Scalable Heterogeneous Multiagent Learning from Demonstration

William Squires and Sean Luke

Department of Computer Science
George Mason University
4400 University Dr
Fairfax, Virginia 22030
wsquires@gmu.edu, sean@gmu.edu

Abstract. We present a method of supervised learning from demonstration for real-time, online training of complex heterogenous multiagent behaviors which scale to large numbers of agents in operation. Our learning method is applicable in domains where coordinated behaviors must be created quickly in unexplored environments. Examples of such problem domains includes disaster relief, search and rescue, and gaming environments. We demonstrate this training method in an adversarial mining scenario which coordinates four types of individual agents to perform six distinct roles in a mining task.

Keywords: Learning from Demonstration; Multiagent Learning; Heterogeneous

1 Introduction

In this paper we introduce an approach to scalable online learning from demonstration of nontrivial, heterogeneous, stateful multiagent and swarm behaviors. The agents in the swarms can coordinate at various levels, and the system can perform scalable load-balancing within the swarm to assign subswarms of heterogeneous agents appropriate for various tasks as called on by the experimenter. This work extends the learning from demonstration method HiTAB, adding an approach to (for the first time) train arbitrarily large, scalable agent swarm hierarchies with agent heterogeneity at every level.

There are many scenarios where training agent behaviors in real-time is highly advantageous. Consider scenarios such as disaster recovery, search, and rescue, which presents considerable challenges: programming and debugging custom behaviors on-the-fly may not be reasonable to do in real-time; and this assumes that the robot handler in the field has sufficient coding skills to achieve this at all. Furthermore, solutions to complex scenarios such as these involve a wide-ranging set of agent capabilities unlikely to be found in a single type of agent and therefore require the coordination of heterogeneous agents to be successful, and this in turn presents a large and complex design space. Finally, these scenarios can span large areas, which may necessitate the need for many heterogeneous teams to work cooperatively.

In scenarios such as these it would be desirable to *instruct* (or if you like, *coach*) the robots how to do various collective behaviors rather than program them. But learning from demonstration in swarms has until recently proven very difficult for two reasons.

First, there is the obvious problem of the per-agent *Curse of Dimensionality*: complex and stateful behaviors imply a high-dimensional learning space and thus a large number of training samples, but this cannot be achieved if we wish training to be online. Less obvious but just as problematic is the *Multiagent Inverse Problem*: even if we could formally quantify the emergent macrophenomenon we wished the agents to achieve, in order to learn, the agents require feedback regarding their individual behaviors. While we potentially have a function which tells us the macrophenomena arising from individual agents — that is, a simulator — we lack the *inverse function* which tells us the individual behaviors needed to achieve a given macrophenomenon. Inverse problems like these are classically solved with optimization (reinforcement learning, evolutionary algorithms), and generally must be offline and/or in simulation given the samples involved; but our scenario requires online learning.

HiTAB overcomes these two problems by manually breaking the task into a subtask hierarchy to be trained bottom-up. Agent behaviors are decomposed into behavior hierarchies, and swarms themselves are decomposed to small and manageable groups. This decomposition allows us to project a complex high-dimensional problem space into many spaces of much lower dimensionality; and it also allows us to overcome the inverse problem by reducing the joint task so much that it becomes obvious what micro-behaviors are need to achieve a given (much simpler and more immediate) macrophenomenon.

HiTAB has been applied to homogeneous agent behaviors with arbitrary numbers of agents [15, 16, 6]; and to heterogeneous agents in small groups or with low heterogeneity [17, 14]. However, it has not to date been applied to training complex heterogeneous multiagent behaviors at multiple levels, scaling to large groups, and coordinating agents with distinctly different capabilities and functions. This is because such a problem would demand nontrivial task allocation and heterogeneous swarm organization.

In this work, we consider an extension to HiTAB which makes it possible to train scalable, nontrivial heterogeneous swarms for the first time. Our approach augments HiTAB with, among other things, automated swarm reorganization and task allocation weighted by which heterogeneous capabilities are needed. We begin by discussing related work, and then review the HiTAB training methods introduced in prior work, followed by further extensions introduced in this work. We then introduce a virtual controller hierarchy scaling algorithm which scales the hierarchy at runtime by creating additional controllers based on the number of individual agents, and forming new sub-teams to perform the learned controller behaviors in parallel. Next, we describe a mining scenario created to challenge and highlight the ability of our training method, and then describe how training was performed for the mining scenario. We then describe the experiments performed and discuss the results of those experiments and future work.

