FastLAS: Scalable Inductive Logic Programming incorporating Domain-specific Optimisation Criteria

Mark Law
Imperial College London, UK
mark.law09@imperial.ac.uk

Alessandra Russo
Imperial College London, UK
a.russo@imperial.ac.uk

Elisa Bertino
Purdue University, USA
bertino@purdue.edu

Krysia Broda
Imperial College London, UK
k.broda@imperial.ac.uk

Jorge Lobo
ICREA - Universitat Pompeu Fabra
jorge.lobo@upf.edu

Abstract

Inductive Logic Programming (ILP) systems aim to find a set of logical rules, called a hypothesis, that explain a set of examples. In cases where many such hypotheses exist, ILP systems often bias towards shorter solutions, leading to highly general rules being learned. In some application domains like security and access control policies, this bias may not be desirable, as when data is sparse more specific rules that guarantee tighter security should be preferred. This paper presents a new general notion of a *scoring function* over hypotheses that allows a user to express domain-specific optimisation criteria. This is incorporated into a new ILP system, called *FastLAS*, that takes as input a learning task and a customised scoring function, and computes an optimal solution with respect to the given scoring function. We evaluate the accuracy of FastLAS over real-world datasets for access control policies and show that varying the scoring function allows a user to target domain-specific performance metrics. We also compare FastLAS to state-of-the-art ILP systems, using the standard ILP bias for shorter solutions, and demonstrate that FastLAS is significantly faster and more scalable.

Introduction

Inductive Logic Programming (ILP) (Muggleton 1991) systems aim to find a set of logical rules, called a hypothesis, that, together with some existing background knowledge, explain a set of examples. Often, many alternative hypotheses can explain the examples, and most systems employ a bias towards shorter solutions, based on Occam's razor (the solution with the fewest assumptions is the most likely).

Choosing the shortest hypothesis often leads to very general hypotheses being learned from relatively few examples. While this can be a huge advantage of ILP over other machine learning approaches that need larger quantities of data, learning such general rules without sufficient quantities of data to justify them may not be desirable in every application domain. For example, in access control, wrongly allowing access to a resource may be far more dangerous than wrongly denying access. So, learning a more general hypothesis, representing a more permissive policy, would be more dangerous than a specific hypothesis, representing a more conservative policy. Equally, for access control where the need for resources is time critical, wrongly denying access could be more dangerous than wrongly allowing access. When learning such policies, and choosing between alternative hypotheses, it would be useful to specify whether the search should be biased towards more or less general hypotheses.

In this paper, we introduce a new ILP system, called *FastLAS*, for learning Answer Set Programs (ASP) (Gelfond and Lifschitz 1988; Brewka, Eiter, and Truszczyński 2011), targeted at solving a restricted version of the context-dependent learning from answer sets tasks defined in (Law, Russo, and Broda 2016), in which tasks require only observational predicate learning. The FastLAS system has two main advantages. Firstly, it takes as input a learning task and a (domain-specific) *scoring function* for hypotheses. The idea is that if there are alternative hypotheses that explain the examples, the hypothesis with the lowest score is preferred. This generalises the standard ILP approach, where hypotheses with the lowest number of literals are normally assumed to be preferred. Secondly, it is specifically designed to be scalable with respect to the hypothesis space – the set of all rules which can appear in a hypothesis. The FastLAS algorithm uses the novel approach of computing a smaller subset of the hypothesis space, called an *OPT-sufficient subset*, that is guaranteed to contain at least one optimal solution with respect to the given scoring function. This smaller search space can be orders of magnitude smaller than the full hypothesis space of a given learning task. We have shown that FastLAS is guaranteed to return an optimal solution w.r.t. a given scoring function.

We evaluated the effect of domain-specific scoring functions by applying FastLAS to real-world datasets for access control policies. Results show that domain-specific scoring functions lead to higher accuracy than the standard ILP bias. We evaluated the scalability of FastLAS against existing state-of-the-art ASP-based ILP systems on real-world datasets, showing that FastLAS is significantly faster than these systems. FastLAS achieves the same if not higher accuracy by exploring, for the same dataset, a much larger hypothesis space.

The next section reviews the necessary background and notation for the paper. In the next sections, (domain-specific) scoring functions and the FastLAS algorithm are presented, followed by an evaluation of the approach. The paper concludes with discussions of related and future work.

Background

This section introduces basic notions used throughout the paper. Given any (first-order logic) atoms $\mathtt{h}$, $\mathtt{b_1}, \ldots, \mathtt{b_n}$, $\mathtt{c_1}, \ldots, \mathtt{c_m}$, a *normal rule* is $\mathtt{h\text{:-}b_1, \ldots, b_n,}$ $\mathtt{not}\ c_1, \ldots,$ $\mathtt{not}\ c_m$, where $\mathtt{h}$ is the *head*, $\mathtt{b_1}, \ldots, \mathtt{b_n}$, $\mathtt{not\ c_1}, \ldots,$ $\mathtt{not\ c_m}$ (collectively) is the *body* of the rule, and "not" represents negation as failure. Rules of the form $\mathtt{\text{:-}b_1, \ldots, b_n,\ not\ c_1, \ldots,\ not\ c_m}$ are called *constraints*. The head (resp. body) of a rule R is denoted $head(R)$ (resp. $body(R)$) and $body^+(R)$ and $body^-(R)$ denote the positive and negative body literals in R, respectively. A rule R is said to be a *subrule* of a rule R' if and only if $head(R) = head(R')$ and $body(R) \subseteq body(R')$ (we call R a *strict-subrule* if $R \neq R'$). In this paper, unless stated otherwise, we assume an ASP program to be a set of normal rules and constraints. The Herbrand Base of a program P, denoted HB_P, is the set of variable free (ground) atoms that can be formed from predicates and constants in P. The subsets of HB_P are called the (Herbrand) interpretations of P. Given a program P and an interpretation $I \subseteq HB_P$, the *reduct* P^I is constructed from the grounding of P in 3 steps: firstly, remove rules whose bodies contain the negation of an atom in I; secondly, remove all negative literals from the remaining rules; and finally, replace the head of any constraint with $\bot$ (where $\bot \notin HB_P$). Any $I \subseteq HB_P$ is an *answer set* of P iff it is the minimal model of P^I. The set of answer sets of a program P is denoted $AS(P)$. *Programs* is the set of all ASP programs and *Rules* is the set of all ASP rules. For an introduction to ASP, please see (Gelfond and Kahl 2014).

We now present the form of examples used in this paper, which were first formalised in (Law, Russo, and Broda 2018b). A *partial interpretation* e is a pair of sets of ground atoms $\langle e^{inc}, e^{exc}\rangle$; we refer to e^{inc} and e^{exc} as the *inclusions* and *exclusions* respectively. An interpretation I is said to *extend* e iff $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$. A *weighted context-dependent partial interpretation* (WCDPI) is a tuple $e = \langle e_{id}, e_{pen}, e_{pi}, e_{ctx}\rangle$, where e_{id} is an identifier for e, e_{pen} is either a positive integer or ∞, called a penalty, e_{pi} is a partial interpretation and e_{ctx} is an ASP program called a *context*. A WCDPI e is *accepted* by a program P if and only if there is an answer set of $P \cup e_{ctx}$ that extends e_{pi}.

