Learning Logic Programs by Combining Programs

Andrew Cropper and Céline Hocquette

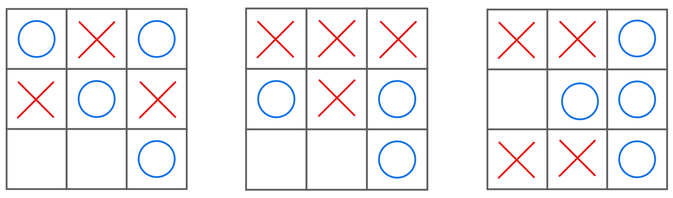
University of Oxford
andrew.cropper@cs.ox.ac.uk; celine.hocquette@cs.ox.ac.uk

1 - Introduction

The goal of inductive logic programming (ILP) is to induce a program (a set of logical rules) that generalises training examples.

Problem: learning large programs is difficult.

Example 1 (Game rules)



win(A,X) ← cell(A,0,X), cell(A,1,X), cell(A,2,X)
win(A,X) ← cell(A,0,X), cell(A,3,X), cell(A,6,X)
win(A,X) ← cell(A,0,X), cell(A,4,X), cell(A,8,X)

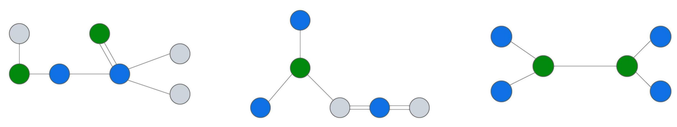
Example 2 (Program synthesis)**.** *Given the examples:*

E^+ = {contains(ecai),
contains(logic),
contains(inductive),
contains(research),
contains(program),
contains(synthesis)}
E^- = {contains(combo),
contains(oxford)}

We might want to learn a program which holds if the input string contains the letter 'a', 'e', or 'i':

contains(A) ← head(A,a)
contains(A) ← head(A,e)
contains(A) ← head(A,i)
contains(A) ← tail(A,B), contains(B)

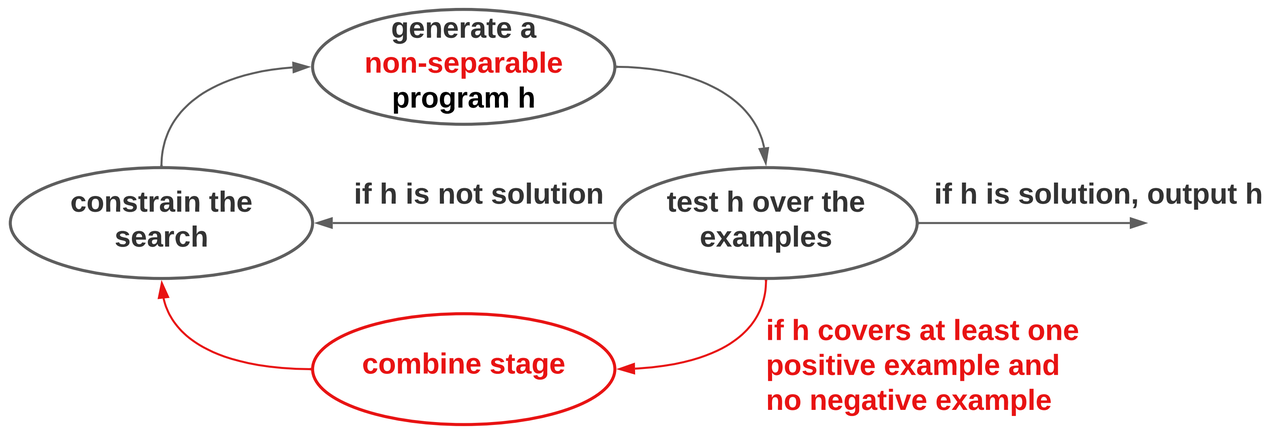
Example 3 (Drug design)



pharma(A) ← zincsite(A,B),
hydrogen_acc(A,C),
single_bond(B,C)
hydrophobic(C)
pharma(A) ← zincsite(A,B),
hydrogen_donor(A,C),
double_bond(B,C)
distance(B,C,D)
leq(D, 2.6)

2 - Our approach (COMBO)

Key idea: learn small *non-separable* programs independently and then try to combine them.



We use answer set programming to search for a combination (a union) of programs that covers as many positive examples as possible, and is minimal in size.