2 Related Work

Multiagent Learning from Demonstration The literature for multiagent learning from demonstration (LfD) outside of HiTAB is quite sparse, and primarily aims to address the complexity of training multiple agents simultaneously. Confidence based LfD was introduced in [4], where robots were trained to cooperatively sort colored balls into the appropriate bin. When a robot was uncertain of the correct action it would request additional demonstration from the trainer. Other work involves learning from the joint

demonstration by multiple trainers. In [10], an approach was developed where the individual sequence of actions for each robot is captured and then the sequence of group behaviors is determined through analysis of the individual action sequences over space and time. In [2], robots learn to collaboratively open a door by extracting a template for the behavior and adapting it to doors in other settings. These methods work well for small teams but become dramatically more complex as more robots are added.

Multiagent LfD approaches may be combined with other learning techniques. In [3], LfD is used as a shaping mechanism for the reward function for reinforcement learning while [8] explores a hybrid approach mixing LfD with more traditional machine learning.

Hierarchies and Agents Hierarchies are a well-known organizational approach for multiagent systems where agents higher in the tree collect increasingly global information and provide instruction to their subordinates [7]. Communication is generally restricted such that subordinate agents communicate information up to parent agents and do not communicate directly with peers, which reduces communication-related scaling problems while also reducing the state information each agent must track.

However, hierarchies can fail due to the loss of a single agent, resulting in multiple single points of failure. These drawbacks are more pronounced when considering hierarchies of real robots [11]. Because of this, most multi-robot coordination algorithms favor decentralized approaches. However, hierarchies can provide a good middle ground between centralized and decentralized coordination [13], and when considering heterogeneous swarms coordination is difficult to achieve without hierarchy [1].

In [5] and [12], heterogeneous swarms used a hierarchical structure to coordinate behaviors between an aerial agent and homogeneous subswarms, extending their capability by providing global information to direct their behavior. The subswarms were able to scale with communication confined to be between subswarm agents and the aerial agent. Multiagent HiTAB operates similarly with regard to the subswarms and communication, but is more flexible in that each subswarm may be a different type of agent.

2.1 HiTAB

Individual Agent Training Below we provide a brief overview of prior work in HiTAB LfD training methods as an evolution from individual agent, to homogeneous multiagent, and finally heterogeneous multiagent training. To train individual agents [9], the trainer decomposes a behavior into a hierarchical finite-state automata (HFA). The HFA states are agent behaviors, with the lowest level containing only atomic agent behaviors. The HFA transitions represent changes in *features* in the environment. Behaviors and features can be parameterized, for example *GoTo(X)* and *DistanceTo(X)* respectively.

The HFA is trained from the bottom up. At each level the trainer selects features needed for training and binds parameters to *targets* in the environment as needed, for example *GoTo(ClosestAgent)* and *DistanceTo(ClosestObstacle)*. The trainer teleoperates the agent; changing the behavior at the appropriate time, each change generating a training sample including the previous behavior and current feature values. When training is complete the trainer observes the behavior and provides corrective training if needed. Once the behavior works as expected, the behavior is saved for training higher-level FA.

Homogeneous Multiagent Training To train homogeneous multiagent behaviors [15], first individual agent behaviors are trained, and then a virtual *controller agent* (or boss) is trained to direct agents to perform their trained behaviors as its atomic behaviors. A hierarchy of controller agent behaviors are themselves trained as compositions of these atomic behaviors. When a controller changes to a new atomic behavior, all subordinate agents in its subswarm switch to the corresponding trained behavior.

Heterogeneous Multiagent Training To train heterogeneous multiagent behaviors [17], virtual controller agents have multiple types of homogeneous agent groups. An atomic behavior in a controller agent corresponds not to a trained behavior learned by its underlings, but to a *set* of top-level behaviors meant to work together. For example, in a group of two agents of type A and one agent of type B, a coordinated behavior C might consist of behavior A_i for each of the A agents and behavior B_j for the B agents. C would correspond to an atomic behavior in the controller agent. When a controller changes to a new atomic behavior, all subordinate agents switch to the appropriate behaviors in the corresponding coordinated behavior set.

