MC-TopLog: Complete Multi-clause Learning Guided by a Top Theory

Stephen Muggleton, Dianhuan Lin and Alireza Tamaddoni-Nezhad

Department of Computing, Imperial College London

Abstract. Within ILP much effort has been put into designing methods that are complete for hypothesis finding. However, it is not clear whether completeness is important in real-world applications. This paper uses a simplified version of grammar learning to show how a complete method can improve on the learning results of an incomplete method. Seeing the necessity of having a complete method for real-world applications, we introduce a method called $\top$-directed theory co-derivation, which is sound and complete for deriving a hypothesis within the declarative bias. The proposed method has been implemented in the ILP system MC-TopLog and tested on grammar learning and the learning of game strategies. Compared to Progol5, an efficient but incomplete ILP system, MC-TopLog has higher predictive accuracies, especially when the background knowledge is severely incomplete.

1 Introduction

As first pointed out by Yamamoto [23], hypotheses derivable from Progol [12] are restricted to those which subsume E relative to B in Plotkin's sense [18]. This type of incompleteness can be characterised as deriving only single-clause hypotheses. In this paper, we compare entailment-incomplete single-clause learning systems to entailment-complete multi-clause learning systems.

Yamamoto uses the learning of odd-numbers to demonstrate Progol's incompleteness. His example involves recursion and mutually dependent predicates (odd and even), making it unclear whether only applications with these properties might be affected by this type of incompleteness. To the authors' knowledge it has not subsequently been demonstrated conclusively that the incompleteness of single-clause learning noticeably restricts the application of single-clause learners. It might reasonably be supposed that in real-world applications learned theories can always be built by sequentially adding single clauses.

Grammar learning is central to language translation software, automated booking systems and grammar checking for word processes. Section 2 uses a simplified version of grammar learning, which is artificially designed and does not involve recursion or mutually dependent predicates, to show how a complete method can improve the learning results of an incomplete method. This is further demonstrated in section 4 via experiments with two real-world data sets. More experiments with real-world applications can be found in [10], where target hypotheses are unknown for knowledge discovery tasks. The focus of this paper is to introduce a new complete approach called $\top$-directed theory co-derivation($\top$DTcD). The following two subsections highlight the two key features that distinguish $\top$DTcD from other multi-clause learning systems.

1.1 Common Generalisation

The idea of common generalisation was first introduced in Plotkin's Least General Generalisation (LGG) [18] and Reynolds' Least Common Generalisation (LCG) [22]. This idea is used in this paper to extend ⊤DTD (⊤-directed theory derivation) to ⊤DTcD. We refer co-generalisation for methods that restrict their search spaces to common generalisations of multiple examples, as opposed to solo-generalisation for methods that generalise a single example.

Although doing co-generalisation can lead to higher efficiency, it has been introduced in few ILP systems. Among the systems based on Inverse Entailment (IE) [12], ProGolem [16] extending from Golem [14] is the only one that can do co-generalisation. Unfortunately, it suffers from similar type of incompleteness as that in Progol. While all the existing complete methods that are IE based can only do solo-generalisation, e.g. CF-Induction [6], XHAIL [21] and IMPARO [8]. These are reflected in table 1, which classifies several typical ILP systems based on their generalisation methods. Although CF-Induction and XHAIL can generalise multiple examples all at once, their search spaces are not bound to the common generalisations, therefore they are *not* in the category of co-generalisation.

The inability to do co-generalisation is less of an issue for ILP systems like HYPER [4] and TILDE [1], which use all the training examples to guide a top-down search. Also the generalisation methods in these systems are not IE based, thus they do not suffer from Progol's incompleteness. On the other hand, they lose the advantage provided by IE. Specifically, their search spaces are not bound to those hold for $B \wedge H \models E$. Also, these systems cannot handle abduction, thus not applicable to the grammar learning example given in this paper, where the background knowledge is incomplete.

1.2 Top Theory and TopLog Family

A top theory $\top$ is a logic program representing a declarative bias. Compared to the mode declaration [12] used in many ILP systems, a top theory has the advantage of encoding a strong declarative bias. Although there exists other forms of declarative bias that are comparable to the top theory in terms of their expressive power in encoding strong declarative bias, they are in the meta-level, such as antecedent description language (ADL) [5] and its extension $\mathcal{D}$LAB [19]. In contrast, a top theory is in the object-level as a logic program. This makes it possible for a top theory to be reasoned directly with background knowledge, so that the derived hypotheses are restricted to those hold for $B \wedge H \models E$. In other words, the top theory not only provides a mechanism for naturally encoding the strong declarative bias, but also facilitate a method to bound the search space. The top theory is also similar to Spectre's starting-point theory (an

	Solo-Generalisation	**Co-Generalisation**
Single-clause	TopLog Progol FOIL	LGG and LCG Golem and ProGolem
Multi-clause	CF-Induction XHAIL IMPARO TAL MC-TopLog (⊤DTD)	 HYPER TILDE MC-TopLog (⊤DTcD)

Table 1: Classifying ILP systems based on their generalisation methods

overly general theory to be unfolded)[3], but top theories make a clear distinction between terminal and non-terminal predicates. This is a powerful mechanism for distinguising between search control and the object language.

$\top$ is first introduced in a method called $\top$-directed hypothesis derivation ($\top$DHD), which is implemented in the ILP system TopLog [15]. To overcome TopLog's limitation of single-clause learning, $\top$DHD is extended to $\top$DTD. The resulting system is named MC-TopLog (Multi-clause TopLog). $\top$DTD and $\top$DTcD correspond to two different learning modes in MC-TopLog: generalising single example or multiple examples. Inherited from $\top$DHD, both $\top$DTD and $\top$DTcD use a logic program called the top theory to represent declarative bias.

