Human-Assisted Neuroevolution Through Shaping, Advice and Examples

Igor V. Karpov, Vinod K. Valsalam and Risto Miikkulainen
Dept. of Computer Science, The University of Texas at Austin
1 University Station C0500, Austin, TX, USA
ikarpov@cs.utexas.edu, vkv@cs.utexas.edu, risto@cs.utexas.edu

ABSTRACT

Many different methods for combining human expertise with machine learning in general, and evolutionary computation in particular, are possible. Which of these methods work best, and do they outperform human design and machine design alone? In order to answer this question, a human-subject experiment for comparing human-assisted machine learning methods was conducted. Three different approaches, i.e. advice, shaping, and demonstration, were employed to assist a powerful machine learning technique (neuroevolution) on a collection of agent training tasks, and contrasted with both a completely manual approach (scripting) and a completely hands-off one (neuroevolution alone). The results show that, (1) human-assisted evolution outperforms a manual scripting approach, (2) unassisted evolution performs consistently well across domains, and (3) different methods of assisting neuroevolution outperform unassisted evolution on different tasks. If done right, human-assisted neuroevolution can therefore be a powerful technique for constructing intelligent agents.

Categories and Subject Descriptors

H.1.2 [**Information Systems**]: User/Machine Systems—*human factors, software psychology*

General Terms

Human Factors, Experimentation, Algorithms

Keywords

human-assisted machine learning, imitation, advice, machine-learning games, neuroevolution, shaping

Track: Digital Entertainment Technologies and Arts

1. INTRODUCTION

When building autonomous agents with complex behavior, such as non-player characters for a video game, or a robotic soccer team, machine learning techniques can be highly useful. For example, evolution of artificial neural networks with augmenting topologies (NEAT) has been demonstrated to be effective in the NERO machine-learning video game, where human players compete to design the most effective policy for agents in a complex virtual environment [10].

However, human domain experts often lack sufficient expertise in machine learning techniques to translate their knowledge into model parameters and training processes. Effective and natural ways to combine their domain expertise with machine learning are needed; such techniques would enable experts and non-experts alike to direct machine learning with their knowledge, thus speeding up the process of creating effective and interesting behaviors for autonomous agents.

Many candidate approaches to this problem of *human-assisted machine learning* have been proposed [4, 8, 1, 6, 12, 14]. In order to compare and contrast their advantages and disadvantages, this article presents a human subject experiment evaluating the relative merit of three possible approaches: giving advice in terms of rules, demonstrating the desired behavior through examples, and shaping machine learning through a sequence of gradually more challenging tasks.

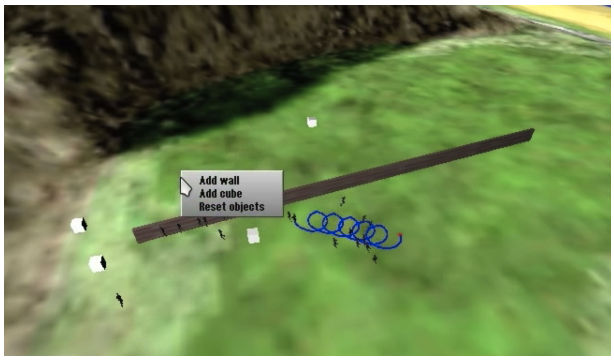


Figure 1: A screenshot of the 3D environment with a human user attempting to solve one of the test tasks using the shaping-assisted neuroevolution method.

In the experiment, the subjects took on the role of agent behavior engineers. Three different behavior design tasks commonly solved as part of the NERO machine-learning game were implemented in OpenNERO, an open-source platform for AI research and education (Figure 1). These tasks

GECCO'11, July 12–16, 2011, Dublin, Ireland.

were a simple navigation around an obstacle (basic navigation), catching a fast moving obstacle (dynamic navigation), and going to a number of targets in sequence (sequential behavior). The experiment held the machine learning algorithm constant—it used the NeuroEvolution of Augmenting Topologies (NEAT) method [11]—and varied the method that the human subjects used to assist neuroevolution in developing the desired policy. In addition to advice, examples, and shaping, neuroevolution alone and a fully manual approach, scripting, were tested as controls.

Each subject was assigned one of the three methods of assisting machine learning. They were then asked to solve each of the three tasks twice, once using the scripting method, and once using their assigned human-assisted neuroevolution method. All the interactions of the subjects with the system were logged via screen capture video and logs created by OpenNERO.

