Applying Artificial Life Techniques to a Symbolic Learning Task

Murray Shanahan

Queen Mary & Westfield College,
Mile End Road,
London E1 4NS,
England.

Tel: +44 (0)171 975 5213
Email: mps@dcs.qmw.ac.uk

December 1995

Abstract

According to the popular folklore of the two fields, the classical, symbolic paradigm of Artificial Intelligence research is incompatible with new ideas from Artificial Life, such as behaviour-based architecture, collective intelligence, and evolutionary computation. This paper seeks to debunk this myth through three experiments which employ techniques from both paradigms. These experiments all concern the acquisition of a declarative representation of the laws of physics of a simple microworld. The first uses a single robot which combines a behaviour-based architecture with the classical ID3 induction algorithm. The second uses a population of such robots. And the third uses a technique resembling a genetic algorithm to evolve a population of robots with appropriate declarative representations.

Keywords: Artificial Life, Machine Learning.

Introduction

It cannot have escaped the notice of many researchers in AI that we are currently in the middle of a paradigm war. The classical, symbolic approach to AI is under attack on various fronts. Many of its recent critics ally themselves with the nascent field of Artificial Life [Langton, 1987], and largely reject the past 40 years of Artificial Intelligence research.

Brooks [1991], for example, advocates a behaviour-based approach to building autonomous agents, in which traditional perception, planning and plan execution modules are replaced by "independent and parallel activity producers which all interface directly to the world through perception and action". Mataric [1992] laments the fact that "traditional AI addresses intelligence as an isolated phenomenon" and instead suggests that "intelligent behaviour is inextricably tied to its [social] context". Harvey, Husbands and Cliff [1992] reject conventional AI's attempts to design cognitive architecture, and instead advocate the "automatic evolution of the architecture without explicit design".

The main aim of this paper is to show that many of the attractive ideas associated with the AI "new wave" can be reconciled with older representational ideas. Whatever the right approach turns out to be, the field can only be hindered by the belief that the spectrum of methodological choices is polarised into the "old fashioned" and the "newfangled". In particular, the paper shows that,

- Behaviour-based control can usefully guide a symbolic learning algorithm,
- Collective behaviour can aid in the construction of symbolic representations, and
- Evolution can be used to generate symbolic representations.

The paper describes three experiments, one to support each of these claims. These experiments all involve the same simulated microworld, populated by one or more robots. And they all concern the construction of the same kind of symbolic representation, namely a set of identification trees (or decision trees) which are intended to capture the laws governing the microworld.

The first experiment involves a single robot which is controlled by a layered behaviour-based architecture, but which uses the classical induction algorithm ID3 to learn about the microworld. The second experiment involves a population of robots, each controlled by a simpler behaviour-based architecture, and each contributing to a central database, which is again processed using the ID3 algorithm. The final experiment preserves the idea of a population of robots, but abandons induction in favour of an evolutionary technique for generating identification trees.

1. The Microworld

The robots' environment in all three experiments is a cellular microworld (a "grid world"). The robot occupies a single cell, and in a single time step can move one

cell either north, east, south or west. The edges of the grid wrap around to join their opposite edges, so the microworld is effectively a torus.

Apart from the robot, cells can be occupied by four kinds of object. These are represented in the simulation by four colours: red, yellow, green and blue. Objects occupy exactly one cell. Only one object may occupy a cell at any given time, with the exception that the robot can occupy the same cell as another object. These four kinds of object behave differently when the robot tries to push them. Their behaviour is summarised in the following rules.

- If the robot tries to push a red object east or west, both robot and object move in that direction.
- If the robot tries to push a red object north or south, neither of them moves.
- If the robot tries to push a blue object north or south, both robot and object move in that direction.
- If the robot tries to push a blue object east or west, neither of them moves.
- If the robot tries to push a green object in any direction, the robot moves in that direction, and the object disappears.
- If the robot tries to push a yellow object in any direction, the object stays where it is and the robot moves on top of it.

These rules compose in a natural way. For example, if the robot pushes a blue object north, and the next cell north of the blue object contains another blue object, then both blue objects will move (see Figure 1). Had it been a red or a yellow object in that cell, then nothing would have moved, because red objects don't move north or south, and yellow objects don't move at all and can't have other objects on top of them besides the robot.

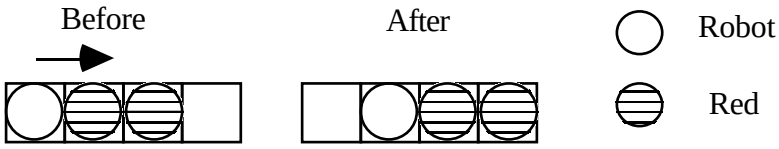

Figure 1: A Microworld Event

There is nothing special about this particular set of rules, and any similar set would have served equally well. However, one notable property is that the robot cannot always undo what it has done, since there is no way to recover green objects once they have gone. Another notable property is that only a relatively small subset of all possible states of the microworld is accessible from any given state. Note that it's possible for the robot to be completely trapped at the start of the run, if it is surrounded by red objects north and south and blue objects in the east and west. When this occurs, the robot is unable to learn anything.

