Learning Efficient Logical Robot Strategies Involving Composable Objects

Andrew Cropper and Stephen H. Muggleton
Imperial College London
United Kingdom
{a.cropper13,s.muggleton}@imperial.ac.uk

Abstract

Most logic-based machine learning algorithms rely on an Occamist bias where textual complexity of hypotheses is minimised. Within Inductive Logic Programming (ILP), this approach fails to distinguish between the efficiencies of hypothesised programs, such as quick sort ($O(n\ log\ n)$) and bubble sort ($O(n^2)$). This paper addresses this issue by considering techniques to minimise both the textual complexity and *resource complexity* of hypothesised robot strategies. We develop a general framework for the problem of minimising resource complexity and show that on two robot strategy problems, 1) Postman 2) Sorter (recursively sort letters for delivery), the theoretical resource complexities of optimal strategies vary depending on whether objects can be composed within a strategy. The approach considered is an extension of Meta-Interpretive Learning (MIL), a recently developed paradigm in ILP which supports predicate invention and the learning of recursive logic programs. We introduce a new MIL implementation, $Metagol_O$, and prove its convergence, with increasing numbers of randomly chosen examples to optimal strategies of this kind. Our experiments show that $Metagol_O$ learns theoretically optimal robot sorting strategies, which is in agreement with the theoretical predictions showing a clear divergence in resource requirements as the number of objects grows. To the authors' knowledge this paper is the first demonstration of a learning algorithm able to learn optimal resource complexity robot strategies and algorithms for sorting lists.

1 Introduction

Commercial robots exist which carry out tasks such as vacuuming a cluttered room [Geringer *et al.*, 2012] or delivering packages [Felder *et al.*, 2003]. However, consider teaching a robot postman to both collect and deliver letters. In the initial state letters are to be collected; in the final state letters have been delivered to their intended destinations (Figure 1). In Section 3 we show that allowing the postman to form *composite objects* by placing letters in a postbag reduces the *resource complexity* of the problem from $O(n+d)$ to $O(nd)$, where n is the number of letters and d is the space size.

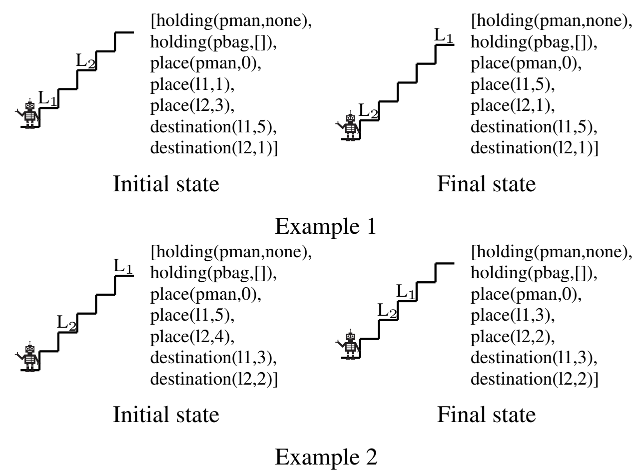

Figure 1: Postman initial/final state examples alongside Prolog representations for a route on a hill. In the initial states letters are to be collected; in the final states letters are at their intended destinations.

However, most logic-based machine learning algorithms rely on an Occamist bias where textual complexity of hypotheses is minimised. Within Inductive Logic Programming [Muggleton *et al.*, 2011] (ILP), this approach fails to distinguish between the efficiencies of hypothesised programs, such as quick sort ($O(n\ log\ n)$) and bubble sort ($O(n^2)$)[1]. For example, Figure 2 shows two strategies from our experiments (Section 5.2) for the postman problem, where strategy (a) was learned by Metagol$_D$[Muggleton *et al.*, 2015], an existing ILP system and strategy (b) was learned by Metagol$_O$, a new ILP system introduced in Section 4. Although the strategies are equal in textual complexity they differ in their resource complexity[2] because Metagol$_O$ minimises both the textual complexity and resource complexity of hypotheses, learning a strategy involving composite objects (i.e. using a

[1]We demonstrate that Metagol$_O$ succeeds in learning quick sort in the experiments described in Section 5.3.