Hypothesis Spaces

Many ILP systems (e.g. (Muggleton 1995; Ray 2009; Srinivasan 2001; Kazmi, Schüller, and Saygın 2017)) use mode declarations as a form of language bias to specify hypothesis spaces. In this paper, we follow a similar approach. A *mode bias* is defined as a pair of sets of mode declarations $\langle M_h, M_b\rangle$, where M_h (resp. M_b) are called the *head* (resp. *body*) *mode declarations*. Each mode declaration is a literal whose abstracted arguments are either $\mathtt{var(t)}$ or $\mathtt{const(t)}$, for some constant $\mathtt{t}$ (called a *type*). Informally, a literal is *compatible* with a mode declaration m if it can be constructed by replacing every instance of $\mathtt{var(t)}$ in m with a variable of type $\mathtt{t}$, and every $\mathtt{const(t)}$ with a constant of type $\mathtt{t}$.[1]

Definition 1. *Given a mode bias $M = \langle M_h, M_b\rangle$, a normal rule R is in the search space S_M if and only if (i) the head of R is compatible with a mode declaration in M_h; (ii) each body literal of R is compatible with a mode declaration in M_b; and (iii) no variable occurs with two different types.*

Scoring Functions

This section presents the new notion of a *scoring function* and formalises the learning task targeted by FastLAS. This new learning task is based on the *Learning from Answer Sets* tasks used by the ILASP (Inductive Learning of Answer Set Programs) systems (Law, Russo, and Broda 2014; 2015a; Law 2018). Specifically, Definition 2 of *Observational Predicate Learning from Answer Sets* tasks is a restriction of Learning from Answer Sets tasks, allowing only *Observational Predicate Learning* (where the predicates in the examples coincide with the predicates defined by the hypothesis), non-recursive hypotheses, and programs with exactly one answer set.

Definition 2. *An* Observational Predicate Learning from Answer Sets *(ILP^{OPL}_{LAS}) task is a tuple $T = \langle B, M, E^+\rangle$ where B is an ASP program called background knowledge, M is a mode bias, and E^+ is a set of WCDPIs such that $\forall e \in E^+$, $|AS(B \cup e_{ctx})| = 1$, and no predicate in M_h occurs in M_b or in the body of any rule in T.*

- *A hypothesis $H \subseteq S_M$ covers an example $e \in E^+$ iff $B \cup H$ accepts e.*
- *H is an inductive solution of T (written $H \in ILP^{OPL}_{LAS}(T)$) iff $\forall e \in E^+$ s.t. $e_{pen} = \infty$, H covers e.*
- *T is* satisfiable *iff $ILP^{OPL}_{LAS}(T) \neq \emptyset$. $\mathcal{T}_{OPL}$ is the set of all ILP^{OPL}_{LAS} tasks.*

Given any task T, we say that an example in E^+ is *uncovered* by a hypothesis H if and only if it is not covered, and use $UNCOV(H, T)$ to denote the set of all such examples.

Definition 3. *A* scoring function *is a function $\mathcal{S}$: Programs $\times\ \mathcal{T}_{OPL} \rightarrow \mathbb{R}_{\geq 0}$. A hypothesis $H \in$ Programs is said to be an* optimal solution *of a task $T \in \mathcal{T}_{OPL}$ w.r.t. a scoring function $\mathcal{S}$ iff $H \in ILP^{OPL}_{LAS}(T)$ and there is no $H' \in ILP^{OPL}_{LAS}(T)$ such that $\mathcal{S}(H', T) < \mathcal{S}(H, T)$.*

The *addition* of two scoring functions $\mathcal{S}_1$ and $\mathcal{S}_2$ is denoted $(\mathcal{S}_1 + \mathcal{S}_2)$, i.e. $(\mathcal{S}_1 + \mathcal{S}_2)(H, T) = \mathcal{S}_1(H, T) + \mathcal{S}_2(H, T)$.

Definition 4. *Let $\mathcal{S}$ be a scoring function. A function $\mathcal{S}^{rule}$: Rules $\times\ \mathcal{T}_{OPL} \rightarrow \mathbb{R}_{\geq 0}$ is a* decomposition *of $\mathcal{S}$ iff for each $P \in$ Programs and $T \in \mathcal{T}_{OPL}$, $\mathcal{S}(P, T) = \sum\limits_{r \in P} \mathcal{S}^{rule}(r, T)$.*

Example 1. Not all scoring functions have decompositions. For instance, consider the scoring function $\mathcal{S}_{noise} = \mathcal{S}_{len} + \mathcal{S}_{pen}$, where $\mathcal{S}_{len}(H, T) = |H|$ and $\mathcal{S}_{pen}(H, T) =$

[1]The set of constants of each type is assumed to be given with a task, together with the maximum number of variables in a rule, giving a set of variables $\mathtt{V_1}, \ldots, \mathtt{V_{max}}$ that can occur in a hypothesis. Whenever a variable $\mathtt{V}$ of type $\mathtt{t}$ occurs in a rule, the atom $\mathtt{t(V)}$ is added to the body of the rule to enforce the type.

$\sum_{e \in UNCOV(H,T)} e_{pen}$. There is clearly a decomposition of $\mathcal{S}_{len}$, $\mathcal{S}^{rule}_{len}(h, T) = |h|$; however, $\mathcal{S}_{pen}$ is not decomposable. To see this, consider the ILP^{OPL}_{LAS} task with an empty background knowledge, the hypothesis space $\{\texttt{p.}, \texttt{q.}\}$ and a single example $\langle \texttt{eg}_1, 1, \langle\{\texttt{p}, \texttt{q}\}, \emptyset\rangle, \emptyset\rangle$. Assume there is such a decomposition $\mathcal{S}^{rule}_{pen}$. $\mathcal{S}^{rule}_{pen}(\texttt{p.}, T)$ must be $\mathcal{S}_{pen}(\{\texttt{p.}\}, T)$ which is 1. Similarly, $\mathcal{S}^{rule}_{pen}(\texttt{q.}, T)$ must also be 1. This would imply that $\mathcal{S}_{pen}(\{\texttt{p}, \texttt{q}\}, T)$ should be 2, but it is 0. Hence $\mathcal{S}_{pen}$, and similarly $\mathcal{S}_{noise}$, are not decomposable.

The FastLAS algorithm presented in the next section supports scoring functions that are either decomposable, or of the form ($\mathcal{S} + \mathcal{S}_{pen}$) where $\mathcal{S}$ is decomposable.

