Learning from Demonstration with Swarm Hierarchies

Keith Sullivan
Department of Computer Science
George Mason University
Fairfax, VA 22030
ksulliv2@cs.gmu.edu

Sean Luke
Department of Computer Science
George Mason University
Fairfax, VA 22030
sean@cs.gmu.edu

ABSTRACT

We present a supervised learning from demonstration system capable of training stateful and recurrent collective behaviors for multiple agents or robots. A model space of this kind is often high-dimensional and consequently may require a large number of samples to learn. Furthermore, the inverse problem posed by emergent macrophenomena among multiple agents presents major challenges to supervised learning methods. Our approach reduces the size of the state space, and shortens the gap between individual behaviors and macrophenomena, by manually decomposing individual behaviors and arranging the agents into a tree hierarchy. This makes it possible to train potentially large numbers of agents using a small number of samples. We demonstrate our system using hundreds of agents in a simulated foraging task, and on real robots performing a collective patrolling task.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*

General Terms

Performance

Keywords

Learning from Demonstration, Multiagent Systems, Robotics

1. INTRODUCTION

Programming agent behaviors is a tedious, time consuming task involving multiple code, test, and debug cycles. Creating these behaviors requires significant programming ability, which makes training the agents attractive. One training approach is Learning from Demonstration (LfD), where the agent learns a behavior in real-time based on examples provided by a human demonstrator. LfD teaches an agent a policy which maps environmental features to agent actions.

Supervised learning methods are a natural fit for LfD, as the trainer is directly providing examples. But we note that supervised *cooperative multiagent training* has a surprisingly small literature. From an extensive survey of cooperative multiagent learning [15] it was found that only a small number of papers deal with supervised learning, and most of those are in the area of *agent modeling*, whereby agents learn about one another, rather than being trained by the experimenter. The lion's share of the remaining literature tends to fall into feedback-based methods such as reinforcement learning or stochastic optimization (genetic algorithms, etc.). For example, in one of the more well-known examples of multiagent layered learning [17], the supervised task ("pass evaluation") may be reasonably described as agent-modeling, while the full multiagent learning task ("pass selection") uses reinforcement learning. This is not unusual.

Why is this so? Supervised training, as opposed to agent modeling, generally requires that agents be told which micro-level behaviors to perform in various situations; but the experimenter often does not know this. He may only know the emergent macro-level phenomenon he wishes to achieve. This inverse problem poses a significant challenge to the application of supervised methods to such problems. The standard response to inverse problems is to use a feedback-based technique. But there is an alternative: to decompose the problem into sub-problems, each of which is simple enough that the gulf between the micro- and macro-level behaviors is reduced to a manageable size. This is our approach.

Our multiagent training method rests upon an LfD system we have developed called *Hierarchical Training of Agent Behaviors* (or HiTAB). In its basic form this system is a *single*-agent training system which learns a hierarchical finite state automaton (HFA) represented as a Moore machine. Individual states in the automaton either correspond to agent behaviors, or may themselves be another HFA. An HFA is constructed iteratively: using with a behavior library consisting solely of atomic behaviors (e.g., turn, go forward), the demonstrator trains an automaton describing a more complicated composed behavior, which is then saved to the behavior library. The now expanded behavior library is again used to train a more abstract and capable automaton, which is likewise saved to the library. This process continues until the desired behavior is trained.

Our goal is to apply this training technique not just to single agents but to supervised training of teams and swarms of arbitrary size. Our approach is as follows. We organize the agents into an *agent hierarchy*, a tree structure where leaf nodes are the individual agents or robots performing tasks, and non-leaf nodes are (possibly virtual) *controller agents*. After we have trained and distributed individual behaviors to the leaf-node agents using HiTAB, we the group them into small, manageable teams (perhaps of size five), each headed

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), June, 4–8, 2012, Valencia, Spain.

by a controller agent. We then train the controller using HiTAB in much the same way that the individual agents were trained: but his states dictate the collective behaviors of his small team. After we have trained the small team, we group controller agents together in teams, each such team headed by a higher-level controller agent. This training and grouping continues until the entire swarm has been organized into a hierarchy.

This tree-structured organization fits between fully decentralized ("swarm"-style) multiagent systems and fully centralized systems. While the tree structure has obvious disadvantages (e.g., it is not robust to agent failure), it has one overriding scaling advantage: regardless of its size, at any position in the structure an agent must deal only with a fixed number of agents (his superior and immediate subordinates). We are taking advantage of this to make the multiagent training task feasible regardless of the size of the swarm. At all times we are training a controller to direct a small number of agents (his immediate subordinates), regardless of the position of the controller in the hierarchy. The micro-to-macro gulf is much smaller and simpler with five or fewer agents than it it is with hundreds or thousands of agents. Furthermore, the use of HFAs at the controller level allows us to decompose complex team behaviors into simpler ones in much the same way that we simplified the problem in the single-agent HiTAB case.

Hierarchies are a natural fit for organizing heterogeneous agent swarms, but interestingly they're also useful for swarms of agents with homogeneous behaviors too. In this paper we show this: we train hierarchies of behaviors to control *homogeneous* agents, and demonstrate trained behaviors which are superior to those found in flat ("swarm"-style) structures.