[2]$O(nd)$ vs $O(n+d)$ respectively where n, d are the number of letters and places for delivery.

postbag), whereas Metagol$_D$ only minimises the textual complexity of hypotheses. Clearly, reducing the complexity of such problems and learning efficient strategies for them is of considerable value in similar real-world applications. In Section 5 we show how strategy optimisation can be used to learn classic efficient sorting algorithms as robot strategies.

```
postman(A,B):- postman2(A,C), postman(C,B).
postman(A,B):- postman2(A,C), go_to_bottom(C,B).
postman2(A,B):- postman1(A,C), go_to_bottom(C,B).
postman1(A,B):- find_next_sender(A,C), take_letter(C,B).
postman1(A,B):- find_next_recipient(A,C), give_letter(C,B).
```

(a) Inefficient strategy learned by Metagol$_D$

```
postman(A,B):- postman2(A,C), postman2(C,B).
postman2(A,B):- postman1(A,C), postman2(C,B).
postman1(A,B):- find_next_sender(A,C), bag_letter(C,B).
postman2(A,B):- postman1(A,C), go_to_bottom(C,B).
postman1(A,B):- find_next_recipient(A,C), give_letter(C,B).
```

(b) Efficient strategy learned by Metagol$_O$

Figure 2: Prolog recursive postman strategies learned by Metagol$_D$ (a) and Metagol$_O$ (b) with resource complexities $O(nd)$ and $O(n + d)$ respectively. Predicates *postmani* are invented and are local to their respective strategy.

Action	**Cost**
find_next_sender	1
take_letter	1
go_to_bottom	1
find_next_recipient	5
give_letter	1
go_to_bottom	5
find_next_sender	3
take_letter	1
go_to_bottom	3
find_next_recipient	1
give_letter	1
go_to_bottom	1
Total	24

(a)

Action	**Cost**
find_next_sender	1
bag_letter	1
find_next_sender	2
bag_letter	1
find_next_recipient	2
give_letter	1
go_to_bottom	5
find_next_recipient	1
give_letter	1
go_to_bottom	1
Total	16

(b)

Table 1: Traces of inefficient (a) and efficient (b) strategies shown in Figure 2 for example 1 in Figure 1.

This paper extends the recently developed Meta-Interpretive Learning (MIL) framework [Lin *et al.*, 2014; Muggleton *et al.*, 2014b]. MIL differs from most state-of-the-art ILP approaches by supporting the use of predicate invention for problem decomposition and the learning of recursive programs. MIL has [Muggleton *et al.*, 2015] been shown to be capable of learning a simple robot *strategy* for building a stable wall from a small set of initial/final state pairs. The learning of such a strategy differs from traditional AI planning techniques which involve the generation of a *plan* as a sequence of actions transforming a particular initial state of the world to a particular final state. By contrast, a *strategy* can be viewed as a (potentially infinite) set of plans. Once learned, such a strategy can be applied without invoking a search through a space of plans. MIL is extended in this paper towards finding programs which have low *resource complexity*. That is, use of resources (eg number of moves, power dissipation etc) is near minimal for the learned strategy over the entire class of initial states.

2 Related work

In AI, planning traditionally involves deriving a sequence of actions to achieve a specific goal from an initial situation [Russell and Norvig, 2010]. The majority of research into AI planning has focused on developing efficient planners, i.e. systems which efficiently learn a plan to accomplish a goal. However, we are often interested in plans that are optimal with respect to an objective function by which the quality of a plan is measured. A common objective function is the length of the plan, i.e. the number of time steps to achieve the goal [Xing *et al.*, 2006]. Plan length alone is only one criterion to be optimised [Eiter *et al.*, 2003]. If executing actions is costly, we may desire a plan which minimises the overall cost of the actions, e.g. to minimise the use of limited resources such energy capacity. The Answer Set Programming literature has already addressed learning optimal plans by incorporating action costs into the learning [Eiter *et al.*, 2003; Yang *et al.*, 2014].

In contrast to these approaches, this paper investigates the construction of strategies which consist of a program containing conditional execution and recursion representing a (sometimes infinite) set of plans. Various machine learning approaches support the construction of strategies, including the SOAR architecture [Laird, 2008], reinforcement learning [Sutton and Barto, 1998], learning from traces [Lau *et al.*, 2003], learning by demonstration [Argall *et al.*, 2009], learning by imitation [Hayes and Demiris, 1994], policy abstraction [Pineau *et al.*, 2002]. and action learning within ILP [Moyle and Muggleton, 1997; Otero, 2005].

Relational Markov Decision Processes [van Otterlo and Wiering, 2012] provide a general setting for reinforcement learning. Strategies can be viewed as a deterministic special case of Markov Decision Processes (MDPs) [Puterman, 2014]. Unlike most work in learning MDPs, MIL involves generation of readable programs by way of problem decomposition using predicate invention and the learning of recursive definitions. This has allowed it to be used in this paper for the classic programming optimisation task of finding optimal solutions for sorting lists (see Section 5.3). While reinforcement learning addresses heuristic policy optimisation it does not generally provide provable convergent means for finding optimal programs.