Algorithm

This section presents the FastLAS algorithm, which is guaranteed to return an optimal solution of any ILP^{OPL}_{LAS} task with respect to any scoring function decomposition. FastLAS consists of four main steps: (1) *initial construction*; (2) *generalisation*; (3) *optimisation*; and (4) *solving*. During the initial construction phase, a subset S^1_M of the given hypothesis space is constructed, which is guaranteed to contain at least one solution of the task if the task is satisfiable. S^1_M is likely to contain very specific rules, each of which has been computed to cover only a single example. The generalisation phase searches for rules that are subrules of one or more rules in S^1_M, which can cover multiple examples leading to a larger hypothesis space S^2_M. The optimisation phase takes each rule R in S^2_M and computes an optimal subrule R', with respect to the scoring function, that is consistent with the exclusions in all examples. The resulting hypothesis space S^3_M is guaranteed to contain an optimal inductive solution of the task (if the task is satisfiable). In the final solving phase, the FastLAS algorithm searches for a subset of S^3_M that covers all the examples and which is guaranteed to be an optimal solution of the full task. For the remainder of this section, let T be the ILP^{OPL}_{LAS} task $\langle B, M, E^+\rangle$. Details on how to download and use the FastLAS system, together with all datasets in this paper, are available from the FastLAS webpage, `https://spike-imperial.github.io/FastLAS/`. Due to space restrictions, most proofs are omitted from this paper, but are available in a document on this webpage.

Initial Construction

We say that a subset of the hypothesis space of a task is *SAT-sufficient* if and only if it either contains at least one solution of the task or the original task is unsatisfiable. This section presents a method for computing a SAT-sufficient subset of a hypothesis space.

Definition 5. *Let $e \in E^+$ and a be a ground atom. A rule R is in the* characteristic ruleset *of T w.r.t. a and e (written $\mathcal{C}(T, a, e)$) if and only if: (i) $R \in S_M$; (ii) there is at least one ground instance R^g of R s.t. $a = head(R^g)$ and $body(R^g)$ is satisfied by the unique answer set of $B \cup e_{ctx}$; and (iii) there is no rule R' that satisfies (i) and (ii) s.t. R is a strict subrule of R'.*

Example 2. Consider a (propositional) task T with an empty background knowledge, mode declarations $M = \langle\{\texttt{p}, \texttt{q}\}, \{\texttt{r}, \texttt{not r}, \texttt{s}, \texttt{not s}\}\rangle$ and two examples $e_1 = \langle 1, \infty, \langle\{\texttt{p}\}, \{\texttt{q}\}\rangle, \{\texttt{r.}\}\rangle$ and $e_2 = \langle 2, \infty, \langle\{\texttt{q}\}, \{\texttt{p}\}\rangle, \emptyset\rangle$.

For each atom a, and each example e, $\mathcal{C}(T, a, e)$ contains the set of all maximal rules that (combined with B and e_{ctx}) prove a. So, for the atoms p and q, and the two examples, the characteristic rulesets are as follows.

- $\mathcal{C}(T, \texttt{p}, e_1) = \{\texttt{p:-r, not s.}\}$.
- $\mathcal{C}(T, \texttt{q}, e_1) = \{\texttt{q:-r, not s.}\}$.
- $\mathcal{C}(T, \texttt{p}, e_2) = \{\texttt{p:- not r, not s.}\}$.
- $\mathcal{C}(T, \texttt{q}, e_2) = \{\texttt{q:- not r, not s.}\}$.

Proposition 1. *Let e be an example in E^+ and $H \subseteq S_M$. $B \cup H$ accepts e if and only if (1) for each $a \in e^{inc}$ there is at least one rule in H that is a subrule of a rule in $\mathcal{C}(T, a, e)$; and (2) for each $a \in e^{exc}$, no rule in H is a subrule of any rule in $\mathcal{C}(T, a, e)$.*

Proposition 1 shows that the characteristic rulesets can be used to check whether a hypothesis (combined with the background knowledge) accepts an example. This leads to the definition of the characterisation of an example.

Definition 6. *The* characterisation *of an example $e \in E^+$ (written $\mathcal{C}(T, e)$) is the pair $\langle e_I, e_V\rangle$, where $e_I = \bigcup_{a \in e^{inc}} \mathcal{C}(T, a, e)$ and $e_V = \bigcup_{a \in e^{exc}} \mathcal{C}(T, a, e)$.*

Theorem 1 shows that the characterisations of examples in T can be used to construct a SAT-sufficient subset of a hypothesis space called the characterisation of T (written $\mathcal{C}(T)$).

Theorem 1. *Let $\mathcal{C}(T) = \{R \mid e \in E^+, R \in e_I\}$. T is satisfiable if and only if $\mathcal{C}(T)$ contains an inductive solution of T.*

Note that the e_V sets are not used when constructing $\mathcal{C}(T)$. The intuition is that the union of all e_I sets contains any maximal rule that proves at least one inclusion, and therefore any maximal rule that could possibly be useful for covering an example. The e_V sets, on the other hand, are the sets of all maximal rules that prove at least one exclusion, and so these are the rules that can not be in any inductive solution. The e_V sets are therefore not important when constructing a SAT-sufficient subset of the hypothesis space. They are, however, crucial in the *optimisation* phase of the algorithm, when the maximal rules in $\mathcal{C}(T)$ are optimised according to the scoring function. The e_V sets give a boundary to this optimisation to prevent exclusions from being proved.

Example 3. Reconsider the task T from Example 2.

- $\mathcal{C}(T, e_1) = \langle\{\texttt{p:-r, not s.}\}, \{\texttt{q:-r, not s.}\}\rangle$.
- $\mathcal{C}(T, e_2) = \langle\{\texttt{q:- not r, not s.}\}, \{\texttt{p:- not r, not s.}\}\rangle$.
- $\mathcal{C}(T) = \{\texttt{p:-r, not s. q:- not r, not s.}\}$

$\mathcal{C}(T)$ is SAT-sufficient, as $\mathcal{C}(T)$ itself is a solution. The rules in the e_V sets give a boundary for optimising the rules in $\mathcal{C}(T)$ (which takes place in a later phase of the FastLAS algorithm). In this case, they show that neither rule can have their first body literal removed without causing one of the exclusions to be proved.

The characterisation of an example can be computed using ASP. FastLAS uses a meta-level ASP encoding $\mathcal{M}(T, a, e)$ for which there is a one-to-one mapping between the subset-maximal[2] answer sets of $\mathcal{M}(T, a, e)$ and the rules in $\mathcal{C}(T, a, e)$. The Clingo5 (Gebser et al. 2016) ASP solver can be used to efficiently compute the subset-maximal answer sets of a program. The first step of the FastLAS algorithm is to compute the characterisations of each example. As characterisations of examples are independent from each other, they can be computed in parallel.

Generalisation

Characterisations of examples contain extremely specific rules. It is necessary to consider rules that are subrules of multiple rules in the characterisations. Definition 7 generalises the characteristic hypothesis space.

