24,25]. Motivated by the problem of learning explanations, we propose an efficient algorithm that exploits sparsity to efficiently learn *sparse* Boolean formulas.

This assumption of sparseness and simplicity of explanations is motivated by the Occam's Razor principle, and is shared across a number of approaches to interpretable machine learning. Techniques based on prediction decomposition, pioneered by Robnik-Sikonja and Kononenko [38] and extended in [10,46], compare the original prediction and the one made by omitting features. This technique is tractable if the number of features being examined is small. Another class of technique focuses on gradient-based learning methods and classification tasks by identifying the local gradients that change the label of a data point as its explanation [4]. Each entry in this gradient vector indicates the weight of a feature. But this approach is limited to finding only explanations that are conjunctions of features. The extension to discovering arbitrary Boolean explanations relies on enumerating these gradient vectors across a small set of selected feature subspaces. Another line of work [26] aims at learning interpretable decision sets where prediction is made by a set of rules where each rule has a set of predicates. The decision sets are constructed by minimizing the number of rules as well as the set of predicates in each rule. Finally, locally interpretable model-agnostic explanations (LIME) methodology [36] aims at approximating decision boundary in the locality of the data point as explanation of the decision at that data point. It relies on sampling neighborhood of the data point to construct this local decision boundary. It uses a lasso regression with preprocessing to select top k most significant features beforehand, named K-LASSO. Thus, all these techniques exploit the sparseness of the explanations. While our approach also uses this assumption, we propose a novel approach to identify these relevant features from a large vocabulary set.

Our approach builds on recent advances in formal synthesis [20,22,23]. We make the following contributions:

- We formulate the problem of finding explanations for decision-making AI algorithms as the problem of learning sparse Boolean formulae.
- We present two algorithms to learn sparse Boolean formulas where the size of required examples grows logarithmically (in contrast to exponentially in the general case) with the size of the overall vocabulary. We theoretically and empirically compare these algorithms.
  - The first algorithm is based on a binary search in the Hamming space. The procedure uses random sampling to find two assignments to variables such that the inputs corresponding to these assignments generate different outcomes from the oracle running the AI algorithm, and then finds inputs, using binary search, differing on only a single (relevant) variable and producing different outcomes. This process is repeated sequentially to find relevant variables.
  - The second algorithm is based on random walk in the Boolean hypercube. The procedure uses results on mixing time of random walks over graphs to find the the required length of walk such that either the algorithm finds neighboring assignments differing in only a single (relevant) variable and producing different outcomes from the oracle, or we know that all the relevant variables have been found with a given confidence.
- We illustrate the effectiveness of our approach on a set of case-studies.

This paper is an extended version of our work (NASA Formal Methods Symposium, 2017) [21]. In addition to the results in [21], we present a novel algorithm for learning sparse

Boolean formula based on random walks on a Boolean hypercube. This algorithm also has logarithmic dependence on the size of the vocabulary but it requires fewer examples (both, theoretically and empirically) than the approach based on binary search in Hamming space presented in [21].

The rest of the paper is organized as follows. We explain the problem using a motivating example in Sect. 2 and present a formal description in Sect. 3. In Sect. 4, we present the overall approach for learning sparse Boolean formula in two steps: the first step finds the relevant variables that form the support set of the explanation, and the second step synthesizes the Boolean formula with this support set. We describe two algorithms for finding the support set of sparse Boolean formulae—the first uses a binary search in the Hamming space and the second is based on a random walk over Boolean hypercube. We analyze the learning complexity of both algorithms in Sect. 5. The application of our approach to a set of case-studies is presented in Sect. 6. We discuss related work in Sect. 7 and conclude in Sect. 8 by identifying promising research directions.

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 [27], 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. Specifically, A* selects an intermediate node n that minimizes $\texttt{totalCost}(n) = \texttt{partialCost}(n) + \texttt{guessCost}(n)$, where `totalCost` is the estimated total cost of the path that includes node n, obtained as the sum of the cost ($\texttt{partialCost}(n)$) of reaching n from the initial node, and a heuristic estimate of the cost ($\texttt{guessCost}(n)$) of reaching the goal from n. The heuristic function `guessCost` is problem-specific: e.g., when searching for the shortest path on a Manhattan grid with obstacles, a good `guessCost` is the straight line distance from the node n to the final destination. Typical implementations of A* use a priority queue to perform the repeated selection of intermediate nodes. This priority queue is known as the open set or fringe. At each step of the algorithm, the node with the lowest `totalCost` value is removed from the queue, and "expanded", This means that the `partialCost` values of its neighbors are updated accordingly based on whether going through n improves them, and these neighbors are added to the queue. The algorithm continues until some goal node has the minimum cost value, `totalCost`, in the queue, or until the queue is empty (in which case no plan exists). The `totalCost` value of the goal node is then the cost of the optimal path. We refer readers to [27] for a detailed description of A*. In rest of this section, we illustrate the need for providing explanations using a simple example map and application of A* on it to find the shortest path.