2 Multi-clause Learning vs. Single-clause Learning

Progol's entailment-incompleteness can be characterized by single-clause learning. Because a hypothesis H will not be derived by Progol, unless it subsumes an example e relative to B in Plotkin's sense. This condition requires H to be a single clause, and this clause is used only once in the *refutation* of the example e. This leads to our definition of single-clause and multi-clause learning as that in Definition 1. Please note that they are defined in terms of the number of hypothesised clauses used in a refutation of an example, rather than the number of clauses in H. Accordingly, even if the number of clauses in H is only one, it can still be multi-clause learning. For example, in Yamamoto's example of learning odd-numbers, the hypothesised clause $odd(s(X)) \leftarrow even(X)$ is used twice when proving the positive example $odd(s(s(s(0))))$, thus deriving such a hypothesis H from that example is multi-clause learning even though H appear to be a single clause. And vice versa: even if the number of clauses in H is more than one, it may be essentially single-clause learning. Such example will be given later.

Definition 1. *Let c_i be a clause, which is either from background knowledge B or hypothesis H. Suppose $R = \langle c_1, c_2, ..., c_n \rangle$ is a refutation sequence that explains a positive example e. Let M be the number of clauses in R that is from H. It is single-clause learning (SCL) if $M = 1$; while it is multi-clause learning (MCL) if $M \geq 1$.*

2.1 Grammar Learning Example

Fig. 1 shows a simplified version of grammar learning, which is used here to exemplify Definition 1. In this grammar learning task, multi-clause and single-clause learning methods will derive $H_{mc} = \{h_4, h_5, h_6, h_7\}$ and $H_{sc} = \{h_1, h_2, h_3\}$, respectively. Although there are multiple clauses in H_{sc}, each of them is derived independently from different examples by a single-clause learner. Specifically, h_1, h_2 and h_3 are generalised independently from e_1, e_2 and e_3, respectively. In contrast, clauses in H_{mc} are dependent, and they have to be generalised together in order to explain an example. For instance, hypothesising h_4 alone is not able to complete the refutation proof of the example e_1, since the definition about np is incomplete in B and the type of the word 'unknown' is also missing from B. Thus another two clauses, either $\{h_5, h_7\}$ or $\{h_8, h_9\}$, have to be derived together with h_4 in order to explain e_1.

In this artificially designed example, H_{mc} is the target hypothesis which is not derivable by a single-clause learner. H_{mc} is also more compressive than

H_{sc}, because H_{mc} has a shorter description length[1] than H_{sc} while covers the same number of examples. The shorter description length of H_{mc} results from a multi-clause learner's ability to hypothesise multiple dependent clauses. For example, the clause about s is simplified in H_{mc}, since clauses about np can be hypothesised together with s. Specifically, h_4 is simpler than any of h_1, h_2 and h_3, because it is derived together with other clauses, such as $\{h_5, h_7\}$ or $\{h_8, h_9\}$.

2.2 Distinctions from MPL and LMC

As discussed earlier, clauses within H_{mc} are dependent. This is similar to that in multiple predicate learning (MPL), where clauses about different predicates depend on each other. However, the MPL discussed in [20] is essentially single-clause learning. Because each predicate to be learned are observable and provided as examples. Therefore, there is only one clause about the observed predicate to be hypothesised for each example. For the learning problem in Fig. 1, an MPL method would require the predicates np and vp to be observable and provided as training examples.

The term *learning multiple clauses* (LMC) is used to describe a global-optimisation approach, in which multiple clauses that compressed from the *whole* set of examples are refined together, as opposed to a local-optimisation approach like the covering algorithm, where clauses compressed from a *subset* of examples are added to the final H iteratively. However, LMC and MCL are related to different issues. LMC is related to the issue of selecting hypotheses globally rather than locally. The hypotheses from which it selects can be derived either by MCL or SCL. Even if a learning algorithm's search space only consists of single clauses derived by SCL, its final hypothesis may still have multiple clauses, which are aggregated from single clauses generalised from different examples. In contrast, MCL is to do with generalising an example to multiple clauses instead of a single clause. It can be combined with a selection method that is either global or local. Specifically, after deriving all candidate hypotheses using a MCL method, the covering algorithm is still applicable to greedily choosing a hypothesis which is locally most compressed.

2.3 Increase in Hypothesis Space

Although the complete hypothesis space of MCL makes it possible to find hypotheses with higher compression than SCL, the trade-off is a much larger search space which affects efficiency. Specifically, the upper bound on the hypothesis space of a single-clause learner is $O(2^N)$, where N is the number of distinct atoms derivable from a hypothesis language. In contrast, it is $O(2^{2^N})$ for a multi-clause learner, because it does not ignore the hypotheses with dependent clauses. Such a large hypothesis space makes MCL not PAC-learnable (Probably approximately correct learnable). Because the number of examples m grows exponentially with increasing N, rather than polynomial as SCL. Specifically, the Blumer bound [2] of SCL and MCL are as given below, in which m stands for the number of training examples, ϵ is the bound on the error, $|H|$ is the cardinality of the hypothesis space and $(1-\delta)$ is the bound on the probability with which

[1] In this paper, the description length (DL) of a clause is defined by the number of literals in the clause; while the compression is defined as $Num_{E+} - Num_{E-} - DL$.

Positive and Negative Examples E:

e_1:$s([an, unknown, alien, hits, the, house], [])$.
e_2:$s([a, small, boy, walks, a, dog], [])$.
e_3:$s([a, dog, walks, into, the, house], [])$.
e_4:$\neg s([dog, hits, a, boy], [])$.

Hypothesis language $\mathcal{L}$:
Predicates $=\{s, np, vp, det, noun, verb...\}$
Variables $=\{S_1, S_2, S_3, ...\}$
Constants $=\{a, the, ...\}$

Background Knowledge B:

b_1:$np(S1, S2) \leftarrow det(S1, S3), noun(S3, S2)$.
b_2:$vp(S1, S2) \leftarrow verb(S1, S2)$.
b_3:$vp(S1, S2) \leftarrow verb(S1, S3), prep(S3, S2)$.
b_4:$det([a|S], S)$. b_5:$det([an|S], S)$. b_{13}:$det([the|S], S)$.
b_6:$noun([dog|S], S)$. b_7:$noun([boy|S], S)$.
b_8:$noun([house|S], S)$. b_9:$noun([alien|S], S)$.
b_{10}:$verb([hits|S], S)$. b_{11}:$adj([small|S], S)$.
b_{12}:$prep([into|S], S)$.

