Continuous and Embedded Learning for Multi-Agent Systems

Zsolt Kira
Mobile Robot Laboratory
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: zkira@cc.gatech.edu

Alan C. Schultz
Navy Center for Applied Research in A.I.
Naval Research Laboratory, Code 5515
Washington, DC 20375
Email: schultz@aic.nrl.navy.mil

***Abstract*— This paper describes multi-agent strategies for applying Continuous and Embedded Learning (CEL). In the CEL architecture [1], an agent maintains a simulator based on its current knowledge of the world and applies a learning algorithm that obtains its performance measure using this simulator. The simulator is updated to reflect changes in the environment or robot state that can be detected by a monitor, such as sensor failures. In this paper, we adapt this architecture to a multi-agent setting in which the monitor is communicated among the team members effectively creating a distributed monitor. The parameters of the current control algorithm (in our case rulebases learned by Genetic Algorithms) used by all of the agents are added to the monitor as well, allowing for cooperative learning. We show that communication of agent status (e.g. failures) among the team members allows the agents to dynamically adapt to team properties, in this case team size. Furthermore, we show that an agent is able to switch between specializing within a section of the domain when there are many team members and generalizing to other parts of the domain when the rest of the team members are disabled. Finally, we also discuss future potential of this method, most notably in the creation of a Distributed Case Based Reasoning system in which the cases are actual Genetic Algorithm population members that can be swapped among team members.**

Index Terms – **Genetic Algorithms, Multi-agent Coevolutionary Systems, Cooperative Coevolution, Multi-agent Communication**

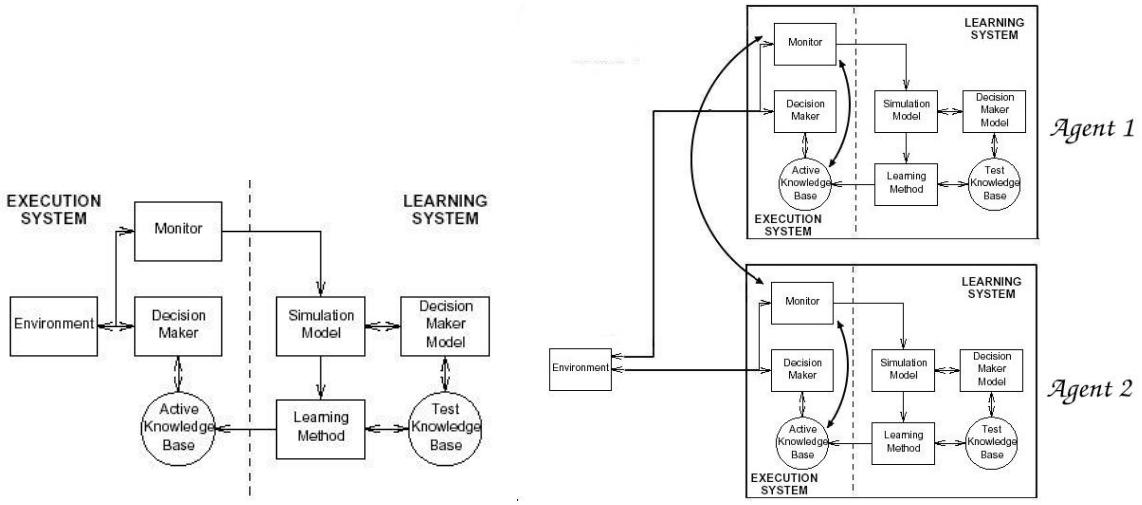
I. Introduction

One considerable challenge in robotics is performing tasks for extended periods of time, and adapting to changes in the environment and robot capabilities that occur throughout this period. An approach that has been designed to deal with such a problem is Continuous and Embedded Learning (CEL) [1], previously referred to as Anytime Learning [2]. The main idea in this approach is for the robot to continually improve its control system by applying a learning algorithm on an internal simulation of its environment, and to update this simulator to reflect major changes within the environment and robot capabilities that will effect performance of the task. The part of the system responsible for detecting such changes is called the *monitor*. Successful control strategies that are learned are continually sent to the actual robot control system that is being used to act in the real world.