The *features* used by an individual agent to determine transitions in its HFA are normally hard-coded sensor capabilities: but what would be the features used by controller agents? [14] introduced *group features* to allow a trainer to quickly define controller features, without programming, by identifying a feature from a subordinate agent group and an aggregator function; for example *Min(At(TheStore))*. Since group features are the features of a controller agent, a higher-level controller can use that feature in a *group feature* of its own, allowing feature information to be passed up the hierarchy.

3 Our Extensions to HiTAB Training

We introduce two extensions to enable training of complex heterogeneous behaviors:

Composite Features Group features aggregate a feature from one subordinate group or one that is common to all groups, but there are cases where training requires the combination of distinct features from multiple groups. *Composite features* allow a trainer to, without programming, define a feature that incorporates feature information from multiple individual agent types. We do this by providing a feature name, a list of *group features*, and an aggregation function that combines the group features into a single value. For example, a controller coordinating driverless shuttles (group 0) and passengers (group 1) needs a feature indicating to go to the parking lot if there are at least 10 passengers or there is more than one shuttle at the station; *ReadyToShuttle* is defined as *Min(TenOrMorePassengers,MultipleAtStation)*. Like group features, composite features can be passed up the hierarchy as input to a group feature defined in the parent controller.

Targets with Hierarchy Context A target in HiTAB is a variable which generalizes a behavior, so we can create general behaviors like "Go to *X*" rather than concrete behaviors like "Go to the ball" or "Go to George". We define a method for individual agent training where the trainer can quickly define a set of agents within the hierarchy to use as targets. For example, in Figure 1, training of agents under the *Support* controller may need to reference some agent in the *Infantry* group of the *Defense* controller as a target. Providing a *level* and a *hierarchy subpath*, the trainer can reference any set of agents in

Algorithm 1 Scale Hierarchy by Least Constrained Agent

```
 1: procedure ScaleHierarchy(rootgroup, agentCnts[])
 2:     topCtlr← CreateTopController(rootgroup)
 3:     ComputeRequirements(topCtlr)
 4:     if ¬ MeetsCtrlRqts(topCtlr, agentCnts, MIN) then Abort( )
 5:     AllocToController(topCtlr, agentCnts, MIN)
 6:     AllocToController(topCtlr, agentCnts, PREF)
 7:     AllocToController(topCtlr, agentCnts, MAX)
 8: procedure AllocToController(ctl, agentCnts[], mode)
 9:     for all grp in ctl do
10:         for all agentType in agentCnts do
11:             computedCnt← ComputedGroupRqt(grp, agentType, mode)
12:             myCnts[agentType]← Min(computedCnt, agentCnts[agentType])
13:         AllocToGroup(grp, myCnts, mode)
14:         agentCnts← AdjustCounts(agentCnts, myCnts)
15: procedure AllocToGroup(grp, agentCnts[], mode)
16:     if grp is a individualAgentType then agentCnt← agentCnts[individualAgentType]
17:     else
18:         newControllerCnt← 0
19:         for all agentType1 in agentCnts do
20:             cnt← agentCnts[agentType1]
21:             ctlCnt← cnt/ComputedControllerRqt(ctl, agentType1, mode)
22:             for all agentType2 in agentCnts do
23:                 cnt← agentCnts[agentType2]
24:                 ctlCntMin← cnt/ComputedControllerRqt(ctl, agentType2, MIN)
25:                 ctlCnt← Min(ctlCnt, ctlCntMin)
26:             newControllerCnt← Max(ctlCnt, tgtCnt)
27:         while numControllers<newControllerCnt do CreateController(ctlType)
28:         for all ctl in group do
29:             myCnts← ControllerCounts(ctl, agentCnts, mode)
30:             AllocToController(ctl, myCnts, mode)
31:             agentCnts← AdjustCounts(agentCnts, myCnts)
32: procedure AdjustCnts(agentCnts[], allocatedCnts[])
33:     for all type in agentCnts do agentCnts[type]←agentCnts[type]−allocatedCnts[type]
34: procedure ControllerCounts(ctl, agentCnts[], mode)
35:     for all type in agentCnts do
36:         tgtCnt← ComputedControllerRqt(ctl, type, mode)
37:         myCnts[type]← Min(tgtCnt, agentCnts[type])
```