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 (Fig. 1a) shows the obstacles and free space. A*

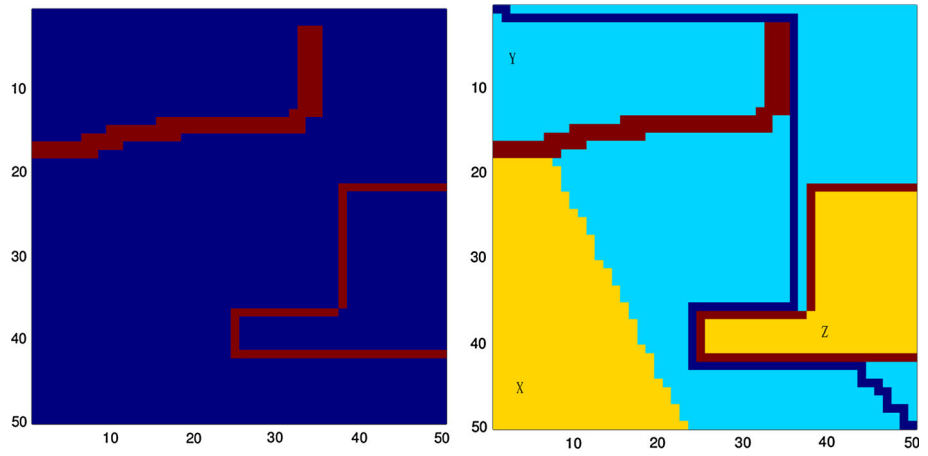

Fig. 1 **a** Input map to A*. **b** Output showing final path and internal states of A*. Cells on the computed optimal path are colored dark blue. Cells which entered A*'s priority queue are colored light cyan, and those cells that never entered the queue are colored yellow

is run to find a path from lower right corner to upper left corner. The output map is shown in Fig. 1b.

Consider the three cells X, Y, Z marked in the output of A* in Fig. 1b and the following inquiries on the optimal path discovered by A*:

- *Why was the cell Y not selected for the optimal path?* Given the output and logged internal states of the A* algorithm, we know that Y was considered as a candidate cell but discarded due to non-optimal cost.
- *Why was the cell X not selected for the optimal path?* If we logged the internal states of the A* algorithm, we would find that X was not even considered as a candidate and it never entered the priority queue of the A* algorithm. But this is not a useful explanation because a non-expert user cannot be expected to understand the concept of a priority queue, or the details of how A* works.
- *Why was the cell Z not selected for the optimal path?* The cell Z was also never inserted into the priority queue and hence, it was never a candidate to be selected on the optimal path similar to cell X. 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 the 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 obstacles, while X is reachable but has higher cost than the cells that were considered.

This example scenario illustrates the need for new information to provide explanation in addition to the usual deduction by AI algorithm 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.

3.1 Defining the Vocabulary for Explanation

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

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

The outputs are the decision variables, while the inputs include the 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. These predicates can either be directly provided by the user or the developer of the AI system, or they can be automatically extracted from the implementation of the AI system by including predicates that appear in the control flow of the AI system. These must be predicates over the input and output variables, that is, `in` and `out`, which are understood by the users. Our approach exploits the sparsity of Boolean formula for learning the explanations and so, the vocabulary can include all possible predicates and variables that might be useful for explaining AI decisions. We propose methods to efficiently find relevant variables where these methods only depend logarithmically on the size of the vocabulary. This ensures that the definition of vocabulary can conveniently include all possible variables, and our approach can automatically find the relevant subset and synthesize the corresponding explanation.

We denote the vocabulary of atomic predicates used in the inquiry from the user and the provided explanation from the system 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.

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

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.

3.2 Explanation Inquiry and Response

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:

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

While conjunction and negation are sufficient to express any Boolean combination, we include disjunction and implication for succinctness of inquiries. 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:

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

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 $\texttt{out} = \texttt{Alg(in)}$. $\phi_R(\mathcal{V}_R)$ is a *sufficient explanation* when $\phi_R(\mathcal{V}_R) \Rightarrow \phi_Q(\mathcal{V}_Q)$.

If the algorithm $\texttt{out} = \texttt{Alg(in)}$ could be modeled 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 modeling 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 modeling 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 probabilistic 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, $\texttt{out} = \texttt{Alg(in)}$ and $0 \leq \kappa \leq 1$. The probability of satisfaction of $\phi_R(\mathcal{V}_R) \iff \phi_Q(\mathcal{V}_Q)$ is computed using uniform distribution over the variables in $\mathcal{V}$. This uniform distribution is not an assumption over the context in which an AI algorithm `Alg` is used. This uniform distribution is only used to estimate the probability of finding the correct explanation. Similarly, $\phi_R(\texttt{in})$ 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 uses 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* which 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, $\mathcal{O}_{\phi_Q,\texttt{Alg}} : \texttt{in} \rightarrow \{\oplus, \ominus\}$.