Definition 7. *For any rule $R \in S_M$, let c_R be the set of all rules R' in $\mathcal{C}(T)$ s.t. R is a subrule of R'. The* generalised characteristic hypothesis space *of T, written $\mathcal{G}(T)$, is the set containing every rule R for which $c_R \neq \emptyset$ and there is no rule $R' \in S_M$ s.t. R is a strict subrule of R' and $c_R = c_{R'}$.*

Example 4. Let T be a task such that $\mathcal{C}(T) = \{R_1, R_2\}$, where $R_1 = \texttt{p:-q,r}$ and $R_2 = \texttt{p:-q,s}$.

- $R_1, R_2 \in \mathcal{G}(T)$.
- $R_3 = \texttt{p:-q} \in \mathcal{G}(T)$ as $c_{R_3} = \{R_1, R_2\}$ (and clearly any rule of which R_3 is a strict subrule could not be a subrule of both R_1 and R_2).
- No other rule is in $\mathcal{G}(T)$. For example, the fact $R_4 = \texttt{p}$ is not in $\mathcal{G}(T)$, as it is a strict subrule of R_3, and $c_{R_3} = c_{R_4}$. Note that R_4 is more general than R_3, and may have a lower score, depending on the scoring function. This is not important when constructing $\mathcal{G}(T)$. Further generalisations, taking into account the scoring function, are described in the next section.

The second step of FastLAS is to compute $\mathcal{G}(T)$ from $\mathcal{C}(T)$. This is done by taking each rule R in $\mathcal{C}(T)$ (in parallel) and searching for subrules which are in $\mathcal{G}(T)$.

Optimisation

We say that a subset of the hypothesis space of a task is *OPT-sufficient* (w.r.t. a scoring function) if and only if it either contains at least one optimal solution of the task or the original task is unsatisfiable. The generalised characteristic hypothesis space contains rules that have been generalised, but only for cases where it is possible to combine multiple rules. In many cases, it is also necessary to generalise rules in order to optimise with respect to the scoring function. As we only focus on decomposable scoring functions in this paper, this computation can be done independently for each rule in $\mathcal{G}(T)$. Definition 8 formalises the notion of an *optimised characteristic hypothesis space*.

[2]Given a program P. A is a subset-maximal answer set of P if and only if $A \in AS(P)$ and A is not a subset of any other answer set in P. Note that as the meta-level encoding contains *choice rules*, it does have some non-subset-maximal answer sets.

Definition 8. *Let $R \in S_M$ and $\mathcal{S}$ be a decomposable scoring function. R' is an* optimisation *of R iff: (i) R' is a subrule of R; (ii) $\nexists e \in E^+$ s.t. $e_{pen} = \infty$ and R' is a subrule of a rule in e_V; and (iii) there is no R'' satisfying (i)-(ii) s.t. $\mathcal{S}^{rule}(R'', T) < \mathcal{S}^{rule}(R', T)$. R is* optimisable *iff it has at least one optimisation. An* optimised characteristic hypothesis space *of T w.r.t. $\mathcal{S}$ is a set of rules containing at least one optimisation of each optimisable rule in $\mathcal{G}(T)$.*

Example 5. Reconsider the task T from Example 2, for which $\mathcal{C}(T)$ is given in Example 3, and the scoring function $\mathcal{S}_{len}$. First note that $\mathcal{G}(T) = \mathcal{C}(T)$. The subrules of the rule `p:-r, not s` are itself, `p:-r`, `p:- not s` and `p`. The last two rules are subrules of `p:- not r, not s`, which is in $(e_2)_V$, and so cannot be optimisations by point (ii) of Definition 8. Clearly the score of the first rule is higher than the second; hence, the only optimisation of the original rule is `p:-r`. A similar argument shows that `q:- not r` is the only optimisation of `q:- not r, not s`. So the only optimised characteristic hypothesis space of T w.r.t. $\mathcal{S}_{len}$ is {`p:-r. q:- not r.`}.

Theorem 2 shows that optimised characteristic hypothesis spaces are guaranteed to be OPT-sufficient.

Theorem 2. *Let O be an optimised characteristic hypothesis space of T w.r.t. to a decomposable scoring function $\mathcal{S}$. If T is satisfiable, then O contains at least one optimal inductive solution of T w.r.t. $\mathcal{S}$.*

Proof. If T is satisfiable then it has at least one optimal inductive solution. Let H^* be an arbitrary such solution. By the definition of $\mathcal{G}(T)$, for each $h \in H^*$, there must be a rule $R_h \in \mathcal{G}(T)$ such that $c_h = c_{R_h}$ and h is a subrule of R_h. As each $h \in H^*$ is part of an optimal inductive solution of T and is a subrule of R_h, it must be an optimisation of R_h,[3] which shows that R_h is optimisable. Hence, for each $h \in H^*$, there is at least one rule $h' \in O$ such that $c_h = c_{h'}$, $\mathcal{S}_{rule}(h, T) = \mathcal{S}_{rule}(h', T)$ and h' is not a subrule of any rule in V_e for any $e \in E^+$ st $e_{pen} = \infty$. Pick an arbitrary $H' \subseteq O$ that contains one such rule for each $h \in H^*$ (and no other rules). $\mathcal{S}(H', T) = \mathcal{S}(H, T)$, and $H' \in ILP^{OPL}_{LAS}(T)$. Hence, H' is an optimal inductive solution of T w.r.t. $\mathcal{S}$. $\square$

Computing Optimised Characteristic Hypothesis Spaces Given a rule $R \in \mathcal{G}(T)$ it is possible to compute an optimisation of R using ASP, provided that the decomposition of the scoring function is expressed in ASP, using a predicate `penalty/2`. The first argument of `penalty` is an integer, representing the value of the penalty, and the second is a term, associating a unique identifier with each penalty. These identifiers are important, as answer sets do not contain repeated elements, meaning that to pay two penalties of 1, there must be two distinct penalty atoms with value 1.

Our ASP representation relies on the following notation. Given an atom a, $rv(a)$ denotes the atom constructed by replacing each variable `V` in a with the ground term

[3]Point (i) of the definition of an optimisation is met because h is a subrule of R_h; point (ii) must be met, or H^* would not be an inductive solution of T; and (iii) must also be met, or H^* would not be an optimal solution of T.

var("V"). For any rule R, $\mathcal{M}(R)$ is the set of facts $\{\texttt{head(rv(a)).} \mid a = head(R)\} \cup \{\texttt{in\_body(pos(rv(a))).} \mid a = body^+(R)\} \cup \{\texttt{in\_body(neg(rv(a))).} \mid a = body^-(R)\}$.

