Department of Computer Science
George Mason University
Technical Report Series

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/
703-993-1530

Selecting Informative Actions Improves Cooperative Multiagent Learning

Liviu Panait
lpanait@cs.gmu.edu

Sean Luke
sean@cs.gmu.edu

Technical Report GMU-CS-TR-2005-5

Abstract

In concurrent cooperative multiagent learning, each agent in a team simultaneously learns to improve the overall performance of the team, with no direct control over the actions chosen by its teammates. An agent's action selection directly influences the rewards received by all the agents; this results in a co-adaptation among the concurrent learning processes. Co-adaptation can drive the team towards suboptimal solutions because agents tend to select those actions that are rewarded better, without any consideration for how such actions may affect the search of their teammates. We argue that to counter this tendency, agents should also prefer actions that inform their teammates about the structure of the joint search space in order to help them choose from among various action options. We analyze this approach in a cooperative coevolutionary framework, and we propose a new algorithm, oCCEA, that highlights the advantages of selecting informative actions. We show that oCCEA generally outperforms other cooperative coevolution algorithms on our test problems.

1 Introduction

Multi-agent learning is challenging because the problem dynamics are often complex and fraught with local optima. Of particular interest to us is cooperative multi-agent learning, where multiple agents learn to work together as a team to accomplish common goals [12]. More specifically, we are interested in concurrent learning, where each agent performs its own learning and has little or no control over the other agents' selection of actions.

Unfortunately, multi-agent learning is problematic for existing machine learning techniques because the concurrent learning processes are not independent. Consider an agent that observes the environment containing other agents and that tries to improve its performance. This leads to a modification in its behavior. This modification is then sensed by the other agents, who change their behaviors in order to improve their performance as well. This "moves the goalpost" on the original agent: its newly-learned behavior may no longer be appropriate. Thus as the agents co-adapt to one another, the environment is essentially changing beneath the agents' feet. Moreover, the agent itself contributes directly to how the landscape changes. Learning in the face of this dynamic is not easy: such co-adaptation can result in cyclical or chaotic adaptive behavior, and may gravitate towards suboptimal solutions.

Most cooperative multiagent learning algorithms assume the agents are rational: each agent searches for actions that fare well when used in combination with the actions currently favored by its teammates. This "best-response" approach usually results in the learners converging to Nash equilibria. Such "rational" convergence to equilibria may well be movement away from globally *team-optimal* solutions [10, 19]. To counter this, we argue that agents must *also* explore actions that inform their teammates about the structure of the space of rewards for joint actions. For example, if an agent identifies an action that helps other agents rank their available actions better, the agent should explore that action to help guide the teammates' learning processes.

Though we believe it to be general, we will demonstrate the application of this approach to a particular multiagent learning method of interest to us, namely *cooperative coevolution* [7, 18]. Coevolution is the use of evolutionary computation learning techniques in a multi-agent setting. Ordinarily, evolutionary computation employs only a single learner to discover a global solution to an optimization problem: the learner first creates an initial pool of randomly-generated candidate solutions (a "population" of "individuals"), then assesses their quality ("fitness") independently of one another, then forms a new population of individuals through iteratively selecting, copying, and modifying ("breeding") individuals from the previous population with an emphasis on the fitter members of that previous population. The new population replaces the old one, and this cycle of fitness assessment, breeding, and population replacement repeats until a sufficiently fit individual is discovered or until resources have expired. Each iteration of this cycle is known as a "generation".

Cooperative coevolutionary algorithms (CCEAs), in the form we will discuss here, use not one but multiple populations, each involved in its own separate learning cycle of fitness assessment, population formation, and population replacement. However, individuals in a given population are no longer assessed independently, but rather in the context of individuals chosen from the other populations. Each population represents a subcomponent of a full solution to the problem, and as part of its fitness assessment, an individual in a given population may be evaluated only by combining it with one individual from each of the other populations to form a complete solution. It is in this fashion that coevolution involves multiple learners (each of the populations' evolutionary search procedures) whose learning trajectories are intertwined (via joint fitness evaluation), and so coevolution must deal with the same co-adaptation challenges as other multi-agent learning methods.

A natural approach to applying CCEAs to cooperative multiagent learning is to assign one population to each of the learning agents in the team. Each individual in the population represents a potential behavior for the agent, and so from now on, for consistency, we will refer to actions rather than individuals. An action may be as simple as a single action in trivial environments, or as complex as policies involving internal states and memory for real-world problems. Thus each population represents a finite sample from an infinite space of possible actions. As the team reward permits only the evaluation of joint actions, an action in one agent's population may be evaluated when combined with actions from the other agents' current populations. Multiple such combinations are usually used.

