Learning logic programs by finding minimal unsatisfiable subprograms

Andrew Cropper and Céline Hocquette
University of Oxford
andrew.cropper@cs.ox.ac.uk; celine.hocquette@cs.ox.ac.uk

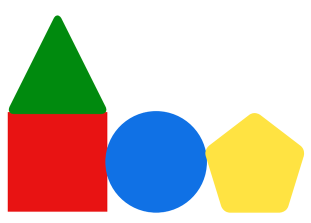
UK Research and Innovation

1 - Introduction

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

In this work, we identify minimal unsatisfiable subprograms (MUSPs) to prune the search space.

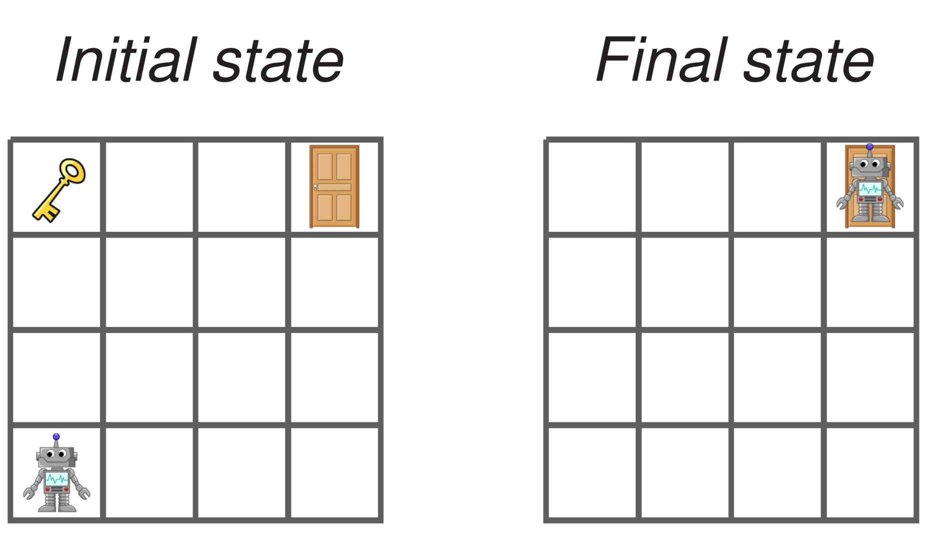
Example 1 (Zendo)



$h = \{ \leftarrow \text{red(Piece), blue(Piece)} \}$

h is an unsatisfiable subprogram because no piece is both red and blue. Therefore, we eliminate from the search space all specialisations of h.

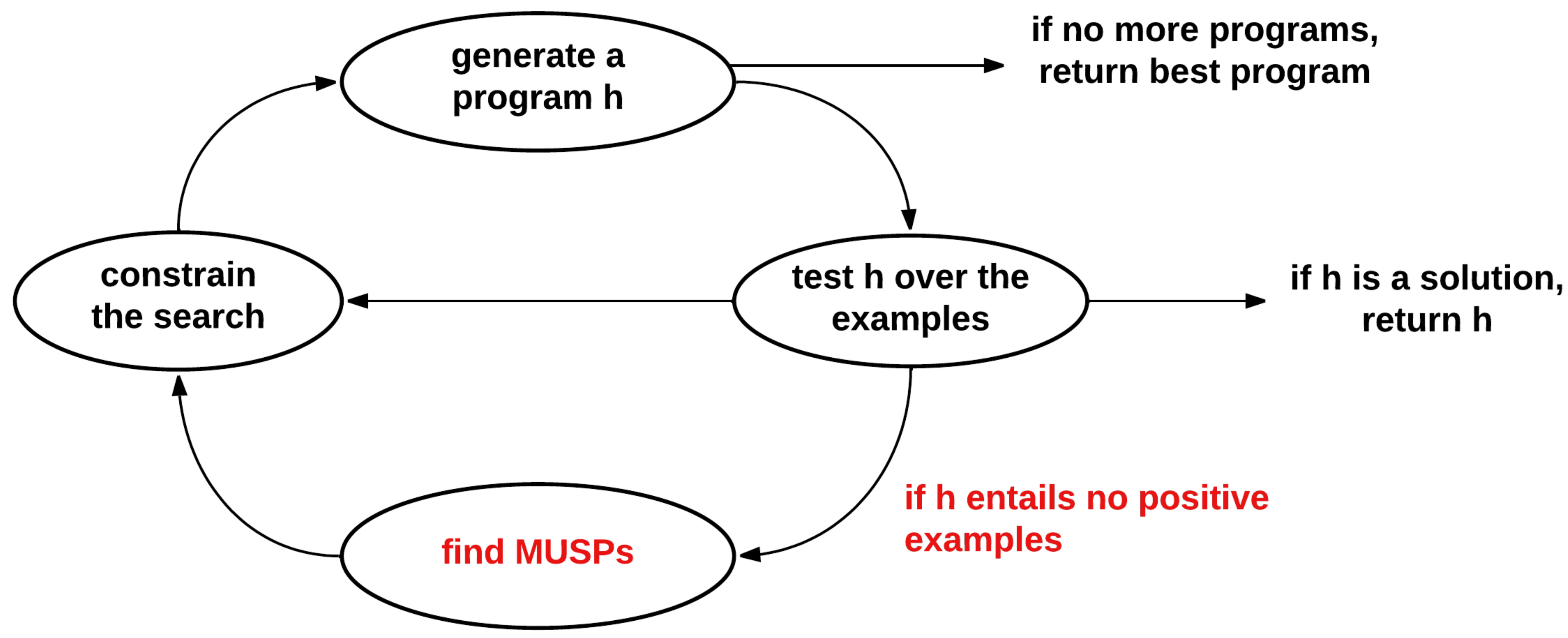
Example 2 (Robot strategy)