Strategy learning can also be classified as a form of inductive programming [Gulwani *et al.*, 2015], in which functional and logic programs are learned from example presentations of input/output pairs. In this case, MIL is unusual in its use of predicate invention for learning recursive definitions. Problem decomposition has been found to be valuable for reusability of sub-programs [Lin *et al.*, 2014], which has also been explored previously in a heuristic form in Genetic Programming [Koza and Rice, 1994].

Our experiments involve robot strategies where objects are composed by robots storing objects in containers, thus increasing carrying efficiency. The idea of composition of objects appears in both the planning [Cambon *et al.*, 2004] and Natural Language literature. For instance, [Eugenio, 1991] describes the action *place a plank between two ladders to create a simple scaffold*.

3 Theoretical framework

3.1 Meta-Interpretive Learning

MIL [Muggleton *et al.*, 2014b; 2015] is a form of ILP based on an adapted Prolog meta-interpreter. Whereas a standard Prolog meta-interpreter attempts to prove a goal by repeatedly fetching first-order clauses whose heads unify with a given goal, a MIL learner attempts to prove a set of goals by repeatedly fetching higher-order metarules (Table 2) whose heads unify with a given goal. The resulting meta-substitutions are saved in an abduction store, and can be re-used in later proofs. Following the proof of a set of goals, a hypothesis is formed by applying the meta-substitutions onto their corresponding metarules, allowing for a form of ILP which supports predicate invention and the learning of recursive theories.

3.2 Resource complexity

In this section we outline a framework for describing the resource complexity of robot strategies. Resource complexity can be vewied as a generalisation of the notion of time-complexity of algorithms, in which time can be viewed as a particular resource. In robot strategies energy consumption and consumption of materials such as solder, glue, or bricks might also be considered as resources.

General formal framework Let O represent an enumerable set of objects in the world. Objects are separated into two disjoint sets O_0 (primitive objects) and O_1 (composite objects) where $O = O_0 \cup O_1$. Composite objects are defined in terms of primitives and other composite objects. Let P represent an enumerable set of places in the world. Let S represent an enumerable set of situations where each situation is a pair $\langle p, o\rangle$ referring to the place $p \in P$ where the object $o \in O$ can be found. In any situation, an object is paired with only one place. Let A represent an enumerable set of actions. Each action a in A is a function where $a : S \rightarrow S$. Action names are separated into two disjoint sets A_0 (primitive actions) and A_1 (composite actions) where $A = A_0 \cup A_1$. Composite actions are defined in terms of primitives and other composite actions. We assume a resource function $r : A \times S \rightarrow \mathbf{N}$ which defines the resources consumed by carrying out action $a \in A$ in situation $s \in S$.

Example 1 Robot Postman (composable). *In the Postman example (Section 1) we have* O_0 = {*letter1, letter2, postman, postbag*}, O_1 = {*postbag containing letter1,* ...}, P = {*place1, place2, place3,* ...}, S = {*<place1,postman>*, ...}, A_0 = {*move_up, move_down,* ...}, A_1 = {*go_to_bottom, go_to_top,* ...}. *For simplicity we assume the resource function gives* $r(a, s) = 1$ *in all cases in which* $a \in A_0$.

Generalised meta-interpreter
$prove([], Prog, Prog).$ $prove([Atom\|As], Prog1, Prog2) :-$ $metarule(Name, MetaSub, (Atom \text{ :- } Body), Order),$ $Order,$ $abduce(metasub(Name, MetaSub), Prog1, Prog3),$ $prove(Body, Prog3, Prog4),$ $prove(As, Prog4, Prog2).$

Figure 3: Prolog code for generalised meta-interpreter

We now give a resource complexity bound for the Postman.

Theorem 1 (Postman composable object bound) *Let* n *be the cardinality of* O_0 *and* d *be the cardinality of* P *in the Postman problem and* a *be a composite action which minimises* $r(a)$ *for* $a \in A_1$. *In this case* $r(a)$ *is* $O(d + n)$.
Proof. The optimal strategy involves the postman using the postbag to hold all n *objects. This involves at most* d *steps for traversal and* n *object pick-ups. The postman then needs to deliver each object to its destination, which again involves at most* d *steps for traversal and* n *object placements. Thus the overall time is bounded by* $2(n + d)$, *which is* $O(n + d)$.