Part of Hypothesis Space $\mathcal{H}$:

h_1:$s(S1, S2) \leftarrow det(S1, S3), S3 = [Word|S4], noun(S4, S5), vp(S5, S6), np(S6, S2)$.
h_2:$s(S1, S2) \leftarrow det(S1, S3), adj(S3, S4), noun(S4, S5), S5 = [Word|S6], np(S6, S2)$.
h_3:$s(S1, S2) \leftarrow np(S1, S3), S3 = [Word|S4], prep(S4, S5), np(S5, S2)$.
h_4:$s(S1, S2) \leftarrow np(S1, S3), vp(S3, S4), np(S4, S2)$.
h_5:$np(S1, S2) \leftarrow det(S1, S3), adj(S3, S4), noun(S4, S2)$
h_9:$np(S1, S2) \leftarrow det(S1, S3), prep(S3, S4), noun(S4, S2)$
h_6:$verb([walks|S], S)$. h_7:$adj([unknown|S], S)$. h_8:$prep([unknown|S], S)$.

Fig. 1: Grammar Learning Example

the inequality holds for a randomly chosen set of training examples. Even when N is small and fixed for a particular learning problem, MCL's hypothesis space is still much larger than that of SCL. That is why it is particularly important for a multi-clause learner to bound its search space to the candidates hold for $B \wedge H \models E$ and makes use of the strong declarative bias that is available to further constrain the hypothesis space.

$$\textbf{Blumer bound } m \geq \frac{1}{\epsilon}(ln|H| + ln\frac{1}{\delta})$$
$$\textbf{SCL's Blumer bound } m \geq \frac{1}{\epsilon}(Nln2 + ln\frac{1}{\delta}),\ |H| = 2^N$$
$$\textbf{MCL's Blumer bound } m \geq \frac{1}{\epsilon}(2^N ln2 + ln\frac{1}{\delta}),\ |H| = 2^{2^N}$$

The Blumer bounds also indicate that MCL has higher bounds on error than SCL. Because when m is fixed for a particular learning problem, the increase in $|H|$ would need to be balanced by an increase in ϵ, i.e. a larger bound on predictive error. However, the Blumer bound argument only holds for the case when the target hypothesis or its approximations are within the hypothesis space of both learning algorithms. In the case that neither the target hypothesis nor its approximations are within the hypothesis space of the incomplete learner (SCL), but within that of the complete learner (MCL), then MCL will have a lower error bound. That is why MCL is shown to have higher predictive accuracies than SCL later in the experiments of this paper. Such case also exists in other real-world applications, as demonstrated in [10].

3 MC-TopLog

This section introduces top theories first, and then explains how to derive a hypothesis using a top theory. Finally, we explain how to constrain the search space to common generalisations using the ⊤DTcD algorithm.

3.1 Top theories as declarative bias

A top theory ⊤ is a declarative bias in the form of a logic program. As a context-free grammar, a top theory consists of the terminals and non-terminals. The terminal literals are those in the hypothesis language, such as $s(X, Y)$ in Fig. 2(b); while the non-terminal literals like $\$body(X, Y)$ in Fig. 2(b) are not allowed to appear in neither the hypothesis language nor background knowledge. In order to distinguish the non-terminals, they are prefixed with the symbol '$'.

Although the non-terminals do not appear in the hypothesis language, they play important role in composing the hypothesis language. More examples of various non-terminals can be found in [9].

Composing Hypothesis Language To compose the hypothesis language from a top theory, there are two necessary operators: SLD-resolution and substitution. By applying SLD-resolution to resolving non-terminal literals, a hypothesis clause with only terminal literals can be derived. For example, a hypothesis clause $s(S1,S2) \leftarrow np(S1,S3), vp(S3,S4), np(S4,S2)$ can be derived from a SLD-derivation sequence $[Th_s, Tb_{np}, Tb_{vp}, Tb_{np}, T_{end}]$. Substitution, which is a special case of subsumption, is required to deal with ground values in clauses. For example, abductive hypotheses are ground facts, while their corresponding top theory are universally quantified, e.g. $noun([X|S],S)$ in Fig. 2(b). Non-unit clauses may need substitutions as well, such example can be seen later in the experiments.

Strong Declarative Bias Fig. 2(a) shows a mode declaration, whose corresponding version of a top theory is in Fig. 2(b). This kind of declarative bias only tells what predicates are allowed in the head/body of a hypothesis clause. However, a stronger declarative bias may exist for a learning task. In that case, it is definitely worth to use that information to further constrain the hypothesis space. For example, in the grammar learning task, we know a noun phrase always consists of a noun and a verb phrase always has a verb. This provides information about how predicates should be connected. However, there is no way for a mode declaration to capture this information, while a top theory can encode it as that in Fig. 2(c). Such a top theory will avoid deriving clauses like $np(S1,S3) \leftarrow det(S1,S2), adj(S2,S3)$, which defines a noun phrase without a noun. Another example of strong bias exists for learning tasks whose target hypothesis is known to be recursive. In that case, it would be more efficient if not to include non-recursive clauses in the hypothesis space. Apart from the strong bias about the connection of predicates, there are other strong biases, such as the restriction on function terms. For example, in Yamamoto's example of learning odd-numbers, it would be undesirable to have a clause like $odd(s(X)) \leftarrow even(s(s(X)))$ in the hypothesis space, since it will lead to the expansion of function terms during reasoning.

$modeh(1, s(+wlist, -wlist))$
$modeh(*, np(+wlist, -wlist))$
$modeh(*, vp(+wlist, -wlist))$
$modeb(1, noun(+wlist, -wlist))$
$modeb(1, verb(+wlist, -wlist))$
$modeb(*, np(+wlist, -wlist))$
$modeb(*, vp(+wlist, -wlist))$
$modeb(1, det(+wlist, -wlist))$ ...