the hierarchy. The level defines the starting point of the hierarchy subpath, where level 0 is the immediate parent controller and increases as we move up the hierarchy. The hierarchy subpath is a string of the form *subpath*=*group*[*agentNum*⟨*:subpath*⟩],... defining a list of agent groups under the controller, each having an agent number and an optional nested subpath. Replacing group or agentNum with a wildcard character indicates all groups or all agents respectively. In this way, an agent below the *Support* controller would reference its formation leader with *level*=1 and *subpath*=1[*], corresponding to any *Infantry* agent below the *Defense* controller.

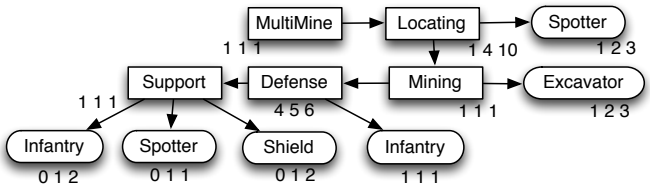

Fig. 1. Mining Behavior Hierarchy. Rectangles are controller groups and rounded rectangles are individual agent groups. Shown with min/preferred/max constraints.

4 Hierarchy Scaling

In this work we train complex behaviors with a relatively small number of agents and then have those behaviors scale, without retraining, to solve large problems given a larger number of individual agents. To do this, we consider the structure of the heterogeneous team, where the size and number of subteams can be increased to improve redundancy and parallelism. The trainer's role in this is to consider an appropriate set of constraints for each subteam by defining its *minimum, maximum,* and *preferred* sizes in operation. In HiTAB, subteams are represented by the agent groups of a virtual controller agent, each including a homogeneous set of individual or virtual controller agents. Given the constraints for each agent group and some allocation of individual agents, we need to adjust agent group sizes to scale the behavior. Scaling a single-level controller is trivial, we simply add available agents to the individual agent groups of the controller (as in [14]). But in a complex agent hierarchy, scaling also involves increasing the size of virtual controller agent groups.

Adding virtual controllers is a difficult problem for the following reasons: First, the number of agents in each agent group must satisfy the *Minimum* and *Maximum* constraints defined by the trainer. We can only add a virtual controller if there are enough individual agents to meet its combined *Minimum* constraints and we can only add individual agents to an existing controller up to the combined *Maximum* constraints. Second, agent types may appear in multiple agent groups of the hierarchy, each having its own constraints for that role in the top-level behavior. And third, we do not know in advance if some type of agent will be much more constrained than others. Below we define a low-cost centralized algorithm to scale an agent hierarchy given the trainer-defined constraints and some allocation of individual agents.

Scale Hierarchy by Least Constrained Agent Shown in Algorithm 1, the *ScaleHierarchy* procedure takes a root group of controller agents that run the top-level behavior and an array of individual agent counts. It begins by creating a *dummy* controller containing the root agent group and then recursively computes the individual agent requirements for the three constraint modes (*Minimum*, *Preferred*, and *Maximum*) for each group and controller in the hierarchy. Then we call *AllocToController* for each constraint mode.

In *AllocToController*, for each agent group we determine how many of each type of agent can be provided given a constraint *mode*. The agent count of each individual agent type is up to the computed group requirement for that type and constraint mode. After allocation to a group is complete, we adjust the individual agent counts to reflect what was allocated and repeat for the next agent group.

The *AllocToGroup* procedure acts in one of two ways. For an individual agent group, the group size is simply increased to the provided agent count. For a controller agent group, we determine how many additional controllers can be added given the available agents and the constraint mode as follows: For each individual agent type, *type1*, we calculate how many controller agents can be created based on the number of *type1* agents and the computed requirement of a controller for that agent type. Within that loop, we check the remaining individual agent types, *type2*, to see how many controllers can be created for that agent type using the *Minimum* constraint. That is, we create additional controllers based on the least constrained agent type so long as the *Minimum* requirements are met for the controller's other individual agent types. If needed, the controller count is reduced so the *Minimum* for the *type2* individual agents is not violated. After calculating the number of additional controllers, we add them to the agent group and then *AllocToController* is called for each controller in the group. Adding the new controllers at the head of the agent group ensures their *Minimum* requirements are met.