Suppose in the above that the postman is disabled from composing objects by adding them to the postbag. In this case the resource complexity for the task is different.

Theorem 2 (Postman non-composable object bound.) *Let* n, d *and* a *be as in Theorem 1, but objects are limited to* O_0. *In this case* $r(a)$ *is* $O(nd)$.
Proof. The optimal strategy involves the postman repeatedly finding and picking up an object and then moving to its destination and placing it there. This involves at most $2d$ *steps for finding and carrying the object plus one pick-up and placement. Thus for all* n *objects this involves a resource of* $n(2d + 2)$ *which is* $O(nd)$.

The difference in the complexities demonstrated in Theorem 1 and 2 show that differences in optimal strategy solutions can exist with assumptions associated with composability of objects. In the experiments described in Section 5 we show that such differences are evident in the output of solutions generated by Metagol$_O$.

4 Implementation

This section describes Metagol$_O$, a variant of Metagol$_D$ [Muggleton *et al.*, 2015], an implementation of the MIL framework which learns programs within the Datalog subset H^2_2, where H^i_j consists of definite Datalog logic programs with predicates of adicity at most i and at most j literals in the body of each clause.

4.1 Metagol$_O$

Figure 3 shows the implementation of Metagol$_O$[3] as a generalised meta-interpreter, similar in form to a standard Prolog meta-interpreter. The metarule set (Table 2) is defined separately, with each metarule having an associated name

[3]Full code for Metagol$_O$ together with all materials for the experiments is available at http://ilp.doc.ic.ac.uk/metagolO.

($Name$), quantification ($MetaSub$), form ($Atom$:-$Body$) and Herbrand ordering constraint ($Order$). Owing to the Turing-expressivity of H_2^2 it is necessary [Muggleton *et al.*, 2015] to constrain ($Order$) the application of the metarules to guarantee termination of the hypothesised program. The termination guarantees are based on these constraints being consistent with a total ordering over the Herbrand base of the hypothesised program. Thus the constraints ensure that the head of each clause is proved on the basis of instances of body atoms lower in the ordering over the Herbrand base. Since the ordering is not infinitely descending, this guarantees termination of the meta-interpreter.

Name	**Metarule**	**Order**
Base	$P(x, y) \leftarrow Q(x, y)$	$P \succ Q$
Chain	$P(x, y) \leftarrow Q(x, z), R(z, y)$	$P \succ Q, P \succ R$
TailRec	$P(x, y) \leftarrow Q(x, z), P(z, y)$	$P \succ Q$, $x \succ z \succ y$

Table 2: Metarules with associated Herbrand ordering constraints where $\succ$ is a pre-defined ordering over symbols in the signature. The uppercase letters P, Q, and R denote existentially quantified higher-order variables. The lowercase letters x, y, and z denote universally quantified first-order variables.

4.2 Iterative descent

The key difference between Metagol$_O$ and Metagol$_D$ is the search procedure. Metagol$_D$ uses iterative deepening to ensure that the first hypothesis returned contains the minimal number of clauses. The search starts at depth 1. At depth i the search returns a consistent hypothesis with at most i clauses if one exists. Otherwise it continues to depth $i + 1$. Metagol$_D$ minimises the textual complexity of the hypothesis rather than resource complexity, which we now define.

Definition 1 (Resource complexity) *Assume the training examples E consist of positive examples E^+ and negative examples E^-. Furthermore a hypotheses $H \in \mathcal{H}$ is a robot strategy chosen from the hypothesis space. The resource complexity of hypothesis H on example set E^+ is*

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

where $r(H(e))$ is the sum of resource costs of primitive actions in applying the strategy H to example e.

To find the hypothesis with minimal resource complexity, we employ a search procedure named *iterative descent*, which works as follows: starting at iteration 1, the search returns a hypothesis H_1 with the minimal number of clauses if one exists. Importantly, on iteration 1 we do not enforce a maximum resource bound. Because the hypothesis space is exponential in the length of the solution [Muggleton *et al.*, 2015], the hypothesis H_1 is the most tractable to learn. The hypothesis H_1 gives us an upper bound on the resource complexity from which to descend. At iteration $i > 1$, we search for a hypothesis H_i with the minimal number of clauses but we also enforce a maximum resource bound set to $r(H_{i-1}, E^+) - 1$. This ensures that any returned hypothesis H_i has a lower resource complexity than any hypothesis H_j, where $j < i$. If a hypothesis H_i exists, the search continues at $i + 1$, until we converge on the optimal hypothesis.

