Learning Logic Programs By Discovering Where Not to Search

Engineering and Physical Sciences Research Council

Andrew Cropper and Céline Hocquette

Departement of Computer Science
University of Oxford
andrew.cropper@cs.ox.ac.uk

UNIVERSITY OF OXFORD

1 - Introduction

The goal of inductive logic programming (ILP) [3] is to search for a hypothesis that generalises training examples and background knowledge (BK).

head(aaai,a)	*tail(aaai,aai)*	*even(2)*
head(ijcai,i)	*tail(ijcai,jcai)*	*even(4)*
head(ecai,e)	*tail(ecai,cai)*	*odd(1)*
tail(ai,i)	*tail(cai,ai)*	*odd(3)*

Fig. 1: Example BK.

head and *tail* are irreflexive, asymmetric, functional, antitriangular and antitransitive.
odd and *even* are mutually exclusive.

We can remove from the hypothesis space rules such as:

$r_1 = h \leftarrow tail(A,A)$
$r_2 = h \leftarrow tail(A,B), tail(B,A)$
$r_3 = h \leftarrow tail(A,B), tail(B,C), tail(A,C)$
$r_4 = h \leftarrow head(A,B), odd(B), even(B)$

We introduce an approach, implemented in Disco, which can:

1. automatically discover functional dependencies and relational properties, such as asymmetry and antitransitivity,
2. substantially reduce learning times by 97%,
3. scale to BK with millions of facts.

2 - Our approach

The key idea is to use the BK to discover constraints to restrict the hypothesis space *before* searching for a solution.

Name	Property	Constraint	Example
Irreflexive	$\neg p(A,A)$	$\leftarrow p(A,A)$	$\leftarrow brother(A,A)$
Antitransitive	$p(A,B), p(B,C) \rightarrow \neg p(A,C)$	$\leftarrow p(A,B), p(B,C), p(A,C)$	$\leftarrow succ(A,B), succ(B,C), succ(A,C)$
Antitriangular	$p(A,B), p(B,C) \rightarrow \neg p(C,A)$	$\leftarrow p(A,B), p(B,C), p(C,A)$	$\leftarrow tail(A,B), tail(B,C), tail(C,A)$
Injective	$p(A,B), p(C,B) \rightarrow A{=}C$	$\leftarrow p(A,B), p(C,B), A{\neq}C$	$\leftarrow succ(A,B), succ(C,B), A{\neq}C$
Functional	$p(A,B), p(A,C) \rightarrow B{=}C$	$\leftarrow p(A,B), p(A,C), B{\neq}C$	$\leftarrow length(A,B), length(A,C), B{\neq}C$
Asymmetric	$p(A,B) \rightarrow \neg p(B,A)$	$\leftarrow p(A,B), p(B,A)$	$\leftarrow mother(A,B), mother(B,A)$
Exclusive	$p(A) \rightarrow \neg q(A)$	$\leftarrow p(A), q(A)$	$\leftarrow odd(A), even(A)$

Table 1: Properties and constraints. We generalise the properties to higher arities.

Our approach works in two stages:

1. **Property identification**: we identify relational properties and functional dependencies [2] (Table 1) from the BK. Implemented as a bottom-up approach [5] in ASP.

   asymmetric(P) ← holds(P,(_,_)), not non_asymmetric(P)
   non_asymmetric(P) ← holds(P,(A,B)), holds(P,(B,A))

2. **Constrain**: we use the properties to build hypothesis constraints to bootstrap an ILP system [1]:

   ← asymmetric(P), b_lit(R,P,(A,B)), b_lit(R,P,(B,A))

Proposition 1 (Optimal Soundness) *The constraints built by our approach are* optimally sound*: they do not prune optimal solutions from the hypothesis space.*

3 - Experiment 1

Q1 Can BK constraint discovery reduce learning times?