5 The Mining Scenario

Demonstrating our learning approach requires a training scenario that is heterogeneous, demands scalability to cover large areas, and requires complex interaction among heterogenous agents. We were unable to find a suitable match in our literature search, so we invented an adversarial mining scenario. In this scenario, *Excavator* agents mine ore from deposits that are located by *Spotter* agents. There are also a number of defensive agent types that cooperate to protect the *Excavator* from adversary agents that seek to stop the mining.

The agent types below are defined by setting attribute values for maximum health and abilities for vision, attacking other agents, shielding attacks from other agents, and what action to take when the agent reaches zero health:

Infantry This agent can attack other agents but cannot shield attacks.

Spotter This agent has zero-value attack and shield values, but has increased visual range and a 360-degree field of view. If an *Infantry* agent is associated with a *Spotter*, then it can attack a *spotted* target with 100 percent accuracy.

Shield This agent can shield other agents from attack, but cannot attack other agents.

Excavator This agent has a greater amount of health, no defensive abilities, and is able mine ore from the ore deposits.

Archer This agent has similar settings to the Infantry agent, but is an adversary to the agents above. When health reaches zero, these agents reappear at spawn points randomly distributed in the environment.

Mining is disrupted when the *Excavator* agent reaches zero health, and resumes when its health is restored (after 1000 time units). When other mining team agents reach zero health, they are disabled for 300 time units, but stay in place to maintain the agent allocations to the scaled hierarchy. *Shield* and *Spotter* agents are not *attackable*, but may be damaged while other agents are being attacked.

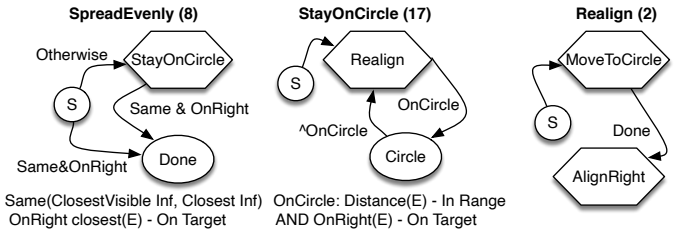

Fig. 2. SpreadEvenly HFA. The samples required to train a behavior is shown in parentheses. Hexagons are trained behaviors; ovals are atomic behaviors.

6 Training the Mining Scenario

To train the mining scenario, we first have to define an agent hierarchy that simplifies training and defines the appropriate subteams. After that we train individual behaviors and then the coordinating behaviors of the virtual controllers.

Defining the agent hierarchy Given the agent types defined above we chose to train the mining scenario behavior using agent hierarchy shown in Figure 1. At the lowest level, the *Support* controller includes the supporting agents of a defense team. To provide a reference point for the training of the defensive formation behavior, we add the *Defense* controller where a single *Infantry* agent acts as a *formation leader* for the *Support* agents in the defense team. The *Mining* controller has *Excavator* agents that mine ore and a number of *Defense* teams to protect them from attack. The *Locating* controller has *Spotter* agents that search for available ore deposits for a *Mining* team. At the top level, the *MultiMine* controller is a homogeneous controller of *Locating* teams that perform the mining behavior in parallel. Note that two agent types have multiple roles in the hierarchy, where the agents are trained for both roles and perform the appropriate role based on their placement in the hierarchy during hierarchy scaling.

When defining the hierarchy, we also defined the constraints for *Minimum*, *Preferred*, and *Maximum* agent counts for each agent group. These are shown in Figure 1 as three integer values next to each agent group. For example, each *Mining* agent has a minimum of four *Defense* agents, with five preferred and 6 at a maximum.

Agent Training The top-level mining behavior combines *over 150 trained behaviors* for the individual and controller agents, so we will not describe all training here. We provide examples below that are part of the behavior to form agents defensively around *Excavator* agents, with final formation examples shown in Figure 4. The figure also shows that the formation is scaled based on agent availability.

Much of the training fell into two categories: training individual agents to position themselves with respect to other agents in the hierarchy and training controller agents to coordinate individual agent behaviors. We describe one instance of each type of training below, highlighting usage of our extensions in this work. Note that all references to *Infantry* agents below are for *formation leader* agents in the second group of the *Defense* controller. Finally, we briefly describe the adversary (*Archer*) behavior.