4.3 Convergence

Metagol$_O$ finds an optimal hypothesis for the training examples as follows: Metagol$_O(E^+, \mathcal{H}) = \text{argmin}_{H \in \mathcal{H}} r(H, E^+)$. The following result demonstrates convergence of Metagol$_O$ to the optimal strategy given sufficiently large numbers of examples.

Theorem 3 (Convergence to optimal hypothesis) *Assume E consists of m examples randomly and independently drawn from instance distribution D. Without loss of generality consider the hypothesis space consists of two hypotheses H_1, H_2 where H_1 has resource complexity $O(f_1(n, d))$ and H_2 has resource complexity $O(f_2(n, d))$ and $f_1(n, d) < f_2(n, d)$ for all $n, d > c$ where c is a positive integer and n, d are two natural number parameters of the examples [4] In the limit Metagol$_O$ will return H_1 in preference to H_2.*

Sketch Proof. *Assume false. However, since $f_1(n, d) > f_2(n, d)$ for $n, d > c$, with sufficiently large m there will exist an example e d such that $r(H_2(e)) > r(H_1(e))$ where $r(H_2(e)) > r(H_2(e'))$ for all other e' in E^+ and $r(H_1(e)) > r(H_1(e'))$ for all other e' in E^+. In this case $r(H_1, E+) < r(H_2, E+)$ and Metagol$_O$ returns H_1, which has the minimum resource complexity. This contradicts the assumption and completes the proof.*

5 Experiments

We now describe experiments in which we use Metagol$_O$ to learn robot strategies involving composite objects in two scenarios: Postman and Sorter. The experimental goals are (1) to support Theorems 1 and 2, i.e. show that resource complexities of optimal strategies vary depending on whether objects can be composed within a strategy, and (2) show that Metagol$_O$ can learn such resource optimal strategies.

5.1 Materials

We use a similar problem representation for both experiments where there is humanoid robot in a one-dimensional space[5]. The world state is a list of Prolog facts. The robot performs actions that change the state. Actions are defined as dyadic predicates and are either primitive or composite. Composite actions are defined in terms of primitive ones. We compare strategies learned with Metagol$_O$ to strategies learned with Metagol$_D$ [Muggleton *et al.*, 2015], an existing ILP system. In both experiments, we provide the same background knowledge and metarules to both systems.

[4]For the Postman f_1, f_2 are sum and product and n, d are as in Theorems 1 and 2. For the Sorter problem n is list length, $d = 0$, $f_1(x) = x^2$ and $f_2 = x\log_2 x$.

[5]This is for simplicity and the learner can handle any n-dimensional space.

5.2 Learning robot postman strategies

Materials

We learn strategies for the postman example introduced in Section 1 where the goal is to learn a strategy to collect and deliver letters. The primitive actions are as follows: *move_up, move_down, take_letter, bag_letter, give_letter*. All primitive actions have a cost of 1. The robot can take and carry a single letter from a sender using the action *take_letter*. Alternatively, the robot can take a letter from a sender and place it in the postbag using the action *bag_letter* to form a composite object consisting of the postbag and the letter. The composite actions are as follows: *go_to_bottom, go_to_top, find_next_sender, find_next_recipient*. The cost of a composite action is dynamic and is based on its constituent actions. For example, the composite action *go_to_top* recursively calls the primitive action *move_up* until the postman is at the top.

Method

To generate training examples we select a random integer d from the interval $[0, 50]$ representing the number of places[6]. We select a random integer n from the interval $[1, d]$ representing the number of letters. For each letter we select random integers i and j from the interval $[1, d]$ representing the letter's start and end positions. To generate testing examples we repeat the same procedure as above but with a fixed number of letters n, as to measure the resource complexity as n grows. We use 5 training and 5 testing examples. We average resource complexities of learned strategies over 10 trials.

Results

Figure 4 shows that the strategies learned with Metagol$_O$ are in agreement with the theoretical *composable* tighter bounds demonstrated in Section 3. By contrast, the strategies learned with Metagol$_D$ are in agreement with the theoretical *non-composable* tighter bounds. Figure 2 shows two recursive strategies learned by Metagol$_D$ (a) and Metagol$_O$ (b) respectively, able to handle any number of places, any number of letters, and different start/end positions for the letters. Although both strategies are equal in textual complexity, they differ in their resource complexity. The strategy learned by Metagol$_O$ (b) is more efficient than the one learned by Metagol$_D$ (a) because it uses composite objects, i.e. the action *bag_letter*, whereas (b) does not. Table 1 illustrates this difference. The explanation for this is that Metagol$_D$ simply had no reason to prefer the more efficient hypothesis, it just happened to find the less efficient hypothesis first.