$modeh(1, det([const|+wlist], -wlist))$
$modeh(1, noun([const|+wlist], -wlist))$
$modeh(1, verb([const|+wlist], -wlist))$

(a) Mode Declaration

Th_s: $s(X,Y) \leftarrow \$body(X,Y)$.
Th_{np}: $np(X,Y) \leftarrow \$body(X,Y)$.
Th_{vp}: $vp(X,Y) \leftarrow \$body(X,Y)$.
Tb_{noun}: $\$body(X,Z) \leftarrow noun(X,Y), \$body(Y,Z)$.
Tb_{verb}: $\$body(X,Z) \leftarrow verb(X,Y), \$body(Y,Z)$.
Tb_{np}: $\$body(X,Z) \leftarrow np(X,Y), \$body(Y,Z)$.
Tb_{vp}: $\$body(X,Z) \leftarrow vp(X,Y), \$body(Y,Z)$.
Tb_{det}: $\$body(X,Z) \leftarrow det(X,Y), \$body(Y,Z)$.
T_{end}: $\$body(Z,Z)$.
Ta_{det}: $det([X|S],S)$.
Ta_{noun}: $noun([X|S],S)$.
Ta_{verb}: $verb([X|S],S)$. ...

(b) Top Theory: Weak Declarative Bias

Th_s: $s(X,Y) \leftarrow \$body(X,Y)$.
Th_{np_noun}: $np(X,Y) \leftarrow \$body(X,M1), noun(M1,M2), \$body(M2,Y)$.
Th_{vp_verb}: $vp(X,Y) \leftarrow \$body(X,M1), verb(M1,M2), \$body(M2,Y)$.
... (The rest are the same as that in Fig. 2(b))

(c) Top Theory: Strong Declarative Bias

Fig. 2: Declarative Bias of Grammar Learning

3.2 ⊤-directed Theory Derivation (⊤DTD)

The ⊤DTD algorithm is to derive all the candidate hypotheses that explain a seed example e with respect to B, that is, hold for equation 1. It uses the top theory to direct the search for such hypotheses. Specifically, based on equation 2, explanations of e can be found using clauses in ⊤ and B. It is the use of clauses in ⊤ that make explanations of e derivable, otherwise, e cannot be explained by B alone, because of the missing clauses to be hypothesised. After deriving all the explanations (refutation proofs) of e, the ⊤ clauses used in the explanations are extracted and translated into their corresponding hypothesis clauses based on equation 3. The ⊤DTD algorithm and its corresponding cover set algorithm are given in Algorithm 1 and 2, respectively. The soundness and completeness of ⊤DTD are also given in Theorem 1 and 2. An example of how ⊤DTD works is given in Example 1.

$$B \wedge H \models e \ (e \in E^+) \tag{1}$$

$$B \wedge \top \models e \ (e \in E^+) \tag{2}$$

$$\top \models H \tag{3}$$

⊤DTD resembles Explanation-based Generalisation (EBG)[7] in that both algorithms find all possible explanations for the seed example first and then construct generalisations based on the derived explanations. However, EBG is essentially deductive learning, while ⊤DTD can achieve inductive learning. Because EBG derives its generalisations from background knowledge, while ⊤DTD's generalisations are derived from a top theory, which can compose hypothesis language that do not exist in the background knowledge.

Algorithm 1 ⊤-directed Theory Deriavation (⊤DTD)

Input: a positive example e, background knowledge B and top theory ⊤
Output: all candidate hypotheses $\mathcal{H} = \{H_i : B \wedge H_i \models e\}$
1: Let $\mathcal{H} = \emptyset$
2: Load ⊤ and B into Prolog's knowledge base
3: Query e in Prolog and record all the refutation proofs found for e into $\mathcal{R}$
4: **for all** R_i in $\mathcal{R}$ **do**
5: Extract the ⊤ clauses from R_i and obtain derivation sequences D_i
6: Translate D_i into hypothesis theory H_i
7: $\mathcal{H} = \mathcal{H} \cup H_i$
8: **end for**
9: **return** $\mathcal{H}$

Algorithm 2 Cover set algorithm of ⊤DTD

Input: examples E, background knowledge B and top theory ⊤
Output: a hypothesis H
1: Let $H = \emptyset$ and let E^+ = all positive examples in E
2: **while** $E^+ \neq \emptyset$ **do**
3: Let e be the first example in E^+
4: Construct all candidate hypotheses $\mathcal{H} = \{H_i : B \wedge H_i \models e\}$ using ⊤DTD (Algorithm 1)
5: Let H_1 be the one in $\mathcal{H}$ with highest compression;
let E' be the positive examples covered by H_1
6: $H = H \cup H_1$
7: $E^+ = E^+ - E'$
8: **end while**
9: **return** H

Theorem 1. Soundness of $\top$DTD *A theory H holds for equation 1, if it is derived by the $\top DTD$ algorithm*

Proof. *Assume the theory H derived using the $\top DTD$ algorithm does not hold for equation 1, which means $H \cup B \cup \neg e \not\models \emptyset$*

However, considering that the theory H is derived by the $\top DTD$ algorithm, H should hold for the equation 2, which means $\top \cup B \cup \neg e \models \emptyset$.

Since the only difference between $\top$ and H are the non-terminals, there must be non-terminals exist in H that are unresolvable. Otherwise, following $\top \cup B \cup \neg e \models \emptyset$, $H \cup B \cup \neg e \models \emptyset$ should hold as well.

This contradict the fact that the non-terminals do not appear in the hypothesis language and all the non-terminals in $\top$ are resolved when composing H.

Theorem 2. Completeness of $\top$DTD *Suppose a theory H is within the hypothesis space defined by $\top$, then the theory H holds for equation 1, only if it is derivable by the $\top DTD$ algorithm*

Proof. *Assume the theory H that holds for equation 1 is not derivable by the $\top DTD$ algorithm. However, according to the condition in the theorem that the theory H is within the hypothesis space defined by $\top$, equation 3 holds. Then the equation 4 holds and the equation 2 can be derived accordingly. This means refutations using clauses in $\top$ and B do exist for the example e. Then following the $\top DTD$ algorithm, the theory H can be derived, which contradicts the assumption that H cannot be derived using the $\top DTD$ algroithm.*

$$B \wedge \top \models B \wedge H \tag{4}$$

Example 1. For the learning task in Fig. 1, one of the explanations for e_1 is as shown in Fig. 3, whose corresponding SLD-refutation sequence is $R_1 = [\neg e_1, Th_s, Tb_{np}, Th_{np\_noun}, Tb_{det}, b_5, Tb_{prep}, Ta_{prep}(unknown), T_{end}, b_9, T_{end}, Tb_{vp}, b_2, b_{10}, Tb_{np}, b_1, b_{13}, b_8, T_{end}]$. Using the extraction algorithm explained in [9], D_1 consisting of three derivation sequences can be extracted from R_1. They are: $d_1 = [Th_s, Tb_{np}, Tb_{vp}, Tb_{np}T_{end}]$, $d_2 = [Th_{np\_noun}, Tb_{det}, Tb_{prep}, T_{end}, T_{end}]$ and $d_3 = [Ta_{prep}(unknown)]$. By applying SLD-derivation and substitution to D_1, $T_1 = \{h_4, h_8, h_9\}$ can be derived.