In all tasks, human-assisted and unassisted neuroevolution were more effective than manual scripting. Unassisted evolution was consistently good across all tasks; However, a different human-assisted method outperformed it in different tasks, revealing interesting differences and opportunities for them:

When using the *example* method, subjects were able to control an agent first person and to record an example behavior in the virtual environment. This example was then introduced as training data for supervised training of the evolving neural network controllers. This approach worked particularly well in the obstacle navigation task where the desired behavior is concrete and specific.

When using the *advice* method, subjects were able to write short snippets of example code and provide them to the evolving population. These snippets were converted into partial artificial neural networks and spliced into the networks within the population [13]. This approach was most effect in the moving obstacle tasks because desired behavior could be expressed as a general rule.

When using the *shaping* method, subjects were able to modify the task by adding and removing objects, moving, scaling and rotating objects, changing the trajectory, and speed of moving targets. The schedule on which this environment modification was combined with evolution was left up to the discretion of the engineer. This approach was best in the sequential task, where it was natural to make the task gradually more complex.

Overall, human-assisted machine learning turned out to be a powerful approach, given that the right method can be chosen for the task. The rest of this article is organized as follows. Section 2 describes related work in combining human design and machine learning. Section 3 briefly reviews the necessary background of real-time neuroevolution in machine-learning video games. Section 4 provides the details of the behavior design tasks created for the experiments, the architecture of the agents' sensors and actuators within the virtual environment, and the three alternative approaches to assisting neuroevolution with human knowledge. Section 5 describes how the experiments were conducted, and section 6 presents the results. Section 7 draws general conclusions about the results and outlines future work in leveraging human expertise with machine learning.

2. RELATED WORK

Almost any application of a machine learning technique to agent behavior design has a human designer bias due to the time spent by the designer in order to pick the right features or representation, tune algorithm parameters, select termination conditions or fitness/reward functions. However, recently a number of methods have been proposed that are specifically designed to be useful to domain experts without detailed access to the underlying machine learning algorithm [4, 8, 1, 6, 12, 14].

One possibility is shaping the learning process via human evaluative feedback. For example, the TAMER system combines human reinforcement with MDP learning in order to speed up the learning process and make it more sample-efficient [5]. Similarly, fitness shaping methods can be used with evolutionary computation as demonstrated with neuroevolution and the NERO machine learning game [10].

Another approach involves formulating knowledge about the domain as rules, and providing these rules as a useful addition to the learning algorithm. An early example of this type of approach includes the work by Maclin and Shavlik, where rules were used to modify a value function approximated with an artificial neural network [7]. More recently, modern natural language processing techniques have been recruited in order to free the human user from the need to use a specially crafted formal language [6].

A third approach to assisting learning agents involves demonstration of expert performance in the domain. For example, imitation learning has been applied to learning driving policies for virtual race cars [2]. A variant of the demonstration-based approaches is apprenticeship learning through inverse reinforcement, where the expert behavior is used to estimate the reward function which is then used for learning in conjunction with a model of the environment [1, 9].

The power of such approaches has been demonstrated many times. For example, apprenticeship learning was used to successfully learn autonomous control for aerobatic helicopters [3]. Additionally, some work has started to consider the human factor in the interaction between the human teacher and the machine learner more directly [15]. However, it remains unclear how these different methods of providing assistance to machine learning methods compare against each other, and against manual policy design methods.

3. BACKGROUND

The particular machine learning technique evaluated in this paper is neuroevolution, i.e. evolving neural networks through genetic algorithms, in particular through the NEAT method. The tasks are chosen from the general framework of the NERO machine learning game, where the human player faces the challenge of designing effective behaviors for the agents in the game.

3.1 Machine Learning Games

The NeuroEvolving Robotic Operatives game (NERO) belongs to a growing new genre of *machine-learning video games*, where agents that learn from experience are inte-

gral to game play [10]. In NERO, the human players compete by using neuroevolution to evolve teams of agents that are effective at combat and other tasks within a complex virtual environment. They do so interactively and in real time, by shaping the real-time neuroevolution of a team of agents through a combination of modifying the virtual environment, adjusting the relative importance of the multiple components of their fitness (or objective) function, and explicitly punishing unwanted behavior by lowering its fitness or interacting with the simulation from a first-person perspective.

OpenNERO is a freely available open source research platform that includes NERO and other environments, and provides several tools for constructing new environments, tasks, and learning algorithms. It includes several reinforcement and other learning methods, supports comparative experiments between them, and provides a way to visualize them using a modern 3D game engine.