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

We now formally define the problem of learning Boolean formula with specified confidence κ given an oracle that labels the 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 ϕ_R 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.

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.

4 Learning 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 [20].

Before delving into details of the 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,

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

$d(in_1, in_2)$ 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:

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

A random walk $\texttt{walk}$ over the Boolean hypercube starts with a random initial input in^0. The input at iteration $t+1$ is $in^{t+1} = \texttt{walk}(in^t)$ and in^{t+1} is obtained by randomly sampling an index j from $[1, n]$ with uniform probability and then flipping the variable at index j of in^t with probability 1/2.

$$\begin{aligned} p(\texttt{walk}(in^t) = in \mid in^t) &= \frac{1}{2} \text{ if } in = in^t \\ &= \frac{1}{2n} \text{ if } d(in, in^t) = 1 \end{aligned}$$

4.1 Learning Based on Binary Search in Hamming Space

We begin with two random inputs in_1, in_2 on which the oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ returns different labels, for example, 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 $\texttt{true}$ or $\texttt{false}$).

We can 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 $\perp$. Lemma 1 justifies why the size m of the samples is sufficient to achieve the probabilistic confidence κ.

Lemma 1 *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 (that is, the oracle is a constant function over $\texttt{mutset}(in, J)$) 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,

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

□

If $\texttt{sample}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in, J, \kappa)$ returns $\perp$, we have found the constant function. Otherwise, it returns two inputs in_1, in_2 on which the oracle disagrees. 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,

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

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 $\texttt{diff}(in_1, in_2, S) = \{j\}$. At this point, we can conclude that $r_j \in \mathcal{V}_R$ is in the support of the explanation ϕ_R because changing the assignment of r_j changes the response of the oracle.

We add r_j to the set of variables $\mathcal{V}_{\phi_R}$. We repeat the above process twice to find the next relevant variable to add to the support set by setting r_j to first true and then false. 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. This continues till the $\texttt{sample}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in, J, \kappa)$ returns $\perp$ and we have found all the relevant variables with probabilistic confidence κ.

This overall algorithm for finding the support of the explanations ϕ_R with probability κ is presented in Algorithm 1 using the oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$. It is a recursive algorithm which is initially called with a randomly generated input $in =$ *random assignment to* $\mathcal{V}_{\phi_R}$, and an empty set $J = \emptyset$.

Algorithm 1 Computation of $\mathcal{V}_{\phi_R}$ using binary search in Hamming space: $\texttt{getSupport}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in, J, \kappa)$

if $\texttt{sample}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in, J, \kappa) = \perp$ **then**
  // The J-restricted Boolean formula is constant function with probability κ.
  return {}
else
  $(in_1, in_2) \Leftarrow \texttt{sample}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in, J, \kappa)$
  while $|\texttt{diff}(in_1, in_2)| \neq 1$ **do**
    $in_1, in_2 \Leftarrow \texttt{introspect}(in_1, in_2)$
  r_i is the singleton element in $\texttt{diff}(in_1, in_2)$
  $J \Leftarrow J \cup \{i\}$
  return $\{r_i\} \cup \texttt{getSupport}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in_1, J, \kappa) \cup \texttt{getSupport}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in_2, J, \kappa)$

Finding sufficient explanations After finding a relevant variable, Algorithm 1 searches for the remaining support of the explanation by setting the found variable to be true and then setting it to false (recursing on in_1 and in_2). So, the number of recursive calls is exponential in k—the size of the support set (relevant variables). But it is not exponential in the size of the vocabulary, and we show later in Sect. 5 that the number of examples required by Algorithm 1 depends only logarithmically on the vocabulary size. The size of the support set is much smaller than the vocabulary size of the k-sparse Boolean formula. 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.

