

ADVANCES IN EVOLUTIONARY AGENT LEARNING

ANNA L. BUCZAK, DAVID G. COOPER, MARTIN O. HOFMANN

Lockheed Martin Advanced Technology Laboratories

Cherry Hill, New Jersey

{abuczak, dcooper, mhofmann}@atl.lmco.com

ABSTRACT

This paper describes a novel methodology for software agent learning. Evolutionary Platform for Agent Learning (EPAL) creates both subtle and drastic changes to agent behavior. Subtle changes arise from learning task parameters. Drastic changes emerge from learning improved workflows containing new programming constructs and tasks. EPAL presents a general approach to agent learning, based on an extension to Genetic Programming that we developed. This approach is suitable for learning by any agent whose behavior can be represented as a workflow that can be further decomposed into building blocks, such as operators, tasks, and parameters. This paper describes a real-world problem that our agents learned to solve that is similar to problems encountered in Navy Fleet Battle Experiment-Juliet. Several sets of results are presented, showing increasing learning capabilities of our agents.

1. INTELLIGENT AGENTS AND LEARNING

Learning plays a fundamental role in most human activities. Humans learn from their experiences, both successes and mistakes. Babies learn to walk and talk, adults learn foreign languages, new skills and concepts. Learning seems to be the fundamental property that allows humans to adapt to changes in the environment and to be successful. The skill of learning is of dynamic nature in humans who continuously acquire and modify representations of the world [Sun, 2001]. On the other hand, artificial systems such as software agents are usually static. For software agents to work properly, humans need to envision all contingencies and code them at the agent development time. Most agents react the same way to similar external stimuli, even when the previous reaction failed. Our goal is for the agent to learn different behavior in response to those stimuli. If software agents could “just” learn from their successes and mistakes, they would be more rapidly deployable and agent systems would have a longer life span. However developing a general learning method for software agents is a difficult proposition. Several attempts have been made in the past such as “conscious learning” mechanism [Ramamurthy, 1998], ABLE agents [Bigus, 2002], layered learning approach [Stone, 1998]. Our method differs from the ones proposed in the literature since it is capable of learning completely new agent behaviors by using Genetic Programming.

2. EXTENDABLE MOBILE AGENT ARCHITECTURE

Lockheed Martin Advanced Technology Laboratories (ATL) developed the Extendable Mobile Agent Architecture (EMAA). EMMA is used in about two dozen projects covering a full range of intelligent systems, including information management for time-sensitive strike [Hofmann, 2001], situation awareness for small military units, and executing user requests entered via spoken language dialogue. Starting with the Domain Adaptive Information System, experimentation with prototypes in military exercises has

guided our research and development towards the adaptable EMAA architecture. EMAA was used in the U.S. Navy Fleet Battle Experiment (FBE) series as a human aiding tool. From these experiments, an Interoperable Intelligent Agent Toolkit (I2AT) was created under DARPA Control of Agent Based Systems (CoABS) funding to allow people to construct agent systems without coding the agents from scratch, but by generating a workflow of tasks and parameters in a graphical fashion.

EMAA agents are designed with a workflow model for agent construction. An EMAA agent's workflow (Fig. 1) is a list of tasks, linked by execution paths that can be conditional or unconditional. Each task can take a certain number of task dependant parameters. The workflow architecture enables a new approach to agent development called agent composition, characterized by configuring and assembling elementary agent tasks into work flows. Typical tasks include queries to relational databases, retrieval of content from Web pages, reading and sending e-mail, etc. Key features have been added to EMAA that support agent learning. The goal of EMAA's new learning capability is allowing agents to adapt to the environment in which they operate.

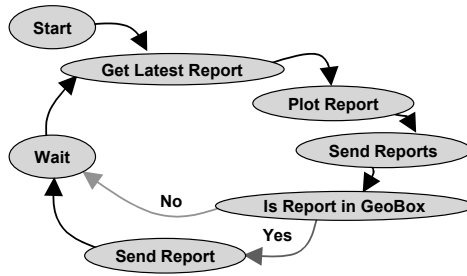


Fig. 1. Example of an EMAA Agent Workflow.