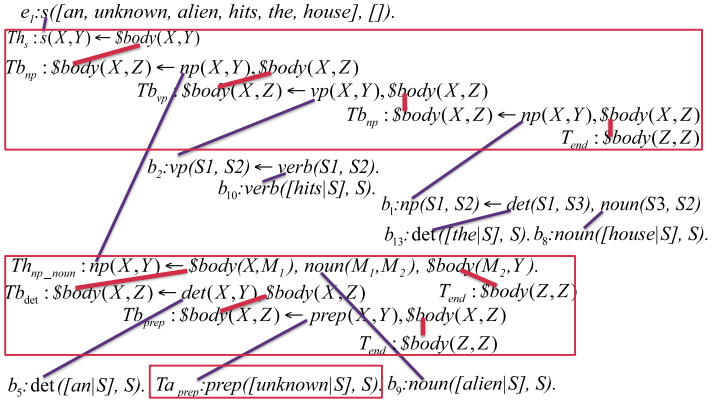


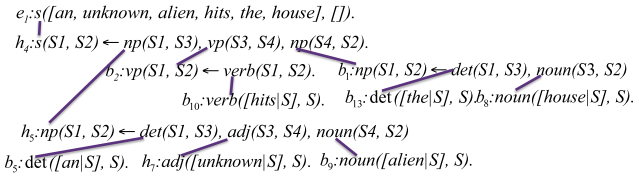
Fig. 3: An explanation of e_1 using clauses in $\top_{strong}$ and B

3.3 ⊤-directed Theory Co-Derivation (⊤DTcD)

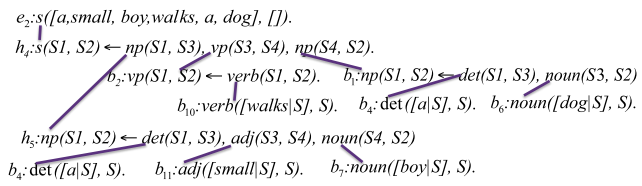
In order to constrain the derivable hypotheses to common generalisations, ⊤DTcD extends ⊤DTD based on co-refutation. Co-refutation is the combination of refutations that are the same except the instantiation of variables. Co-refutation is feasible via program transformation. Specifically, literals of the same predicate can be combined into one literal by combining their corresponding arguments into a compound. For example, the refutation proof in Fig 4(c) is the result of combining the two refutation proofs in Fig 4(a) and Fig 4(b). Co-refutation has the advantage of proving several examples together. More importantly, it proves them using the same non-ground clauses.

The design of ⊤DTcD is based on the fact that if a theory is common to multiple examples E, then the refutation proofs of each example in E using that common theory will have the same structure, that is, the proofs are the same except the instantiation of variables. Those same-structure refutation proofs can be combined into co-refutation by combining corresponding arguments. It is the combined proof that forces the co-generalised examples to be proved using the same non-ground rules.

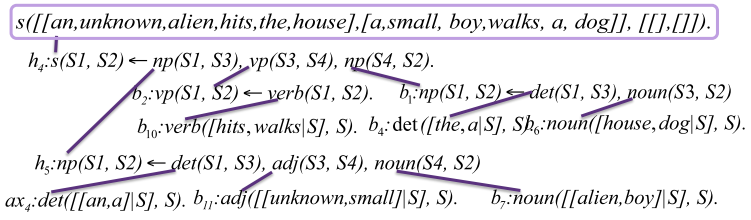
The next question is how to choose the examples to be generalised together. Rather than randomly sample a pair of examples as that in ProGolem, ⊤DTcD takes all positive examples as input, while those do not fit are filtered out along the derivation of a refutation proof. At the end of a refutation, not only a hypothesis is derived, but also the maximum set of examples that can be explained by that hypothesis.



(a) A refutation-proof of e_1



(b) A refutation-proof of e_2



(c) Co-refutation of e_1 and e_2

Fig. 4: Combine same structure refutation-proofs

The algorithm of ⊤DTcD is given in Algorithm 3. It is the same as the ⊤DTD algorithm 1 except (1) its input and output; (2) its step 3 and 4, where it combines examples to be generalised together into a compound and queries the compound instead of a single example. The cover set algorithm of ⊤DTcD is also similar to that of ⊤DTD, but they are slightly different since the output of ⊤DTcD contains the candidate hypotheses for all the positive examples, rather than just for one example. The soundness and completeness of ⊤DTcD are given in Theorem 3 and 4. We also give an example of how ⊤DTcD works in Example 2.

Algorithm 3 ⊤-directed Theory co-Deriavation (⊤DTcD)

Input: All positive examples E^+ , background knowledge B and top theory $\top$
Output: all candidate hypotheses $\mathcal{H} = \{H_i : B \wedge H_i \models E_i\}$, where $E_i \subset E$ and $|E_i| > 1$
1: Let $\mathcal{H} = \emptyset$
2: Load $\top$ and B into Prolog's knowledge base
3: Aggregate all positive examples E^+ into a compound example $e_{compound}$
4: Query $e_{compound}$ in Prolog and record all the refutation proofs found for $e_{compound}$ into $\mathcal{R}$
5: **for all** R_i in $\mathcal{R}$ **do**
6: Extract the $\top$ clauses from R_i and obtain derivation sequences D_i
7: Translate D_i into hypothesis theory H_i
8: $\mathcal{H} = \mathcal{H} \cup H_i$
9: **end for**
10: **return** $\mathcal{H}$