3.2 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) and its real-time, steady-state variant Real-Time NEAT (rtNEAT) are two evolutionary machine learning methods well suited for designing complex controllers [11]. NEAT evolves artificial neural network controllers by starting with a population of single layer, fully connected feed-forward neural networks with random weights. During evolution, it complexifies these networks through crossover and mutation operators that add new nodes and connections to the network, as well as change its weights. NEAT has been shown powerful in several applications involving evolution of intelligent agent behavior (including NERO); it also supports several human-assisted methods naturally, and is therefore used as the machine learning method for the experiments in this paper.

4. SYSTEM DESCRIPTION

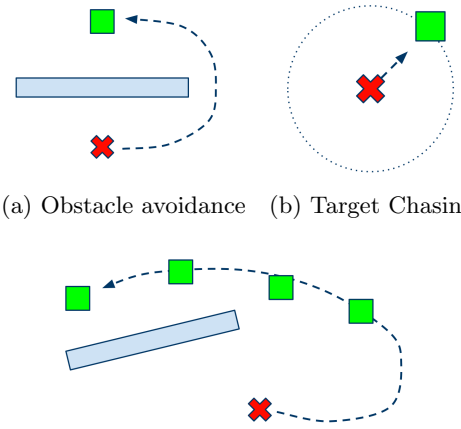
The experiments described in this article were performed using three *behavior design tasks* implemented within the machine learning video game OpenNERO (Section 4.1). The tasks involved training a team of virtual agents (Section 4.2) using one of three approaches to human-assisted neuroevolution (Section 4.4) and a manual scripting approach.

4.1 Behavior Design Tasks

In order to explore the breadth of possible tasks that might face a designer of game bot behavior, three tasks were created for the experiment (Figure 2).

One task commonly encountered by game bots is that of *obstacle avoidance*, where the bots have to navigate to a target around some level geometry. In order to test this scenario, the subjects were presented with an obstacle avoidance task (Figure 2a). The fitness on the task is defined as 1.0 or greater if the agent starting at the origin reaches the goal and collects the target cube positioned there.

The second task tested in the experiment is that of a moving target chasing (Figure 2b). The task consists of a target moving along a predefined trajectory (in this case, a circle). The goal of the agent is to reach the moving target and to collect the 1.0 fitness by doing so. Another part of the fitness is normalized to range between 0.0 (agent ending far away from the target) and 0.1 (agent ending close to the target).



(a) Obstacle avoidance (b) Target Chasing

(c) Resource gathering task

Figure 2: The three tasks used during the experiments. The blue bars represent walls, the red crosses represent the spawning location of the agents, and the green boxes represent the targets that the agents were required to reach in order to complete the task.

The third task tested in the experiment is that of multiple resource gathering (Figure 2c). The task consists of four resources positioned behind two walls. It is possible for an agent to move around the walls and collect all four, but not with a lot of time to spare. The agent receives a +1 towards its fitness for reaching a resource, and a fraction of 0.1 for getting close to one.

4.2 Agent Configuration

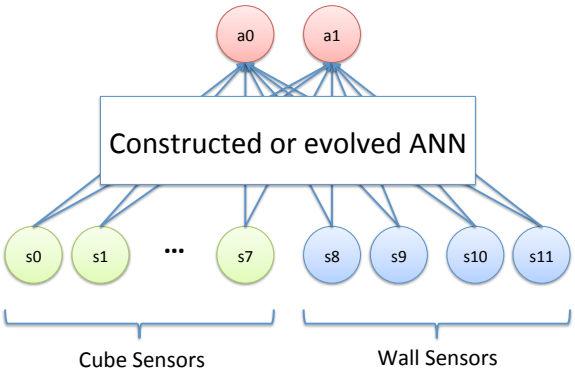
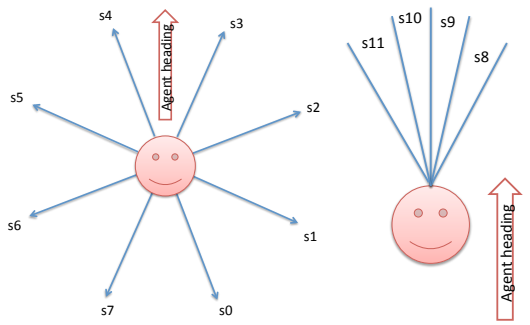


Figure 3: The architecture of the agent. The agent has twelve sensors and two actuators. The actuators determine the speed and direction of motion, while the sensors provide information about the location of the targets and the walls in the environment around the agent. The controller is always a neural network, either constructed manually from rules, or evolved using NEAT (with or without assistance).