Section 2 highlights related cooperative multiagent learning algorithms. Following, Section 3 introduces a novel learning algorithm where agents pay special attention to informative actions. We compare it against other cooperative multiagent learning techniques in Section 4. The paper concludes with a brief discussion of our findings, accompanied by directions for future work.

2 Related Work

Two learning algorithms that are guaranteed to find the globally optimal joint action in a stateless environment are proposed in [1]. Both algorithms have two phases: agents first explore the entire space of joint actions (either deterministically or randomly); this is followed by a greedy selection of only the action (one per agent) that returned the highest reward. Both algorithms find the global optimum in polynomial time in the number of actions for each agent. However, scaling these algorithms to environments with states or with possibly infinite numbers of actions per agent may be problematic.

Instead of choosing actions deterministically or randomly, Claus and Boutilier [5] argue that agents should be more optimistic about their teammates: an agent should not prefer actions that do well in the context of the actions currently preferred by its teammates, but rather the agent should prefer actions that do well in the context of better actions that its teammates might learn. The application of this heuristic assumption results in additional multiagent reinforcement learning algorithms for stateless environments, such as the ones in [9, 8]. Unfortunately, scaling these algorithms from simple coordination games to more complex domains is nontrivial.

The cooperative coevolution literature has followed a similar path. The properties of cooperative coevolutionary algorithms are analyzed in [3, 4, 21]; results of such experiments indicate that assessing the fitness of an action based on the maximum of multiple joint rewards works better than if it were based on the minimum or on the average. Recent work has analyzed the conditions under which coevolutionary systems gravitate towards suboptimal solutions [20], has provided a visual illustration of the basins of attraction for simple cooperative multiagent domains [16], and it has proposed a biased CCEA that is more likely to find the global optimum [14].

Bucci and Pollack [2] apply recent advances from *competitive* coevolution research to improve CCEAs, resulting in the pCCEA algorithm. The authors argue that the aggregation of multiple joint rewards to compute the fitness of an action may result in loss of useful information. Instead, pCCEA uses all joint rewards to compute the set of actions in each population that are Pareto non-dominated: given two actions a_1 and a_2 for one agent, a_1 dominates a_2 if and only if (1) for any action b for the other agent, a_1 receives higher or equal reward when joined with b than a_2 does, and (2) there exists an action c for the other agent such that a_1 receives a strictly higher reward when joined with c than a_2 does. This set of non-dominated actions, termed an *archive*, is is automatically copied to the next generation to help evaluate the new population of actions. After the evaluation is completed for that generation, a new archive is computed for each population. Our experiments in Section 4 indicate that pCCEA's archive tends to converge to the Pareto frontier, which unfortunately may be infinite in even simple cooperative multiagent domains.

3 The oCCEA Algorithm

When multiple agents learn concurrently, each of them is afforded only a partial glimpse at the overall search space. Specifically, each agent may weight its actions using only a projection of the entire space, a projection that is influenced by the actions currently chosen by the other agents according to their own learning processes. Differences among such projections are illustrated in [13]: projections at early stages of learning may provide more information about the search space, because the agents' actions are more randomly distributed and so sample the joint space better. As the agents start to converge, the projections may become skewed, sometimes losing all information about the globally optimal solutions. As each agent's choice of actions influences the projections searched by the other agents, we may view each of the multiagent learning algorithms in Section 2 as recipes for agents to influence each other's learning processes.

We argue that the team of learning agents may benefit if each agent is concerned about the projection of the search space that it provides to its teammates via the actions it explores. In other words, agents should not necessarily explore only their most promising actions, but also those actions that provide the other agents with accurate projections of the joint search space. We propose a coevolutionary algorithm, oCCEA, to illustrate the advantages of such an approach.

For simplicity, we present the pseudocode for the algorithm using only two agents (and hence two populations), although it may be extended to arbitrary numbers of agents. Given an agent, the other agent is referred to as its *teammate*. We will assume that all populations have equal size *PopSize*, though the algorithm can easily be extended to allow for different sizes. We define $Reward_p(i,a)$ as the reward received when an agent (whose actions are represented in population p) selects action i and its teammate selects action a.

oCCEA follows the standard architecture of a generational cooperative coevolutionary algorithm [18]. As is the case for other CCEAs, oCCEA assumes that an agent can perceive the actions chosen by its teammates as well as the reward they receive. In oCCEA, as in pCCEA, each agent maintains a population of actions, a subset of which is defined as an *archive* responsible for ensuring that some actions exist primarily to keep the teammate's projection well informed. oCCEA agents learn concurrently, meaning, the populations advance through their generation cycles together, rather than one population advancing, then the other.

The evaluation process tests actions by combining them with actions from the teammate's archive, plus possibly some additional actions in the teammate's population. In addition to computing an action's fitness, the evaluation process stores the joint reward of any pair of actions that are evaluated together; this information is later used to update the archives of each agent.