Algorithm 4 Cover set algorithm of ⊤DTcD

Input: examples E, background knowledge B and top theory $\top$
Output: a hypothesis H
1: Let $H = \emptyset$ and let E^+ = all positive examples in E
2: Construct all candidate hypotheses $\mathcal{H} = \{H_i : B \wedge H_i \models E_i\}$ $(E_i \subset E \wedge |E_i| > 1)$ using ⊤DTcD (Algorithm 3)
3: **while** $E^+ \neq \emptyset$ **do**
4: Let H_1 be the one in $\mathcal{H}$ with highest compression; let E' be the positive examples covered by H_1; let $\mathcal{H}'$ be the ones in $\mathcal{H}$ that only cover a subset of of E'
5: $H = H \cup H_1$
6: $E^+ = E^+ - E'$
7: $\mathcal{H} = \mathcal{H} - \mathcal{H}'$
8: **end while**
9: **return** H

Theorem 3. Soundness of TDTcD *A theory H holds for $\forall e_i \in E_j, B \wedge H \models e_i$, where $E_j \subset E^+$ and $|E_j| > 1$ if it is derived by the ⊤DTcD algorithm*

Proof. *Assume $\exists e_i \in E_j, B \wedge H \not\models e_i$, while H is derived by co-generalising E_j using the ⊤DTcD algorithm.*

Considering H is derived using ⊤DTcD algorithm, there should be co-refutations of E_j using clauses in $\top$ and B, which means $\forall e_i \in E_j, B \wedge \top \models e_i$.

Then following the ⊤DTD algorithm, a hypothesis H can be derived from $\top$. According to the soundness of topDTD, this hypothesis H also meets $B \wedge H \models e_i$, which contradicts the assumption set at the beginning, thus the theorem holds.

Theorem 4. Completeness of ⊤DTcD *Suppose a theory H is within the hypothesis space defined by $\top$, then the theory H holds for $B \wedge H \models E_j$ ($E_j \subset E^+, |E_j| > 1$ and each $e_i \in E_j$ shares the same structure of refutation proofs), only if it is derivable by the ⊤DTcD algorithm*

Proof. *Assume the $\top DTcD$ algorithm cannot derive a theory H that holds for $B \wedge H \models E_j$, where $E_j \subset E^+ \wedge |E_j| > 1$ and each $e_i \in E_j$ shares the same structure of refutation proofs. Then according to the condition in the theorem that the theory H is within the hypothesis space defined by $\top$, the equation $\top \models H$ holds, therefore the equation $B \wedge \top \models B \wedge H$ holds. Since the equation $B \wedge H \models E_j$ holds as well, the equation $B \wedge \top \models E_j$ can be derived accordingly. This means refutations using clauses in $\top$ and B do exist for the set of examples E_j that share the same structure of refutation proofs. Then following the $\top DTcD$ algorithm, the theory H can be derived, which contradicts the assumption that H cannot be derived using the $\top DTcD$ algroithm.*

Example 2. For all the positive examples in Fig. 1, the ⊤DTcD method first combines them into a compound example like s([[an,unknown,alien,hits,the,house],[a, small,boy,walks,a,dog],[a,dog,walks,into,the,house]],[[],[],[]]), and then proves this compound example using clauses in B and $\top$. In this way, we can derive the hypothesis $H_2 = \{h_4, h_5, h_7\}$ that co-generalises examples e_1 and e_2. Please note that H_2 does not cover e_3, since e_3 is filtered out during the derivation of H_2. This is visualised in Fig. 5, where it shows that e_3 is filtered out when proving the goal marked with a cross symbol in Fig. 5). This is because the second word 'dog' in e_3 is known to be a noun, rather than adjective, thus does not fit into the compound proof that derives H_2. Here also gives an example of the hypotheses that are pruned since they are not common generalisations: the hypothesis $H_1 = \{h_4, h_8, h_9\}$ derived when generalising e_1 alone is no longer derivable because it cannot generalise either e_2 or e_3, apart from e_1. Specifically, both e_2 and e_3 have their second words known as non-prepositions according to the given background knowledge, therefore they do not fit into the co-refutation using H_1.

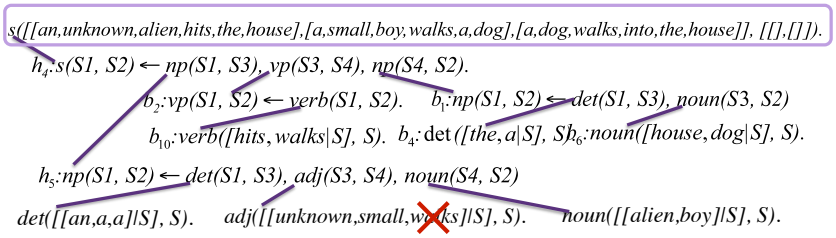


Fig. 5: Filter

4 Experiments

The null hypotheses to be empirically investigated in the study are as follows. (a) A multi-clause learning method does not have higher predictive accuracies than a single-clause learning method. MC-TopLog and Progol5 [13] are the two ILP systems used in this experiment. (b) The search space of a co-generalisation method (⊤DTcD) is not smaller than that of a solo-generalisation method (⊤DTD). All materials used in the experiments can be found at http://ilp.doc.ic.ac.uk/mcTopLog.

4.1 Grammar Learning

Materials The complete theory for parsing a grammar is in Fig. 6. The background knowledge B for each learning task is generated by randomly removing

certain number of clauses from the complete theory, and those left-out clauses form the corresponding target hypothesis. Part of the training examples are in Fig. 7. There are 50 training examples and half of them are negative. Therefore the default accuracy is 50%.

```
s(S1,S2) :- np(S1,S3), vp(S3,S4), np(S4,S2).
s(S1,S2) :- np(S1,S3), vp(S3,S4), np(S4,S5), prep(S5,S6), np(S6,S2).
np(S1,S2) :- det(S1,S3), noun(S3,S2).
np(S1,S2) :- det(S1,S3), adj(S3,S4), noun(S4,S2).
vp(S1,S2) :- verb(S1,S2).
vp(S1,S2) :- verb(S1,S3), prep(S3,S2).
det([a|S],S).      det([the|S],S).
adj([big|S],S).    adj([small|S],S).    adj([nasty|S],S).
noun([man|S],S).   noun([dog|S],S).     noun([house|S],S).    noun([ball|S],S).
verb([takes|S],S). verb([walks|S],S).   verb([hits|S],S).
prep([at|S],S).    prep([to|S],S).      prep([on|S],S).       prep([in|S],S).    prep([into|S],S).
```