It has been shown that CEL can be successfully used to adapt to environmental and robot changes, usually using genetic algorithms for learning. For example, Schultz and Grefenstette showed that a robot can adapt to sensor failures [1]. This system was extended by applying Case-Based Reasoning (CBR) for the initialization of the genetic algorithm populations that are used for learning [3]. CEL has also been applied in a domain involving coevolution of form and function, which has slightly different characteristics than multi-robot systems [6].

In team settings, the changes that occur also include changes in the team structure, size, or team member capabilities. The main purpose of this paper is to explore how communication among multiple agents in such a setting can be leveraged to create a distributed monitor and to adapt to these types of changes. In addition, we add the rulebases of the other agents as part of the monitor. This allows for team members to share their knowledge of other agents' behavior, in addition to their own, which enables cooperative coevolution. This information can be communicated explicitly by members of the team (as is done in this work) but can be possibly learned through observation when no such communication is available (e.g. enemies will not willingly communicate their rulebases). Furthermore, we demonstrate that the system allows for the ability of agents to switch between specializing to parts of the domain when the team is large and generalizing to larger areas of the domain when the team is small.

Related work includes case-based initialization of genetic algorithms, multi-agent learning, and adaptation to changing team properties. Using Case-based Reasoning to bootstrap initial populations used in Genetic Algorithms has been studied previously, although not in multi-agent settings [3][4][5]. Some of the work



(a) Single agent system

(b) Multi-agent system

Fig. 1. **Left:** The Continuous and Embedded Learning Model (from [1]). **Right:** Multi-agent CEL architecture with communication between monitors and inclusion of rulebases in the monitor.

involving reinforcement learning with respect to specialization and generalization of tasks, e.g. [7], is also relevant although there is no notion of monitors or CBR to switch between different strategies when team properties change. With regard to specific adaptation to team changes, the focus tends to be on creating algorithms that are robust despite team member failures but which do not adapt to this change. The advantage of our system is that it is able to explicitly adapt to the team properties, resulting in better performance when certain strategies are better after the team changes. Some exceptions include the L-ALLIANCE architecture [8] which is a behavior-based distributed architecture designed to be tolerant to faults or changes in the agents, market-based task allocation schemes [9] that distribute tasks based on market-like bidding and allow failed agents to retract bids, and physics-based control of swarms that can reorganize due to agent failures [10]. Note that in our work there is no explicit coordination performed by the robots, only cooperative evolution. In other words, the agents do not explicitly communicate for the purposes of accomplishing the task; all communication is solely to aid learning to accomplish the task. Adding such coordination on top of the current system is certainly possible.

There is also a large body of research in cooperative coevolution. The underlying genetic algorithm implementation used here was extended to multi-agent domains in [11], but not using CEL. The resulting cooperative coevolution architecture has been analyzed since then, especially with regard to the issue of choosing among the population members of coevolving agents when determining the overall fitness (our method essentially takes the most fit population member of other agents, communication permitting) [12]. Other multi-agent work has looked at Punctuated Anytime Learning, where the computational complexity of evaluating *entire* populations against each other is reduced by only doing this periodically [13]. This can be applied in our work by communicating entire populations instead of the best known rulebases, although this would increase the amount of communication and computational complexity. No application of CEL to coevolutionary multi-agent domains using real communication has been explored nor has the idea of a distributed monitor.

II. Architecture

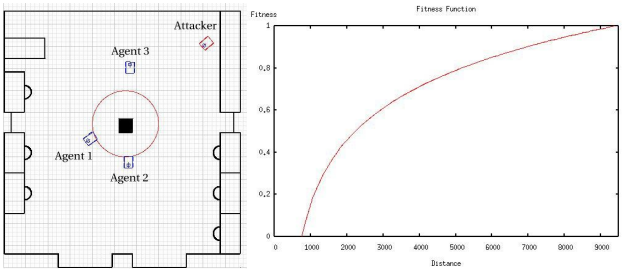
The original CEL approach is depicted in Figure 1a. There are two main modules in the CEL model. The execution module controls the robot's interaction with the environment using the decision maker which receives input from the knowledge base. The learning module continuously tests new strategies for the robot against a simulation model of the environment. When a new strategy is discovered that performs better than the robot's current strategy, the control system is notified and updated. Conversely, when the robot detects sensor or environmental changes relevant to the simulator, the learning module is updated. The module that detects such changes is called the monitor.