The overall agent architecture is depicted in Figure 4. Each agent can sense the virtual environment through a set of egocentric sensors and act in the environment using a set of actuators. The sensors consist of target sensors (Fig-



(a) Target sensors (b) Wall sensors

Figure 4: The sensor of the agent consist of target sensors (a), and wall sensors (b). The observations of the target values vary with proximity and with with number of targets, while the wall sensors only report whether or not a collision in that direction has occurred.

ure 4a) and wall sensors (Figure 4b). All observations were normalized to lie within the range $[0, 1]$.

The target sensors' value increases linearly with relative proximity of the target object (a blue cube) and with angular alignment of the sensor axis with the ray cast towards the location of the target. The value is cumulative for all targets in the environment. The targets disappear from agent's view after they are visited once. The wall sensors return binary observations that report whether or not a ray cast in the direction of the sensor intersects a wall in the environment.

The agents have two continuous action outputs. The actuator a_0 sets the speed of forward motion: it stops the agent when its value is set to 0, and moves the agent at the maximum allowed speed as the value approaches -1 or 1. The actuator a_1 sets the turn rate, ranging from -1 (maximum left turn) to 1 (maximum right turn).

The fitness is defined as $+1.0$ for each target reached, and a fraction of 0.1 for approaching the target faster, for getting closer to a target, or for closely following the example trace.

4.3 Manual Design Through Scripting

One possible approach to designing a complex behavior (and the one usually used in game industry) is to simply write a deterministic program to execute it. Thus each subject in the experiment was asked to solve the tasks using a custom scripting language that consists of a collection of if-statements that determine the actions of the agent given its current inputs (figure 5).

A simple procedural language based on KBANN (Knowledge-Based Artificial Neural Networks; [7]) and in earlier work on human-assisted neuroevolution [13] was implemented in OpenNERO. The language consists of simple if-then-else constructs, where the if-statement conditions are based on binary comparisons of sensor values and then- and else-clauses can assign constant values to output actuators (Figure 6) or to temporary register values. The script is processed using a lexical analyzer and parsed using a parser generated from a set of context-free grammar rules (Figure 5).

Using KBANN, the scripts written in this language are compiled into neural networks that implement the desired behavior. In the fully manual scripting case, these networks

rules → rule | rules rule
rule → **if** conds setrules iftail
rule → **if** conds setrules iftail
iftail → | **elif** setrules iftail
conds → term | conds **and** term
term → **false** | **true** | variable termop varvar
termop → ≤ | < | > | ≥
varval → **variable** | **value**
setrules → setvar | setrules setvar
setvar → **variable** = expr
expr → eterm | expr + eterm | expr − eterm
eterm → **value** | **variable** | −**variable** | **value***variable

Figure 5: The context-free grammar used in parsing script and advice. The primary construct is an if-then-else clause, and the primary statement is assignment. The ground symbol **value** stands for a floating point number, and **variable** can be a sensor name, an actuator name, or a temporary register name.

are used as the sole controllers. However, these networks can also be used as pieces of advice to the neuroevolution, as described next.

```
# Go to a target
if s3 < s4 {
    a0 = 1
    a1 = 1
} else {
    a0 = 1
    a1 = -1
}
```

```
# Go around a wall
if s9 > 0.5 {
    a0 = 1
    a1 = 1
} elif s10 > 0.5 {
    a0 = 1
    a1 = -1
}
```

Figure 6: Example rules for going to a target (a), and going around the wall (b). Variable names starting with 's' refer to sensor values, and those starting with 'a' refer to actuator settings. These rules can be used either as standalone scripts, or combined with neuroevolution as pieces of advice. While these scripts are relatively simple, the process of formulating such rules for an agent behavior can take a long time.

4.4 Human-Assisted Neuroevolution

Neuroevolution provides a suitable testbed to compare different ways of integrating human knowledge because it is capable of supporting many of them. For the experiment presented in this article, the subjects were asked to train populations of NEAT agents to complete a number of tasks using one of four approaches defined below.

4.4.1 Advice

The scripting language described above can also be used to formulate pieces of advice, instead of the entire behavior at once. Such advice can then be converted into small neural networks, and spliced into the networks of the actively evolving population. It is easy for NEAT to incorporate such advice because it adds nodes and connections are to evolving networks already as part of evolution. If the advice network provides a useful modification to the individual's behavior, it will be incorporated into subsequent generations, speeding

up learning. If the piece of advice is not beneficial, it will be selected against or even reused for something else later in evolution.