Fig. 6: A Complete Theory for Parsing a Grammar

```
s([the,dog,takes,the,ball,to,the,house],[]).      ¬s([the, dog],[]).
s([the,small,dog,walks,on,the,house],[]).         ¬s([dog,the,man,the,walks],[]).
s([a,ball,hits,the,dog],[]).                      ¬s([ball,a,dog,a,hits],[]).
```

Fig. 7: Part of the Training Examples for Grammar Learning

Methods The null hypothesis(a) was investigated by comparing the learning results of MC-TopLog and Progol5[13] for randomly chosen samples. For each size of leave-out, we sampled ten times and the predictive accuracies results of ten samples were averaged. The predictive accuracies were measured by leave-one-out cross validation. The null hypothesis(b) was studied by comparing the search spaces of $\top$DTD and $\top$DTcD. The search space is measured by the number of candidate hypotheses generated during learning.

Results The x-axis in Fig. 8 corresponds to the percentage of clauses remaining in the background knowledge. The smaller the percentage, the more clauses are left-out and to be learned. The line marked with 'before' represents the predictive accuracies before learning, which shows the degree of incompleteness in the background knowledge. Progol's predictive accuracies line is above the 'before learning' line, which shows the effectiveness in learning. However, when the percentage of remaining clauses decreases to half, Progol fails to reconstruct the multiple missing clauses due to its single-clause limitation, therefore its accuracy drops to default. In contrast, MC-TopLog's ability of deriving multi-clause hypotheses makes it possible to hypothesise the missing clauses or their approximations even when half of the background knowledge is left-out. Therefore MC-TopLog's predictive accuracies are always higher than that of Progol, and their difference increases as the background knowledge becomes more incomplete. Therefore, the null hypothesis (a) is refuted. The accuracies line of MC-TopLog actually has two lines overlapped. Specifically, there is no significant difference between $\top$DTD and $\top$DTcD in terms of predictive accuracies.

Fig. 9 shows the improvement of $\top$DTcD over $\top$DTD in terms of efficiency. The solid line denotes the learning mode of singleEx which applies $\top$DTD to generalize a single example; while the dash line corresponds to the learning mode of multipleExs which applies $\top$DTcD to generalize multiple examples together. As shown in Fig. 9, the search space is reduced by more than half when the learning mode switches from singleEx to multipleExs. Therefore the null hypothesis (b) is refuted.

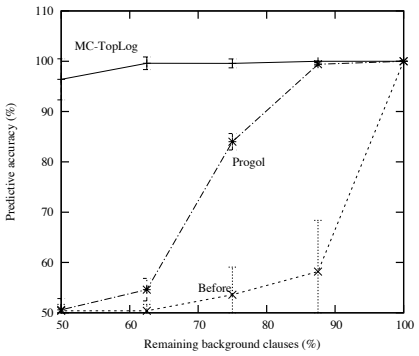


Fig. 8: Predictive Accuracies

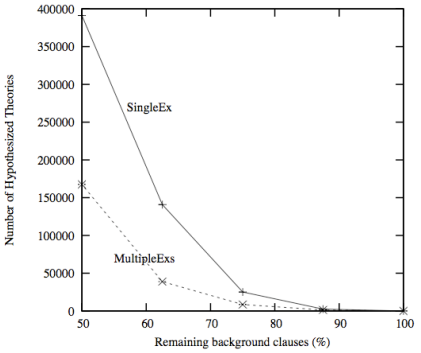


Fig. 9: Comparison of Search Spaces

4.2 Learning Game Strategies

Materials We choose the game Nim [17] for this experiment, because the target hypothesis not only has recursion, but also involves non-observable predicate learning. The learning task is to generalise a theory for identifying a P-position, which is a position that players are guaranteed to win if continue to play optimally, that is, identifying the precondition for grasping the winning strategy. Although [17] has suggested a single-clause hypothesis as $play(HeapN_1, HeapN_2, HeapN_3) \leftarrow xor(HeapN_1, HeapN_2, HeapN_3)$, this is not the target hypothesis unless the number of heaps N is fixed to be three. To handle a more general case where N is not fixed, that hypothesis is too specific and needs to be further generalised. The background knowledge available for this learning task includes the definition of mathematical functions like *and*, *or* and *xor*. The training examples are in the form of $play([3,4,5])$, in which the number sequence records the number of sticks in each heap.

$play([HeapN_1, HeapN_2, HeapN_3]) \leftarrow$
$\quad xor(HeapN_1, HeapN_2, HeapN_3).$
$play([HeapN_1, HeapN_2, HeapN_3, HeapN_4]) \leftarrow$
$\quad xor(HeapN_1, HeapN_2, MidResult),$
$\quad xor(MidResult, HeapN_3, HeapN_4).$

(a) H_s by Progol

$play(Heaps) \leftarrow compute(Heaps, 0, Result).$
$compute([Heap|Heaps], ResultSofar, Result) \leftarrow$
$\quad xor(Heap, ResultSofar, NewResultSofar),$
$\quad compute(Heaps, NewResultSofar, Result).$

(b) H_m by MC-TopLog

Fig. 10: Hypotheses suggested by different ILP systems

Methods Similar to the experiment of grammar learning, the null hypothesis(a) was investigated by comparing the learning results of MC-TopLog and Progol5. However, different from the previous experiment, the background knowledge is fixed, since its size is too small to be randomly sampled. The accuracy curves in Fig. 11 are drawn with the number of examples on the x-axis. The null hypothesis(b) was studied by comparing the search spaces and running time of $\top$DTD and $\top$DTcD. Again, we varied the number of examples to see how the search space shrinks with more examples available to be co-generalised.