In this work, we use a Java implementation of SAMUEL [2], called Jaga-Samuel[1] [14] for the learning system. SAMUEL is a genetic algorithm that evolves rulebases containing condition-action rules where conditions are based on sensor values and the corresponding outputs are actions to be performed. In addition to traditional genetic algorithm mechanisms (selection, mutation, and crossover) Lamarckian operators are added that modify rules (i.e. modify the genetic material) based on the actual interaction between the agent and the environment. There are also methods to assign strengths to the rules that are used when determining which rule to enact, allowing for a type of structural credit assignment.

The Multi-agent CEL architecture is shown in Figure 1b. In the previous system, with only a single agent, there was only one execution module and one learning module. In the new architecture, there are multiple execution and learning modules, with communication between the monitors of the individual agents. In addition, the best rulebases discovered by the agents are sent to the monitor, so that this information can be relayed to other agents through communication. This allows the agents to co-evolve. The agents communicate if they are within communication distance (here it was infinite), and send both their current best rulebase as well as the latest knowledge from their monitors (which includes the best rulebase of all the other agents). This creates a distributed monitor in which robots can pass on environmental, robot, or team knowledge to each other. In order to ascertain which agent's knowledge is more recent, all updates are tagged with a timestamp.

In addition, each agent has a Case-Based Reasoner, where the indexing to the cases is based on the situation and the output of the case is an entire genetic algorithm learner (population and rulebase) that has been suspended. Matching during retrieval is based on the nearest neighbor algorithm with hamming distance (i.e. the number of features in the case that match) as the distance metric. If there are enough matching features, as determined by a threshold, the case is returned. If not, then no case is returned and the agent creates a new case with a learner that starts with an initial population. If there are multiple matching cases, one is randomly picked. Note that when agents communicate their best rulebases to each other, this does not trigger a case retrieval and is not part of the indexing. This is because rulebases are highly dynamic and it is unlikely that agents will have similar rulebases as in the past.

[1]For questions about Jaga-Samuel or information on obtaining it, please send an email to samuel@aic.nrl.navy.mil.



(a) The multi-agent domain.

(b) The fitness function based on the distance of the enemy from the center of the ship. If the enemy is within the safety radius, then no fitness is given to any of the defenders.

Fig. 2. The domain and fitness function used in the experiments.

III. Experimental Design

The multi-agent task chosen to test the resulting model is a defense task in which several agents guard a centrally located ship against incoming attackers. Figure 2a shows the domain, which is 12,000 units in height and 11,000 units in width. In the experiments conducted here, there were three defenders and one enemy attacker. The defender positions were fixed around the ship that is to be defended, represented in the middle as a square. The circle around the ship with a radius of approximately 1,400 units shown in Figure 2a represents the "critical region" that the attackers must stay out of. If the enemy reaches the ship the domain is restarted. The fitness function outside the circle is defined by the following logarithmic function, where d is the distance from the enemy agent to the center of the ship (the function is plotted in Figure 2b): $log(\frac{(\frac{d-380}{380})^{1.05}}{3.325})$. This rewards the defending agents a great deal for keeping the attackers away from the ship and additional reward tapers off as the distances increase. Also note that the defenders are assumed to be slightly more aggressive than the attackers. The features used in indexing for the Case-Based Reasoners include which defenders are disabled as well as sensor failure information, although in these experiments no sensor failure situations were created.

All experiments were conducted using the Player/Stage simulation environment [15], with laser, beacon, and odometry sensor data and a model of the Pioneer 3DX robot. The sensors used by the defending agents were relative range and bearing to the enemy (if seen) and the ship. The sensors used by the attacking agents were the relative range and bearing to the nearest two defenders (if seen) and the ship. Action involved a turn signal ranging from 90 to -90 degrees and a speed signal. The initial rulebases for the agents consisted

of 19 rules consisting of several turn and speed values with no IF conditions, effectively making the robot move randomly. Each genetic algorithm (or Case in the case library) had 45 population members which were uniformly initialized to the initial rulebases described previously. Proportional selection, where the fitness of a member is used to determine its probability of being selected, was used. Four evaluations were performed for each of the evaluation phases of SAMUEL.