SpreadEvenly *Infantry* agents are trained to spread evenly in a circle around *Excavator* agents below the *Mining* controller. When this behavior begins, the agents are already

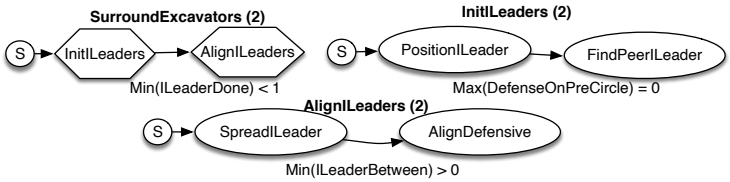

Fig. 3. SurroundExcavators HFA. See also Figure 2.

positioned on a circle around *Excavator* agents and a peer *Infantry* agent is visible. The HFA for the *SpreadEvenly* behavior is shown in Figure 2.

Realign positions the agent at the desired radius to the closest *Excavator* and orients it so that the *Excavator* is directly to the right. *StayOnCircle* circles the closest *Excavator* and realigns if the distance or orientation are not within acceptable limits, which can happen if there are multiple *Excavator* agents to circle. Finally, *SpreadEvenly* moves the agent around the circle until the closest visible peer is also the closest peer. Without further coordination, the agents continue to circle at a relatively even distance because they cannot all be at the stopping condition at the same time.

To train *SpreadEvenly*, we used two different *targets with hierarchy context*. The *Realign* and *StayOnCircle* behaviors are trained with respect to an *Excavator* using *level=1* and *subpath*=1[*]. The *SpreadEvenly* behavior is trained with respect to peer *Infantry* agents using *level*=0 and *subpath*=0[*:1[*]].

SurroundExcavators The *Mining* controller agent is trained to coordinate *Infantry* agents to position evenly on a circle around its *Excavator* agents. The HFA is shown in Figure 3, where the ellipses are *joint behaviors* and the features are *group features*.

InitILeaders instructs *Infantry* agents to move to a radius around the closest *Excavator* agent using *PostionILeader* and, when they are all at the proper distance, they are instructed to move around the circle until a peer is visible using *FindPeerILeader*. *AlignILeaders* runs the *SpreadEvenly* behavior described above using *SpreadILeader* and, when all agents are roughly equidistant to their two closest peers, they are instructed to orient outward from the *Excavator* using *AlignDefensive*. The top-level behavior switches from *InitILeaders* to *AlignILeaders* when all of the agents are in a *Done* state.

While no *composite features* were used in training *SurroundExcavators*, two were used in other training to indicate when the defensive formation is complete and mining can start. First, the *DefenseAligned* composite feature in the *Support* controller is the Max of three group features indicating all support agents are on their respective circles around the *Excavator*. This feature is passed up through the *Defense* and *Mining* controllers as the *SupAligned* group feature. The *DefenseAligned* composite feature of the *Mining* controller is the Max of *SupAligned* and the *Max(AlignedRear)* group feature indicating all formation leaders are pointing away from the *Excavators*.

Archer Behavior The adversary agents are trained with the *AttackMined* behavior, which moves the agent randomly in the environment until it is within 150 units of an ore deposit that is being mined. It then moves toward the closest ore deposit being mined and attacks the *Infantry* and *Excavator* agents near the ore.

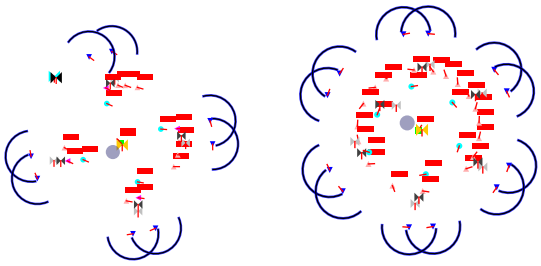

Fig. 4. Mining Formation for 4 and 6 defensive teams

7 Experiments

In our experiments we run the top-level mining behavior in environments of two sizes, 580x580 and 1080x1080 units, with 12 ore deposits. Deposits are randomly distributed in grid locations in the environment and reappear in an open location randomly when depleted. *Archer* agents are randomly distributed in the environment, with 160 in the smaller environment and 555 in the larger environment to keep the density equal. When an *Archer* dies it reappears at one of 18 spawn locations, which are randomly distributed in grid locations at start time. Experiments are run with three allocations of mining agents that result in one, two, and four mining teams respectively after hierarchy scaling.