We also ran experiments (omitted for brevity) where we removed the ability to form composite objects, i.e. the action *bag_letter*. In this scenario, the strategies learned with Metagol$_O$ are in agreement with the theoretical *non-composable* tighter bounds.

5.3 Learning robot sorting strategies

Materials

In this experiment, we investigate recursive sorting algorithms learned as recursive robot strategies. In the initial state there is an unsorted list; in the final state there is a sorted list.

[6] 50 is an arbitrary limit, and the learner handles any finite limit.

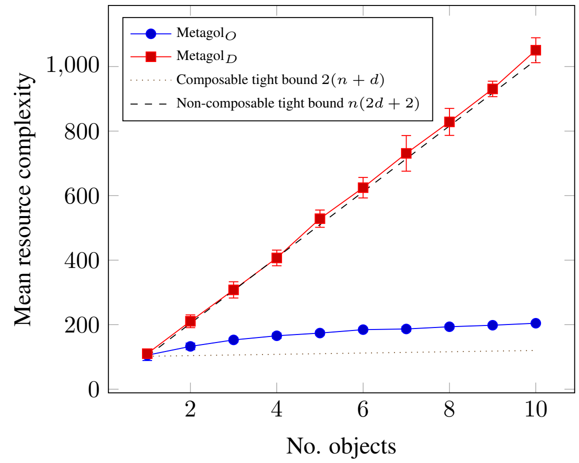

Figure 4: Mean resource complexity of learned postman strategies with varying numbers of letters for 50 places.

The robot can traverse the list moving sideways. The leftmost element represents the smallest element in the sorted list. We provide the learner with the actions to perform robot translations of the quick sort and bubble sort algorithms. We provide four composite actions: *compare_adjacent, split, combine, go_to_start*. The action *compare* compares two adjacent elements and swaps them if the one to the right is smaller than the one to the left. The action *split* allows the robot to move through the list comparing each element with the element in the robot's left hand (the pivot). If an element is less than or equal to the pivot, then the item is placed in a left bag; otherwise it is placed in a right bag. Both bags are stacks. The action *combine* empties the right bag, drops the pivot, and empties the left bag. We also provide two primitive actions: *decrement_end, pick_up_left*. The *decrement_end* action decrements a counter in the state representing the position of the last element of the list. The composite actions call other primitive actions that are unknown to the learner, e.g. *move_left, move_right*. The full list of such actions is omitted for brevity. All of the primitive actions have an action cost of 0, except the action *compare_holding* which has an action cost of 1. Thus, the optimal strategy is the one that minimises element comparisons.

Method

To generate training examples we select a random integer n from the interval $[2, 50]$ representing the length of the list. We use Prolog's *randseq/3* predicate to generate a list of n unique random integers from the interval $[1, 100]$ representing the input list, with this list sorted representing the output list. To generate testing examples we repeat the same procedure as above but with a fixed list length n as to measure the resource complexity as n grows. We use 5 training and 5 testing examples. We average resource complexities of learned strategies over 10 trials.

Results

Figure 6 shows that the strategies learned with Metagol$_O$ are in agreement with the theoretical average-case expectations of quick sort[7] ($O(n \ \ log \ \ n)$). By contrast, the strategies learned with Metagol$_D$ are closer to the average-case expectations of bubble sort ($O(n^2)$). Figure 5 shows the two recursive strategies learned by Metagol$_D$ (a) and Metagol$_O$ (b) respectively. Metagol$_O$ learns a variant of quick sort, which uses composite objects, whereas Metagol$_D$ learns a variation of bubble sort, which does not use composite objects.

```
metasort(A,B):- metasort1(A,C), metasort(C,B).
metasort1(A,B):- comp_adjacent(A,C), metasort1(C,B).
metasort1(A,B):- decrement_end(A,C), go_to_start(C,B).
metasort(A,B):- metasort1(A,C), go_to_start(C,B).
```

(a) Inefficient strategy learned by Metagol$_D$

```
metasort(A,B):- metasort1(A,C), metasort(C,B).
metasort1(A,B):- pick_up_left(A,C), split(C,B).
metasort1(A,B):- combine(A,C), go_to_start(C,B).
metasort(A,B):- split(A,C), combine(C,B).
```

(b) Efficient strategy learned by Metagol$_O$

Figure 5: Robot sorting strategies learned by Metagol$_D$ (a) and Metagol$_O$ (b) respectively.