This microworld is very simple, but can serve as a testbed for various AI techniques, such as learning, path finding, planning, and map building. In addition, it can be used as a laboratory for comparing different approaches to the same

problem. In the present paper, the problem studied is that of building a symbolic representation of the microworld's "laws of physics", and the techniques investigated are induction, behaviour-based control, collective behaviour, and evolution.

2. Wandering and Learning

The first of the three experiments involves a single robot using a classical induction algorithm. The architecture of the robot is the product of what Brooks [1991] calls a behaviour-based decomposition. That is to say, its components are not the functional modules of classical AI, such as perception, planning, plan execution, etc., but rather are layers giving rise to different activities. Each such layer connects the robot's sensors directly to its effectors, and is responsible for a particular behavioural tendency (see also [Agre & Chapman, 1987]). The separate layers are usually activated by particular circumstances, and frequently have to compete with other active layers to influence the robot's behaviour.

The three layers in the learning robot described here are Wander, Explore and Discover. In keeping with Brooks's ideas, these layers reflect a hierarchy of control. The Discover layer subsumes the Explore layer, which in turn subsumes the Wander layer. That is to say, the Wander layer is active all the time, but if a layer above it becomes active, that layer can override the Wander layer and take over control of the robot.

The Wander layer simply executes a random walk around the microworld, by randomly choosing one of the four possible directions of motion. The Explore and Discover layers take over under special circumstances in which they can potentially speed up the learning process. The Explore and Discover layers can each be "unplugged" and the robot will continue to move around its world, learning as it goes. Its learning will simply be slower than it would be with those layers working. Unplugging the Wander layer, however, would leave the robot stationary. If other objects in microworld moved about, a passing object might trigger the Discover layer, but the implemented system includes only static objects.

In addition to these layers, a background learning process is continually running. The learning algorithm used is ID3 ([Quinlan, 1986]), which constructs identification trees for events in the robot's field of interest.[1] The robot's *field of interest* is the cell it occupies plus the next two cells along in the direction it is facing (that is, the direction of its attempted movement) (see Figure 2). An *event* in the robot's field of interest is a pair comprising a cell (Cell 0, Cell 1 or Cell 2) and an event type (Move, Stay or Vanish). The robot maintains a "diary" of events which is used as input to ID3. An example of an event would be ⟨Cell 1,Vanish⟩, denoting the disappearance of the object in Cell 1. This would in fact occur if Cell 1 contained a green object, and it is the task of the learning algorithm to find such laws as this.

[1] In the present implementation, ID3 is run from scratch after each of the robot's moves, but only on trees potentially affected by the move. It would no doubt be faster to use an incremental version of ID3, such as that described by Utgoff [1989], which could assimilate events one by one.

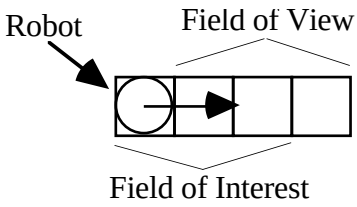

Figure 2: The Robot's Field of Interest and Field of View

The features used to construct the identification tree are action types (attempts to move in a particular direction, in this case) and properties of the robot's field of view before the action. The robot's *field of view* is the next three cells along in the direction it is facing (see Figure 2). To be more precise, a *feature* is a pair comprising an attribute (Dir, Cell 1, Cell 2, or Cell 3) and a value. The possible values for Dir (the direction of attempted movement) are North, East, South and West. The possible values for Cells 1, 2 or 3 are Empty, Red, Yellow, Green and Blue. An example of a feature would be ⟨Dir,North⟩, denoting that the robot tried to move north.

An *identification tree* for an event is a tree whose non-leaf nodes are attributes, whose leaf nodes are either YES or NO, and whose arcs are labelled by values. Each non-leaf node for an attribute A has a number of sub-trees, each corresponding to a possible value of A. Figure 3 shows an identification tree for the (nearly correct) rule that an object in the next cell along from the robot will move if it is red and is pushed east or west, or if it is blue and is pushed north or south.[2] Directions are abbreviated to N, E, S and W.