The results of the experiments are averages of 19 complete runs for each condition (with and without CEL). Each run consists of approximately 2800 evaluations in the real world, which is also in simulation for these experiments. Each evaluation consists of an attacker progressing towards the ship in a straight line starting from an initial position chosen from several possible positions around the edge of the world, with the addition of obstacle avoidance behaviors. After every 200 evaluation, the team size alternates between three and one defender (agent 3, on the northern side of the domain, is the one that remains active). In the non-CEL condition, this is not known by the defenders (since there is no monitor) and so they continue to train using simulations containing three defenders.

In the CEL condition, the monitor detects this change, which is communicated to other agents if they are within communicating distance. This causes the single remaining agent to evaluate the situation and either retrieve an existing case from its case library similar to the current situation or start learning with a new population if it has never encountered the situation. We gathered statistics regarding the fitness received by the agents in the real world in addition to the best rulebases at all times and the fitnesses obtained by the best member of the population during each generation (used to plot the learning curves). The behavior of the best rulebases after all of the runs were also analyzed, both in terms of the defense behaviors used during incoming attacks as well as exploration behaviors used when no enemies are visible.

IV. Experimental Results

In this section we describe results that demonstrate that using CEL in a coevolutionary multi-agent environment can allow the team to be more adaptive, in this case to different team sizes. Figure 3 shows the fitness in the real world (which in this case is also in simulation), with the team size alternating between 1 and 3 about every 200 evaluations (note that there is noise due to the randomization in attacker placement). The switch in team sizes is evident from the graph since the fitnesses are much higher when there are three agents defending as opposed to one. Without a distributed monitor (lighter color in Figure 3), the agent that is remaining when the other two are disabled still continues to learn using a simulation with three agents. Since this does not reflect the real situation, its performance does not improve significantly. It is interesting to note that there is a slight improvement through time, meaning that strategies that the remaining agent learned for the three agent situation were also somewhat helpful when it was alone.

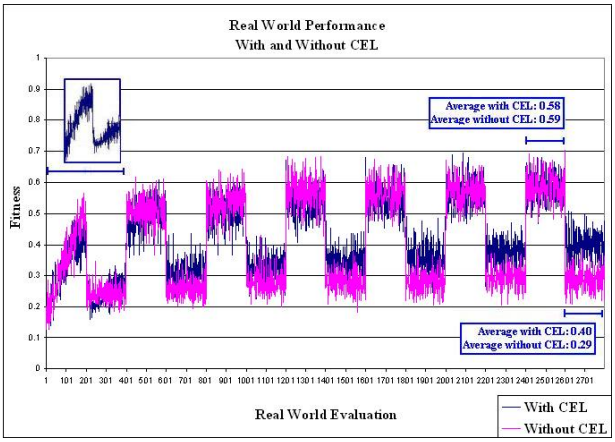


Fig. 3. Real world performance with and without CEL. The box on the upper left zooms in on the CEL performance during the first 400 evaluations, showing the two separate learning stages (for each case in the library) which are not exhibited in the non-CEL system. The averages in the boxes on the right demonstrate that using CEL, the lone agent was able to learn better strategies when its monitor detected a change in team size. Case 1 (average shown on top) shows no difference because both trained on this situation, whereas there is a significant difference for Case 2 (average shown on bottom).

When using CEL and a distributed monitor (darker color in Figure 3), however, all the agents receive information from the monitor or from communications with other agents that the two agents have been disabled. Since the agents initially do not have a case for this situation, learning begins from scratch using the initial genetic algorithm population. On the upper left in Figure 3, the performance of the CEL architecture alone for the first 400 evaluations is shown for clarity. As can be seen, after the first change (at evaluation 200, where two agents are disabled) a steep learning phase similar to the previous case occurs. This is because the agent has been informed of a change, could not retrieve any similar case encountered during previous experiences, and began learning again with a new population. Without CEL, there is only one steep learning phase after which the performance levels off.