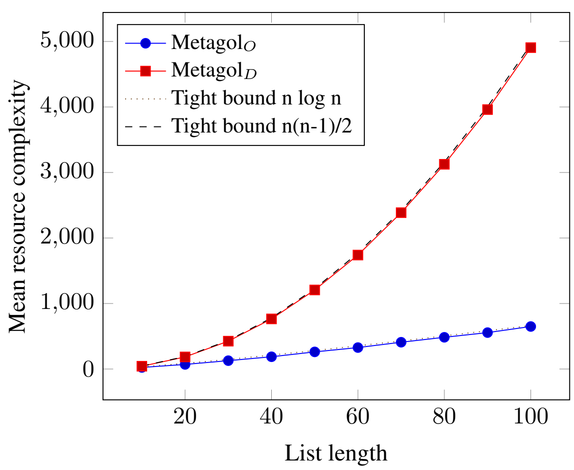

Figure 6: Mean resource complexity of robot sorter strategies with varying lengths of input.

6 Conclusions and further work

To the authors' knowledge this paper represents the first demonstration of a learning algorithm which provably optimises the resource complexity of robot strategies. The approach differs from traditional AI planning techniques which involve the identification of a sequence of actions to achieve a single goal from a single initial situation (e.g. moving from the door to the table). By contrast, a learned strategy is a program which can be applied to a multiplicity of initial situations to achieve a multiplicity of corresponding goal situations (e.g. deliver all the letters to their destinations). Once learned, such a strategy can be applied without the need for searching a space of plans each time. This paper proves the existence (Theorems 1 and 2) of particular cases (Postman) in which classes of learnable strategies have different resource complexities. A new MIL implementation, Metagol$_O$, is shown (Theorem 3) to converge with sufficiently large numbers of examples to the most efficient strategy. Our experiments demonstrate that the theoretical bounds of Theorem 3 hold in practice. The approach suggests the ability to build delivery and sorting robots which can learn resource efficient strategies from examples.

[7] Resource complexity proofs are omitted for brevity.

6.1 Further work

A limitation of this paper is the lack of details regarding the computational requirements of Metagol$_O$ to converge on an optimal solution. We will explore this in future work, including exploring methods to optimise the iterative descent search procedure. For example, instead of decrementing the energy bound by 1, binary search would be more efficient.

In Section 5 we compared our implementation, Metagol$_O$, to an existing ILP system, Metagol$_D$. In future work we intend to run comparisons with non-ILP systems. We also want to test this approach on a broader range of program induction tasks that include resource optimisation. For instance, in the learning of *proof tactics* for theorem proving, *game tactics strategies*, and *string transformation functions*.

The approach taken in this paper can be generalised in several ways. For instance, the use of dyadic Datalog programs could be generalised by using a richer set of metarules. We also intend to further explore the notion of object composition and investigate the resource complexity reduction of inventing new objects in the world. For instance, in the postman example, we provide a postbag in the background knowledge. In future work we would like to investigate methods for the learner to invent such an object.

In this work we have assumed noise-free data. To move beyond this assumption we could consider probabilistic variants such as those investigated in Statistical Relational Learning [De Raedt *et al.*, 2007; Muggleton *et al.*, 2014a].

We also hope to better characterise the value of recursion in strategy-learning tasks. In [Broda *et al.*, 2009] the authors consider continuous actions such as *move hand until you touch the table*. We aim to investigate continuous actions in further work.

All actions described in this paper have positive resource costs. We would like to consider actions with negative costs, i.e. benefits. For instance, an action which recharges the robot's battery, or actions in which the robot collects other resources, such as glue or bricks, or recruits other robots to help in a task.

To summarise, we believe that this paper opens exciting new avenues in a variety of areas in AI for understanding the value of machine learning efficient strategies.

Acknowledgments

The first author thanks the BBSRC for his PhD studentship. The second author acknowledges support from his Syngenta Royal Academy of Engineering Research Chair at the Department of Computing at Imperial College London.

References

[Argall *et al.*, 2009] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[Broda *et al.*, 2009] Krysia Broda, Keith Clark, Rob Miller, and Alessandra Russo. *SAGE: a logical agent-based environment monitoring and control system*. Springer, 2009.

[Cambon *et al.*, 2004] Stéphane Cambon, Fabien Gravot, and Rachid Alami. A robot task planner that merges symbolic and geometric reasoning. In *ECAI 2004 Proceedings of the 16th European Conference on Artificial Intelligence, August 22-27, 2004*, volume 16, page 895. IOS Press, 2004.