Task	Popper	Disco	Change
trains1	5 ± 0.1	4 ± 0.1	**-20%**
trains2	5 ± 0.2	4 ± 0.3	**-20%**
trains3	27 ± 0.8	22 ± 0.6	**-18%**
trains4	24 ± 0.8	20 ± 0.5	**-16%**
zendo1	8 ± 2	6 ± 1	**-25%**
zendo2	32 ± 2	31 ± 2	**-3%**
zendo3	33 ± 2	31 ± 1	**-6%**
zendo4	24 ± 3	24 ± 3	0%
imdb1	1 ± 0	1 ± 0	0%
imdb2	2 ± 0.1	2 ± 0	0%
imdb3	366 ± 23	287 ± 17	**-21%**
krk	48 ± 6	9 ± 0.6	**-81%**
rps	37 ± 1	6 ± 0.2	**-83%**
centipede	47 ± 2	9 ± 0.2	**-80%**
md	142 ± 7	13 ± 0.4	**-90%**
buttons	686 ± 109	25 ± 1	**-96%**
attrition	410 ± 20	57 ± 2	**-86%**
coins	496 ± 19	345 ± 18	**-30%**
buttons-goal	11 ± 0.2	5 ± 0.1	**-54%**
coins-goal	122 ± 6	76 ± 2	**-37%**
dropk	4 ± 0.3	3 ± 0.2	**-25%**
droplast	41 ± 3	23 ± 2	**-43%**
evens	33 ± 7	9 ± 1	**-72%**
finddup	51 ± 8	32 ± 4	**-37%**
last	4 ± 0.4	3 ± 0.2	**-25%**
len	31 ± 5	16 ± 2	**-48%**
sorted	74 ± 5	23 ± 1	**-68%**
sumlist	554 ± 122	320 ± 40	**-42%**

Table 2: Learning times in seconds.

- **Disco can drastically reduce learning times.**

4 - Experiment 2

Q2 What effect does BK constraint discovery have on learning times given larger hypothesis spaces?

Task	Size	Popper	Disco	Change
md	5	12 ± 0.9	3 ± 0.3	**-75%**
md	6	113 ± 2	10 ± 0.1	**-91%**
md	7	864 ± 156	23 ± 0.9	**-97%**
md	8	*timeout*	47 ± 2	**-96%**
md	9	*timeout*	48 ± 3	**-96%**
md	10	*timeout*	52 ± 0.1	**-95%**

Table 3: Learning times (seconds) when progressively increasing the maximum rule size and thus the hypothesis space.

- **Disco can drastically reduce learning time as the hypothesis space grows relative to Popper, by up to 97%.**

5 - Experiment 3

Q3 How long does our BK constraint discovery approach take given larger BK?

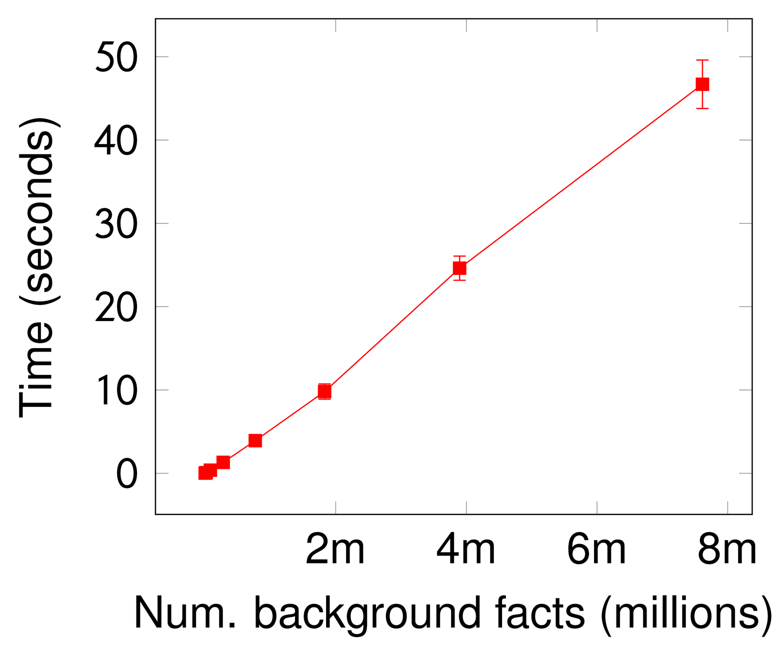

Fig. 2: BK constraint discovery time (seconds) when increasing the number of background facts.

- **Disco scales linearly in the size of the BK and can scale to millions of facts.**

6 - Conclusion

- **Bias discovery approach to improve learning performance**
- **Our approach can substantially reduce learning times.**
- **Our approach can scale to BK with millions of facts.**

Future work and Limitations:

- BK with an infinite grounding
- relax the closed-world assumption [4]
- more general properties and constraints

References

[1] A. Cropper and R. Morel. Learning programs by learning from failures. *Mach. Learn.*, 2021.

[2] H. Mannila and K-J. Räihä. Algorithms for inferring functional dependencies from relations. *Data Knowl. Eng.*, 12(1):83–99, 1994.

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

[4] R. Reiter. On closed world data bases. In *Logic and Databases, Symposium on Logic and Databases*, pages 55–76, 1977.

[5] I. Savnik and P. A Flach. Bottom-up induction of functional dependencies from relations. In *AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 174–185, 1993.

Article

Code