3. EVOLUTIONARY PLATFORM FOR AGENT LEARNING

Our approach to agent learning is based on Genetic Programming (GP). John Koza invented GP as a means of program synthesis by genetically breeding a population of computer programs using the principles of Darwinian evolution [Koza, 1993]. The basic operators of reproduction, crossover and mutation operate on individuals in the population and a fitness function describes how good a given individual is. In GP each individual is represented as a tree. In our approach (Fig. 2) GP evolves an agent workflow instead of directly evolving a computer program. The first step is to represent agent workflows in a GP tree form (these workflows are used in the initial population). GP individuals are translated from a tree format into a workflow, using their jar file descriptions from I2AT toolkit. The Activity Graph is then used to instantiate a composable agent, which is injected into the simulated system. Fitness is collected while the agent is run in the simulated environment.

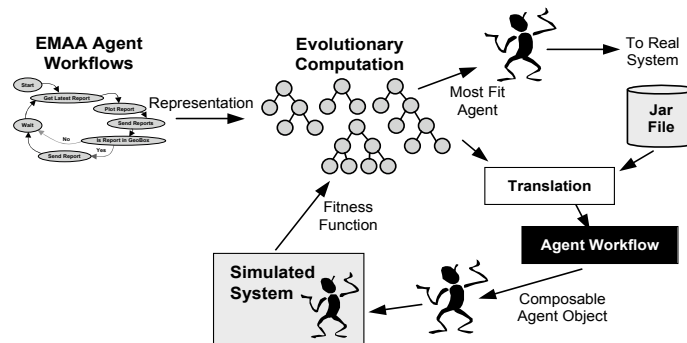


Fig. 2. Evolutionary Platform for Agent Learning.

GP operators work on agents' genetic material to generate new agents that have learned to overcome certain problems in their environment. Once the agents have learned, the most fit are injected into the real system and take the place of their parents who did not know how to overcome the problems. The method proposed is a general method that can generate completely new agent workflows, as well as related workflows but with new parameters. Augmenting an agent's workflow is synonymous with changing the agent's behavior.

In order to use GP for agent learning, several extensions to standard GP had to be developed. These extensions include: a novel GP tree representation that instead of two types of nodes has three node types (operators, tasks, task parameters), and restricted crossover and mutation operators that are guaranteed to produce legal offspring(s) from legal parent(s). The special mutation operators introduced include parameter mutation, task mutation, and operator mutation. The details of the representation and the new operators are described in our previous publication [Buczak, 2003].

Our GP implementation is based on Evolutionary Computation in Java (ECJ). ECJ [ECJ] is a flexible genetic algorithm and genetic programming freeware. Its source code is readily available which allowed us to make modifications necessary for agent learning.

4. FLEET BATTLE EXPERIMENTS: A REAL WORLD LEARNING SCENARIO

The scenario that we are using to demonstrate our agents' learning capabilities is similar to a real problem encountered in Fleet Battle Experiment-Juliet (FBE-J). In our scenario agents transmit vehicles' track information over a network and plot all the reports. Those track reports are coming from two vehicles: a fast vehicle moving at 100 knots and a slow vehicle moving at 10 knots. As message arrival rates increase, the network bandwidth becomes insufficient to transmit all messages, message queues grow and delivery delays grow. A solution to the problem encountered, suggested by a SPAWAR expert, was to exploit the differences in message urgency, i.e. match the rate of sending track updates to the urgency of the message. This typically means that slow moving tracks can be sent less frequently than fast moving tracks. Another possible solution would be to reduce the scope of the updates i.e. send updates in a smaller area of interest (smaller GeoBox).

The overarching goal for agents is to improve the network latency. One way agents can achieve this goal is to discriminate between fast and slow moving vehicles and to skip a number of slow reports. The appropriate ratio for filtering is proportionate to the ratio of the vehicle speeds (1:10). We make discrimination and filtering tasks available for EPAL with the filtering task parameterized by the number of messages to skip. We also make a GeoBox task available so that agents can choose a second way of filtering.