2. RELATED WORK

Agent Hierarchies. Hierarchies have long been employed to control a robot programmatically, from the traditional multi-tier planner/executive/control hierarchical frameworks, to *behavior hierarchies* establishing precedence among competing robot behaviors, of which an early example is the Subsumption architecture [2]. A significant body of literature has constructed groups of agents, with each agent employing its own internal hierarchical behavior mechanism [16, 5, 21]. Hierarchies *among* agents are less common, for example [6]. Some recent literature has focused on *hierarchies of control* among heterogeneous agents [8]. Hierarchies may also be constructed dynamically as a mechanism for task allocation [12].

Learning from Demonstration. Much of the learning from demonstration literature may be divided into systems which learn plans (for example [14, 20]) and those which learn *policies*, that is, stateless mappings from the agent's feature vector to a desired action [1, 4, 10, 13]. Some work involves stateful models related to ours, notably via Hidden Markov Models. For example, [9] treat states not as behaviors but as hidden world conditions which the learner is attempting to discover and optimize for. [7] learns transitions between states corresponding to behaviors, though it does not label the transitions.

Multiagent Learning. One of the primary challenges addressed by this paper is in applying learning from demonstration — at its heart a supervised task — to the multiagent case. As noted in [15], supervised learning methods are not very common in multiagent learning: by far the lion's share of literature is based on reward-based methods such as reinforcement learning or stochastic optimization. Of those supervised methods, many fall in the category of *agent modeling*, where agents learn about one another rather than about a task given to them by demonstrator. For example, in the celebrated [18], the supervised task ("pass evaluation") is reasonably described as agent modeling, while the full multiagent learning task ("pass selection") uses reinforcement learning. Multiagent learning may also be achieved via confidence estimation rather than reinforcement learning [3]. An alternative way to bridge the macro-micro gulf is to eliminate the macrobehaviors entirely by issuing separate micro-level training directives each individual agent [11, 19]. We argue that this approach is unlikely to scale.

3. AGENT BEHAVIOR TRAINING

HiTAB develops behaviors in the form of hierarchical finite-state automata (HFA), where each state in an automaton is either an atomic behavior in the agent, or another automaton. Multiple states in the automaton may map to the same atomic behavior or lower-level automaton. As the objective is to enable learning from demonstration with a minimum number of samples, the trainer first defines the behavior by manually decomposing it into a hierarchy of sub-behaviors. For each sub-behavior, he then selects the *features* of the environment and the *states* needed to learn the behavior. HiTAB thus learns only the *transition functions* from each state within the HFA. These simplifications, made possible by decomposition, radically reduce the dimensionality of the problem and enable learning on a much smaller number of samples. Formally, the HFA is a tuple $\langle S, B, F, T \rangle \in \mathcal{H}$:

- $S = \{S_1, \ldots, S_n\}$ is the set of *states* in the automaton. Included is one special state, the *start state* S_0, and zero or more *flag states*. Exactly one state is active at a time, designated S_t.

  The purpose of a flag state is simply to raise a flag in the automaton to indicate that the automaton believes that some condition is now true. Two obvious conditions might be *done* and *failed*, but there could be many more. Flags in an automaton appear as optional features in its *parent* automaton. For example, the *done* flag may be used by the parent to transition away from the current automaton because the automaton believes it has completed its task.

- $B = \{B_1, \ldots, B_k\}$ is the set of *basic behaviors*. Each state is associated with either a basic behavior or *another automaton* from $\mathcal{H}$, though recursion is not permitted.

- $F = \{f_1, \ldots, f_m\}$ is the set of observable *features* in the environment. At any given time each feature has a numerical value. The collective values of F at time t is the environment's *feature vector* $\vec{f_t} = \langle f_1, \ldots, f_m \rangle$.

- $T = \vec{f_t} \times S \to S$ is the *transition function* which maps the current state S_t and the current feature vector $\vec{f_t}$ to a new state S_{t+1}.

- Optional free variables (parameters) $G_1, \ldots, G_n$ for basic behaviors and features generalize the model: each behavior B_i and feature f_i are replaced as $B_i(G_1, \ldots, G_n)$ and $f_i(G_1, \ldots, G_n)$. The evaluation of the transition function

and the execution of behaviors are based on ground instances of the free variables. For example, rather than have a behavior called *go to the ball*, we can create a behavior called *goTo(A)*, where *A* is left unspecified. Similarly, a feature might be defined not as *distance to the ball* but as *distanceTo(B)*. If such a behavior or feature is used in an automaton, either its parameter must be bound to a specific *target* (such as "the ball" or "the nearest obstacle"), or it must be bound to some higher-level parent of the automaton itself. Thus HFAs may themselves be parameterized.

Single-Agent Training. Training is an iterative process between a *training mode* and a *testing mode.* In the training mode, the agent performs exactly those behaviors as directed by the demonstrator. During training, each time the demonstrator chooses a new behavior, the agent records a training example: a tuple $\langle S_t, \vec{f}_t, S_{t+1} \rangle$ which stores the current feature vector, along with the states corresponding to the old and new behaviors. If state S_{t+1} must be executed exactly once, then no additional examples are recorded. Otherwise, a default example is stored: $\langle S_{t+1}, \vec{f}_t, S_{t+1} \rangle$, which tells the agent to continue in the current state if the given feature vector is observed again. The feature vector is specified by the user from a library of predefined but parameterizable features selected for the behavior.