Example Let us consider the target 2-sparse Boolean formula to be $x_1 \vee x_2$ over the vocabulary set x_1, x_2, x_3, x_4, x_5. It is a sparse formula which depends on only 2 out of the 5 atomic propositions. We need to learn the formula using oracle $\mathcal{O}$ for the formula. $\mathcal{O}$ is an oracle that can label any input assignment to be positive or negative depending on whether it satisfies the Boolean formula $x_1 \vee x_2$ or not. Given two random samples (T, F, T, F, F) and (F, F, F, T, T)—the first is labeled positive by the oracle $\mathcal{O}$ and the second is negative. The `diff` set is $\{1, 3, 4, 5\}$ and the `bisect` produces a new example (T, F, T, T, T) which is labeled positive. So, the next pair is (T, F, T, T, T) and (F, F, F, T, T). The `bisect` now produces new example (T, F, F, T, T) which is labeled positive. Now, the difference set `diff` between (T, F, F, T, T) labeled positive and (F, F, F, T, T) labeled negative, is a singleton set $\{1\}$. The binary search over the Hamming space of input assignment ends when the difference set `diff` is a singleton set. So, x_1 is in the support set of ϕ_R. Setting x_1 to F, we repeat the above process again. Consider the two initial random inputs (F, T, T, T, T) and (F, F, T, F, F) where the first is labeled positive and the second is labeled negative. The `bisect` function finds (F, T, T, F, F) which is labeled positive, and the new example pair becomes (F, T, T, F, F) (positive) and (F, F, T, F, F) (negative). Again, the difference set is a singleton set $\{2\}$. So, x_2 is also in the support set of ϕ. Setting x_1 to T, we find that the Oracle computes the constant function T and `sample` function returns $\perp$ after sampling enough examples for the given probabilistic confidence κ. Setting x_1 and x_2 also makes the restricted function constant with respect to the other three non-relevant variables. Thus, we have identified $\{x_1, x_2\}$ as the set of relevant variables.

4.2 Learning Based on Random Walk in Boolean Hypercube

The algorithm for finding relevant variables using random walk in Boolean hypercube starts with a random initial input in^0. This input is a vertex in the n-dimensional Boolean hypercube of the $\mathcal{V}_R$ variables. The algorithm performs a random walk from this vertex, proceeding in iteration t from the vertex in^t to the vertex $\texttt{walk}(in^t) = in^{t+1}$. As defined earlier, `walk` probabilistically either chooses to stay at the same vertex, or to move to a vertex which is 1 Hamming distance away where the differing variable in $\mathcal{V}_R$ is uniformly selected from the n variables. This random walk is performed for at most $L(k', \kappa) = 2^{k'}(2k'^2)(1 + \log k') \log(k'/\kappa)$ steps when searching for k' relevant variables. Using results from mixing time, we will later show that random walk of this length must either find two neighboring vertices on which the oracle produces different labels or the oracle computes a constant function with probabilistic confidence κ. When we find neighboring vertices with different labels, we can find the single variable on which these two inputs differ and clearly, this variable is a relevant variable since changing its assignment changes the output of the oracle. This is repeated to find all the relevant variables. The recursive Algorithm 2 is initially called as $\texttt{getSupportRW}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in_0, \{\}, \kappa, k')$ with random initial input in^0 and an empty set as the so far found variables, $J = \{\}$. For learning sparse Boolean formula, we may not know the size of the support set and so, Algorithm 2 is repeated with $k' = 1, 2, \ldots, k$ till we can't find more relevant variables. We analyze the complexity of this algorithm in Sect. 5, and show that the number of examples required to find all the relevant variables is logarithmic in the size of the vocabulary $\mathcal{V}_R$. Lemma 2 establish that $L(k', \kappa) = 2^{k'}(2k'^2)(1+\log k') \log(n/(1-\kappa))$ is a sufficiently long random walk that the function must be constant if all the vertices have the same label.

Lemma 2 *The expected mixing time of the uniform random walk in Algorithm 2 is smaller than $k(1 + \log k)$ where $k = |\mathcal{V}_R|$ is the number of relevant variables, that is, the expected*

Algorithm 2 Computation of $\mathcal{V}_{\phi_R}$ using random walk over Boolean hypercube: $\texttt{getSupportRW}(\mathcal{O}_{\phi_Q,\texttt{Alg}}, in, J, \kappa, k')$

```
t = 0
while t ≤ L(k', κ) do
  in^{t+1} = walk(in^t)
  if O_{φ_Q,Alg}(in^{t+1}) ≠ O_{φ_Q,Alg}(in^t) then
    r_i is the singleton element in diff(in^{t+1}, in^t)
    J ⇐ J ∪ {i}
    return getSupportRW(O_{φ_Q,Alg}, in, J, κ, k')
  t = t + 1
return J
```

number of steps starting from any vertex in the hypercube, after which sampling is identical to uniform sampling from all the 2^n possible assignments is less than $k(1+\log k)$. This is a formulation of the well known "coupon collector's problem" [7].

Proof We define a new random variable C_j for $1 \leq j \leq k$ that counts the number of steps since $j-1$ unique relevant variables have been found until the j-th relevant variable is identified. Given the probabilistic definition of the walk, C_j is a geometric random variable with success probability $(k-j+1)/k$, and consequently the mixing time is

$$E\left[\sum_{j=1}^{k} C_j\right] = \sum_{j=1}^{k} E[C_j] = \sum_{j=1}^{k} \frac{k}{k-j+1} = k\sum_{j=1}^{k} \frac{1}{j} = kH_k < k(1+\log k)$$

where E denotes the expected value for the random walk and H_k is the partial harmonic sum. The final inequality results from the upper bound on the partial harmonic sum. □