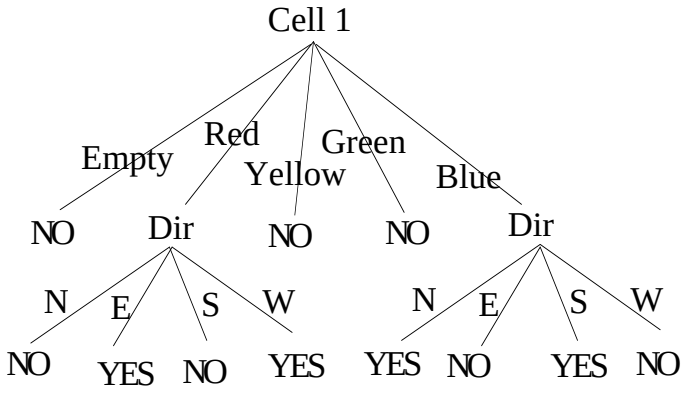

Figure 3: An Identification Tree for the Event ⟨Cell 1,Move⟩

A semantics for these identification trees is conveniently provided by the Situation Calculus [McCarthy & Hayes, 1969]. I won't present the translation here, but it should be easy to reconstruct for anyone familiar with the Situation Calculus formalism. There is one action, Move(d) where d is one of the four directions in which the robot can move, and two fluents: Occupied(x,y,c) representing that cell ⟨x,y⟩ is occupied by an object whose colour is c, and Robot(x,y) representing that

[2] The robots in the present implementation cannot learn rules that depend on hidden states such as the contents of cells outside their fields of view.

the robot is in cell ⟨x,y⟩, where x and y are cartesian co-ordinates in the microworld.

3. Exploring

The robot's current collection of identification trees enables it to make predictions about the outcome of possible actions, based on the state of its field of view. This gives it a capacity for lookahead, which is exploited by both the Explore and Discover layers. Both these layers exist to speed up the learning process, by improving on the simple random walk strategy of the Wander layer whenever certain opportunities arise to do so.

Let's consider the Explore layer first. If the robot spends all its time in one corner of the microworld, it will never encounter phenomena which are unique to configurations of objects only to be found elsewhere in the microworld. The Explore layer tries to prevent this from happening. The microworld is divided up into nine square regions, and the robot maintains a record of how many moves it has attempted in each region. When the robot finds itself near the boundary of a region, the Explore layer becomes active.

If the neighbouring region is relatively unexplored — that is, if the number of moves attempted in that region is less than a certain proportion of those attempted in the present region — then the Wander layer is inhibited and the robot tries to move towards it. If it has the chance to cross over into such a region, then it will. If the robot is in a position where many actions provide such opportunities, such as in the corner cell of a region, then it chooses one at random.

Initially, when the robot has an empty set of rules, the Explore layer is ineffectual, because it relies on the predictive power of these rules to determine which actions will successfully take it towards an unexplored region. For example, if there is a blue cell immediately to the east in an unexplored region, then the laws of the microworld dictate that moving to the east will have no effect. Because of such possibilities, the robot will only try to move towards an unexplored region when its current set of rules suggests that the attempt will be successful. The rules may still be wrong, of course, in which case an attempt to move towards an unexplored region may be unsuccessful even when they suggest otherwise.

The currently implemented version of the Explore layer needs some development. It can cause the robot to oscillate back and forth across region borders, since it only encourages the robot into an unexplored area and does nothing to keep it there. There are some obvious ways around this problem. But for present purposes, the Wander layer is mainly there in case a two layer behaviour-based architecture seems too trivial to justify the claims of the paper.

4. Discovering

Both the Wander layer and the Explore layer are subsumed by the Discover layer, which is activated when the robot has an opportunity to check one of its less well-established rules. Such an opportunity arises whenever the robot's field of view in

some direction matches the antecedent of one of these rules. The robot will take the best of any such opportunities that arise, and will try to move in the corresponding direction. The meaning of a "well-established rule" and "the best opportunity" will be made precise below.

In addition, if no such opportunities immediately present themselves, the robot uses its current set of rules to look one move ahead. If, according to those rules, it can perform an action which will result in its field of view matching one of its less well-established rules, and which will therefore provide it with a good opportunity to check the rule, then it will go ahead and perform that action. So the feedback from the current set of rules back into the rule learning process is along two paths. First, the current set of rules biases the choice of which action to perform towards those which have the best chance of disconfirming a rule. And second, when there is no chance of a disconfirmation, the predictive power of the current set of rules can guide the robot into a position where there is such a chance.

Figure 4 shows the robot disconfirming a rule. In Stage 1, the robot employs a well-established rule in its current rules set, "Cell 1 Vanishes if Cell 1 is Green," to predict that by trying to move north it will end up with a red object to its east. This anticipated situation would match the antecedent of one of its not so well-established rules, "Cell 1 Moves if Cell 1 is Red and Dir is East." So the robot goes for it. Indeed it does end up where it thought it would in Stage 2, giving it the opportunity to check the rule by moving east, which it does. Contrary to the rule, nothing happens. The object in Cell 1 doesn't move because it is jammed up against a blue object. This new event (or rather non-event), which will be recorded in the robot's diary, will force the robot's rules into better shape when ID3 is run.