Once enough examples are collected, the demonstrator switches to the *testing mode*, which causes the agent to learn the transition functions within the finite-state automaton. For a given state S_i, HiTAB takes all examples of the form $\langle S_i, f_t, S_j \rangle$ and reduces them to $\langle f_t, S_j \rangle$. HiTAB then applies a classification algorithm to these examples, using the f_t as data and S_j as their labels. At present HiTAB uses decision trees with probabilistic leaf nodes for our classifiers.

After all the transition functions are built, the agent begins performing the learned behavior. If the demonstrator observes the agent performing an incorrect behavior, he may issue corrections, causing the agent to switch back to training mode and collect additional examples, then reenter testing mode with revised training functions. This continues until the demonstrator is satisfied with the behavior, and saves it to the behavior library. At this point, unused states and features are trimmed from the automaton, and any parameterized behaviors and features are bound to a target (e.g., "nearest obstacle"), or to a parameter of the automaton itself. The behavior is now available as a state for training another, higher-level HFA.

Multiagent Training. Once we have trained a library of useful individual behaviors using HiTAB, how might we extend this to training collective multiagent behavior? The obvious (distributed) approach is to simply endow all agents with the same top-level behavior. An alternative centralized approach is to define a single master *controller agent* in charge of all subsidiary agents. The subsidiary agents all have the same behaviors in their libraries; but the controller agent has its own separate library of behaviors, both basic behaviors and learned automata. A controller agents' basic behaviors do not manipulate the controller, but instead correspond to a unique behavior in the libraries of the subsidiaries. When a controller agent transitions to a new basic behavior, this directs the subsidiaries to immediately start performing the corresponding behavior in their libraries.

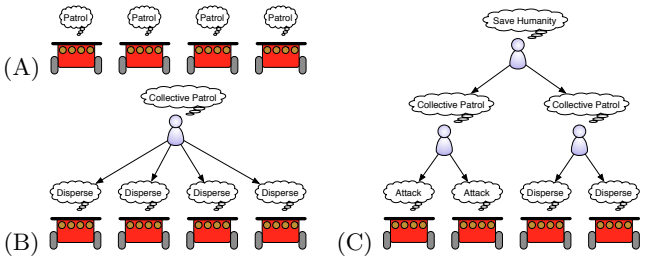

Figure 1: Three notions of homogeneity. (A) Each agent has the same top-level behavior, but acts independently. (B) The top-level behavior all agents is the same, but may all be switched according to a higher-level behavior under the control of a controller agent. (C) Squads in the team are directed by different controller agents, whose behaviors are the same but may all be switched by a higher-level controller agent (and so on).

Our framework is in-between: we define a hierarchy of controller agents. The basic agents are grouped into subgroups, each headed by a level-1 controller agent; then various level-1 controller agents are grouped as subsidiaries to level-2 agents, and so on, up to level-m agents forming one or more roots. Just as all basic agents have the same behaviors, all controller agents at a given level have the same behaviors. The actual structure of the hierarchy (number of levels, number of agents per controller, etc.) is pre-defined by the user. Depending the configuration of the hierarchy, this framework can range from fully distributed to fully centralized, with many points in-between.

The agents are trained starting at the leaf nodes, and working towards the root node one level at a time. After training basic agents in the usual fashion, we may then train a level-1 controller agent, then a level-2 controller agent, and so on. All agents at a given level within the hierarchy run the same HFA, but at any time they may be in different states of that HFA. controller agent training is essentially the same as for basic agents: the user directs the controller agent to perform various behaviors, which in turn cause the controller's subsidiaries to perform behaviors. This adds examples to a database from which transitions are learned.

While the basic behaviors for a controller agent are straightforward, what is a controller agent's set of features? We presume that, unlike a basic agent, a controller agent isn't embodied: his features are derived from statistical results from his subsidiaries: for example "a basic agent in my group is stuck (or isn't)", or "all my immediate subsidiaries are 'done' (or not)", or "the average Y position of basic agents in my group". Typically a controller agent only accesses its immediate subsidiary agents, but there are no restrictions as to how deep in the hierarchy the controller agent can gather information. Like an agent's basic features, the choice of features available to a controller agent are domain-specific.

We ultimately plan to use this method to develop heterogeneous team behaviors: but for now we are concentrating on homogeneous behaviors. We note that this embedding of the HFA training into an agent hierarchy suggests at least three different notions of "homogeneous" behaviors, as shown in Figure 1. First, all agents may simply perform the exact same HFA, but independent of one another. But we can go

Figure 2: Learned multi-robot behavior. Demonstrator is holding a green ("intruder") target.

further than this and still stay within the aegis of homogeneity: we may add a controller agent which controls *which* HFA the agents are performing. It does so by running its own HFA with those subsidiary HFA as basic behaviors. Coordination may continue further up the chain: second- or higher-level controller agents may also dictate their subsidiaries' choice of HFAs.

4. ROBOT DEMONSTRATION

We begin with a simple demonstration which illustrates this approach on actual robots, using a simple hierarchy of four Pioneer robots under the control of a single controller agent. We trained this group to perform a pursuit task while also deferring to and avoiding a "boss". Each robot had a color camera and sonar, and was marked with colored paper. The boss, intruders to pursue, and a home base were also marked with paper of different colors. (See Figure 2).