4.4.2 Examples

A second way to assist evolution of agents is to provide them with a trace of an example behavior. The subjects were asked to provide such a trace by controlling an agent from a third-person perspective via keyboard commands.

In order to match the agent behavior with examples of different length and complexity, a segment-based approach was developed. A trace is recorded and stored as a sequence of agent positions. It is then broken up into segments 10 steps long. Each neural network is first evaluated on how well it matches the example, and then it is trained to match it better using backpropagation.

During evaluation, the network is allowed to move the agent for 10 steps starting at the beginning point of the first segment. If its path deviates too much from the example, the network is trained with backpropagation on this segment. If the network's path follows the example segment to within a small deviation, the next segment of the path is evaluated, until all segments have been processed. The network's fitness is a fraction of segments successfully matched.

During backpropagation training, the output actions that would have generated the example path are used as targets, and the input activations resulting from the agent's location as the input. Each step through the segment generates one backpropagation input/output pair; the network is trained on each pair once, backpropagating the error through the network topology until all weights leading from the inputs are updated. The backpropagation changes are then encoded back into the genetic representation of the network, and are thus inherited by its offspring.

4.4.3 Shaping via Environment

A third way to assist evolution of agents is to shape their behavior by modifying the task gradually. The tasks were instrumented with a graphical user interface with which the subjects could modify the environment and task parameters, thereby guiding the process of evolution. The idea is that the engineer will be able to find the appropriate *task decomposition* using such a method: After a simpler version of the task is solved, a more complicated version of the task may become easier for neuroevolution to master. Several different kinds of shaping approaches were possible:

In free environment modification, the user adds or removes targets and walls at any location on the field. The user can also scale and rotate these objects, as well as move them from one location to another.

In restricted environment modification, specific keys on the keyboard are used to add, reset or remove the objects that were part of the original task.

In parametrized environment modification, the subjects are able to modify the speed and trajectory of the moving target in the dynamic target domain.

The subjects did indeed use each of these in the process of attempting to discover a useful shaping strategy.

Table 1: Attempted solutions by task and method

	Obstacle	Chasing	Resource	**Total**
Scripting	14	16	14	44
Examples	5	6	6	17
Shaping	6	6	6	18
Advice	4	3	2	9
Total	29	31	28	88

5. EXPERIMENTS

In order to compare the different methods of assisting evolution in constructing agent behavior, a human subject study was conducted. Participants in the study took on the roles of engineers attempting to create agents with a particular behavior in order to solve three machine-learning game tasks defined below.

5.1 Experiment Setup

The interactions with the system were recorded and analyzed to extract the success rate in solving the tasks using the different methods, how long it took to solve them, and how many times they had to restart before reaching a solution.

Additionally, in order to compare the performance of the human-assisted evolution and human manual scripting to evolution alone, 30 runs of evolution were performed on each of the tasks defined below.

In all of the experiments that use neuroevolution, the same algorithm (NEAT) with the same set of parameters was used. A population of size 50 was used in all cases. When time comparisons are made, the time is always wall clock time. Both the evolution runs and the human subject experiments were conducted on identical machines with *Intel* Core 2 Duo CPUS running at 1.83 GHz and configured with 2 GB of RAM.

5.2 Experimental Protocol

Sixteen subjects were drawn from a freshman research course. The subjects had experience using neuroevolution in the context of a machine-learning game, and had a lecture describing how the method works.

All subjects were briefed before the experiment, describing the timeline of the experiment, the methods they were allowed to use, the user interface they were using for the experiment, the tasks they were solving, and the goal they had to achieve within each task.

Each subject was randomly assigned a sequence of three tasks and a collection of two methods. They then used the two methods to solve each of the the three tasks, in order, for a total of six task and method combinations. Subjects were not allowed to work longer than 30 minutes on each combination. The subjects were given a total of 150 minutes to complete as many combinations as they could. The number of task and method combinations attempted are summarized in Table 1. Each subject could attain at most one successful solution for each of the six task-method combinations they were assigned.

All subjects filled out a survey immediately after the experiment rating the quality of their solutions, the quality of the approaches they used, how many times they had to restart on each task-method combination, and whether or not they understood the task.

When running neuroevolution, subjects had the choice of running it in "visible" mode that shows every action of every agent, or in "headless" mode, that runs evolution in the background and can make progress four to five times faster. Running neuroevolution alone was done using the fastests "headless" mode.