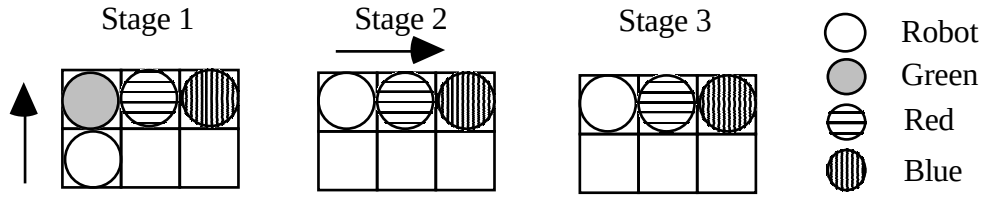

Figure 4: The Robot Disconfirms a Rule

It seems clear, intuitively, that this discovery strategy is subject to the law of diminishing returns. The more often the robot checks a current rule, modifying it when necessary to take account of disconfirmations, the better that rule will get at predicting events. Eventually a rule will become very well-established, reflecting the fact that it is close to perfect, and there will be little to gain in continuing to check it. When this happens, it would be better if the Wander layer retained control, maximising the chances that something new is discovered.

How is this law of diminishing returns reflected in the implementation? Before each action performed by the robot, the Discover layer gives a score to each possible move[3] — the lower the score the better the move — and if the score for any move is below a certain threshold (the robot's "boredom threshold"), the Discover layer

[3] In fact, in a situation identical to one it has encountered before, the robot ignores moves it has already tried in that situation.

becomes active and takes control of the robot, executing the move with the lowest score.

To calculate this score, the robot records, for each event for which it has a rule, the number of occurrences of that event. With ID3 running continually, the corresponding rule will be correct for each such occurrence, so this number will reflect the degree of confirmation of the rule. In the implementation, to yield a number between 0 and 1, the calculated score is (1–1/(N+1)), where N is the number of occurrences of the event.

5. Some Empirical Results

The microworld and robot were implemented in LPA MacProlog on an Apple Power Macintosh 6100. Although little attempt was made to optimise the code, the system runs fast enough to make quite entertaining real-time viewing. For instance, because the robot (rightly) has no way of knowing which of several equally parsimonious rules (according to ID3) is the right one, its arbitrary choice is sometimes rather bizarre when it has had few training instances.

It might, for example, decide that the rule "Cell 1 vanishes if Cell 2 is Blue" is a good explanation for the disappearance of a green object which just happened to be next to a blue one. It will then attempt to make every object it comes across which is next to a blue object disappear, until ID3 takes into account its inevitable failures and supplies an improved rule. Obviously, any attempt to avoid this sort of behaviour by imposing appropriate heuristics would be to cheat by tailoring the robot for this particular microworld's physics.

A number of experiments were peformed to assess the effect of combining the Wander layer with the Discover layer. In these experiments, the Explore layer was disconnected.[4] Thirty 15 by 15 microworlds each containing 25 objects were randomly generated. First, a robot with just a Wander layer was let loose in each of these microworlds. At the end of each run, the identification trees produced by the robot were tested. Then the same process was repeated, on the same set of microworlds, but using a robot with both Wander and Discover layers.

To test a set of identification trees, the following procedure was used. The set of trees was tested against each possible combination of up to two objects of any colour in the field of view, in each of the four directions. There are 244 such combinations — 61 in each direction. For every correct prediction yielded by the trees, one point was awarded, and for every incorrect prediction one point was taken away. No points are awarded or deducted when the trees yield no prediction at all. The mean score over all 244 combinations was recorded for each run. The maximum possible mean score, which would result from a perfect set of rules, is 2.77. Conversely, the worst possible mean score is -2.77.

The tables in Figure 5 summarise a pairwise comparison of the means of the resulting mean scores, with and without the Discover layer, for runs of 20, 40 and 60 moves, and for two values of the boredom threshold (b.t.). The leftmost table

[4] Collecting empirical data about an improved Wander layer is on the agenda.

shows the mean of mean scores $\bar{x}$ without the Discover layer. The next two tables are comparative. The first columns show the mean of mean scores $\bar{x}$ with the Discover layer. The mean difference $\bar{d}$ between the mean scores with and without the Discover layer, and the standard deviation of the differences s_d^2 are shown. Recall that the sample size is thirty, corresponding to the number of randomly generated microworlds tried.

	No Discover layer	b.t. = 0.666666			b.t. = 0.75			
Moves	$\bar{x}$		$\bar{x}$	$\bar{d}$	s_d^2		$\bar{x}$	$\bar{d}$	s_d^2
20	0.47	20	0.52	0.05	0.02	20	0.53	0.06	0.03
40	0.55	40	0.61	0.06	0.05	40	0.62	0.07	0.08
60	0.62	60	0.68	0.05	0.10	60	0.68	0.06	0.09

Figure 5: Empirical Evaluation of the Discover Layer

A pairwise *t*-test applied to these results reveals that the difference between the scores is statistically significant at the five percent level for 20 moves for both values of the boredom threshold. This suggests that the incorporation of the Discover layer improves the robot's learning performance to a worthwhile degree. However, the statistical significance of the results decreases as the number of moves increases. A more thorough set of tests would be required to firmly establish and quantify the utility of the technique. This is not the purpose of the present paper.