$h = \{ \leftarrow \text{move\_left(Initial,State)} \}$

h is an unsatisfiable subprogram because the robot cannot move left from the initial state. Therefore, we eliminate from the search space all specialisations of h.

2 - Our approach (MUSPer)



A program is *unsatisfiable* if it does not entail any positive example.

Key idea: identify the MUSPs of unsatisfiable programs and build constraints from them.

Consider the positive examples $E^+ = \{f([\,],0), f([e,c,a,i],4)\}$, an appropriate BK, and the program:

$$h = \{ \; f(A,B) \leftarrow empty(A),\, head(A,B),\, tail(A,C),\, head(C,B) \; \}$$

h has the MUSPs:

$$\{ \leftarrow empty(A),\, head(A,B) \}$$
$$\{ \leftarrow empty(A),\, tail(A,C) \}$$
$$\{ f(A,B) \leftarrow head(A,B) \}$$

These programs are MUSPs of h because:

1. they are subprograms of h,
2. they are unsatisfiable, and
3. there are no unsatisfiable subprograms of h with strictly smaller size.

We prune specialisations of each of these MUSPs, such as:

$$\{ \leftarrow empty(A),\, head(A,B),\, one(B) \}$$
$$\{ \leftarrow empty(A),\, tail(A,C),\, tail(C,D),\, tail(D,E) \}$$
$$\{ f(A,B) \leftarrow head(A,B),\, tail(B,C) \}$$

3 - Theoretical Analysis

Theorem 1 Constraints from MUSPs are optimally sound.

Theorem 2 Using MUSPs to build constraints leads to more pruning.

4 - Experiment

Q1 Can identifying MUSPs reduce learning times?
Q2 How does MUSPer compare to other approaches?

Domain	Popper	MUSPer	Change	Disco	MUSPer	Change
trains	13 ± 1	13 ± 1	0%	14 ± 1	13 ± 1	**–7%**
zendo	36 ± 4	36 ± 4	0%	40 ± 5	36 ± 4	**–10%**
imdb	193 ± 51	148 ± 39	**–23%**	250 ± 73	148 ± 39	**–40%**
iggp	617 ± 66	207 ± 34	**–66%**	583 ± 66	207 ± 34	**–64%**
graph	12 ± 5	8 ± 2	**–33%**	8 ± 2	8 ± 2	0%
synthesis	343 ± 39	199 ± 33	**–41%**	327 ± 38	199 ± 33	**–39%**
sql	594 ± 63	13 ± 1	**–97%**	505 ± 58	13 ± 1	**–97%**

Table 1: Learning times (seconds).

Q1 Identifying MUSPs can drastically improve learning times whilst maintaining high predictive accuracies.
Q2 MUSPer can substantially improve learning performance, both in terms of predictive accuracies and learning times, compared to existing approaches.

5 - Conclusion

- **An approach that identifies MUSPs to prune the search space.**

Article