The task was as follows. Ordinarily all agents would *Disperse* in the environment, wandering randomly while avoiding obstacles (by sonar) and each other (by sonar or camera). Upon detecting an intruder in the environment, the robots would all *Attack* the intruder, servoing towards it in a stateful fashion, until one of them was close enough to "capture" the intruder and the intruder was eliminated. At this point the robots would all go to a home base (essentially *Attack* the base) until they were *all* within a certain distance of the base. Only then would they once again *Disperse*. At any time, if the boss entered the environment, each agent was to *Run Away* from the boss: turn to him, then back away from him slowly, stopping if it encountered an obstacle behind.

This task was designed to test and demonstrate every aspect of the hierarchical learning framework: it required the learning of hierarchies of individual agent behaviors, stateful automata, behaviors and features with targets, both continuous and categorical features, multiple agents, and learned hierarchical behaviors for a controller agent.

Each robot was provided the following simple basic behaviors: to continuously go *Forwards* or *Backwards*, to continuously turn *Left* or *Right*, to *Stop*, and to *Stop* and raise the *Done* flag. Transitions in HFAs within individual agents were solely based on the following simple features: whether the current behavior had raised the *Done* flag; the minimum value of the *Front Left*, *Front Right*, or *Rear* sonars; and the *X Coordinate* or the *Size* of a blob of color in the environment (we provided four colors as targets to these two features, corresponding to *Teammates*, *Intruders*, the *Boss*, and the *Home Base*). Each robot was dressed in the *Teammate* color.

We began by training agents to learn various small parameterized HFAs, as detailed in Figure 3, Subfigures 1 through 7. Note that the *Servo* and *Scatter* HFAs are stateful: when the target disappeared, the robot had to discern which direction it had gone and turn appropriately. Since our system has only one behavior per state, we enabled multiple states with the same behavior by training the trivial HFAs in subfigures 3A through 3D. Training these behaviors required approximately 30 minutes.

We then experimented with the "basic" homogeneous behavior approach as detailed in Figure 1(A): each agent simply performing the same top-level behavior but without any controller agent controlling them. This top-level behavior was *Patrol* (Figure 3, Subfigure 8), and iterated through the three previously described states: dispersing through the environment, attacking intruders, and returning to the home base. We did not bother to add deferral to the "boss" at this point.

Coordinated Homogeneity. Simple homogeneous coordination like this was insufficient. In this simple configuration, when an agent found an intruder, it would attack the intruder until it had "captured" it, then go to the home base, then resume dispersing. But other agents would not join in unless they too had discovered the intruder (and typically they had not). Furthermore, if an agent captured an intruder and removed it from the environment, other agents presently attacking the intruder would not realize it had been captured, and would continue searching for the now missing intruder indefinitely!

These difficulties highlighted the value of one or more controller agents, and so we have also experimented with placing all four robots under the control of a single controller that would choose the top-level behavior each robot would perform at a given time. The controller was trained to follow the *CollectivePatrol* behavior shown in Figure 3, Subfigure 9. This HFA was similar to the *Patrol* behavior, except that robots would attack when *any* robot saw an intruder, would all go to the Home Base when *any* robot had captured the intruder, and would all resume dispersing when *all* of the robots had reached the Home Base. This effectively solved the difficulties described earlier.

We provided the controller with three simple features: whether any robot had seen the Intruder's color; whether any robot was *Done*, and whether all robots were *Done*. We trained a simple hierarchical behavior on the controller agent, called *CollectivePatrolAndDefer* (Subfigure 10). We first added a new statistical feature to the controller agent: whether anyone had seen the Boss color within the last $N-10$ seconds. The controller agent would perform *CollectivePatrol* until someone had seen the Boss within the last 10 seconds, at which point the controller agent would switch to the *RunAway* behavior, causing all the agents to search for the Boss and back away from him. When no agent had seen the Boss for 10 seconds, the controller would resume the *CollectivePatrol* behavior (Figure 2).

Summary. This is a reasonably comprehensive team behavior, with a large non-decomposed finite-state automaton, spanning across four different robots acting in sync. We do not believe that we could train the agents to perform a behavior of this complexity without decomposition, and certainly not in real-time. There are too many states and aliased states, too many features (at least 12), and too many transition conditions. However decomposition is straightfor-