After approximately 2600 evaluations, the divergence in performance between the conditions with CEL and without CEL become apparent. There is a statistically significant difference in the performance of Case 2 (with one agent) with an average fitness of 0.399 with CEL

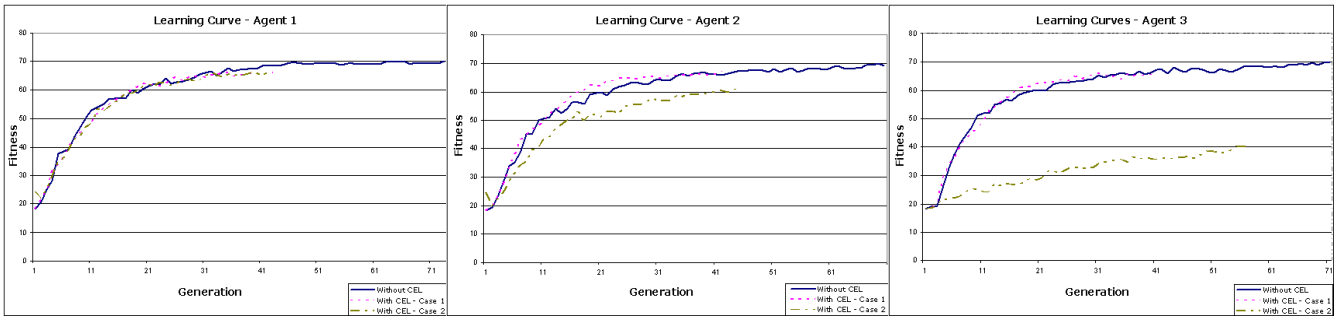
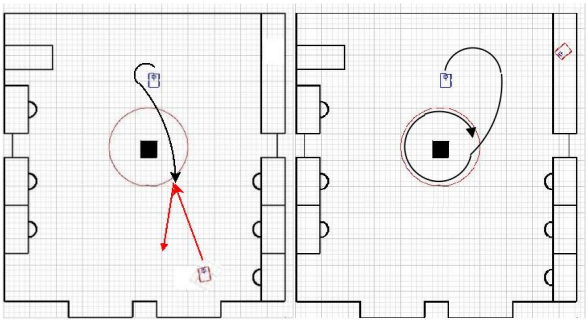


Fig. 4. Learning curves for the three agents. Without CEL, the simulator used for learning is one with three defenders. Using CEL, there are two cases in the case library: Case 1 represents the situation when there are three defenders, and Case 2 represents the situation when two defenders are disabled and there is only one active defender.

and 0.289 without (95% confidence interval difference > 0.1075). In Case 1, however, there is no statistically significant difference with an average fitness of 0.579 with CEL and 0.587 without (95% confidence interval difference < 0.005) since both conditions employed training for this situation. Note that at any point in time, the condition without CEL has trained for more generations than with CEL because it only has one simulator on which it trains. Using CEL, whenever the team size changes the agent loads the case for that situation and suspends learning for the previous situation. This is why it takes a while for the CEL system to reach the same fitness for learning in Case 1 than the non-CEL system.

Figure 3 showed the performance of the agents in the real world, with agents acting under rulebases that are continuously updated to the best one found so far in simulation. Figure 4, on the other hand, shows the learning curves for the individual agents. Note that the curves represent the averages of all the runs, and only generations for which *all* experiments finished are plotted. The non-CEL condition has almost twice as many generations completed because only one case is trained, whereas with CEL there are two cases. Note that in Figure 4c the single agent training in Case 2 with the other agents disabled accomplishes learning for more generations than in Case 1 because it is a harder task and hence the enemies reach the ship more often. Since the simulation is stopped when the enemies reach the ship, the evaluations take less time and hence more generations are completed.

Although the two disabled agent were not acting in the real world, we allowed them to continue learning in a simulation where only the one other disabled agent does not act. This did not effect real world performance since they could not act when disabled, but allowed us to gain insight into the difficulty of the domain with two active defenders. In all cases, without CEL the three agents trained on a simulator with all three agents being active. The learning curves for situations with two defenders



(a) Depiction of a defense behavior learned with a single defender in the CEL condition. The defender is able to search for enemies by rotating and intercepts the enemy in time.

(b) Depiction of an exploration behavior learned when there is no enemy in sight (it has been moved within the walls where it is not visible), for one trial of the CEL system.

Fig. 5. Defense behaviors in rulebases learned in the CEL condition.

are similar to those with three and performance levels off at a fitness of about 0.70 and starts at around 0.20. Note that this represents a great deal of difference in terms of distances (approximately 2,800 units) since the fitness function is logarithmic with respect to the actual distances at which the enemy was warded off.