If we consider random walk of length $m' = 2k(1+\log k)$, then the probability that the m-th vertex is uniformly sampled is

$$1 - P\left(\sum_{j=1}^{k} C_j \geq m'\right) \geq 1 - \frac{E\left[\sum_{j=1}^{k} C_j\right]}{m'} > 1 - \frac{E\left[\sum_{j=1}^{k} C_j\right]}{2E\left[\sum_{j=1}^{k} C_j\right]} = \frac{1}{2}$$

If the vertices are uniformly randomly sampled, then the probability of finding neighboring vertices which have different labels and hence, differ in a relevant variable is the probability of picking one of the k relevant variables and picking one of the two vertices of the 2^k possible assignments which have different labels and differ in the assignment of the relevant variable. Each relevant variable has at least two such vertices. In general, a relevant variable with higher sensitivity would have more than 2 such vertices. Thus, the probability of finding a relevant variable under uniform sampling is $\frac{1}{2^k} \cdot \frac{1}{k}$.

Let us consider m'' length random walk after reaching uniform sampling threshold, then the probability of not discovering any of the relevant variable from the vocabulary of size n is $n(1-\frac{1}{k.2^k})^{m''}$. For a given probabilistic confidence κ, we require $n(1-\frac{1}{k.2^k})^{m''} \leq 1-\kappa$, and using $1-x \leq e^{-x}$, we get that $m'' \leq k2^k \log \frac{n}{1-\kappa}$.

Thus, after $L(k,\kappa) = m'm''$ length random walk, the probability of finding a relevant variable is at least κ.

$$L(k,\kappa) = m'm'' = 2k(1+\log k) \times k2^k \log \frac{n}{1-\kappa} = 2^k 2k^2(1+\log k)\log(n/(1-\kappa))$$

Algorithm 2 is repeated for $k' = 1, 2, \ldots, k$ because we don't know the number (k) of relevant variables apriori. We analyze the overall example complexity of the algorithm in Sect. 5.

Algorithm 3 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

4.3 Learning Boolean Formula ϕ_R with Given Support

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 [20]. This algorithm starts with a single random input in_1. The oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ is queried with the example in_1 and the oracle either labels it to be positive or negative. 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 search for an input which distinguishes ϕ_R^c and ϕ_R^a, and then query the oracle with this new *distinguishing* input. The oracle labels this new input as positive or negative. This label is guaranteed to be inconsistent with one of the explanations R^c or R^a since they differ on the label assigned to this input. Thus, the new label provided by the oracle to this *distinguishing* input refutes one of the two explanation formulae R^c or R^a. This procedure of finding two consistent explanations followed by search for a distinguishing input is repeating until we converge to a single Boolean formula which is the uniquely consistent explanation. In the worst-case, all 2^k inputs must be generated but usually a small set of inputs is sufficient. Algorithm 3 summarizes this learning procedure.

4.4 Impact of Incomplete Vocabulary

If the vocabulary is complete, that is, an explanation can be constructed using the variables in the vocabulary, then the presented approach finds the correct support set with the given probabilistic confidence and synthesizes the Boolean formula using this support set. But if the

vocabulary is not complete and we are missing variables that are critical to the explanation, it results in one of the following scenarios.

- Algorithm 3 finds a set of labeled examples E^+ and E^- such that there is no Boolean formula with the given support set that is consistent with E^+ and E^-. These examples serve as an evidence that the vocabulary is insufficient to generate the explanation.
- Algorithm 3 finds a set of labeled examples E^+ and E^- such that there is a unique Boolean formula consistent with these examples but not the correct explanation. In this case, our approach would produce this incorrect explanation. We note that stumbling on such a set of labeled examples which admits a unique Boolean formula by chance, is very unlikely in practice.

Thus, the probabilistic soundness and completeness of our approach in finding the correct explanation requires the vocabulary to be complete. Since our approach scales logarithmically in the size of the vocabulary, this issue can be alleviated in practice by including all potential variables into the vocabulary that might be useful in generating the explanation.

5 Complexity Analysis

5.1 Complexity of Binary Search in Hamming Space

The efficiency of the introspection process to obtain each variable is summarized in Lemma 3.

Lemma 3 *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: $|J(i)| \leq |J(i-1)|/2 + 1$. 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 $O(\ln n)$. □

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 3. So, the `while` loop in Algorithm 1 makes at most $O(\ln n)$ queries. In Lemma 1, 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. □

5.2 Complexity of Random Walk on Boolean Hypercube

Theorem 2 *Finding all the relevant variables using random walk on the Boolean hypercube requires at most $2^{k+1}2k^2(1+\log k)\log(n/(1-\kappa))$ examples.*

Proof Finding k' relevant variable requires at most $L(k', \kappa) = 2^{k'}2{k'}^2(1+\log k')\log(n/(1-\kappa))$ examples. We repeat Algorithm 2 for $k' = 1, 2, \ldots, k$ because we don't know the number

(k) of relevant variables apriori. Thus, the total number of examples is $\sum_{k'=1}^{k} 2^{k'} 2k'^2(1 + \log k') \log(n/(1-\kappa)) \leq 2^{k+1} 2k^2(1 + \log k) \log(n/(1-\kappa))$. □

Observation 3 *The number of examples required by the random walk approach to find all the relevant variables is smaller than the number of examples required by the binary search in Hamming space. It is smaller by a factor of* $2^k / poly(k)$ *where* $poly(k)$ *denotes a polynomial in* k.