6. Collective Learning

The second experiment also uses induction to construct identification trees, but the microworld is inhabited by many robots. The robots are autonomous and independent, without any central control, but they communicate with a central database. Each robot has a Wander layer, which has the same role as in the single robot version, and an Avoid layer that keeps the robots apart from each other. A robot's Avoid layer comes into play when one or more other robots come within the robot's field of view in any direction. Other robots are like repellent forces, and the Avoid layer resolves these forces to determine possible directions in which to try to move. To implement this, each robot's repertoire of actions has to be extended to include a DoNothing action for the case when it is completely surrounded by other robots. As well as an object, each cell can now be occupied by another robot, and the set of possible events is correspondingly expanded.

Each robot contributes its separate record of events to a central knowledge base, and ID3 is run on this collective record. This parallelism, in the spirit of suggestions in [Brooks & Flynn, 1989], speeds up the learning process considerably. The experimental results below suggest that this speed up is, on average, even better than n-times for n robots. This is no doubt due to the geographical spread of the robots. A single robot wastes much of its time exploring parts of the microworld too familiar to yield any further improvement to its rules. With robots distributed all over the microworld, this isn't a source of inefficiency.

The tables in Figure 6 summarise the results of an experiment with five robots. The same collection of thirty microworlds was used as in the previous experiment. The robots were run for 4, 8 and 12 moves each, giving total numbers of 20, 40 and 60 moves, as in the previous experiment. This permitted a comparison with the results previously obtained for the single robot with no Discover layer. The comparison is summarised in the right-hand table. The left-hand table is copied straight from Figure 5. A pairwise *t*-test again showed that the difference is statistically significant at the five percent level for 20 moves, but that the significance decreases as the number of moves increases.

One Robot

Moves	$\overline{x}$
20	0.47
40	0.55
60	0.62

5 Robots

Moves	$\overline{x}$	$\overline{d}$	s_d^2
4	0.56	0.09	0.07
8	0.60	0.05	0.12
12	0.67	0.05	0.17

Figure 6: Empirical Evaluation of the Multi-Robot Version

7. Evolution Instead of Induction

The third and final experiment departs from induction, and instead employs evolution to generate identification trees. Two techniques were tried: A and B. Technique A is very similar to a genetic algorithm [Holland, 1992], but with unconventional crossover and mutation operations. Technique A works on a population of collections of identification trees, and does not directly involve robots or the microworld. Technique B uses the same crossover and mutation operations, but it works on a population of robots, and is less like a genetic algorithm because members of the population inhabit a common microworld and interact with each other. With Technique A, evolution has the advantage of working directly with the cost function used to evaluate its end products.

Using Technique A, we start with a randomly generated population of collections of identification trees. Each such collection includes exactly one randomly generated tree (see below), for a randomly chosen concept. (For the purposes of this paper, a concept is an event.) Each collection of trees is assessed according to the scoring method used in previous experiments, and the high scoring collections are crossed and mutated to produce a new generation of collections. The process is then repeated.

In Technique B, the standard scoring method is not coded into the evolutionary loop as it is in Technique A. Initially, a randomly generated microworld is populated by robots with randomly generated collections of identification trees. The robots, which are controlled by the same two layer architecture as in the previous experiment, then make a certain number of moves each. As they go, they use their identification trees to bet on the outcomes of the actions they perform. They are awarded one mark for each correct prediction and lose one mark for each faulty prediction.

After they have peformed a fixed number of moves, the robots with the fewest marks are removed. The trees of randomly chosen pairs of successful robots — those with most marks — are then crossed and mutated to produce a new generation of robots, and the process is repeated. After each generation, a new microworld is generated randomly.

What is mean by a "randomly generated tree" here? To randomly generate a tree, an attribute is selected at random for the root. Each attribute has an equal chance of being selected. Then, a random subset of possible values for that attribute is chosen. Any given value will be included in such a subset with probability 0.5. The root is then extended with leaf nodes corresponding to each value in this subset. Each of these leaf nodes will be either YES or NO with equal probability.

Genetic algorithms manipulate a population of bit strings, for which mutation and crossover operations are easily defined: crossover involves swapping segments of bit strings, and mutation involves flipping bits. In genetic programming [Koza, 1992], crossover is defined for Lisp S-expressions, and involves swapping sub-expressions (which can be thought of as swapping sub-trees).

For identification trees, these operations are necessarily more complicated, because swapping arbitrary sub-trees of two parents is not guaranteed to produce a useful identification tree. For example, it's pointless for a node labelled with attribute A to have sub-trees which also mention attribute A. The recursive algorithm in the next section gets around this problem by rearranging one of the parent trees so that both trees have the same attribute at the root.

8. Crossover and Mutation

The function Cross, defined below, takes two collections of identification trees and crosses them, returning the result. Collections of identification trees are represented as sets of pairs ⟨C,T⟩, where C is a concept and T is an identification tree. The algorithm only has to cross trees when both parents have a tree for the same concept. If only one parent has a tree for a concept, then the child simply inherits that tree for that concept.