In the learning condition with two agents remaining (Case 2 for agent 1 and 2), the learning rate was similar to that where all agents are active. However, the fitness of Case 2 of agent 3, where *both* of the other agents are disabled, is significantly lower since one agent has difficulty warding off the enemy coming from different locations. This shows that this domain is significantly easier with two or three active and much more difficult when there is only one agent. However, since the agent in the CEL system recognized that it was the only one left and specifically began learning for that condition in its simulation, it was able to adapt and learn to become

a more general defender. In the non-CEL case the agent never realized this and continued learning to defend with two other agents besides it, but did not perform as well in the real world because its simulations did not reflect reality accurately.

Having established the performance gains of the multi-agent CEL architecture, we now look at the qualitative behavior of the robots in the two different cases. Figure 5a depicts successful behavior using rulebases learned in CEL trials with the single lone agent defending against an attacker coming from a different area. As can be seen, the defender in this case is able to find the enemy by spinning around, and heads towards it quickly enough to intercept. When using the rulebases learned in the non-CEL condition, the lone agent was usually not able to defend against the attacker unless the attacker was near its region. In the case with three agents, successful intercepts included both a team effort where multiple defenders went towards the enemy and instances with one or two defenders defending against the enemy while the others explored around the environment.

One interesting thing to look at is the exploration behavior of the single robot (when the others are disabled) while the enemy is not in sight. Figure 5b shows the behavior of the single agent with CEL when no enemy is in sight using one of the rulebases learned. Interestingly, in one trial in the CEL system the agent learns to go in circles around the ship at about the same radius that the safety radius defines. In other words, the agent has generalized to searching the entire area for enemies, as opposed to sticking near its own area when there are three active agents. Behaviors similar to this are in four of the CEL trial and although they occur in the non-CEL trials (albeit less frequently), the non-CEL rulebases appear to be less varied in their exploration strategies and many times stay near the quadrant the agent started in.

This can seen in Figure 6, which shows a discretized representation of the areas that are heavily explored. The best rulebase of each of the 19 runs for each condition were used without any enemy around, and the position of the defender at each timestep recorded. The figure shows a discretized histogram of visited locations, where lighter areas indicated more heavily visited locations. As can be seen, in the CEL condition Case 2, where the rulebase is the result of training using a lone defender, the exploration is more uniform around the ship and more compact as well. In the non-CEL system there is some circular exploration around the ship, but it is not compact and the majority of the time is spent in the northern region of the environment. The CEL condition Case 1 is similar, but less structured. Although non-CEL condition and Case 1 of the CEL condition trained on the same simulator, the differences could be accounted for by the fact that the non-CEL condition trained on this situation for almost twice as many generations. Overall, the behavior of the agents in the CEL system may be an interesting finding in light of research on specialization versus generalization. One benefit of CEL is that it is able to specialize or generalize the agents' abilities based on the current team properties. Since all of the behaviors are maintained in a case library, the agents can switch from specialists to generalists dynamically as the situation warrants.

V. Conclusions

In this paper, we have shown that the Continuous and Embedded Learning architecture used previously to adapt to sensor failures in single agents can be extended to multi-agent coevolutionary domains to adapt to changes in the team properties. This is done by allowing the agents to communicate the monitor resulting in a distributed monitor, which now also includes the rulebases used by the agents. Using this system, the single agent was able to adapt and learn how to defend when the other defenders were disabled. We have also shown that the system enables agents to dynamically switch between task specialization and generalization in a domain. When there are other team members, it was beneficial to split the domain among the agents and concentrate on defending in a certain region of the domain. When there was only one defending agent, however, the remaining defender learned to explore in order to effectively intercept all the attacks.

These results are very promising, and there is a great deal of future work that can be done to further explore Multi-agent CEL. First, there are several ways to make the domain more challenging. For example, several attackers can attack at once and can run away when they encounter a defender, and they can be made more aggressive in terms of avoiding the defenders. Also, it would be preferable if the real world simulation was continuously running in order to gauge whether the defenders are able to learn to stay near the ship when there are no attackers. One change in the fitness function that can encourage this behavior is to explicitly reward the defenders for being close to the ship when the enemies are a certain large distance away from the ship.