5.3 Complexity of Learning Sparse Boolean Formula

Theorem 3 *The overall algorithm to generate* k*-sparse explanation* ϕ_R *for a given query* ϕ_Q *takes* $O(2^{2k} \ln(n/(1-\kappa)))$ *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.

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

Thus, our algorithm finds 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 [24,25] 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.

6 Experiments

In this section, we describe the result of empirical evaluation of the proposed approaches (Table 1).

6.1 Explaining A* Decisions

We begin by describing the results on the motivating example of A* presented in Sect. 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

Table 1 Parameters used in different case studies

Case Study	$\lvert V\rvert$	$\lvert V_Q\rvert$	$\lvert V_R\rvert$	Number of possible explanations	Confidence κ
A* algorithm	2677	2500	177	1.9×10^{53}	0.9
Reactive Strategy	174	96	78	7.9×10^{28}	0.9
MNIST classification	586	10	576	2.5×10^{173}	[0.8–0.99]

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 min 4 s (48 examples) using binary search and 2 min 56 s (65 examples) using random walk. 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 min 44 s (57 examples) using binary search in Hamming space and 2 min 18 seconds (38 examples) using random walk. 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 min 48 s (45 examples) using binary search in Hamming space and 1 min 15 s (37 examples) using random walk. Given the 177 obstacles, a naive approach of enumerating all possible explanations would require 1.9×10^{53} 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.

6.2 Explaining Reactive Strategy [33]

We also applied our approach to a reactive switching protocol for multi-robot systems generated according to the approach described in [33]. We only applied the binary search based method due to the simplicity of explanation. 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 Fig. 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 s, 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 s 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.

6.3 Explaining Classification Error in MNIST [28]

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 Fig. 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

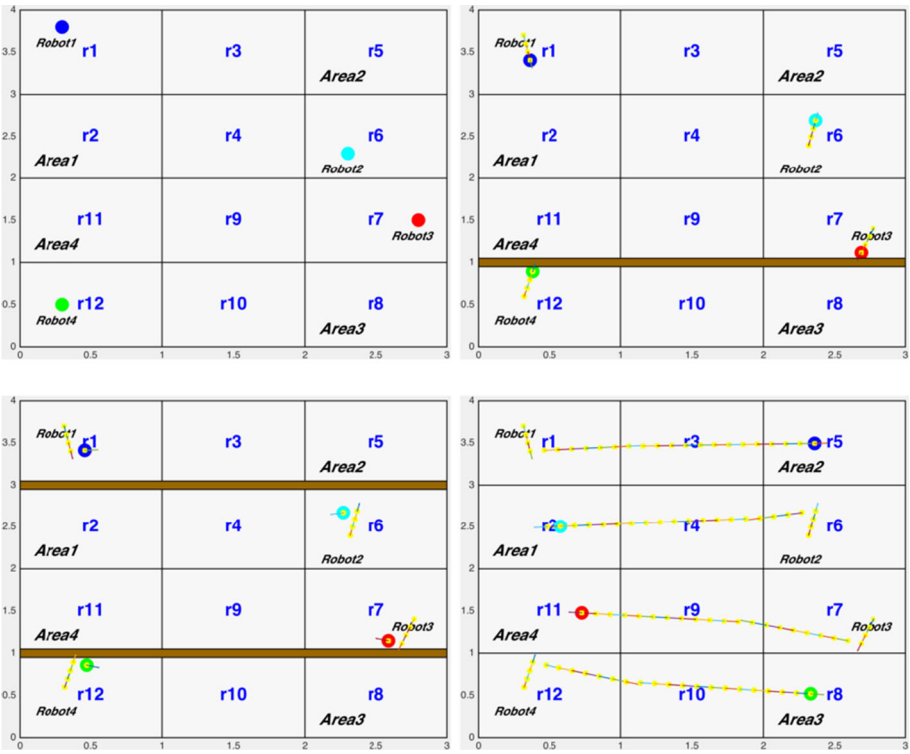

Fig. 2 Execution of reactive strategy for particular sequence of door closings. Each Robot i is initially assigned to goal Area i, but they can swap if needed to achieve the global goal (each marked area must eventually get one robot). Brown lines indicate closed doors preventing the robots' motion.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

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

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 [37]) 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. We initialized the algorithm with the images of 4 and 9 in the figure below.