5. PARAMETER LEARNING

5.1 Experiment Description

The goal of the first EPAL test was to show parameter learning. This is a simpler form of learning where all agents in the initial population have the right workflow but none of them has the right task parameters. As such, the goal is to maximize the agent's fitness by adjusting the task parameters without changing the agent's workflow. In the parameter learning only parameter mutation was used (with probability 100%).

An important part of GP is the fitness of an individual. Our ultimate goal is to make agents learn everything, similarly to what humans do. Consequently we do not want to give too much guidance to the GP individuals. The fitness criteria we used to evaluate the

agents during learning: all fast reports should be displayed; and, slow reports should be displayed at the ratio between the average speed of the slow reports and the speed of the fast reports. This fitness is collected when the agent is executed, counting the number of slow and fast reports displayed. This is done realistically, and when the network slows down the number of fast and slow reports read and displayed by the agent changes.

In the experiments three experimental settings were varied: type of selection, maximum number of mutations per individual, and existence/absence of user guidance. The type of selection refers to the tournament or roulette wheel selection for choosing GP individuals for the next generation. The maximum number of mutations per individual specifies the number of times mutation will be called per individual (2 or 3). There is a 100% probability of the first parameter mutation, a 50% probability that there is a second mutation, and (in the case of 3 mutations) a 25% probability that there is a third mutation.

The existence/absence of user guidance specifies whether each parameter can undergo mutation (no guidance - undirected) or specifies which parameters can be mutated (user guidance - directed). In order to better explain it, let's look at Fig. 3 where the circles around the parameters indicate the way they can be mutated during learning. In case of undirected experiment, all the parameters in any type of circle can be mutated. In case of user directed experiment, only the parameters that are in solid line circles can undergo mutation. Since during mutation a new value is chosen at random from a set of legal values for a given parameter, there is always a non-zero probability that the same value will be chosen i.e. the parameter will not change (for most of the parameters this probability is very low).

For Parameter Learning we performed four sets of experiments. The first experiment uses tournament selection, a maximum of 2 mutations per agent, and no user guidance. The second experiment uses tournament selection, a maximum of 3 mutations per agent, and no user guidance. The third experiment uses tournament selection, a maximum of 2

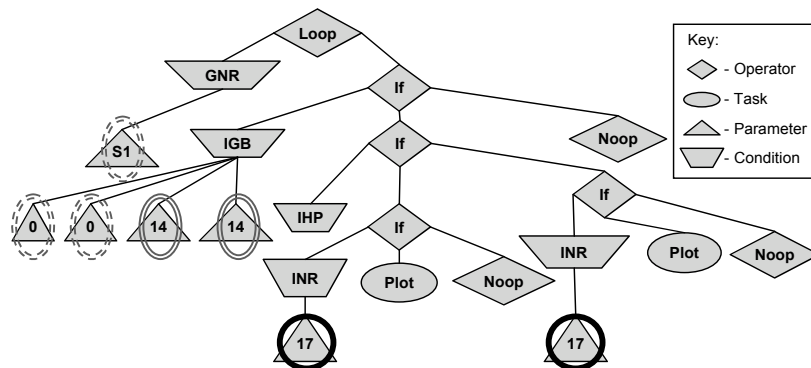


Fig. 3. The tree of the initial agent in parameter learning. The tree is read from the top down and left to right. GNR = Get Next Report, IGB = Is in GeoBox, IHP = Is High Priority, INR = Is Nth Report. In this case:

```
while (getNextReportFrom ("Sensor 1")) {
  GeoBox geoBox = new GeoBox(0,0,14,14);
  if (geoBox.isInside(report) { if (isHighPriority(report)) {
    fastCounter++;if((fastCounter modulo 17) == 0) {Plot(report);}
  } else { // report is low priority
    slowCounter++; if((slowCounter modulo 17) == 0) {Plot(report);}}}}
```

mutations per agent, and uses guidance. The fourth experiment performs roulette wheel selection, a maximum of 2 mutations per agent, and uses guidance. Each experiment was repeated several times (at least 32), and minimum, maximum, and mean generation number for finding the best agent were recorded. All the runs used a population of 50 GP trees. In the experiments the filtering values (under INR node on Fig. 3) are integers that can range from 1 to 100. The latitude length and longitude length (under IGB node) are rational numbers from 1.0 to 20.0, and all other parameters can assume only one value.