[De Raedt *et al.*, 2007] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its applications in link discovery. In R. Lopez de Mantaras and M.M Veloso, editors, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 804–809, 2007.

[Eiter *et al.*, 2003] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. Answer set planning under action costs. *Journal of Artificial Intelligence Research*, pages 25–71, 2003.

[Eugenio, 1991] B. Di Eugenio. Action representations for nl instructions. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, pages 333–334, Stroudsburg, PA, 1991. Association for Computational Linguistics.

[Felder *et al.*, 2003] Robin Felder, Randy Turner, William Holman, and Chris Estey. Robotic pick up and deliver system, April 8 2003. US Patent 6,543,983.

[Geringer *et al.*, 2012] J. Geringer, R. Watson, and J. Cooper. Robot vacuum cleaner, November 13 2012. US Patent D670,877.

[Gulwani *et al.*, 2015] S. Gulwani, J. Hernandez-Orallo, E. Kitzelmann, S.H. Muggleton, U. Schmid, and B. Zorn. Inductive programming meets the real world. *Communications of the ACM*, 2015. In press.

[Hayes and Demiris, 1994] Gillian M Hayes and John Demiris. *A robot controller using learning by imitation*. University of Edinburgh, Department of Artificial Intelligence, 1994.

[Koza and Rice, 1994] John R Koza and James P Rice. *Genetic programming II: automatic discovery of reusable programs*, volume 40. MIT press Cambridge, 1994.

[Laird, 2008] J. E. Laird. Extending the soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, pages 224–235, 2008.

[Lau *et al.*, 2003] Tessa Lau, Pedro Domingos, and Daniel S Weld. Learning programs from traces using version space algebra. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 36–43. ACM, 2003.

[Lin *et al.*, 2014] 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.

[Moyle and Muggleton, 1997] S. Moyle and S.H. Muggleton. Learning programs in the event calculus. In N. Lavrač and S. Džeroski, editors, *Proceedings of the Seventh Inductive Logic Programming Workshop (ILP97)*, LNAI 1297, pages 205–212, Berlin, 1997. Springer-Verlag.

[Muggleton *et al.*, 2011] S.H. Muggleton, L. De Raedt, D. Poole, I. Bratko, P. Flach, and K. Inoue. ILP turns 20: biography and future challenges. *Machine Learning*, 86(1):3–23, 2011.

[Muggleton *et al.*, 2014a] S.H. Muggleton, D. Lin, J. Chen, and A. Tamaddoni-Nezhad. Metabayes: Bayesian meta-interpretative learning using higher-order stochastic refinement. In Gerson Zaverucha, Vitor Santos Costa, and Aline Marins Paes, editors, *Proceedings of the 23rd International Conference on Inductive Logic Programming (ILP 2013)*, pages 1–17, Berlin, 2014. Springer-Verlag. LNAI 8812.

[Muggleton *et al.*, 2014b] S.H. Muggleton, D. Lin, N. Pahlavi, and A. Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94:25–49, 2014.

[Muggleton *et al.*, 2015] S.H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 2015. Published online: DOI 10.1007/s10994-014-5471-y.

[Otero, 2005] R. Otero. Induction of the indirect effects of actions by monotonic methods. In *Proceedings of the Fifteenth International Conference on Inductive Logic Programming (ILP05)*, volume 3625, pages 279–294. Springer, 2005.

[Pineau *et al.*, 2002] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Policy-contingent abstraction for robust robot control. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 477–484. Morgan Kaufmann Publishers Inc., 2002.

[Puterman, 2014] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[Russell and Norvig, 2010] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, New Jersey, 2010. Third Edition.

[Sutton and Barto, 1998] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[van Otterlo and Wiering, 2012] Martijn van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement Learning*, pages 3–42. Springer, 2012.

[Xing *et al.*, 2006] Zhao Xing, Yixin Chen, and Weixiong Zhang. Optimal strips planning by maximum satisfiability and accumulative learning. In *Proceedings of the International Conference on Autonomous Planning and Scheduling (ICAPS)*, pages 442–446, 2006.

[Yang *et al.*, 2014] Fangkai Yang, Piyush Khandelwal, Matteo Leonetti, and Peter Stone. Planning in answer set programming while learning action costs for mobile robots. *AAAI Spring 2014 Symposium on Knowledge Representation and Reasoning in Robotics (AAAI-SSS)*, 2014.