A program *h* is *separable* when (i) it has at least two rules, and (ii) no predicate symbol in the head of a rule in *h* also appears in the body of a rule in *h*. This program is separable:

$$\left\{\begin{array}{l} happy(A) \leftarrow rich(A) \\ happy(A) \leftarrow friend(A,B), famous(B) \\ happy(A) \leftarrow married(A,B), beautiful(B) \end{array}\right\}$$

A program is *non-separable* when it is not separable. This program is non-separable:

$$\left\{\begin{array}{l} happy(A) \leftarrow rich(A) \\ happy(A) \leftarrow married(A,B), happy(B) \end{array}\right\}$$

Why does it work? The union of the logical consequences of each rule is equivalent to the consequences of the whole program.

Why does it help? Searching over non-separable programs only can vastly reduce the hypothesis space.

3 - Experiment

Q1 Can combining non-separable programs improve learning times?
Q2 How does COMBO compare against other approaches?

Task	COMBO	POPPER	DCC	ALEPH
trains1	4 ± 0	5 ± 0	8 ± 1	**3 ± 1**
trains2	4 ± 0	82 ± 25	10 ± 1	**2 ± 0**
trains3	18 ± 1	*timeout*	*timeout*	**13 ± 3**
trains4	**16 ± 1**	*timeout*	*timeout*	136 ± 55
zendo1	3 ± 1	7 ± 1	7 ± 1	**1 ± 0**
zendo2	49 ± 5	*timeout*	3256 ± 345	**1 ± 0**
zendo3	55 ± 6	*timeout*	*timeout*	**1 ± 0**
zendo4	53 ± 11	3243 ± 359	2939 ± 444	**1 ± 0**
imdb1	**2 ± 0**	3 ± 0	3 ± 0	142 ± 41
imdb2	**3 ± 0**	11 ± 1	**3 ± 0**	*timeout*
imdb3	**547 ± 46**	875 ± 166	910 ± 320	*timeout*
krk1	28 ± 6	1358 ± 321	188 ± 53	**3 ± 1**
krk2	3459 ± 141	*timeout*	*timeout*	**11 ± 4**
krk3	*timeout*	*timeout*	*timeout*	**16 ± 3**
md	13 ± 1	3357 ± 196	*timeout*	**4 ± 0**
buttons	**23 ± 3**	*timeout*	*timeout*	99 ± 0
rps	87 ± 15	*timeout*	*timeout*	**20 ± 0**
coins	**490 ± 35**	*timeout*	*timeout*	*timeout*
buttons-g	**3 ± 0**	*timeout*	*timeout*	86 ± 0
coins-g	105 ± 6	*timeout*	*timeout*	**9 ± 0**
attrition	**26 ± 1**	*timeout*	*timeout*	678 ± 25
centipede	**9 ± 0**	1102 ± 136	2104 ± 501	12 ± 0
adj_red	**2 ± 0**	5 ± 0	6 ± 0	479 ± 349
connected	**5 ± 1**	112 ± 71	735 ± 478	435 ± 353
cyclic	**35 ± 13**	1321 ± 525	1192 ± 456	1120 ± 541
colouring	**2 ± 0**	6 ± 0	5 ± 0	2373 ± 518
undirected	**2 ± 0**	6 ± 0	6 ± 0	227 ± 109
2children	**2 ± 0**	7 ± 0	6 ± 0	986 ± 405
dropk	7 ± 3	17 ± 2	14 ± 2	**4 ± 1**
droplast	**3 ± 0**	372 ± 359	13 ± 1	763 ± 67
evens	3 ± 0	29 ± 3	25 ± 2	**2 ± 0**
finddup	11 ± 5	136 ± 14	149 ± 7	**0.8 ± 0**
last	**2 ± 0**	12 ± 1	11 ± 1	**2 ± 0**
contains	**17 ± 0**	299 ± 52	158 ± 48	64 ± 5
len	3 ± 0	52 ± 5	45 ± 2	**2 ± 0**
reverse	40 ± 5	1961 ± 401	1924 ± 300	**3 ± 0**
sorted	127 ± 78	111 ± 11	131 ± 10	**1 ± 0**
sumlist	4 ± 0	256 ± 27	221 ± 12	**0 ± 0**

Table 1: Learning times (s) with a 60 minutes timeout.

- **Learning non-separable programs and combining them can drastically improve learning performance.**

4 - Limitation

- Learning programs from noisy examples by finding combinations that cover as many positive examples and as few negative examples as possible.

Article