Second, there is also future work in exploring a Distributed Case-Based Reasoning system in which an agent can retrieve cases from nearby agents (in addition to their own case libraries) and seed its population with the contents of these cases instead of starting from scratch when a situation has never been encountered before. This leverages the experiences of other agents through

(a) Without CEL

(b) CEL Case 1 (learned on simulator with three defenders)

(c) CEL Case 2 (learned on simulator with one defender)

Fig. 6. Exploration when there is no enemy, for the two conditions. Lighter areas indicate regions visited by the defender more often. In the CEL condition Case 2, where the remaining defender trained explicitly for defending alone, the agent tends to explore more away from its starting location and more compactly around the ship. In the non-CEL condition the agent does have some nice circular exploration, but tends to stay in the northern part of the domain a great deal and the exploration is less uniform.

communication. Varying communication distances of the agents might also yield interesting results in that perhaps defenders that are closer may have more pertinent cases than agents that are farther away. Finally, the Case-Based Reasoning system can be tested further by adding other types of changes at the same time, for example sensor failures of individual agents.

VI. ACKNOWLEDGMENT

The authors would like to thank Bob Daley for developing and help on using Jaga-Samuel, the Java implementation of SAMUEL, as well as valuable discussions. We would also like to thank Arya Irani, Jeff Bassett, Paul Wiegand, Joseph Blumenthal, and Mitchell Potter for discussions and ideas related to this work.

REFERENCES

[1] A.C. Schultz and J.J. Grefenstette, “Continuous and Embedded Learning in Autonomous Vehicles: Adapting to Sensor Failures”, *Unmanned ground vehicle technology II*, 2000.J.J. Grefenstette and C.L. Ramsey, “An Approach to Anytime Learning”, *International Workshop on Machine Learning*, 1992.

[2] J.J. Grefenstette, C. L. Ramsey, and A. C. Schultz, “Learning sequential decision rules using simulation models and competition”, *Machine Learning*, 5(4), pp. 355-381, 1990.

[3] C.L. Ramsey and J.J. Grefenstette, “Case-based initialization of genetic algorithms. ”, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo: Morgan Kaufmann, pp. 84–91, 1993.

[4] S.J. Louis and J. Johnson, ”Solving similar problems using genetic algorithms and case-based memory”, In Proceedings of the Seventh International Conference on Genetic Algorithms, pages 283–290. Morgan Kauffman, San Mateo, CA, 1997.

[5] X. Liu, *Combining Genetic Algorithm and Case-based Reasoning for Structure Design*. University of Nevada, Reno. M.S. Thesis, Department of Computer Science.

[6] D. Bugajska and A.C. Schultz, “Anytime Coevolution of Form and Function”, *Proceedings of the Congress on Evolutionary Computation*, 2003.

[7] P. Ulam and T. Balch, "Niche Selection in Foraging Tasks in Multi-Robot Teams Using Reinforcement Learning", *2nd International Workshop on the Mathematics and Algorithms of Social Insects*, Atlanta, Georgia, 2003.

[8] L.E. Parker, “ALLIANCE: An architecture for fault tolerant multi-robot cooperation”, *IEEE Transactions on Robotics and Automation*, 1998.

[9] B.P. Gerkey and M.J. Mataric, “Sold!: Market methods for multi-robot control”, *IEEE Transactions on Robotics and Automation Special Issue on Multi-Robot Systems*, 2001.

[10] W.M. Spears, D.F. Spears, J.C. Hamann, and R. Heil, “Distributed, Physics-Based Control of Swarms of Vehicles”, *Autonomous Robots*, 2004.

[11] M.A. Potter, K. De Jong, and J.J. Grefenstette, “A Coevolutionary Approach to Learning Sequential Decision Rules”, *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1995.

[12] P. Wiegand, W. Liles, and K. De Jong, “Cooperative coevolution: An architecture for evolving coadapted subcomponents”, *Evolutionary Computation*, 8(1), pp.1–29. 2001.

[13] H.J. Blumenthal and G.B. Parker,“Punctuated Anytime Learning for Evolving Multi-Agent Capture Strategies”, *Proceedings of the Congress on Evolutionary Computation*, 1820-1827, 2004.

[14] R. Daley, “JAGA SAMUEL - A Java Implementation of John Grefenstette’s SAMUEL Learning System User Manuel”, *NRL Report*, Naval Research Lab, Washington, DC, 2004.

[15] B. Gerkey, R.T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems", *Proceedings of the 11th International Conference on Advanced Robotics*, pp. 317–323, Coimbra, Portugal, June 2003.