Table 2 Change in runtime with varying confidence κ

Confidence κ	0.80	0.85	0.90	0.95	0.99
Runtime (s) [Hamming]	112	156	228	308	510
Runtime (s) [Random walk]	46	61	78	212	485

We repeated the experiment with different values of confidence level κ and found our approach is suitable for getting confidence as high as 0.99. Our approach can be easily parallelized since each sample can be independently queried from the AI algorithm or machine learning model. For simplicity, we sequentially loop over the samples currently (Table 2).

7 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 the literature [1,24] in both exact and probably approximately correct (PAC) setting. Exact learning Boolean formula [3,25] 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 [2,30,32]. 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 [12,32]. Changing the representation from CNF to DNF form can lead to exponential blow-up. In contrast, we consider only sparse Boolean formulas 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 [16,19], 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 [20,22]. 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 [5,35,41], composition of component Boolean functions or uplifting to bitvector form [9,17,20,44], inferring transducers and finite state-machines [6,8,15], and synthesis of invariants [40,42], 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 [29]. Specific approaches have been introduced to discover explanations in specific domains such as MDPs[13], HTNs[18] and Bayesian networks[45]. 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 [34]. Inductive logic programming [14] 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 [37,43]. 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.

8 Conclusion and Future Work

We proposed a novel algorithm to first find the support of any sparse Boolean formula using two alternative methods, 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 is crucial for overcoming barriers to the adoption of AI in safety-critical applications of autonomy. We identify two dimensions along which our work can be extended. First, the approach needs to be generalized to non-deterministic AI systems by learning Boolean formula from a noisy oracle. Second, we cannot yet infer multiple valid explanations in response to an inquiry. Further, we are working on combining this model-agnostic method for generating explanations by interrogating the model with white-box methods for analyzing neural networks [11]. 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.

Acknowledgements The authors acknowledge support from the National Science Foundation(NSF) Cyber-Physical Systems #1740079 Project, NSF Software & Hardware Foundation #1750009 Project, and US ARL Cooperative Agreement W911NF-17-2-0196 on Internet of Battle Things (IoBT).

References

1. Abouzied, A., Angluin, D., Papadimitriou, C., Hellerstein, J.M., Silberschatz, A.: Learning and verifying quantified boolean queries by example. In: ACM Symposium on Principles of Database Systems, pp. 49–60. ACM (2013)
2. Angluin, D.: Computational learning theory: survey and selected bibliography. In: ACM Symposium on Theory of Computing, pp. 351–369. ACM (1992)
3. Angluin, D., Kharitonov, M.: When won't membership queries help? In: ACM Symposium on Theory of Computing, pp. 444–454. ACM (1991)
4. Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., Ažller, K.-R.M.: How to explain individual classification decisions. J. Mach. Learn. Res. **11**(Jun), 1803–1831 (2010)
5. Bittner, B., Bozzano, M., Cimatti, A., Gario, M., Griggio, A.: Towards pareto-optimal parameter synthesis for monotonie cost functions. In: FMCAD, pp. 23–30 (2014)
6. Boigelot, B., Godefroid, P.: Automatic synthesis of specifications from the dynamic observation of reactive programs. In: TACAS, pp. 321–333 (1997)
7. Boneh, A., Hofri, M.: The coupon-collector problem revisiteda survey of engineering problems and computational methods. Stoch. Models **13**(1), 39–66 (1997)
8. Botinčan, M., Babić, D.: Sigma*: symbolic learning of input-output specifications. In: POPL, pp. 443–456 (2013)
9. Cook, B., Kroening, D., Rümmer, P., Wintersteiger, C.M.: Ranking function synthesis for bit-vector relations. FMSD **43**(1), 93–120 (2013)
10. de Fortuny, E.J., Martens, D.: Active learning-based pedagogical rule extraction. IEEE Trans. Neural Netw. Learn. Syst. **26**(11), 2664–2677 (2015)
11. Dutta, S., Jha, S., Sanakaranarayanan, S., Tiwari, A.: Output range analysis for deep neural networks. arXiv preprint, arXiv:1709.09130 (2017)
12. Ehrenfeucht, A., Haussler, D., Kearns, M., Valiant, L.: A general lower bound on the number of examples needed for learning. Inf. Comput. **82**(3), 247–261 (1989)