6. RESULTS

The results of the experiments are divided into two broad categories. The survey results are self-reported by the subjects who filled out a short online survey in the lab immediately after participating in the experiment. The recorded results are extracted from the game logs and from the video screen capture of the sessions.

All results presented in this section exclude the first task-method combination for each subject. This was done because leave-one-out analysis shows an acclimatization effect of the subjects where they improve their performance after having gotten used to the interface during the first trial.

6.1 Survey Results

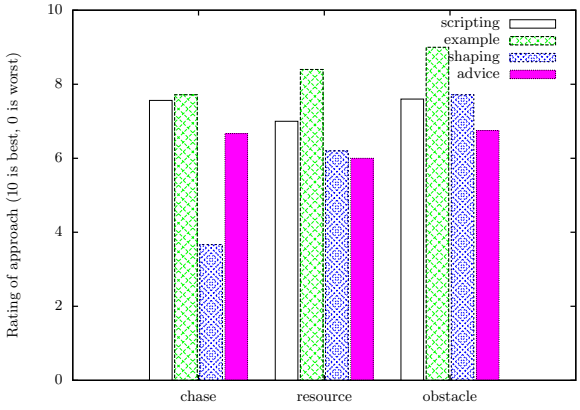


Figure 7: Average approach quality ratings collected during the post-experiment survey. While these scores are subjective (in particular, example is rated highly because it is most fun to use), some trends correlate with the recorded results: shaping is really difficult on the chase task, advice is rated lowest on resource collection, and all methods are rated relatively highly on the obstacle task.

During the post-experiment surveys, the subjects were asked to rate the quality of the approach they were using to solve the tasks (Figure 7). The highest rating was given to example, followed by scripting, shaping, and advice, respectively. These subjective ratings did not always correlate with the objective rate of success achieved on the tasks. They appear rather to contain a substantial subjective "ease of use" components that do not directly translate into functionality and effectiveness but can make the process of solving the task more fun.

6.2 Recorded Results

In order to collect objective measures of the performance of these methods, the interactions of the users with the OpenNERO interface were logged and recorded via screen capture. Three main measures were collected - the rate of success (i.e. whether or not the subject was able to solve a particular task using a particular method within the allotted time), the time to solution (how long the subject took to solve the task), and the number of restarts (how many times the subject restarted the interface before reaching a solution).

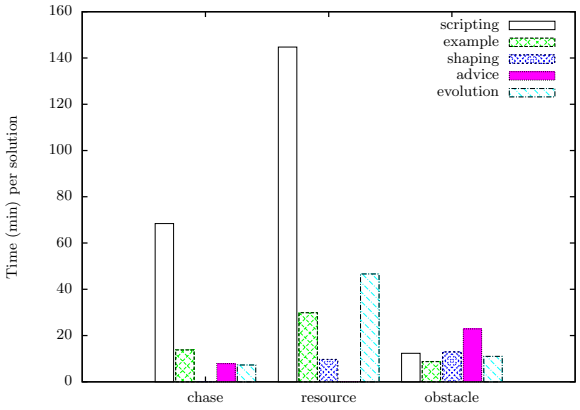


Figure 8: Average time spent by subjects before successfully solving each task. Bars not shown indicate an unknown value because no successful solutions were seen in the data. Evolution alone does consistently well and outperforms scripting, but is always outperformed by one of the human-assisted methods.

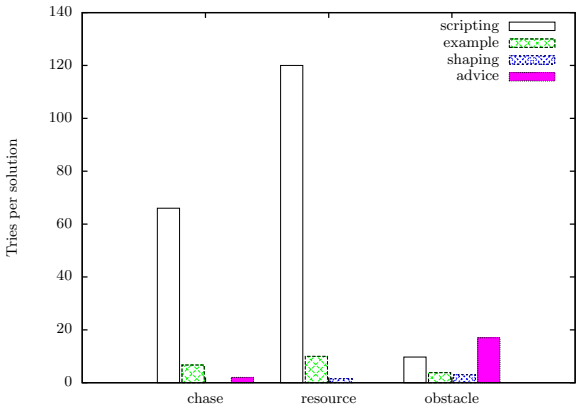


Figure 9: Average number of retries by subjects before successfully solving each task. Bars not shown indicate an unknown value because no successful solutions were seen in the data.