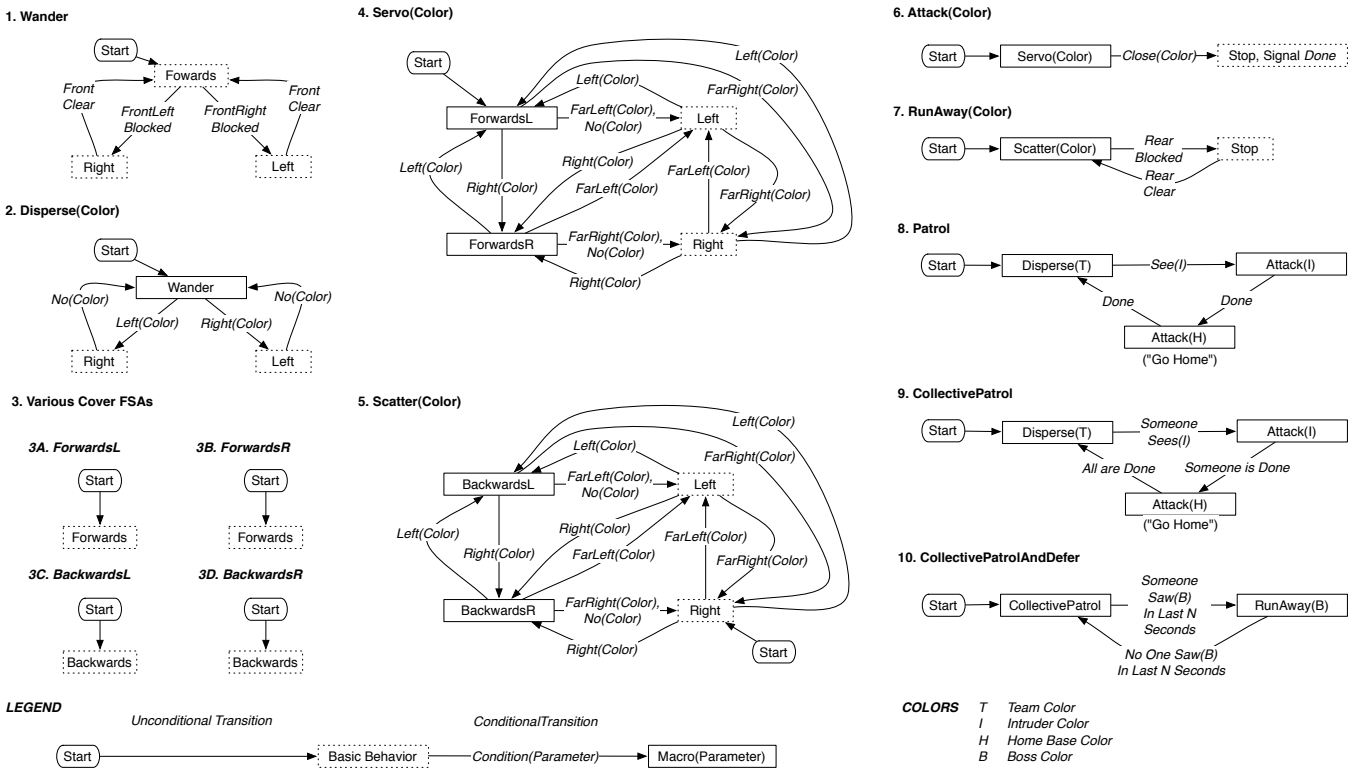

Figure 3: Decomposed hierarchical finite-state automaton learned in the demonstration. See discussion in the text on each subfigure. Most behaviors form a hierarchy within an individual robot, but *CollectivePatrol* and *CollectivePatrolAndDefer* form a separate hierarchy within the team controlling agent. Though the transition condition descriptions here are categorical sounding, most are in fact derived from continuous values: for example, *Left(Color)* is trained based on *X* coordinates of the color blob in the field of view.

ward into simple, easily trained behaviors with small numbers of features and states, simple (indeed often trivial) and easily trained transition functions, and features and states which may vary from behavior to behavior.

5. SIMULATION EXPERIMENTS

After conducting the robot demonstration above, we proceeded to conduct simulation experiments to quantify the benefit of controller agents, particularly as the hierarchy grew from a single controller to multiple levels of controllers. We applied our multiagent homogeneous hierarchies to a simulated box foraging problem: agents hunt for boxes, then pull them to a known deposit location. The boxes are randomly distributed throughout the environment, and after collection at the deposit, the box disappears and a new box is placed randomly in the environment. The environment consists of various circular "boxes" of different sizes, which likewise require different numbers of agents (5, 25, 125) to pull them.

We performed experiments involving *swarms* of independent agents, *groups* of agents under a single layer of controllers (called Level 1 controllers), and groups of agents under multiple layers of controllers (Level 1, Level 2, and so on). To perform these experiments required training three kinds of behaviors. First, we trained behaviors for each basic agent, then we trained behaviors for Level 1 controllers (designed to control basic agents), and finally we trained behaviors for Level N≥2 controllers (designed to control other controllers). This set of behaviors was sufficient to scale to any number of levels. We now describe the basic behaviors and features, and trained decompositions.

Basic Agent Behavior Decomposition. We decomposed and trained a basic agent's behavior hierarchy as follows:

- Agents' basic features were *DistanceTo(X), DirectionTo(X), ICanSeeABox, IAmAttachedToABox,* and *Done*. The first two features were parameterizable to either visible boxes or to the deposit location. The last feature was true when the *done* flag had been raised. Boxes could only be seen if they were within 10 units.
- Agents' basic behaviors were *Forward, RotateLeft, RotateRight, GrabBox, ReleaseBox, ReleaseBoxAndFinish,* and *Done*. Both *ReleaseBox* and *Done* would raise the *done* flag and (as normal) immediately transfer to the start state. *ReleaseBoxAndFinish* would as well, except that it would also raise a *finished* flag in the agent which could be detected by controllers as a feature. Boxes could only be grabbed if they were sufficiently close (5 units).
- Using *Forward, RotateLeft,* and *RotateRight*, we trained *Wander*, which wandered randomly.
- Using *Forward, RotateLeft,* and *RotateRight*, plus the *DistanceTo(X)* and *DirectionTo(X)* features, we trained the behavior *Goto(X)*, which servoed to a given target.

