Learning efficient programs

Andrew Cropper and Stephen H. Muggleton

Department of Computing, Imperial College London

Abstract. Program induction approaches typically rely on an Occamist bias to select programs with minimal textual complexity. This bias is inadequate when the efficiencies of programs differ, such as permutation sort ($O(n!)$) and merge sort ($O(n\ log\ n)$). We address this issue by introducing an inductive logic programming approach which given sufficient numbers of examples is proven to learn minimal *resource complexity* robot strategies. Our experiments demonstrate learning of optimal strategies in all cases, in contrast to existing approaches, which learn non-optimal strategies. We also propose future work to generalise the approach to a broader class of programs.

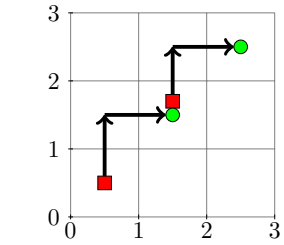
1 Introduction

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

(a) Initial state (b) Final state

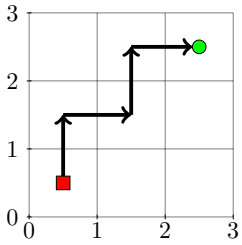
Fig. 1: Robot planning example

However, most program induction approaches rely on an Occamist bias to select programs with minimal textual complexity and cannot distinguish between the efficiencies of programs, such as merge sort ($O(n\ log\ n)$), and bubble sort ($O(n^2)$). We address this issue by introducing an inductive logic programming (ILP) [5] technique 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




```
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

Fig. 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.

particular resource. Our main contribution is the introduction of a framework for minimising the resource complexity of logical robot strategies and the demonstration of a learning algorithm which given sufficient numbers of examples is proven to learn minimal resource complexity strategies.

2 Completed work

In [2], we describe an approach to learn optimal resource complexity robot strategies based on the meta-interpretive learning (MIL) framework [6, 7, 1, 3], a form of ILP which supports predicate invention and learning recursive programs. A strategy is a logic program composed of actions and fluents which transforms an initial state S_i to a final state S_j. The resource complexity of a strategy is defined as follows.

Definition 1. *Let $e = \langle S_1, S_2 \rangle$ be an example where S_1 and S_2 are initial/final states respectively and H a strategy. Then the resource complexity $r(H(e))$ is the sum of the action costs in applying the strategy H to e to transform S_1 to S_2.*

We assume a user-provided function $r : P_a \rightarrow \mathbf{N}$ which defines the resource cost of calling an action $p \in P_a$. In robot strategies, energy consumption and consumption of materials may be considered as resources. We define the resource complexity of a strategy over a set of examples.

Definition 2. *Let E^+ be a set of positive examples. Then the resource complexity of a strategy H is defined as follows:*

$$r(H, E^+) = arg\,max_{e \in E^+} r(H(e))$$

To find the minimal resource complexity strategy given a set of examples, we introduced Metagol$_O$, an implementation of the MIL framework which uses *iterative descent* to find resource optimal strategies.

The main idea of iterative descent is to first find the minimal textual complexity strategy H_1, which is the most tractable to learn because the hypothesis space is exponential in the solution length [4]. The resource complexity $r(H_1, E^+)$ of strategy H_1 provides an upper bound from which to descend, i.e. to search for a more efficient strategy with a resource complexity less than H_1. In [2], we prove convergence of this search procedure to resource optimal strategies. Our experimental results agree with the theoretical optimal predictions 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 inducing efficient programs. We intend to generalise the approach to a broader class of logic programs. In [2], the resource complexity of a hypothesised strategy is maintained in the state description. Each dyadic action increases 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 logic programs, we need a more general representation to calculate the resource complexity. In addition, we assume a user-provided function to assign 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.

Overall, 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. Logical minimisation of meta-rules within meta-interpretive learning. In *ILP*, volume 9046 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2014.
2. Andrew Cropper and Stephen H. Muggleton. Learning efficient logical robot strategies involving composable objects. In *IJCAI 2015*, pages 3423–3429, 2015.
3. Andrew Cropper and Stephen H. Muggleton. Learning higher-order logic programs through abstraction and invention. In *IJCAI*, pages 1418–1424. IJCAI/AAAI Press, 2016.
4. D. Lin, E. Dechter, K. Ellis, J.B. Tenenbaum, and S.H. Muggleton. Bias reformulation for one-shot function induction. In *Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI 2014)*, pages 525–530, Amsterdam, 2014. IOS Press.
5. S.H. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8(4):295–318, 1991.
6. S.H. Muggleton, D. Lin, N. Pahlavi, and A. Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94:25–49, 2014.
7. 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.