Baseline Training We establish a baseline of comparison by training individual agent behaviors for the mining task without a controller hierarchy. For this training we defined an additional agent type, *Seeker*, to perform the *ore seeking* behaviors of the *Spotter* agent. The individual agent behaviors were trained as follows: (1) A *Seeker* agent moves randomly, goes to and attaches to a visible ore deposit with no attached agents, and then repeats the behavior when mining begins on the deposit. (2) An *Excavator* agent moves randomly until it is within 150 units of an attached *Seeker* agent, then goes to, attaches to, and mines the ore deposit, and then repeats the behavior when the deposit is depleted. (3) An *Infantry* agent moves randomly in the environment until within 150 units of active mining, then goes to the mining location and attacks *Archer* agents, and repeats the behavior when the deposit is depleted. (4) A *Spotter* agent moves randomly until it sees an *Infantry* agent, then moves to and follows the *Infantry* agent while keeping it between the nearest *Archer* agent and itself. (5) A *Shield* agent moves randomly until it sees an *Infantry* agent, then moves to and follows the *Infantry* agent while trying to shield it from the nearest *Archer* agent.

The agent allocations for experiments are shown in Table 1, where *h* is the hierarchical behavior and *b* is the baseline behavior. At the beginning of each simulation mining teams are randomly distributed, with agents in a 50x50 unit square. Though the baseline agents are not true *teams*, we group them similarly to create a fair comparison at startup. Each experiment was run for 100,000 timesteps. For each environment, we compared

Table 1. Agent Allocations

Allocation	*Infantry(h,b)*	*Shield(h,b)*	*Excav(h,b)*	*Spotter(h)*	*Seeker(b)*	*Spotter(b)*
One Team	8	5	1	7	5	2
Two Teams	16	10	2	14	10	4
Four Teams	32	20	4	28	20	8

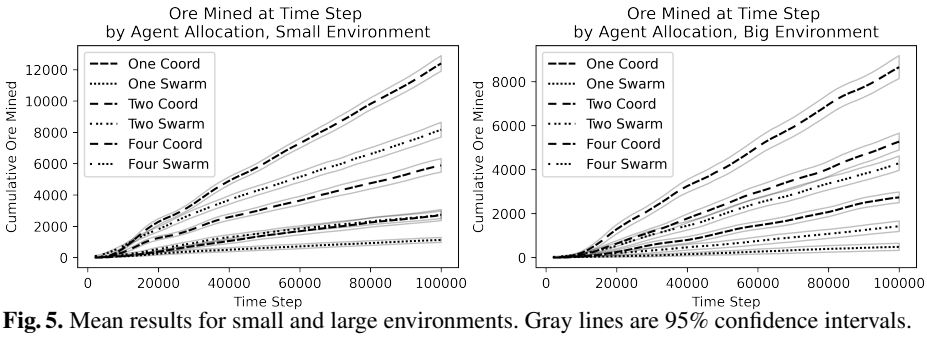

Fig. 5. Mean results for small and large environments. Gray lines are 95% confidence intervals.

the average ore gathered at step 100,000 over 50 runs for the team configurations in the two environments using two-tailed t-tests, p-value=0.01, with Bonferroni correction.

Figure 5 shows the mean results with the baseline results having dots indicating the number of teams and the controller hierarchy having dashes indicating the number of teams. In both environments the scaling resulted in a clear but slightly sublinear improvement in results for additional mining teams. The virtual controller hierarchy results in a clear improvement over the baseline behavior. The difference is even more pronounced in the larger environment, with 2 hierarchy teams outperforming 4 baseline teams, suggesting that the added communication and coordination provided by the virtual controller agent hierarchy becomes more important as the environment grows.

In comparing the relative efficiency per team, the increase to four teams shows a small drop in ore mined per team. We attribute this to the increased competition for ore resources. This difference is more pronounced in the big environment, where the cost of teams exploring the same area is higher. Training a top-level behavior to encourage teams to explore other areas of the environment may help to address the decreased efficiency.