Definition 9. *Let P be an ASP program, which defines the predicate* penalty/2. *The scoring function $\mathcal{S}[P]$ is defined in terms of its decomposition. For any $h \in$ Rules and $T \in \mathcal{T}_{OPL}$, $\mathcal{S}[P]^{rule}(h, T) = \min\limits_{A \in AS(\mathcal{M}(h) \cup P)} \sum_{\texttt{penalty(x, y)} \in A} \texttt{x}$.*

Example 6. To express the $\mathcal{S}_{len}$ scoring function in FastLAS, we can use the ASP program P_{len}. $\mathcal{S}_{len} = \mathcal{S}[P_{len}]$.

$$P_{len} = \left\{ \begin{array}{l} \texttt{penalty(1, head) :- head(\_).} \\ \texttt{penalty(1, body(X)) :- in\_body(X).} \end{array} \right\}$$

Note that the second rule has a variable X in the head so a penalty of 1 is paid per body literal. Without this variable the penalty paid for the body of a rule could only be 0 or 1.

For any ASP-based decomposition, we compute the optimisation of a rule R using the ASP encoding in Definition 10.[4]

Definition 10. *Let $\mathcal{S}[P]$ be a scoring function and R be a rule. $\mathcal{M}_{opt}(P, R, T)$ is the program containing:*

1. *P.*
2. *$\mathcal{M}(R)$, where all* in_body *facts "*a.*" have been replaced with choices "*0{a}1.*".*
3. *The weak constraint* :~ penalty(X, Y).[X@1, Y].
4. *For each $e \in E^+$ and $R' \in e_V$ s.t. $head(R) = head(R')$, the rule* $\texttt{v(e}_{\texttt{id}}\texttt{) :- not in\_body(a}_1\texttt{)}, \ldots,$ $\texttt{not in\_body(a}_n\texttt{)}$. *where $\{a_1, \ldots, a_n\}$ are the literals[5] in the body of R that do not occur in the body of R'.*
5. *For each $e \in E^+$ s.t. $e_{pen} = \infty$, the constraint* $\texttt{:- v(e}_{\texttt{id}}\texttt{)}$.

Any answer set A of $\mathcal{M}_{opt}(P, R, T)$ can be mapped back into an ASP rule, by interpreting the head and in_body atoms in A. $\mathcal{M}^{-1}_{rule}(A)$ denotes the rule extracted from A.

Theorem 3. *Let $\mathcal{S}[P]$ be a scoring function and R be a rule. Let AS be the optimal answer sets of $\mathcal{M}_{opt}(P, R, T)$. $\{\mathcal{M}^{-1}_{rule}(A) \mid A \in AS\}$ is the set of all optimisations of R.*

FastLAS computes an optimisation of each optimisable rule in $\mathcal{G}(T)$ using the $\mathcal{M}_{opt}$ encoding, the correctness of which is proved by Theorem 3. By Theorem 2, this set of optimisations is guaranteed to be OPT-sufficient.

Noise. Point (ii) of Definition 8 means that no optimisation of any rule can prove an exclusion of an example with an infinite penalty. This is because no such rule could ever appear in an inductive solution of the task. However, we must consider such rules for examples with finite penalties (which can be left uncovered). In this setting we use an iterative method, formalised by Algorithm 1. $opt(P, R, T)$ iteratively constructs a set of rules RS such that in each iteration, the new rule R_{new} added to RS satisfies the following properties: (1) R_{new} is a subrule of R; (2) R_{new} is not a subrule of $R' \in e_V$ for any $e \in E^+$ such that $e_{pen} = \infty$; (3) for each $R' \in RS$ there is at least one example $e \in E^+$ for which R' is a subrule of at least one rule in e_V and R_{new} is not a subrule of any rule in e_V; and (4) R_{new} is optimal w.r.t. $\mathcal{S}[P]$. Conditions (1), (2) and (4) are enforced by $\mathcal{M}_{opt}$. Condition (3) is enforced by the constraints in CS. Theorem 4 shows that Algorithm 1 can be used to compute an OPT-sufficient subset of the hypothesis space.

Algorithm 1 $opt(P, R, T)$

```
procedure OPT($P, R, T$)
    $CS = \emptyset$; $RS = \emptyset$;
    while $AS(\mathcal{M}_{opt}(P, R, T) \cup CS) \neq \emptyset$ do
        Fix $A$ to be an optimal answer set of the program
        $R_{new} = \mathcal{M}^{-1}_{rule}(A)$;
        $RS = RS \cup \{R_{new}\}$;
        $CS = CS \cup \left\{ \texttt{:-} \bigwedge_{\texttt{v(id}_\texttt{i}\texttt{)} \in \texttt{A}} \texttt{v(id}_\texttt{i}\texttt{)}. \right\}$;
    end while
    return $RS$;
end procedure
```

Theorem 4. *Let $\mathcal{S}[P]$ be a scoring function. $\bigcup\limits_{R \in \mathcal{G}(T)} opt(P, R, T)$ is OPT-sufficient w.r.t. $(\mathcal{S}[P] + \mathcal{S}_{pen})$.*

Solving

Once an OPT-sufficient subset of the hypothesis space has been computed, it is possible to pass the task, along with this hypothesis space to an off-the-shelf ILP system, such as ILASP (Law, Russo, and Broda 2015a), which can find an optimal solution of the task. In fact, as the learning tasks that FastLAS solves are a simplification of the full $ILP^{context}_{LAS}$ tasks solved by ILASP, it is more efficient to use a specialised ASP encoding to find the optimal inductive solution. This ASP encoding is given in the proofs document on the FastLAS webpage, together with a proof of its correctness.

FastLAS is sound and complete w.r.t. the optimal inductive solutions of any ILP^{OPL}_{LAS} task under any decomposable scoring function (or any decomposable scoring function added to $\mathcal{S}_{pen}$), meaning that if FastLAS is used to solve an ILP^{OPL}_{LAS} task with the scoring function $(\mathcal{S}_{len} + \mathcal{S}_{pen})$, then it has exactly the same guarantees as the state-of-the-art ILASP systems. However, we show in our evaluation that FastLAS is significantly faster than ILASP on these tasks.

Evaluation

This section contains an evaluation of the FastLAS approach. The aim of the evaluation is to answer two questions: firstly, if we use standard scoring functions, is the FastLAS algorithm faster than state-of-the-art approaches that use the same standard scoring function; and secondly,

[4] This uses a wider subset of ASP than other programs in this paper. Roughly speaking, a *choice rule* "0{a}1." is equivalent to "a :- not $\hat{\texttt{a}}$. $\hat{\texttt{a}}$:- not a." (where for any atom a, $\hat{\texttt{a}}$ is a new atom, representing the complement of a) and generates two answer sets, one with and one without a. The *weak constraint* :~ penalty(X, Y).[X@1, Y] creates a preference ordering over answer sets, s.t. *optimal* answer sets A minimise $\sum_{\texttt{penalty(x, y)} \in A} \texttt{x}$.

[5] Positive literals a are replaced with pos(rv(a)) and negative literals not a are replaced with neg(rv(a)).

System	F_1	Running Time
INSPIRE	0.733 / 0.712	– / –
ILASP3	0.757 / 0.777	210.4s / 1051.4s
FastLAS	0.751 / 0.768	0.909s / 4.5s

Table 1: Results for INSPIRE, ILASP3 and FastLAS on the sentence chunking dataset. Each entry is of the form `a / b`, where `a` and `b` are the results for tasks with 100 and 500 examples, respectively.

in cases where we have a domain-specific notion of performance, can we use domain-specific scoring functions to improve performance w.r.t. this measure.