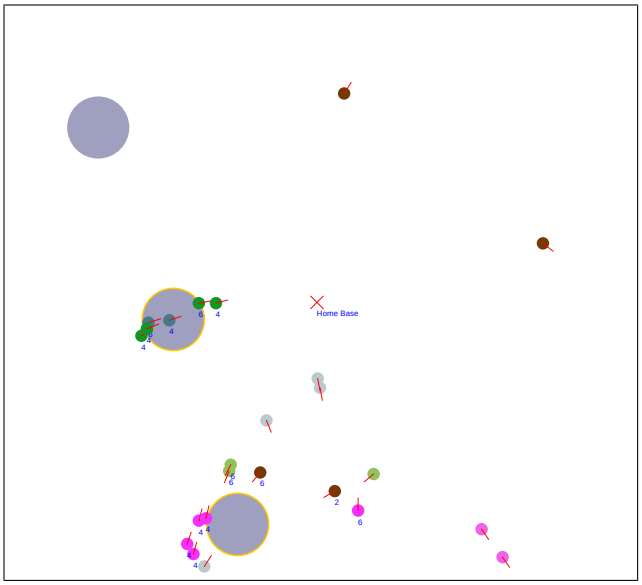

Figure 4: A screenshot of our system in action (showing part of the environment). The large grey circles are the boxes, and the X in the middle is the collection location. Note that while the agents pulling the box on left are all from the same subgroup, the box in the bottom is being pulled by agents from different subgroups.

- Using *Goto(X)*, *GrabBox*, *ReleaseBoxAndFinish*, and *DistanceTo(X)*, we trained *ReturnWithBox*, which pulled the box back to the deposit location and released it when the agent was close enough to home.

- Using *Wander* and *ReturnWithBox*, we trained *Forage*, a simple top-level composition which foraged for boxes and brought them to the deposit.

If agents were acting on their own (they had no controller), their top-level behavior would be simply *Forage*. When acting under a Level 1 controller, the current behavior of the agent would be determined by the controller. Training the agent's behavior required approximately 30 –40 minutes.

Level 1 Controller Agent Behavior Decomposition. A Level 1 controller's behavior hierarchy was as follows:

- A controller's basic features were *SomeoneIsFinished* and *SomeoneIsAttachedToABox*. The latter feature was true if any subsidiary agent had raised its *finished* flag. A controller also had access to an additional target: *closest-attached-agent*, which pointed to the subsidiary agent which had grabbed the box (if any).

- A controller's basic behaviors corresponded to the full set of behaviors of its subsidiary agents: *Forward, RotateLeft, RotateRight, GrabBox, ReleaseBoxAndDone, Done, Wander, Goto(X), ReturnWithBox,* and *Forage.*

- Using *Goto(closest-attached-agent)*, *ReleaseBox*, *ReturnWithBox*, *Forage*, *SomeoneIsAttachedToABox*, and *SomeoneIsFinished*, we trained the behavior *ControlForage*, which directed agents to *Forage* until an agent found a box.

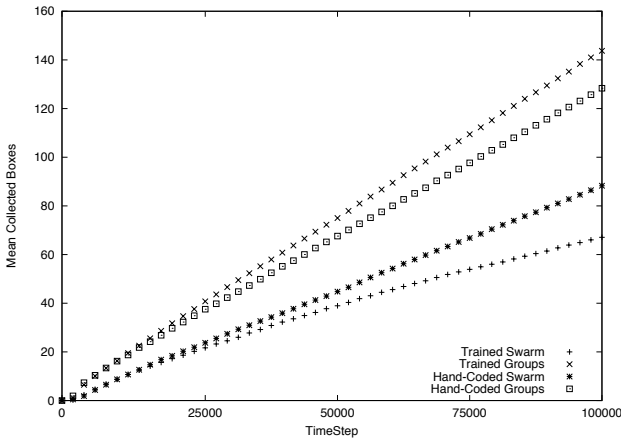

Figure 5: Mean number of boxes collected over time for the first experiment.

Then, the controller would direct agents to *Goto(closest-attached-agent)*; once agents where close to the attached agent, they would grab the box and begin pulling it towards the deposit location. Once one agent finished pulling the box, the controller would direct the agents to *ReleaseBox*, and to resume *Forage*. We also trained trivial *ReturnWithBox* and *Goto(X)* behaviors which simply called their corresponding basic behaviors.

If the agents were acting on their own (they had no Level 2 controller), their top-level behavior would be simply *ControlForage*. When acting under a Level 2 controller, the current behavior of the Level 1 controllers would be determined by their Level 2 controllers. Training Level 1 controller agents required a few minutes.

Level N≥2 Controller Agent Behavior Decomposition. All controller agents at levels ≥ 2 used exactly the same behavior hierarchy, which was:

- A controller's basic features were *SomeoneIsFinished* and *SomeoneNeedsHelp*. The former feature is true if a subsidiary agent knows of a box which requires more agents to push it than are available to the subsidiary agent. A Level N≥2 controller also had an additional target: *biggest-attached-agent*, which is the agent attached to the largest box that the N≥2 controller "knows about". The controller would learn of such boxes from its superior, or from subsidiary controllers unable to manage the box themselves.

- A Level N≥2 controller's basic behaviors corresponded to behaviors from its subsidiary controllers: *ControlForage*, *ReturnWithBox*, *Goto(X)*.