5.2 Results

Table 1 shows the results of the four experiments described in the previous section. EPAL was able to find the best agent in all the runs, regardless of the number of mutations used, type of selection, and whether user direction was given. The best results were obtained for experiment 3; on average only 11.03 generations were needed to find the optimal agent, and the maximum number of generations needed was the smallest of all experiments (32). This experiment uses user direction. User directed experiments (3 and 4) give better results than the undirected experiments—the reason being that the search space is smaller since only 2 parameters can be mutated instead of 7. A very large improvement occurs when the maximum of 3 mutations is used instead of the maximum of 2 (experiment 2 vs. experiment 1): on average 14.35 generations are needed, instead of 24.76. Experiment 2 results were very close to experiment 4 results, even though the search space was so much larger. This shows the power of performing more mutations on the agent. The best agent found in parameter learning experiments has the same workflow as on Fig. 3 but the parameters of two INR tasks are set to 1 and 10, thus it performs the perfect filtering.

Table 1. The results for parameter learning.

	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Ave number generations	24.76	14.35	11.03	13.94
Std deviation	4.99	1.044	0.59	0.55
Number runs	33	66	106	32
% runs best agent found	100%	100%	100%	100%
Min number generations	6	1	1	2
Max number generations	96	44	32	35

6. WORKFLOW LEARNING

6.1 Experiment Description

Workflow learning is a more difficult form of learning, since the agents do not have the right workflow, or the right parameters. When performing this type of learning, EPAL needs to generate new elements in the workflows, add new tasks and programming constructs, remove other tasks and programming constructs, and adjust the task parameters. Hence, workflow learning generates new agent behaviors. For this experiment the probabilities of different operations were as follows: reproduction - 0.25, crossover - 0.1, parameter mutation - 0.2, task mutation - 0.4 and operator mutation - 0.05. The fitness function is the same as in case of parameter learning.

In FBE-J, the agents had the workflow shown in Fig. 4. Those agents were getting reports from a sensor, and sending them if those reports fell inside a Geo Box. They did not perform any filtering of reports based on priority.

We used a population of 50 GP individuals with 50% of initial population with random workflows and 50% with skeleton workflows as the one on Fig. 4. None of the

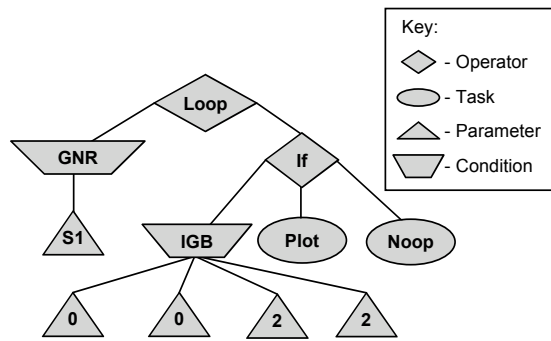


Figure 4. Best agent in generation 0 of workflow learning experiments. GNR = Get Next Report, IGB = In GeoBox:

```
while (getNextReportFrom ("Sensor 1")) {
  geoBox = new GeoBox(0,0,2,2);
  if (geoBox.isInside(report) {Plot(report);}}
```

This experiment was run about 10 times, and the “best agent” was found on average in 84 generations. EPAL found many solutions that exploited the ordering of the slow and fast reports, an example is shown in Fig. 5. In the lower right part of the tree, the check for the 18th report (INR) is outside of checking if the report is high priority (IHP). This means that the agent is filtering every 18th report, and if the reports are interleaved, it is equivalent to plotting every 9th of slow reports (ratio 1:9). However should the reports come in a different order, the filtering done by the agent won’t be acceptable. This means that the agent is taking advantage of the interleaved nature of fast and slow reports. To counter these effects, we designed a second experiment.