Results As shown in Fig. 11, MC-TopLog only needs 6 examples to achieve accuracy of 100%, while Progol is not able to achieve accuracy of 100% even given 50 examples. Therefore the null hypothesis (a) is refuted. Progol's significantly lower accuracies results from its single-clause hypotheses which is too specific. For example, $\forall c_i \in H_s, H_m \models c_i$, where H_s and H_m are in Fig. 10(a) and 10(b), respectively. H_m not only consists of a recursive clause, but also involves a non-observable predicate 'compute', therefore even methods that can learn recursive theories (e.g. [11]) are not able to derive H_m.

MC-TopLog's accuracy line in Fig. 11 is derived under the mode of co-generalisation, while solo-generalisation is impractical for this learning task. Because there are so many mathematical functions which can be fit into a single example that the size of candidate hypotheses is much larger than what YAP (a Prolog interpreter) can handle. We not only demonstrate in Fig. 11 that ⊤DTcD is applicable for this learning task while ⊤DTD fails, but also show that the power of co-generalisation is more effective with more examples. As can be seen from Fig. 12, the number of search nodes decreases dramatically with the increasing number of examples. This is consistent with the fact that the common part of different sets shrinks as the number of sets increases. The running time also decreases accordingly. However, Fig. 13 shows that the running time slightly increases after the number of examples increases to 20. This is due to the counteracting effect of binding more variables.

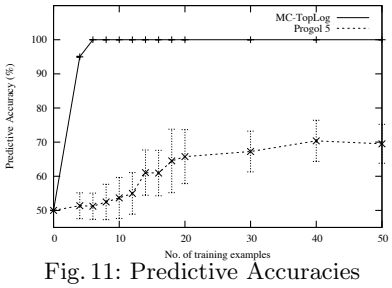


Fig. 11: Predictive Accuracies

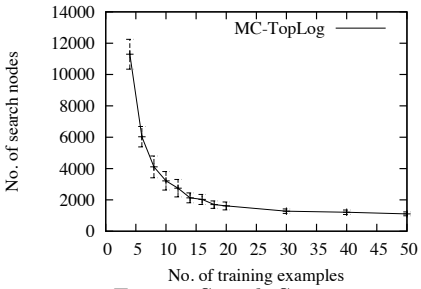


Fig. 12: Search Spaces

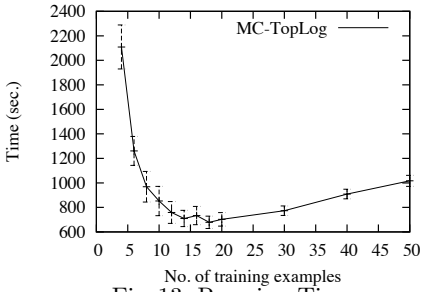


Fig. 13: Running Time

5 Conclusions and Future work

The simplified version of grammar learning shows the importance of having a complete method, even for learning problems without recursion and mutually dependent predicates. Both ⊤DTD and ⊤DTcD are sound and complete for deriving hypotheses, but ⊤DTcD is more efficient than ⊤DTD, while the improvement in efficiency is not a trade-off for its predictive accuracy. We intend to compare ⊤DTcD to other complete methods like CF-induction in future work.

Acknowledgements

This work is part of the Syngenta University Innovation Centre (UIC) on Systems Biology at Imperial College, which is funded by Syngenta. The first author also would like to thank the Royal Academy of Engineering and Microsoft for funding his present 5 year Research Chair. The authors also want to thank Changze Xu for providing the experimental data set about learning game strategies.

References

1. H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1–2):285–297, 1998.
2. A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.
3. H. Boström and P. Idestam-Almquist. Induction of logic programs by example-guided unfolding. *The Journal of Logic Programming*, 40:159–183, 1999.
4. I. Bratko. Refining complete hypotheses in ILP. In *Proceedings of ILP-99*, volume 1634, pages 44–55, Berlin, 1999. Springer-Verlag.
5. W. Cohen. Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68:303–366, 1994.
6. K Inoue. Induction as consequence finding. *Machine Learning*, 55:109–135, 2004.
7. S.T. Kedar-Cabelli and L.T. McCarty. Explanation-based generalization as resolution theorem proving. In *Proceedings of ICML-87*, pages 383–389, Los Altos, 1987. Morgan Kaufmann.
8. T. Kimber, K. Broda, and A. Russo. Induction on failure: Learning connected Horn theories. In *LPNMR 2009*, pages 169–181, Berlin, 2009. Springer-Verlag.
9. D. Lin. Efficient, complete and declarative search in inductive logic programming. Master's thesis, Imperial College London, September 2009.
10. D. Lin, J. Chen, H. Watanabe, S.H. Muggleton, P. Jain, and et al. Does multi-clause learning help in real-world applications? In *Proceedings of ILP-11*, Berlin, 2011. Springer-Verlag.
11. D. Malerba. Learning recursive theories in the normal ilp setting. *Fundamenta Informaticae*, 57:39–77, 2003.
12. S.H. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.
13. S.H. Muggleton and C.H. Bryant. Theory completion using inverse entailment. In *Proceedings of ILP-00*, pages 130–146, Berlin, 2000. Springer-Verlag.
14. S.H. Muggleton and C. Feng. Efficient induction of logic programs. In *ALT90*, pages 368–381, Tokyo, 1990. Ohmsha.
15. S.H. Muggleton, J. Santos, and A. Tamaddoni-Nezhad. Toplog: ILP using a logic program declarative bias. In *ICLP 2008*, pages 687–692, 2008.
16. S.H. Muggleton, J. Santos, and A. Tamaddoni-Nezhad. Progolem: a system based on relative minimal generalisation. In *ILP09*, pages 131–148. Springer-Verlag, 2010.
17. S.H. Muggleton and C. Xu. Can ILP learn complete and correct game strategies? In *Late-breaking proceedings of ILP*. Imperial College London Press, 2011.
18. G.D. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, August 1971.
19. L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
20. L. De Raedt, N. Lavrac, and S. Dzeroski. Multiple predicate learning. In *IJCAI*, pages 1037–1043, 1993.
21. O. Ray. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329–340, 2009.
22. J.C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 135–151. Edinburgh University Press, Edinburgh, 1969.
23. A. Yamamoto. Which hypotheses can be found with inverse entailment? In N. Lavrač and S. Džeroski, editors, *ILP97*, pages 296–308. Springer-Verlag, 1997.