Meta-interpretive learning of efficient logic programs

Andrew Cropper
Department of Computing
Imperial College London
a.cropper13@imperial.ac.uk

Stephen H. Muggleton
Department of Computing
Imperial College London
s.muggleton@imperial.ac.uk

Abstract

Most forms of program induction cannot distinguish between the efficiencies of programs, such as mergesort ($O(n\ log\ n)$) and bubblesort ($O(n^2)$), and instead rely on an Occamist bias to learn simple programs, typically based on textual complexity. To address this issue, we introduce techniques to learn efficient logic programs with minimal resource complexity.

1 Introduction

Suppose we are machine learning robot plans from initial/final state examples. Figure 1 shows a scenario where a robot is learning to move a ball from square 1/1 to square 3/3. Assume the robot can move *north*, *south*, *east*, and *west*, and can *grab* and *drop* the ball, then Figure 2 shows two plans with corresponding Prolog programs for this problem. Although both programs transform the initial state to the final state and both have the same textual complexity, they differ in their efficiencies. Program (a) is inefficient because it involves two *grab* and two *drop* operations, whereas program (b) is efficient because it requires only one *grab* and one *drop* operation.

(a) Initial state (b) Final state

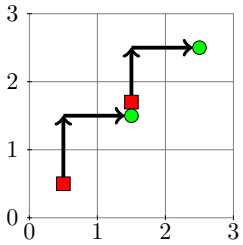
Figure 1: Robot planning example

However, most forms of program induction cannot distinguish between the efficiencies of programs, and instead rely on an Occamist bias to learn programs with minimal textual complexity. We address this issue by introducing techniques to learn minimal resource complexity logic programs. Resource complexity is a generalisation of the notion of time-complexity of algorithms, in which time is a particular resource. Our main contribution, thus far, is the introduction of a learning algorithm proven to learn minimal resource complexity robot strategies.

2 Completed work

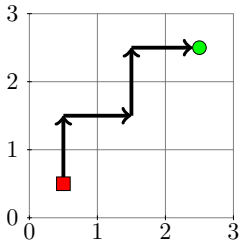
In [1], we describe an approach to learn optimal resource complexity robot strategies based on the meta-interpretive learning (MIL) framework [3, 4, 2], a form of inductive logic programming

1st Workshop on Neural Abstract Machines & Program Induction (NAMPI), @NIPS 2016, Barcelona, Spain.




```
move(X,Y):-p3(X,Z),p3(Z,Y).
p3(X,Y):-p2(X,Z),drop(Z,Y).
p2(X,Y):-grab(X,Z),p1(Z,Y).
p1(X,Y):-north(X,Z),east(Z,Y).
```

(a) Inefficient solution




```
move(X,Y):-p3(X,Z),drop(Z,Y).
p3(X,Y):-grab(X,Z),p2(Z,Y).
p2(X,Y):-p1(X,Z),p1(Z,Y).
p1(X,Y):-north(X,Z),east(Z,Y).
```

(b) Efficient solution

Figure 2: Prolog programs for the planning example in Figure 1. A red square denotes a *grab* action and a green circle denotes a *drop* action.

which supports predicate invention and recursion. A strategy is a logic program composed of actions and fluents which transforms an initial state to a final state. The resource complexity of a strategy is the sum of the action costs in applying a strategy to an example. We introduced Metagol$_O$, an implementation of the MIL framework, and proved that given sufficient numbers of examples, Metagol$_O$ converges on resource optimal strategies. Our experimental results support this claim and show, for instance, that when learning to sort lists, Metagol$_O$ learns an efficient quick sort strategy, rather than an inefficient bubble sort strategy.

3 Conclusions and future work

By focusing on robot strategies, we have made an initial attempt at learning efficient logic programs. We intend to generalise the approach to a broader class of logic programs. In [1], the resource complexity of a hypothesised strategy is maintained in the state description. Each dyadic action has an input state as the first term and an output state as the second term. Executing a dyadic action increments the resource cost in the input state to form the output state. However, predicates in logic programs do not not necessarily have *input* and *output* arguments, for instance when learning a monadic predicate. Therefore, to generalise the approach to arbitrary logic programs, we need a more general representation to calculate the resource complexity. In addition, we have assumed a user-provided function to assign resource costs to each robot action. However, such information is not necessarily available and in future work we intend to investigate whether we can learn efficient time-complexity algorithms without user-provided costs.

This work demonstrates that it is possible to learn efficient programs and opens new avenue of research, such as allowing algorithm designers to discover novel efficient algorithms, and for software engineers to automatically build efficient software.

References

[1] Andrew Cropper and Stephen H. Muggleton. Learning efficient logical robot strategies involving composable objects. In *IJCAI*, pages 3423–3429. AAAI Press, 2015.

[2] Andrew Cropper and Stephen H. Muggleton. Learning higher-order logic programs through abstraction and invention. In *IJCAI*, pages 1418–1424. IJCAI/AAAI Press, 2016.

[3] Stephen H. Muggleton, Dianhuan Lin, Niels Pahlavi, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94(1):25–49, 2014.

[4] Stephen H. Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.