Comparison of FastLAS with the state-of-the-art

FastLAS was evaluated on two datasets that have previously been used to evaluate the state-of-the-art ASP-based ILP systems: OLED (Katzouris, Artikis, and Paliouras 2016), INSPIRE (Kazmi, Schüller, and Saygın 2017) and ILASP3 (Law, Russo, and Broda 2018b).[6]

INSPIRE (Kazmi, Schüller, and Saygın 2017) has been evaluated using a sentence chunking (Tjong Kim Sang and Buchholz 2000) dataset (Agirre et al. 2016), where the goal is to learn to split a sentence into short phrases called chunks. (Kazmi, Schüller, and Saygın 2017) describes how to transform each sentence into a set of facts consisting of part of speech (POS) tags, forming a pre-processing step. We ran FastLAS on the processed version of the dataset using each of these sets of facts as an example, learning rules for whether to split the sentence between each pair of tags. Both FastLAS and ILASP3 outperformed INSPIRE in terms of the average F_1 score. FastLAS and ILASP3 achieved similar scores.[7] INSPIRE is an *approximate* system that, although using the same scoring function as FastLAS and ILASP3, does not guarantee optimality. This explains the significantly better F_1 scores of FastLAS and ILASP3. One might expect that this would mean that INSPIRE would be faster than FastLAS and ILASP3. This does not seem to be the case – running times are not reported in (Kazmi, Schüller, and Saygın 2017), but a timeout of 30 minutes is used, after which the best solution computed so far was returned, which would indicate that at least some of the experiments did not complete in 30 minutes. On the other hand, every experiment for FastLAS and ILASP3 completed inside 30 minutes. Furthermore, FastLAS is over two orders of magnitude faster than ILASP3.

System	F_1	Running Time
OLED	0.792	107s
ILASP3	0.837	523.3s
FastLAS	0.907	263.8s

Table 2: Results for OLED, ILASP3 and FastLAS on the CAVIAR dataset.

We compared FastLAS to OLED (Katzouris, Artikis, and Paliouras 2016) and ILASP3 on a dataset containing data gathered from a video stream (Fisher, Santos-Victor, and Crowley 2004). Information such as the positions of people has been extracted from the stream, and humans have annotated the data to specify when any two people are interacting. Specifically, we consider a task from (Katzouris, Artikis, and Paliouras 2016), in which the aim is to learn rules to define initiating and terminating conditions for two people meeting. As ILASP3 enumerates the hypothesis space in full, it is not able to use large hypothesis spaces. A small subset of the hypothesis space used by OLED was used in the ILASP3 experiments, restricting the number of literals in the body, employing several "common sense" constraints, such as a person cannot be walking and running at the same time, and forbidding multiple uses of the same predicate in the body of a single rule. As this is real data, not constructed with a target hypothesis in mind, there may be better hypotheses outside this restricted subset. In the FastLAS experiments, we allowed a much larger hypothesis space without these restrictions, containing over 2^{44} non-isomorphic rules, compared with the hypothesis space used in the ILASP3 experiment, which had only 3370. Even so, the average running time for FastLAS is faster. The average F_1 is also significantly higher. The hypotheses learned by FastLAS are outside the restricted space used by ILASP3, indicating that the larger hypothesis space contains better quality solutions. OLED is significantly faster than both ILASP3 and FastLAS on this dataset; however, as it is an approximate system that does not guarantee optimality, this is unsurprising. OLED's average F_1 score is significantly lower than both ILASP3's and FastLAS's.

Domain-specific scoring functions

We experimented with three scoring functions (each added to $\mathcal{S}_{pen}$ to account for noise), aimed at encouraging progressively more generalisation. Their decompositions were $\mathcal{S}^{rule}_{len}$, $\mathcal{S}^{rule}_{cov}(h,T) = \frac{1000}{COV(h,T)}$, where $COV(h,T)$ is the number of inclusions of examples in T which are covered by h, and $\mathcal{S}^{rule}_{uni}(h,T) = -1$.[8] We considered two datasets:

[6]The FastLAS and ILASP experiments were run on an Ubuntu 18.04 desktop machine with a 3.6 GHz Intel® Core™ i7-4790 processor and with 16GB RAM. For a fair comparison, the final ASP program in the FastLAS solving phase was solved using Clingo 5.3.0 with the same arguments as in the similar phase for ILASP3 (the arguments were `--opt-strat=usc,stratify`). The results for INSPIRE and OLED are quoted from (Kazmi, Schüller, and Saygın 2017) and (Katzouris, Artikis, and Paliouras 2016).

[7]The small differences between these scores is due to FastLAS finding a different optimal solution to the one found by ILASP3. Starting with a null hypothesis that there is no difference between FastLAS and ILASP3's mean F_1 scores and testing with a paired two-tailed t-test yields a p-value of .072, meaning that at $p < .05$ the difference is not statistically significant, whereas a similar comparison of FastLAS and INSPIRE yields a p-value of .0001, meaning that at $p < .05$ the difference is statistically significant.

[8]Note that the last scoring function is technically not a valid scoring function, as it returns a negative integer. This means that FastLAS is not guaranteed to return the optimal solution in this case. In fact, rather than returning the hypothesis with the most rules, FastLAS returns the hypothesis with the most rules which prove at least one inclusion of an example (the rules which do not satisfy this criterion are not considered by FastLAS).

Scoring function	Recall	Precision	F_1
$\mathcal{S}_{len}$	0.782 / 0.976	0.957 / 0.948	0.861 / 0.962
$\mathcal{S}_{cov}$	0.792 / 0.972	0.956 / 0.949	0.867 / 0.960
$\mathcal{S}_{uni}$	0.820 / 0.973	0.955 / 0.949	0.882 / 0.961

Table 3: Results for the Amazon dataset, both for learning `accept` and `reject` rules. Each entry is of the form (`accept` / `reject`).

Scoring function	Recall	Precision	F_1
$\mathcal{S}_{len}$	0.905 / 0.974	0.951 / 0.935	0.928 / 0.954
$\mathcal{S}_{cov}$	0.892 / 0.969	0.949 / 0.941	0.919 / 0.955
$\mathcal{S}_{uni}$	0.991 / 0.966	0.917 / 0.965	0.952 / 0.966

Table 4: Results for the Project Management dataset, both for learning `accept` and `reject` rules. Each entry is of the form (`accept` / `reject`).

Amazon (Amazon 2013) and Project Management (Xu and Stoller 2014). Both datasets are based on access logs, where many requests are made by various users to access various resources, based on the attributes of the requester and the resource. The only attribute of resources in the Amazon dataset is the resource id, whereas the Project Management dataset also includes attributes such as the resource type. The goal of this task was to learn rules for each resource to determine whether a user should be allowed access to different resources. We performed 10-fold cross validation, experimenting with learning `accept` and assuming that a request should be rejected if our learned rules do not say it should be accepted, and learning `reject`, assuming a default of `accept`.