8 Conclusions and Future Work

In this work, we successfully trained a complex heterogeneous multiagent behavior and showed that it scales without retraining. We also showed that the virtual controller hierarchy offers a clear benefit over agents attempting to coordinate on their own. We introduced the mining scenario as a means of demonstrating complex and scalable heterogeneous agent training. We also made further refinements to the HiTAB heterogeneous multiagent learning method that were an important part of the mining behavior training.

The *Scale Hierarchy by Least Constrained Agent* algorithm successfully incorporated additional agents, but in empirical testing we noted certain cases where the distribution of available agents was not always balanced. Better hierarchy scaling algorithms could be created to address this, but without applying our method to a greater number of scenarios it is hard to tell if one algorithm would be suitable for all problems.

Finally, the current hierarchy scaling is based on central knowledge of the individual agents and where they are in the environment. While this knowledge is available for a simulation environment, we cannot assume this information is available for robots in a large environment where they are likely to be initially separated from their team.

Future work will center around the creation of a distributed hierarchy scaling algorithm and other methods that will allow the agent hierarchy to be built, scaled, and balanced as agents find each other in the environment.

References

1. Barca, J.C., Sekercioglu, Y.A.: Swarm robotics reviewed. Robotica **31**(3), 345–359 (2013)
2. Blokzijl-Zanker, M., Demiris, Y.: Multi robot learning by demonstration. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3. pp. 1207–1208. International Foundation for Autonomous Agents and Multiagent Systems (2012)
3. Brys, T., Harutyunyan, A., Suay, H.B., Chernova, S., Taylor, M.E., Nowé, A.: Reinforcement learning from demonstration through shaping. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
4. Chernova, S., Veloso, M.: Confidence-based multi-robot learning from demonstration. International Journal of Social Robotics **2**(2), 195–215 (2010)
5. Elston, J., Frew, E.W.: Hierarchical distributed control for search and tracking by heterogeneous aerial robot networks. In: Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on. pp. 170–175. IEEE (2008)
6. Freelan, D., Wicke, D., Sullivan, K., Luke, S.: Towards rapid multi-robot learning from demonstration at the robocup competition. In: Robot Soccer World Cup. pp. 369–382. Springer (2014)
7. Horling, B., Lesser, V.: A survey of multi-agent organizational paradigms. The Knowledge engineering review **19**(4), 281–316 (2004)
8. Le, H.M., Yue, Y., Carr, P., Lucey, P.: Coordinated multi-agent imitation learning. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 1995–2003. JMLR. org (2017)
9. Luke, S., Ziparo, V.A.: Learn to behave! rapid training of behavior automata. In: Proceedings of Adaptive and Learning Agents Workshop at AAMAS 2010 (2010)
10. Martins, M.F., Demiris, Y.: Learning multirobot joint action plans from simultaneous task execution demonstrations. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. pp. 931–938. International Foundation for Autonomous Agents and Multiagent Systems (2010)
11. Parker, L.E.: Multiple mobile robot systems. Springer Handbook of Robotics pp. 921–941 (2008)
12. Pinciroli, C., O'Grady, R., Christensen, A.L., Dorigo, M.: Coordinating heterogeneous swarms through minimal communication among homogeneous sub-swarms. In: International Conference on Swarm Intelligence. pp. 558–559. Springer Berlin Heidelberg (2010)
13. Soule, T., Heckendorn, R.B.: A developmental approach to evolving scalable hierarchies for multi-agent swarms. In: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation. pp. 1769–1776. ACM (2010)
14. Squires, W.G., Luke, S.: Lfd training of heterogeneous formation behaviors. In: AAAI Spring Symposia (2018)
15. Sullivan, K., Luke, S.: Learning from demonstration with swarm hierarchies. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1. pp. 197–204. International Foundation for Autonomous Agents and Multiagent Systems (2012)
16. Sullivan, K., Luke, S.: Real-time training of team soccer behaviors. In: Robot Soccer World Cup. pp. 356–367. Springer (2012)
17. Sullivan, K., Wei, E., Squires, B., Wicke, D., Luke, S.: Training heterogeneous teams of robots. In: Autonomous Robots and Multirobot Systems (ARMS) (2015)