The average time per solution (the sum of the total time taken by all subjects divided by the number of successful solutions) is shown in Figure 8. An equivalent measure is also shown for neuroevolution without human assistance. This measure represents the expected amount of time one would have to wait before a solution is obtained, for each task and method. In all three cases, evolution performs well and outperforms scripting. Additionally, at least one human-assisted method outperforms neuroevolution alone on each task. The average number of restarts per successful solution is shown in Figure 9 and confirms this trend.

Another interesting result of the experiments becomes apparent when considering the detailed timelines of the success

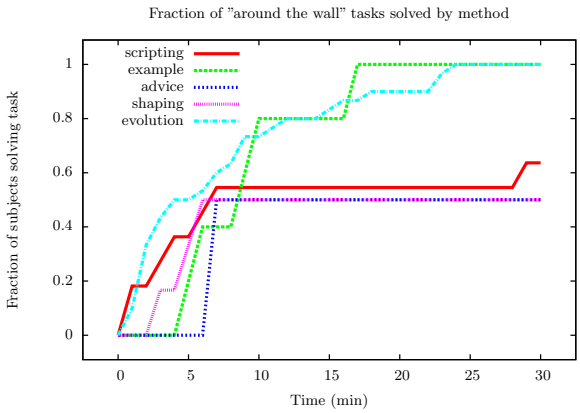


Figure 10: Fraction of "around the wall" tasks solved by each method over time. Here, scripting, shaping, and advice do not work as well as evolution alone, but example allows all the subjects to solve their task before evolution does, using fewer samples of the simulation in the process.

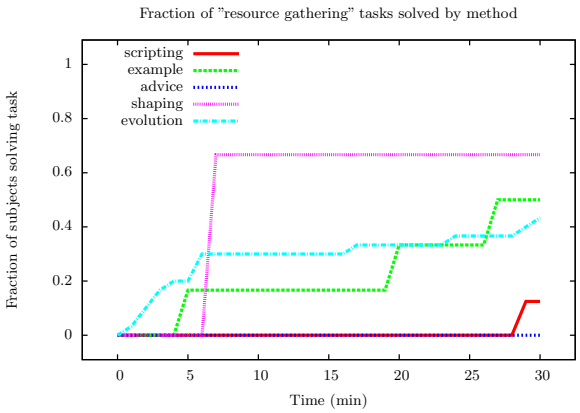


Figure 12: Fraction of "resource gathering" tasks solved by each method over time. Here, the task is more challenging for all methods, but the best method turns out to be shaping because the task can be easily decomposed by modifying the environment.

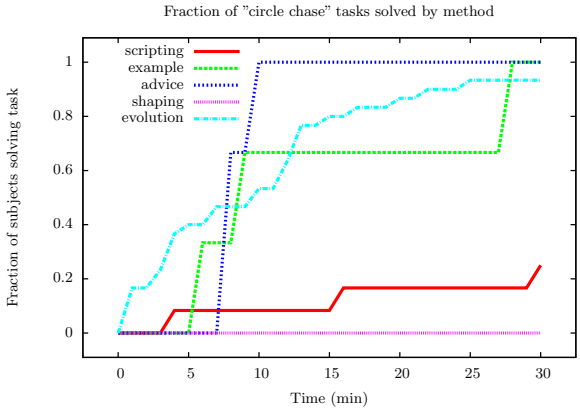


Figure 11: Fraction of "circle chase" tasks solved by each method over time. In this task, shaping and scripting do not work at all, but advice allows the subjects to solve all the tasks before evolution alone.

rates of the methods (Figures 10, 11, and 12). In each case, unassisted and human-assisted evolution perform much better than scripting. While neuroevolution performs consistently well across the domains, it is outperformed by one of the human-assisted methods in each task - a different one in each case. While example traces work best in obstacle avoidance, advice works best in target chasing and shaping works best in the resource gathering task.

These results confirm that both neuroevolution and human-assisted neuroevolution outperform the manual scripting approach and that human-assisted neuroevolution is most effective, given the right approach to the task.

7. DISCUSSION AND FUTURE WORK

The result that different human assisted neuroevolution methods are best in different tasks makes sense when one considers the inherent characteristics of the tasks.

First, in the simple obstacle avoidance task, all methods work relatively well and the intuitive nature and ease of use of the example method allow users to complete the task in a short amount of time.

In the target chase task, in contrast, while it remains possible to solve the task with examples, the examples that are first attempted by the subjects are not as helpful, and the process therefore takes a longer time. Also, the task does not lend itself well to shaping because it contains a conceptual discontinuity: the behavior required to reach slow moving targets or targets that move within a smaller radius is drastically different from that of the complete solution. Thus shaping by modifying the environment is not a good fit for this task. However, formulating what needs to be done as either advice or script is relatively easy in this task, and neuroevolution with advice has an advantage because it can recover from mistakes made during rule design and learn some of the task autonomously.