6.3 Experiment 2 Results

Our goal in the second experiment was to make the learning conditions for agents more

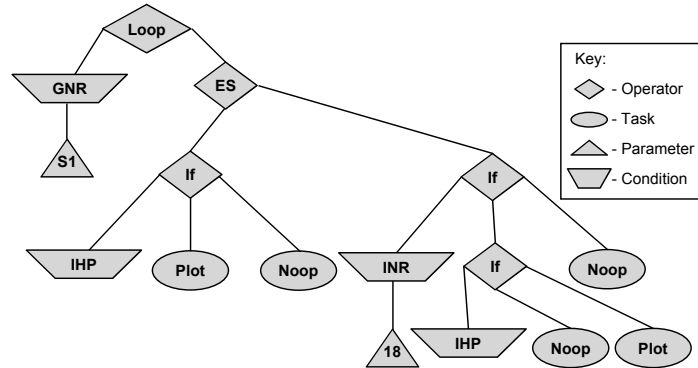


Figure 5. Best agent from one run of Experiment 1. GNR = Get Next Report, IGB = In GeoBox, IHP = Is High Priority, INR = Is Nth Report:

```
while (getNextReportFrom ("Sensor 1")) {
  if (isHighPriority(report)) {Plot(report);} counter++;
  if((counter modulo 18) == 0) {if (not(isHighPriority(report))) {Plot(report);}}
```

skeleton agents perform the filtering needed to satisfy the requirements. In order to be a fit agent, slow and fast re-ports need to be differentiated, and the slow reports need to be filtered. The tasks needed for this are “Is High Priority” task and “Is Nth Report” task.

6.2 Experiment 1 Results

In our first experiment, the input was very regular: fast and slow reports were always interleaved (fast, slow, fast, etc.) and the two vehicles’ x, y positions stayed the same during the tests.

realistic. The first real-world modification was that slow and fast reports could come in any order, with the probability of getting a slow or a fast report set at 0.5. The second modification was for the values in sensor reports themselves (describing x, y positions) not to be constants but to be provided by a sensor model interfacing with a Joint Semi Automated Forces (JSAF) simulator.

These changes led to very noisy fitness values, meaning that the fitness would significantly change for the same agent from generation to generation (depending on the order of slow and fast reports that the agent was scored on). To alleviate this problem, we modified EPAL to evaluate each agent three times on three random sequences of reports and additionally to compute an average fitness value based on all evaluations that the agent performed so far. If an agent survives many generations, the average fitness would be based on more evaluations, and would be less likely to have a good value due to idiosyncrasies with one set of sensor inputs. By using the realistic sensor data, the exact ratio of 10:100 knots in the speed of the slow to fast targets did not hold anymore. Moreover this ratio was not constant, varying between 9 and 10, and it was rarely an integer. Those changes made the learning problem much more difficult, albeit realistic.

EPAL learning was conducted for 400 generations. The experiment was run four times due to the length of each run. One run is discussed here. In generation 0 the best agent had the workflow of Fig. 4. Small improvements are made in consecutive generations. At around generation 117 EPAL creates an agent with the right workflow, but the filtering ratio is 1:7 instead of 1 to 9 or 10. By generation 337, the best agent in the run was found (Fig. 6) with a ratio of 1:8. It performs a loop in which it gets the next sensor report (GNR), checks if the report is high priority (IHP), if yes it plots the report. If the report is not high priority it checks if the report is in GeoBox (IGB - will always evaluate to true in this case). If it is, it plots every 8th report (INR), and does nothing with other reports. In other words this agent plots all fast reports and every 8th of slow reports.