The results show that for both datasets there is an ordering $\mathcal{S}_{len}$, $\mathcal{S}_{cov}$, $\mathcal{S}_{uni}$, in terms of the recall when learning `accept` rules. This indicates that $\mathcal{S}_{uni}$ encourages learning more general hypotheses, which allow more requests. There is a corresponding reverse ordering for the precision. Interestingly, although the differences are less significant, the orderings in both cases are reversed when learning `reject` rules, as a more general hypothesis for reject *denies* more requests and therefore accepts fewer. The ability of these scoring functions to encourage or discourage generalisation has the potential to make a significant impact. In security domains, where false positives are potentially more dangerous than false negatives, a scoring function which optimises the precision should be preferred, whereas in healthcare, where false negatives may be more dangerous, the reverse is true.

Universal F_1 In (Cotrini, Weghorn, and Basin 2018) it was argued that standard evaluation measures such as accuracy and the F_1 score may be misleading when evaluating a learner in an access control setting. This is because certain types of request may not be likely to appear in the training/test data, meaning that, although they should be rejected, there are no negative instances in the data. The standard quality measures would not penalise overly permissive rules that accept these missing requests. A new method of *universal cross validation* was proposed in (Cotrini, Weghorn, and Basin 2018), which is based on a modification of the F_1 score. We refer to it as the *universal* F_1 or UF_1 score.

When using an access control log for learning a policy for accepting a particular resource r, we construct the examples as follows. The positive and negative examples are as normal (the accepted and rejected requests, respectively). Additionally, there is a third set of examples of "unlabelled" requests, which consists of every person that occurs in the log, but does not request access to r (they have requested access to other resources). The implicit assumption behind universal F_1 is that it is safer to treat these as negative instances, and therefore when measuring the quality of a hypothesis, it should be penalised for any of the unlabelled examples that it says should be accepted. The definition of precision is therefore altered to *universal precision*: $UP = \frac{tp}{tp+fp+u}$, where tp and fn is the number of true positives and false negative, as usual, and u is the number of unlabelled examples predicted by the hypothesis. UF_1 is then defined similarly to F_1, but using *UP* rather than standard precision.[9] As the number of unlabelled examples tends to be much higher than the number of examples in the logs, and some of the unlabelled examples are likely to be requests that should be accepted, even the best algorithms in (Cotrini, Weghorn, and Basin 2018) had relatively low UF_1 scores.

We compared FastLAS against the two best performing methods from (Cotrini, Weghorn, and Basin 2018) using the UF_1 score, learning rules for `accept` on the same four resources. The three scoring functions used earlier in this section perform poorer than the two previous methods, because they each encourage generalisation. In (Cotrini, Weghorn, and Basin 2018), the full set of users in the log is assumed to be given to the algorithms, and is used when selecting rules. We took a similar approach, defining a new scoring function to penalise overly general rules: $\mathcal{S}^{rule}_{UF_1}(h,T) = 1 + \frac{|Predicted(h)|}{\sqrt{|PosUsers \cap Predicted(h)|}}$, where *PosUsers* is the set of users who appear as an accept request in the training data and *Predicted*(h) is the set of users who are predicted as accept by h. The intuition of this scoring function is as follows: the 1 term is to make this a valid scoring function, which always returns a positive integer; we multiply by the total number of predicted users to penalise for generality; the number of predicted positive instances is in the denominator to encourage these cases. There may be scoring functions with better performance; however, on this dataset, compared with the two previous methods, FastLAS with $\mathcal{S}^{rule}_{UF_1}(h,T)$ performs as well as the two previous methods.

We make no claims about the legitimacy of universal F_1 as a statistical measure, although we agree with its motivation. The experiments in this section were designed to show that, if the goal of a learner is to maximise UF_1, standard scoring functions in ILP perform poorly, as they encourage generalisation. If we instead use a specialised scoring function that discourages over-generalisation, we can increase the UF_1 score. This demonstrates that we can inject domain-

[9]The standard definition of precision is $Pr = \frac{tp}{tp+fp}$, recall is $Re = \frac{tp}{tp+fn}$ and F_1 is the harmonic mean of Pr and Re.

Method	Resource 25993	Resource 4675	Resource 75078	Resource 79092
Rhapsody	0.04	0.10	0.10	0.04
CTA	0.04	**0.12**	0.10	0.04
FastLAS: $\mathcal{S}_{len}$	0.02	0.04	0.02	0.02
FastLAS: $\mathcal{S}_{cov}$	0.02	0.04	0.02	0.02
FastLAS: $\mathcal{S}_{uni}$	0.01	0.04	0.02	0.02
FastLAS: $\mathcal{S}_{UF_1}$	**0.07**	0.10	**0.11**	**0.05**

Table 5: Universal F_1 scores for the four resources in the Amazon dataset, learning `accept` rules.

specific preferences over the properties of hypotheses in order to maximise a domain-specific notion of performance.

Related Work

The construction phase of the FastLAS algorithm is very similar to early *bottom clause* ILP approaches used by Progol (Muggleton 1995), Aleph (Srinivasan 2001) and later generalised by HAIL (Ray, Broda, and Russo 2003). A key difference is that these early systems all used iterative *cover loop* approaches to construct a hypothesis. A single positive example (corresponding to a single inclusion in this paper) is considered in each iteration. The systems compute the best rule (or rules in the case of HAIL) that covers the example, and add it to the hypothesis. This means that none of these cover loop systems guarantee finding an optimal solution, as although each iteration might find an optimal rule to add to the hypothesis, the final hypothesis may still be sub-optimal.

The ILASP (Law, Russo, and Broda 2015a) systems also learn ASP programs. Our evaluation shows that FastLAS is significantly faster than ILASP when applied on the same learning tasks. This increase in speed is for two main reasons: (1) ILASP3 starts by constructing the hypothesis space in full, whereas FastLAS constructs a small OPT-sufficient subset of the hypothesis space; and (2) ILASP3 translates each example into a set of structural constraints on hypotheses and processes a single example at a time, whereas FastLAS processes the examples in parallel. On the other hand, ILASP is much more general as it can (resources permitting) learn any ASP program (Law, Russo, and Broda 2018a), including programs with choice rules and weak constraints (Law, Russo, and Broda 2015b), and supports recursion and predicate invention (Law 2018). In future work, we intend to lift some of FastLAS's current restrictions, in order to support more of the applications supported by ILASP.

The idea of allowing customised scoring functions has been considered before. Aleph has nine built-in evaluation functions, which are evaluated over single rules and mostly defined over the coverage of the examples, but with some also involving the length of the rule. Aleph also allows a user defined evaluation function to be given, but the Aleph manual (Srinivasan 2001) notes that it is usually not possible to define admissible pruning strategies for an arbitrary evaluation function, so when using this feature the user must also give an admissible pruning strategy. On the other hand, FastLAS can solve a task with any ASP-decomposition of a scoring function. A key difference between the approaches is that Aleph's evaluation function gives a measure of the utility of a rule, whereas FastLAS's scoring functions define the cost of a rule. As noted above Aleph is not guaranteed to find an optimal solution w.r.t. a given scoring function.