Function Cross(M,F)
    NewTrees := {}
    For each concept C for which one parent has a tree T but for which the other parent has no tree
        NewTrees := NewTrees ∪ {⟨C,T⟩}
    For each concept C for which M has tree T1 and F has tree T2
        NewTrees := NewTrees ∪ {⟨C,CrossTrees(T1,T2)⟩}
    Return NewTrees

Crossing two trees is a recursively defined operation. In the base case, at least one of the parent trees is a leaf node, and the function randomly chooses one of the parents (possibly the one that is just a leaf node) and returns it. Otherwise, before the recursive application of the algorithm, one of the parents may have to be rearranged so that they both have the same attribute at the root. Given a tree whose

root attribute is A, one of whose possible values is V, SubTree(T,V) denotes the sub-tree of the root for value V.

```
Function CrossTrees(T1,T2)
    If T1 or T2 is a leaf node
    Then NewTree := either T1 or T2
    Else If root attribute of T1 ≠ root attribute A of T2
    Then T1 := Rearrange(T1,T2)
    Let A be root attribute of NewTree
    For each value V of A
        SubTree(NewTree,V) := CrossTrees(SubTree(T1,V),SubTree(T2,V))
    Simplify(NewTree)
    Return NewTree
```

The function Simplify removes the redundant foliage in branches whose leaf-nodes are all YES or all NO. To rearrange a tree T so that it has attribute A at the root involves making a tree whose root is A, and whose sub-trees are modified copies of T, one for each value V of A. These modified copies are simplified by taking into account the fact that A must be V within that sub-tree, so any A nodes can be removed.

```
Function Rearrange(T1,T2)
    Let Tree have the root attribute A of T2
    For each value V of A for which T2 has a sub-tree
        T' := T1
        Replace all nodes N in T' labelled with attribute A by SubTree(N,V)
        SubTree(Tree,V) := T'
    Return Tree
```

An example of a crossover operation is given in Figure 7. The result shown is, of course, just one of several possible outcomes of crossing the trees T1 and T2.

Mutation, in the present implementation, is defined in terms of crossover. To mutate a tree, it is simply crossed with a randomly generated tree. This can result in a more dramatic change to a tree than is normally expected from a mutation operation. Alternative mutation algorithms are under investigation.

9. Empirical Results for Evolution

The evolutionary techniques described were implemented in LPA MacProlog on an Apple Power Macintosh 6100. For both Technique A and Technique B, populations of twenty robots were used (which is relatively small for an evolutionary algorithm of this kind). With both techniques, the top ten robots from each generation were retained for the next generation, and these were crossed and mutated to produce ten new robots to replenish the population, two of which, chosen at random, were subjected to mutation.

Ten runs of Technique A were performed, starting from random trees, and each time seeding the random number generator with a different value. These runs suggest that Technique A is quite effective at producing collections of trees of ever

increasing quality. The table in Figure 7 summarises the ten runs. A number n in row x and column y indicates that, in run x, generation n was the first in which the best robot achieved a score greater than y.