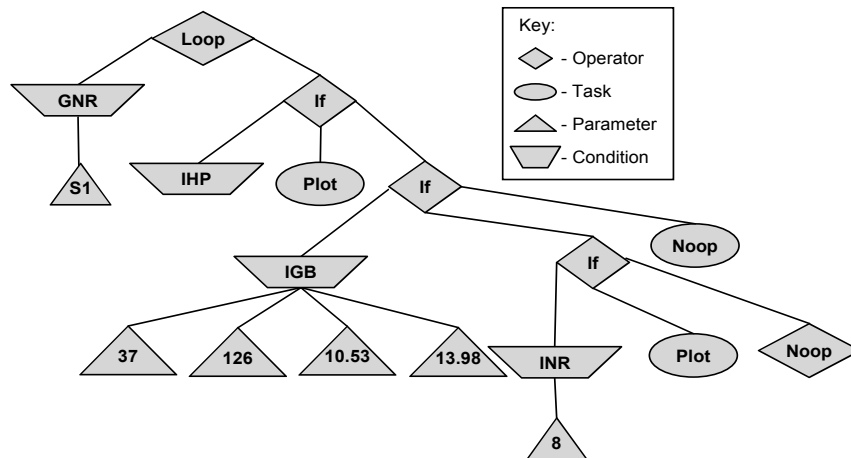


Fig. 6. Experiment 2, Best Agent of one run found in generation 337.

```

while (getNextReportFrom ("Sensor 1")) {
if (isHighPriority(report)) {Plot(report);}
else { GeoBox geoBox = new GeoBox(37,126,10.53,13.98);
      if (geoBox.isInside(report) { slowCounter++;if((slowCounter modulo 8) == 0) {
        Plot(report);}}}
}

```

By comparing the initial agent (Fig. 4) with the final one (Fig. 6), it is apparent how different the workflows are. The tasks added (in the right location) were IHP and INR. Comparing the workflow of the best agent from this run with the workflow that we made for parameter learning, notice that the filter (If, INR) for high priority reports is missing in the learned workflow. This makes sense since the problem requires no High Priority (fast) reports being filtered. The other difference is that the GeoBox is only utilized for slow reports. This didn't make a difference since the GeoBox is so large that it does not exclude any reports.

The overall results of workflow learning are very encouraging. Our agents were able to learn the new desired behavior on average in 350 generations. They did that even though the fitness function was noisy and constructed in such a way as not to suggest how to solve the problem. Very often the resulting workflows had many parts that were never executed: we developed a pruning algorithm that removes those parts.

7. CONCLUSIONS

EPAL is a framework for software agents to automatically learn how to better perform in their environment. The learning methodology we developed for intelligent agents is revolutionary. It has a capability to generate and discover completely new behaviors as well as adapt existing behaviors by changing task parameters. EPAL is a general framework for intelligent agent learning that can be used for any intelligent software agents as long as they are described by a workflow that can be further decomposed into simple building blocks. The uniqueness of EPAL comes from its ability to generate real software agents that can be executed in real military/other scenarios. To our knowledge the solution we formulated is the first platform in which real software agents can learn from their experience to perform standard tasks better. In the future, as we augment the set of tasks that agents can build their workflows from, agents should be able to "learn anything", similarly to what humans do.

8. REFERENCES

- Bigus, J.P., Schlosnagle, D.A., Pilgrim, J.R., Mills III, W.N., Diao, Y., 2002, "ABLE: A Toolkit for Building Multiagent Autonomic Systems," *IBM Systems Journal*, Vol. 41, No. 3.
- Buczak, A.L., Cooper, D.G., Hofmann, M.O., (2003), "Evolutionary Platform for Agent Learning", Intelligent Engineering Systems Through Artificial Neural Networks, eds. C.H. Dagli, A.L. Buczak, J. Ghosh, M.J. Embrechts, O. Ersoy, Vol. 13., pp. 201-206, ASME Press, New York.
- ECJ, www.cs.umd.edu/projects/plus/ec/ecj/.
- Hofmann, M.O., Chacón, D., Mayer, G., Whitebread, K.R., Hendler, J., 2001, "CAST Agents: Network-Centric Fires Unleashed," *Proceedings of the 2001 National Fire Control Symposium*, Lihue, HI, August 12-30.
- Koza, J., 1993, "Genetic Programming – On the Programming of Computers by Means of Natural Selection," Cambridge, Massachusetts, MIT Press.
- Ramamurthy, U., Bogner, M., Franklin, S., 1998, "Conscious Learning in an Adaptive Software Agent," *Proceedings of Second Asia Pacific Conference on Simulated Evolution and Learning*, Canberra, Australia.
- Stone, P., 1998, "Layered Learning in Multi-Agent Systems," PhD Thesis, CMU.
- Sun, R., Merrill and E., Peterson, T., 2001, "From implicit to Explicit Knowledge: A Bottom-up Model of Skill Learning," *Cognitive Science*, Vol. 25, No. 2, pp. 203-244.