The variation of the INSPIRE system in (Schüller and Benz 2018) used a scoring function with many parameters, such as a cost for each variable, a cost for each positive/negative body literal, a cost for each variable that occurs only in the head of the rule (and many others). The total cost of a rule is the sum of its values for these costs. Each of the cost parameters used by INSPIRE could be implemented in FastLAS scoring functions. The flexibility of INSPIRE is that a user can set the values for these costs, whereas the flexibility in FastLAS is that a user can write a completely new scoring function, defining their own parameters.

Metaopt (Cropper and Muggleton 2019) aims to learn the most efficient Prolog program that covers a set of examples. Metaopt uses a *program cost function*, which measures the cost of proving each positive example (atom), given the learned program. Metaopt guarantees to find the program which minimises the maximum program cost over the positive examples. The program cost function can be used to represent computational efficiency, and Metaopt guarantees finding the program that optimises the worst-case efficiency over the examples. Our abstract notion of scoring function is more general, and does not need to be defined in terms of the costs of positive examples, although it can be; however, the current version of FastLAS would not be able to support the program cost functions, as they are not decomposable. Furthermore, there are fundamental differences between Metaopt, which learns definite clauses, but supports recursion and predicate invention, and FastLAS, which learns ASP programs, including negation as failure, but does not support recursion or predicate invention. A further fundamental difference is that FastLAS supports learning in the presence of noise, whereas Metaopt does not, and can only find hypotheses which cover all of the examples.

Conclusion and Future Work

The FastLAS algorithm presented in this paper solves a restricted form of an $ILP_{LAS}^{context}$ task with either a purely decomposable scoring function or the combination of a decomposable scoring function with the $\mathcal{S}_{pen}$ function to allow for noisy examples. The evaluation showed that even with these restrictions, FastLAS shows significant improvements over the state-of-the-art, both by improving scalability w.r.t. the size of the hypothesis space and by allowing domain-specific biases to be incorporated into the learning process.

The first step of future work is to lift the restrictions to allow full $ILP_{LAS}^{context}$ tasks to be solved, allowing general

ASP programs, with multiple answer sets, to be learned. This would immediately widen the scope of scoring functions that could be defined; for example, cycles in the dependency graph of the program impact the number of answer sets of a program, and scoring functions could be designed to minimise/maximise the number of cycles, or other similar properties. Such scoring functions will necessitate a generalisation of FastLAS for non-decomposable scoring functions, which is being developed in current work.

In future work, it would also be interesting to further investigate scoring functions whose decompositions return negative integers, such as $\mathcal{S}^{uni}_{rule}$. These scoring functions can represent the utility, rather than the cost, of rules.

Acknowledgements

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. Jorge Lobo was supported by the Spanish Ministry of Economy and Competitiveness under Grant Numbers TIN201681032P, MDM20150502, and the U.S. Army Research Office under agreement number W911NF1910432.

References

Agirre, E.; Gonzalez Agirre, A.; Lopez-Gazpio, I.; Maritxalar, M.; Rigau Claramunt, G.; and Uria, L. 2016. Semeval-2016 task 2: Interpretable semantic textual similarity. In *SemEval-2016. 10th International Workshop on Semantic Evaluation.* ACL.

Amazon. 2013. Amazon.com employee access challenge. http://www.kaggle.com/c/amazon-employee-access-challenge.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12):92–103.

Cotrini, C.; Weghorn, T.; and Basin, D. 2018. Mining abac rules from sparse logs. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 31–46. IEEE.

Cropper, A., and Muggleton, S. H. 2019. Learning efficient logic programs. *Machine Learning* 108(7):1063–1083.

Fisher, R.; Santos-Victor, J.; and Crowley, J. 2004. CAVIAR: Context aware vision using image-based active recognition. `http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/`. Accessed: 2019-08-28.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Wanko, P. 2016. Theory solving made easy with clingo 5. In *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Gelfond, M., and Kahl, Y. 2014. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach.* Cambridge University Press.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, 1070–1080.

Katzouris, N.; Artikis, A.; and Paliouras, G. 2016. Online learning of event definitions. *Theory and Practice of Logic Programming* 16(5-6):817–833.

Kazmi, M.; Schüller, P.; and Saygın, Y. 2017. Improving scalability of inductive logic programming via pruning and best-effort optimisation. *Expert Systems with Applications* 87:291–303.

Law, M.; Russo, A.; and Broda, K. 2014. Inductive learning of answer set programs. In Fermé, E., and Leite, J., eds., *Proceedings of the Fourteenth European Conference on Logics in Artificial Intelligence, 2014, Funchal, Madeira, Portugal, September 24-26, 2014.*, volume 8761 of *Lecture Notes in Computer Science*, 311–325. Springer.

Law, M.; Russo, A.; and Broda, K. 2015a. The ILASP system for learning answer set programs. `www.ilasp.com`.

Law, M.; Russo, A.; and Broda, K. 2015b. Learning weak constraints in answer set programming. *Theory and Practice of Logic Programming* 15(4-5):511–525.

Law, M.; Russo, A.; and Broda, K. 2016. Iterative learning of answer set programs from context dependent examples. *Theory and Practice of Logic Programming* 16(5-6):834–848.

Law, M.; Russo, A.; and Broda, K. 2018a. The complexity and generality of learning answer set programs. *Artificial Intelligence* 259:110–146.

Law, M.; Russo, A.; and Broda, K. 2018b. Inductive learning of answer set programs from noisy examples. *Advances in Cognitive Systems.*

Law, M. 2018. *Inductive Learning of Answer Set Programs.* Ph.D. Dissertation, Imperial College London.

Muggleton, S. 1991. Inductive logic programming. *New Generation Computing* 8(4):295–318.

Muggleton, S. 1995. Inverse entailment and Progol. *New Generation Computing* 13(3-4):245–286.

Ray, O.; Broda, K.; and Russo, A. 2003. Hybrid abductive inductive learning: A generalisation of progol. In *Inductive Logic Programming*. Springer. 311–328.

Ray, O. 2009. Nonmonotonic abductive inductive learning. *Journal of Applied Logic* 7(3):329–340.

Schüller, P., and Benz, M. 2018. Best-effort inductive logic programming via fine-grained cost-based hypothesis generation. *Machine Learning* 107(7):1141–1169.

Srinivasan, A. 2001. The aleph manual. *Machine Learning at the Computing Laboratory, Oxford University*.

Tjong Kim Sang, E. F., and Buchholz, S. 2000. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, 127–132. Association for Computational Linguistics.

Xu, Z., and Stoller, S. D. 2014. Mining attribute-based access control policies from logs. In *IFIP Annual Conference on Data and Applications Security and Privacy*, 276–291. Springer.