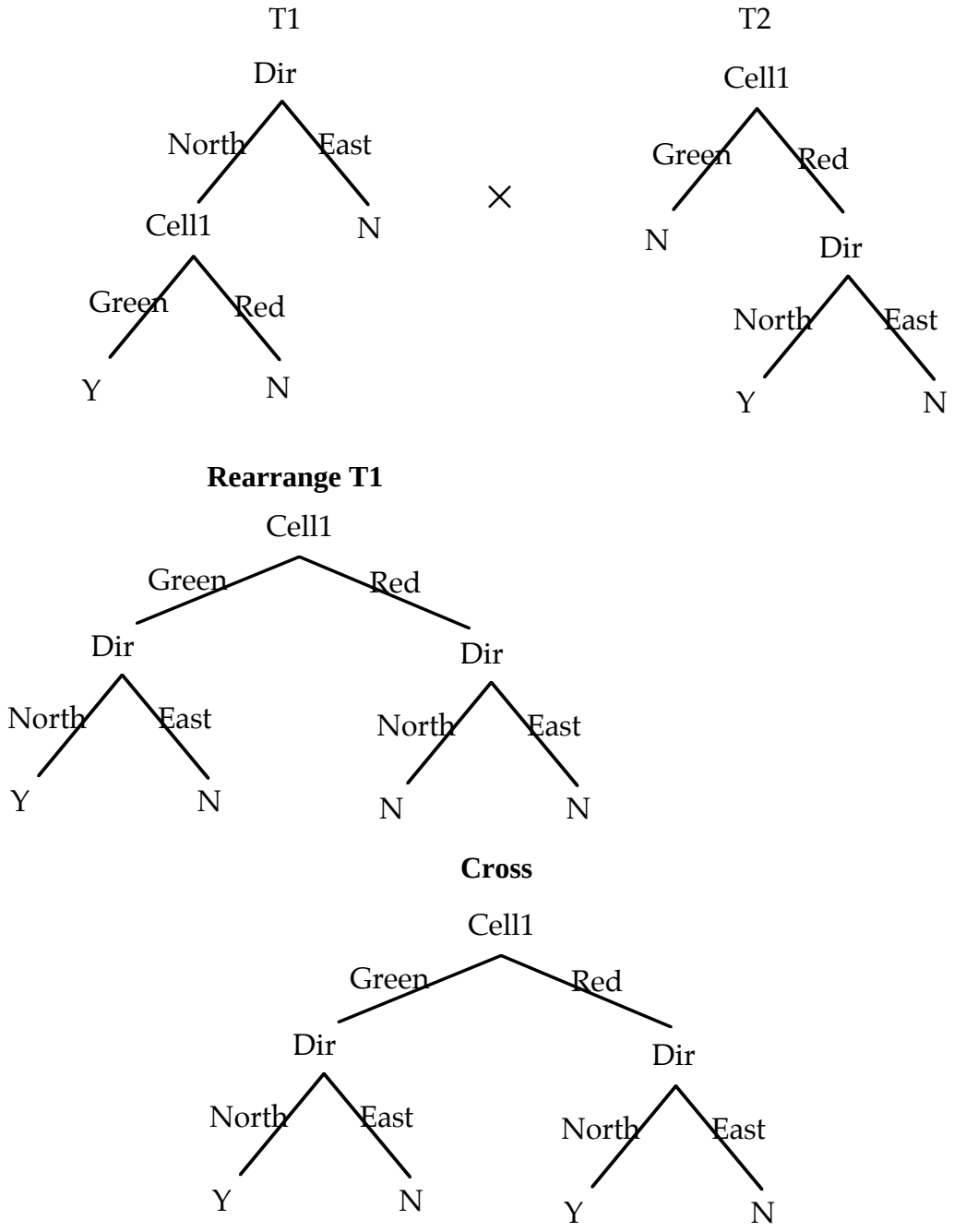

Figure 7: An Example of Crossover

Best Score \ Run	1	2	3	4	5	6	7	8	9	10
0.4	2	1	2	10	3	8	5	1	4	1
0.5	6	3	3	16	9	10	9	2	6	3
0.6	8	4	3	18	9	11	92	18	14	12
0.7	11	18	24	22	14	14	110	18	14	21

Figure 8: Empirical Evaluation of Technique A

For Technique B, a 25 by 25 grid was used for the microworld. Each randomly generated microworld included 120 randomly placed and coloured objects. Each generation lasted 20 moves. Ten runs of Technique B were performed. The trees of each generation's best robot were evaluated, using the same scoring scheme as before. The most successful robot in each of generation 20, 30, 40 and 50 was evaluated according to the standard scoring method. The results are summarised in Figure 8.

Generation \ Run	1	2	3	4	5	6	7	8	9	10
20	0.29	0.40	0.12	0.36	0.00	0.19	0.25	0.72	–0.07	0.24
30	0.25	0.18	0.12	0.32	0.47	0.30	0.24	0.64	–0.24	–0.23
40	0.07	0.27	0.16	0.24	0.31	0.30	0.50	0.67	–0.23	0.24
50	–0.46	0.27	0.15	0.21	0.22	0.30	0.40	0.47	–0.23	0.14

Figure 9: Empirical Evaluation of Technique B

As Figure 8 shows, Technique B does not tend to evolve ever better collections of trees, unlike Technique A. Perhaps this is unsurprising, since in Technique B the criterion according to which the resulting trees are evaluated is different from the cost function used in the algorithm. However, the induction algorithm in the previous experiments was similarly disadvantaged. The power of inductive learning is that it can reach beyond a few examples towards a set of general rules. These experiments suggest that evolutionary techniques might be less able to do this. However, these results are preliminary, and much work needs to be done, varying the parameters of the experiment, to get a clearer picture.

10. Discussion

Three experiments have been presented that draw on ideas both from classical, symbolic Artificial Intelligence, and from the new field of Artificial Life. The experiments have shown that, contrary to AI and A-Life folklore, neither approach excludes the other. It's not the aim of the work reported here to match the achievements of either paradigm, but simply to highlight the possible benefits of a reconciliation. The empirical results obtained, whilst preliminary, do suggest that the combination of paradigms is potentially fruitful.

To my knowledge, no similar work to that reported here has been done on mechanisms which evolve declarative representations of knowledge. On the other hand, there's a considerable amount of related work on situated learning and learning in microworlds.

Brooks, for example, has investigated learning in the context of behaviour-based robotics with his celebrated walking robots [Maes & Brooks, 1990]. These robots have to deal with the messy complexities of the real world, whilst the microworld described in this paper is very clean and simple. As in most systems that use reinforcement learning [Sutton, 1990], the task for Brooks's robots is to learn how to perform a task rather than to learn *that* certain facts are true. By contrast, classical symbolic learning has been largely concerned with learning that, in other words with acquiring declarative knowledge, whose purpose is not necessarily known at the time of learning.