13. Elizalde, F., Sucar, E., Noguez, J., Reyes, A.: Generating Explanations Based on Markov Decision Processes, pp. 51–62 (2009)
14. Feng, C., Muggleton, S.: Towards inductive generalisation in higher order logic. In: 9th International Workshop on Machine learning, pp. 154–162 (2014)
15. Godefroid, P., Taly, A.: Automated synthesis of symbolic instruction encodings from i/o samples. SIGPLAN Not. **47**(6), 441–452 (2012)
16. Goldsmith, J., Sloan, R.H., Szörényi, B., Turán, G.: Theory revision with queries: horn, read-once, and parity formulas. Artif. Intell. **156**(2), 139–176 (2004)
17. Gurfinkel, A., Belov, A., Marques-Silva, J.: Synthesizing Safe Bit-Precise Invariants, pp. 93–108 (2014)
18. Harbers, M., Meyer, J.-J., van den Bosch, K.: Explaining simulations through self explaining agents. J. Artif. Soc. Soc. Simul. **12**, 6 (2010)
19. Hellerstein, L., Servedio, R.A.: On pac learning algorithms for rich boolean function classes. Theor. Comput. Sci. **384**(1), 66–76 (2007)
20. Jha, S., Gulwani, S., Seshia, S.A., Tiwari, A.: In: Oracle-guided component-based program synthesis. In: ICSE, pp. 215–224. IEEE (2010)
21. Jha, S., Raman, V., Pinto, A., Sahai, T., Francis, M.: On learning sparse boolean formulae for explaining AI decisions. In: NASA Formal Methods—9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16–18, 2017, Proceedings, pp. 99–114 (2017)
22. Jha, S., Seshia, S.A.: A theory of formal synthesis via inductive learning. In: Acta Informatica, Special Issue on Synthesis (2016)
23. Jha, S., Seshia, S.A., Tiwari, A.: Synthesis of optimal switching logic for hybrid systems. In: EMSOFT, pp. 107–116. ACM (2011)
24. Kearns, M., Li, M., Valiant, L.: Learning boolean formulas. J. ACM **41**(6), 1298–1328 (1994)
25. Kearns, M., Valiant, L.: Cryptographic limitations on learning boolean formulae and finite automata. J. ACM **41**(1), 67–95 (1994)
26. Lakkaraju, H., Bach, S.H., Leskovec, J.: Interpretable decision sets: a joint framework for description and prediction. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 1675–1684. ACM (2016)
27. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
28. Lecun, Y., Cortes, C.: The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/
29. Lee, J., Moray, N.: Trust, control strategies and allocation of function in human–machine systems. Ergonomics **35**(10), 1243–1270 (1992)
30. Mansour, Y.: Learning boolean functions via the Fourier transform. In: Theoretical Advances in Neural Computation and Learning, pp. 391–424 (1994)
31. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco (2004)
32. Pitt, L., Valiant, L.G.: Computational limitations on learning from examples. J. ACM **35**(4), 965–984 (1988)
33. Raman, V.: Reactive switching protocols for multi-robot high-level tasks. In: IEEE/RSJ, pp. 336–341 (2014)
34. Raman, V., Lignos, C., Finucane, C., Lee, K.C.T., Marcus, M.P., Kress-Gazit, H.: Sorry Dave, I'm Afraid I can't do that: Explaining unachievable robot tasks using natural language. In: Robotics: Science and Systems (2013)
35. Reynolds, A., Deters, M., Kuncak, V., Tinelli, C., Barrett, C.: Counterexample-Guided Quantifier Instantiation for Synthesis in SMT, pp. 198–216 (2015)
36. Ribeiro, M.T., Singh, S., Guestrin, C.: Why Should I Trust You?: explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144. ACM (2016)
37. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why Should I Trust You?": explaining the predictions of any classifier. In: KDD, pp. 1135–1144 (2016)
38. Robnik-Šikonja, M., Kononenko, I.: Explaining classifications for individual instances. IEEE Trans. Knowl. Data Eng. **20**(5), 589–600 (2008)
39. Russell, J., Cohn, R.: OODA loop. In: Book on Demand (2012)
40. Sankaranarayanan, S.: Automatic invariant generation for hybrid systems using ideal fixed points. In: HSCC, pp. 221–230 (2010)
41. Sankaranarayanan, S., Miller, C., Raghunathan, R., Ravanbakhsh, H., Fainekos, G.: A model-based approach to synthesizing insulin infusion pump usage parameters for diabetic patients. In: Annual Allerton Conference on Communication, Control, and Computing, pp. 1610–1617 (2012)
42. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constructing invariants for hybrid systems. FMSD **32**(1), 25–55 (2008)

43. Štrumbelj, E., Kononenko, I.: Explaining prediction models and individual predictions with feature contributions. KIS **41**(3), 647–665 (2014)
44. Urban, C., Gurfinkel, A., Kahsai, T.: Synthesizing Ranking Functions from Bits and Pieces, pp. 54–70 (2016)
45. Yuan, C., Lim, H., Lu, T.-C.: Most relevant explanation in bayesian networks. J. Artif. Intell. Res. (JAIR) **42**, 309–352 (2011)
46. Zintgraf, L.M., Cohen, T.S., Adel, T., Welling, M.: Visualizing deep neural network decisions: prediction difference analysis. arXiv preprint arXiv:1702.04595 (2017)