- We trained a version of *ControlForage* similar to the Level 1 *ControlForage* behavior. The difference is that the Level N≥2 behavior directs agents to *Goto(biggest-attached-agent)* when a Level 1 controller requires help. We also trained trivial *ReturnWithBox* and *Goto(X)* behaviors which simply called their corresponding basic behaviors. Training level N $\geq$ 2 controller agents required a few minutes for each level.

A Level N≥2 controller's top behavior was *ControlForage*.

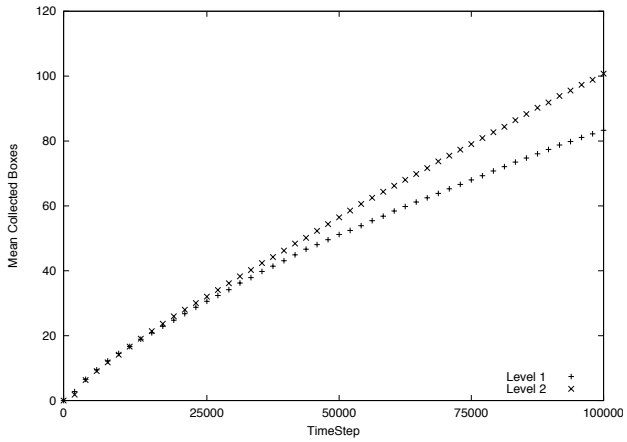

Figure 6: Mean number of boxes collected over time for the second experiment.

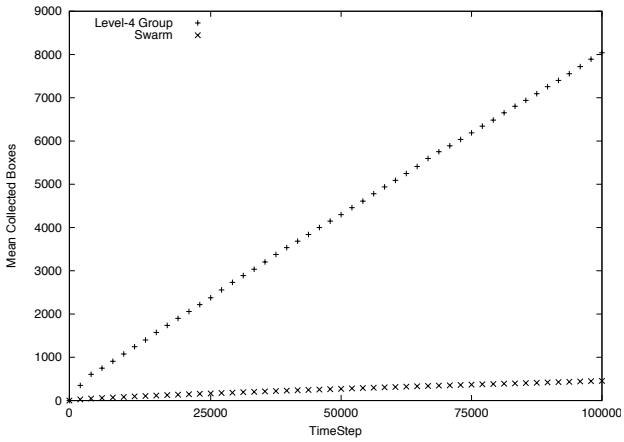

Figure 7: Mean number of boxes collected over time for the third experiment.

5.1 Experiments

For the first two experiments, we considered three hierarchical structures: (1) 50 independent agents (2) ten independent Level 1 controller agents, each heading a five-agent subgroup (3) two independent Level 2 controller agents, each heading five Level 1 controller agents, each heading a five-agent subgroup. These structures roughly correspond to the notions illustrated in Figure 1.

In the first experiment, we sought to demonstrate that a simple hierarchy can out-perform a group of independent agents; and additionally, that the behaviors learned in this experiment would perform adequately compared to fine-tuned hand-coded behaviors. In the second experiment, we sought to demonstrate that a two-layer hierarchy could outperform a one-layer hierarchy. In these experiments, the environment was 200×200 units and agents moved 0.1 units per timestep. Agents started at uniformly randomly distributed locations.

Finally, we tried a scalability experiment, comparing a different, even larger hierarchy against 625 independent agents.

Each experimental run lasted 100,000 timesteps, and each treatment had 100 independent runs. Treatments were gauged based on the mean number of boxes returned. Differences in results were measured at the final timestep with a 95% confidence, using Bonferroni-corrected two-tailed t-tests.

First Experiment: 1-Level Hierarchies. We began by comparing an entirely distributed *swarm* of 50 agents against a *group* of ten controller agents, each in charge of five basic agents. For each of these configurations, we performed runs using a set of trained behaviors and using a set of hand-coded behaviors. Figure 5 shows the results of all four sets of runs.

Independent Agents Versus Hierarchies: We expected the controller agents to outperform a distributed swarm due to the semi-centralized coordination available, because the controller enabled specific groups of agents to work together on a single box. Without controller agents, agents could become stranded at boxes waiting for other agents to help pull. These waiting agents simply relied on random discovery of the box by other agents to gather enough helpers. Figure 5 verifies the expected improvement due to the controllers. The improvement was statistically significant in both cases.

Hand-Coded Versus Trained: We then compared the trained versions of the two previous structures with hand-coded versions of the same. Figure 5 again shows the results. We had expected the hand-coded solutions to perform better, since trained solutions contained significant training error. But in fact, in the Level 1 hierarchy case, the trained solution actually performed statistically significantly better than the hand-coded solution! This was due to a more random exploration strategy which allowed agents to disperse throughout the environment better. This same exploration strategy didn't fare as well in the swarm case, however, because this strategy resulted in too many agents distributed across multiple boxes rather than pulling on the same box. While the results do not present a clear advantage to either training or programming, they do suggest that training the agents will crucially not *significantly* impair performance.

Second Experiment: 2-Level Hierarchies. We then compared the same Level 1 hierarchy as before against a two-level hierarchy: two Level 2 controllers, each in charge of five Level 1 controllers, each in charge of five agents. We changed the scenario to favor two levels of coordination: the environment now had eight boxes which each required five agents to pull, and two boxes which each required twenty-five agents to pull. Just as the first experiment was constructed so as to demonstrate the value of some degree of homogeneous coordination, the second experiment is meant to show the value of homogeneous coordination at two levels.