In the resource gathering task depicted in Figure 2c, the complicated nature of the environment and the possibility to decompose the task into logical subtasks allow shaping to outperform advice and example. In particular, an effective and simple shaping strategy is to simply remove all the targets except for the one that should come first in the sequence, and allow neuroevolution to master that task first, after adding the next target in the sequence and repeating until the complete task is learned.

In addition to performing well compared to unassisted neuroevolution in real time, it is important to note that human assisted neuroevolution is much more *sample-efficient* in that it requires many fewer interactions with the environment per unit time. In particular, while the entire time during the neuroevolution runs was spent evaluating neural network controllers in the environment at the highest possible speed, the human-assisted methods dedicate a fraction of time to other activities, such as writing scripts and advice pieces, providing example behavior traces, running neuroevolution in the slower real-time mode in order to verify the agents' performance, or simply thinking about the problem. Therefore it is a particularly good approach when testing candidate solutions is expensive.

The results of such experiments can be used to select and customize human-assisted evolution methods for building autonomous agents in virtual environments, as well as autonomous robots that interact with and learn from humans. They may also be used to train machine learning agents that adapt to a changing task on the fly. Most importantly, they give human engineers a degree of control that may make it easier for them to incorporate advanced machine learning techniques into the design process.

8. CONCLUSIONS

This article described the design and results of a human subject experiment evaluating three different approaches to combining human domain knowledge with neuroevolution, comparing them with both completely manual design and completely automated discovery through neuroevolution. The results demonstrate that (1) human-assisted evolution outperforms a manual scripting approach, (2) unassisted evolution performs consistently well across domains, and (3) different methods of human-assisted neuroevolution outperform unassisted evolution on different tasks. Therefore, if done right, human-assisted neuroevolution can be powerful technique for constructing intelligent agents in the future.

9. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grants IIS-0757479, IIS-0915038, and DBI-0939454. The authors would also like to acknowledge the Freshman Research Initiative at the University of Texas at Austin and the students of the Computational Intelligence in Game Design stream for participating in the study.

10. REFERENCES

[1] P. Abbeel and A. Ng. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the twenty-first International Conference on Machine Learning ICML'04*, page 1. ACM, 2004.

[2] L. Cardamone, D. Loiacono, and P. L. Lanzi. Learning Drivers for TORCS through Imitation Using Supervised Methods. In *Proceedings of the IEEE 2009 Symposium on Computational Intelligence and Games (CIG'09)*, 2009.

[3] A. Coates, P. Abbeel, and A. Y. Ng. Apprenticeship Learning for Helicopter Control. *Commun. ACM*, 52:97–105, July 2009.

[4] W. Knox and P. Stone. Interactively Shaping Agents via Human Reinforcement: the TAMER Framework. In *Proceedings of the 5th International Conference on Knowledge Capture*, pages 9–16. ACM, 2009.

[5] W. Knox and P. Stone. Combining Manual Feedback with Subsequent MDP Reward Signals for Reinforcement Learning. In van der Hoek, Kaminka, Lespérance, Luck, and Sen, editors, *Proceedings of the 9th Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 5–12, May 2010.

[6] G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *The AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems*, July 2004.

[7] R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22:251–281, 1996.

[8] M. N. Nicolescu and M. J. Mataric. Learning and Interacting in Human-Robot Domains. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 31(5):419–430, 2002.

[9] D. Ramachandran and E. Amir. Bayesian Inverse Reinforcement Learning. In *Twentieth International Joint Conference on Artificial Intelligence*, 2007.

[10] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, 2005.

[11] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[12] L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Skill acquisition via transfer learning and advice taking. In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML'06)*, pages 425–436, Berlin, Germany, 2006.

[13] C. H. Yong, K. O. Stanley, I. V. Karpov, and R. Miikkulainen. Incorporating Advice into Neuroevolution of Adaptive Agents. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE'06)*, pages 98–104, 2006.

[14] P. Zang, A. Irani, P. Zhou, C. L. Isbell, and A. Thomaz. Using Training Regimens to Teach Expanding Function Approximators. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

[15] P. Zang, R. Tian, C. L. Isbell, and A. Thomaz. Batch versus Interactive Learning by Demonstration. In *Proceedings of the Ninth International Conference on Development and Learning (ICDL)*, 2010.