But classical learning has placed little emphasis on situatedness. Exceptions include recent work on learning declarative knowledge from a microworld by Drescher [1991] and Shen [1993]. Both employ declarative representation, Shen in a more classical sense than Drescher. Mitchell [1990] describes a robot architecture which employs (but does not learn) declarative representations of the world in order to learn reactive stimulus response rules for performing tasks.

With present day robotics technology, learning declarative knowledge is hard to study using an embodied system in the real world. From the point of view of robotics, there is a strong argument for abandoning the study of the learning of declarative representations. The real world, so the argument goes, is far too messy and complicated to be described by declarative representations. So why bother to study techniques for acquiring them? If you want to build robots that actually do something in a realistic environment, the world is its own best model [Brooks, 1991]. Following this line of reasoning, learning how to perform tasks seems to be the only useful kind of learning.

People find this argument persuasive because the success of traditional AI techniques in toy microworlds contrasted so sharply with their failure in the real world, a failure highlighted by work in robotics. But the traditional approach was "thrown in at the deep end." There simply didn't exist anything intermediate between the Blocks World and the real world. This is where Artificial Life, despite itself, can come to the rescue of symbolic representation.

The manifesto of the field of Artificial Life advocates the synthesis and simulation of life-like phenomena, including intelligence, in non-physical media, such as computers [Langton, 1987]. These non-physical media are, at present, much less messy and complicated than the real world. It is possible to see research in Artificial Life as a source of microworld environments of ever increasing complexity (see Wilson [1990]).[5] The gradual nature of this increase in complexity might allow the development, at a similarly gradual pace, of hybrid AI systems, employing both A-Life inspired techniques and classical AI technology.

[5] Computer games might serve a similar purpose.

Acknowledgements

Thanks to Pat Langley and Stewart Wilson for advice and encouragement. The author is supported by an EPSRC Advanced Research Fellowship.

References

[Agre & Chapman, 1987] P.E.Agre and D.Chapman, Pengi: An Implementation of a Theory of Activity, *Proceedings AAAI 87*, pages 268-272.

[Brooks, 1991] R.A.Brooks, Intelligence Without Representation, *Artificial Intelligence*, vol 47 (1991), pages 139-159.

[Brooks & Flynn, 1989] R.A.Brooks and A.M.Flynn, Fast, Cheap and Out of Control: A Robot Invasion of the Solar System, *Journal of the British Interplanetary Society*, vol 42 (1989), pages 478-485.

[Drescher, 1991] G.L.Drescher, *Made-Up Minds: A Constructivist Appproach to Artificial Intelligence*, MIT Press, 1991.

[Harvey, Husbands & Cliff, 1992] I.Harvey, P.Husbands and D.Cliff, Issues in Evolutionary Robotics, *Proceedings 2nd International Conference on the Simulation of Adaptive Behavior*, 1992, pages 364-373.

[Holland, 1992] J.H.Holland, *Adaptation in Natural and Artificial Systems* (2nd edition), MIT Press (1992).

[Koza, 1992] J.Koza, *Genetic Programming*, MIT Press (1992).

[Langton, 1987] C.G.Langton, Artificial Life, *Proceedings Artificial Life I*, pages 1-47.

[McCarthy & Hayes, 1969] J.McCarthy and P.J.Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence, in *Machine Intelligence 4*, ed D.Michie and B.Meltzer, Edinburgh University Press (1969).

[Maes & Brooks, 1990] P.Maes and R.A.Brooks, Learning to Coordinate Behaviors, *Proceedings AAAI 90*, pages 796-802.

[Mataric, 1992] M.Mataric, Designing Emergent Behaviors: From Local Interactions to Collective Intelligence, *Proceedings 2nd International Conference on the Simulation of Adaptive Behavior*, 1992, pages 432-441.

[Mitchell, 1990] T.Mitchell, Becoming Increasingly Reactive, *Proceedings AAAI 90*, pages 1051-1058.

[Quinlan, 1986] J.R.Quinlan, Induction of Decision Trees, *Machine Learning*, vol 1 (1986), pages 81-106.

[Shen, 1993] W.M.Shen, Discovery as Autonomous Learning from the Environment, *Machine Learning*, vol 10 (1993), pages 143-166.

[Sutton, 1990] R.S.Sutton, Reinforcement Learning Architectures for Animats, *Proceedings 1st International Conference on the Simulation of Adaptive Behavior*, 1990, pages 288-296.

[Utgoff, 1989] P.Utgoff, Incremental Induction of Decision Trees, *Machine Learning*, vol 4 (1989), pages 161-186.

[Wilson, 1990] S.W.Wilson, The Animat Path to AI, *Proceedings 1st International Conference on the Simulation of Adaptive Behavior*, 1990, pages 15-21.