As shown in Figure 6, two levels significantly outperformed a single level, and for similar reasons as the first experiment. If in the one-level case a group discovered a 25-agent box, 4 other groups had to randomly discover the box before it could be moved and all the groups freed. But with two layers of coordination, we could train agents to work together not only in 5-agent groups but also in 25-agent groups.

Third Experiment: Large Numbers of Agents. Finally, we reran the first experiment using a *four*-level hierarchy: a single Level 4 controller in charge of five Level 3 controllers, each in charge of five Level 2 controllers, each in charge of five Level 1 controllers, each in charge of five agents. This arrangement results in 625 agents and 156 controller agents.

We compared this against a swarm of 625 independent agents. With the larger number of agents, we expanded the environment to 225×225, and provided the environment with 25 size-5 boxes, five size-25 boxes, and one size-125 box.

As would be expected, this was no contest: the swarm of agents were simply outclassed, as shown in Figure 7. This somewhat unfair contest was not intended to show the *efficacy* of the hierarchy, but simply that it this approach is capable of scaling to large numbers of agents and more complex environments.

6. CONCLUSIONS

This paper demonstrates a novel approach to the challenging task of multiagent learning from demonstration. The approach makes progress against the inherent inverse problem by performing macro-behavior decomposition throughout a swarm hierarchy, to the point that the differences between the micro-level and macro-level behaviors are small enough to be surmounted. It also applies behavior decomposition on the individual agent level, enabling a variety of tricks to significantly reduce the complexity and dimensionality of the learning space, so as to get by on a very small number of samples.

Though the obvious target for hierarchies in swarms is heterogeneous behaviors (and that is our first task as future work), the demonstrations here show that hierarchies may still be of benefit in the homogeneous behavior case. In this case, such hierarchies provide an alternate method of consistent, learnable coordination among agents.

We note that, from a machine learning perspective, the *individual* learned behaviors shown here are often quite simple. But this is exactly the point. Our goal is to enable rapid agent and robot behavior development. From this perspective, decomposition of a very complex joint model into many simple models promises to allow even novices to build multiagent behaviors rapidly because the number of samples need not be large.

7. REFERENCES

[1] D. C. Bentivegna, C. G. Atkeson, and G. Cheng. Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47(2-3):163–169, 2004.

[2] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal Of Robotics And Automation*, RA-2:14–23, April 1986.

[3] S. Chernova. *Confidence-based Robot Policy Learning from Demonstration*. PhD thesis, Carnegie Mellon University, 2009.

[4] J. Dinerstein, P. K. Egbert, and D. Ventura. Learning policies for embodied virtual agents through demonstration. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1257–1252, 2007.

[5] E. H. Durfee and T. A. Montgomery. A hierarchical protocol for coordinating multiagent behaviors. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 86–93, Boston, MA, USA, 1990. AAAI Press.

[6] D. Goldberg and M. J. Mataric. Design and evaluation of robust behavior-based controllers. In T. Balch and L. E. Parker, editors, *Robot Teams: From Diversity to Polymorphism*, pages 315–344. A. K. Peters, 2002.

[7] D. Goldberg and M. J. Mataric. Maximizing reward in a non-stationary mobile robot environment. *Autonomous Agents and Multi-Agent Systems*, 6:2003, 2002.

[8] R. Grabowski, L. E. Navarro-Serment, C. J. J. Paredis, and P. K. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, July 2000.

[9] G. E. Hovland, P. Sikka, and B. J. McCarragher. Skill acquisition from human demonstration using a hidden markov model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2706–2711, 1996.

[10] M. Kasper, G. Fricke, K. Steuernagel, and E. von Puttkamer. A behavior-based mobile robot architecture for learning from demonstration. *Robotics and Autonomous Systems*, 34(2-3):153–164, 2001.

[11] M. F. Martins and Y. Demiris. Learning multirobot joint action plans from simultaneous task execution demonstrations. In *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, pages 931–938, 2010.

[12] J. McLurkin and D. Yamins. Dynamic task assignment in robot swarms. In *Robotics: Science and Systems Conference*, 2005.

[13] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2-3):79–91, 2004.

[14] M. N. Nicolescu and M. J. Mataric. A hierarchical architecture for behavior-based robots. In *The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 227–233. ACM, 2002.

[15] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[16] L. Parker. ALLIANCE: An architecture for fault tolerance multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2), 1998.

[17] P. Stone and M. Veloso. Layered learning and flexible teamwork in robocup simulation agents. In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, volume 1856 of *Lecture Notes in Computer Science*, pages 65–72. Springer Berlin / Heidelberg, 2000.

[18] P. Stone and M. M. Veloso. Layered learning. In R. L. de Mántaras and E. Plaza, editors, *11th European Conference on Machine Learning (ECML)*, pages 369–381. Springer, 2000.

[19] B. Takács and Y. Demiris. Balancing spectral clustering for segmenting spatio-temporal observations of multi-agent systems. In *IEEE International Conference on Data Mining (ICDM)*, pages 580–587, 2008.

[20] H. Veeraraghavan and M. M. Veloso. Learning task specific plans through sound and visually interpretable demonstrations. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2599–2604. IEEE, 2008.

[21] T. Vu, J. Go, G. Kaminka, M. Veloso, and B. Browning. MONAD: a flexible architecture for multi-agent control. In *AAMAS 2003*, pages 449–456, 2003.