Each generation, every action in a population is first evaluated by testing it in combination with every action from the teammate's archive. As the very first generation's populations have not built an archive yet, their entire population is used as an archive (an expensive process but one which ensures a thorough exploration of the joint space and good bootstrapping for the archive in future generations). This is meant to provide an accurate ranking of the actions. If the maximum size of the archives is less than $MaxEvals$, the actions are also tested with enough randomly-chosen actions from the teammate's population (in the archive or not) to provide at least $MaxEvals$ tests per action.

The fitness of action i is then set to maximum of F_i^j over all actions j in the teammate's population (F_i^j equals $-\infty$ if i was not tested in combination with j). If the archive size is 1, this reduces to a common evaluation approach for CCEAs [18, 21]: the evaluation is equivalent to using the best action (from the previous generation) plus some actions chosen at random from the other population. The pseudocode for the evaluation process is:

```
oCCEA-Evaluation
Parameters
  $MaxEvals$: maximum evaluations per action
Initial Settings
  For each population $p$
    $ArchiveSize_p = PopSize$
Evaluation phase (at each generation)
  For each population $p$
    $p' =$ other population than $p$
    For each action $i$ in $p$
      For each action $j$ in $p'$
        $F_i^j = -\infty$
      For each action $a$ in $Archive_{p'}$
        $F_i^a = Reward_p(i,a)$
        $F_a^i = Reward'_p(a,i)$
  $MaxArchive = \max_p ArchiveSize_p$
```

```
oCCEA-Evaluation (continued)
  Repeat for $\max(0, MaxEvals - MaxArchive)$ times
    For each population $p$
      Shuffle $p$
    For each index i in $1..PopSize$
      $a_1$ = action in population $p^1$ with index $i$
      $b_1$ = action in population $p^2$ with index $i$
      $F_{a_1}^{b_1} = Reward_1(a_1, b_1)$
      $F_{b_1}^{a_1} = Reward_2(b_1, a_1)$
  For each population $p$
    $p' =$ other population than $p$
    For each action $i$ in $p$
      $Fitness(i) = \max_{j \in p'} F_i^j$
```

The breeding and population reassembly phase of oCCEA proceeds similarly to the one in pCCEA: the archive members are selected from the old population and are copied directly into the new population, and remainder of the new population is filled with children bred using standard evolutionary computation algorithms applied to the old population (including the old archive). The whole previous population (including the archive) competes for breeding. The pseudocode is straightforward:

```
oCCEA-Breeding
**Breeding phase (at each generation)**
  For each population
    Select its new archive with oCCEA-Archive-Selection
    Copy the archive into the new population
    Fill the remainder of the new population using
          standard EC breeding
```

Archive selection is intended to select those actions which revealed features of the projected joint space useful to the other teammate. Specifically, we would like to select as an archive a minimal set of actions from each agent's population such that when assessing the fitness of actions in a given popualtion, testing them against the full teammate's population would not change the rank ordering of their fitnesses beyond just testing against the teammate's archive. The hope is that this archive would provide an accurate evaluation and ranking of the teammate's actions in the next generation as well. We'd like this set to be as small as possible, because as each action in an agent's population is tested in combination with *every* action from the other agent's archive, large archives imply a prohibitive $O(n^2)$ evaluation cost. Therefore we add actions to the archive only if they cause actions in the other population to improve significantly enough so as to effect the ranking — causing actions to *worsen* is not considered helpful information. Of the various actions which change this ranking, we will select the ones which do so by raising fitnesses to the highest levels.

The archive selection process starts from the empty set and proceeds iteratively. For each action i not in an agent's archive, and for each action x in its teammate's population, we compute the fitness of x if evaluated in combination with all actions in the current archive, that is, $M1Fit_x = \max_{j \in Archive_p} F_x^j$. We also compute the fitness of x if i were added to the current archive, $M2Fit_x^i = \max_{j \in Archive_p \cup \{i\}} F_x^j$. Note that $M2Fit_x^i \geq M1Fit_x$. Our first criterion for adding i to the archive (it improves upon the current ordering of the teammate's population) translates into finding two actions x and y such that adding i to the archive would change their ranking relative to one another ($M1Fit_x \leq M1Fit_y$ and $M2Fit_x^i > M2Fit_y^i$, or equivalently $M3Fit_{x,y}^i \neq -\infty$). Of all actions i that meet this first criterion, we greedily prefer the one that changed the ranking by raising the fitness of a teammate's action to the highest level. Note that the first action to be selected for the archive is always the action with the highest fitness. The pseudocode is:

```
oCCEA-Archive-Selection
**Parameters**
  $MaxArchiveSize$: maximum archive size
**Archive Selection (at each generation)**
  For each population $p$
    $p' =$ other population than $p$
    $Archive_p = \emptyset$
    While $Size(Archive_p) \leq MaxArchiveSize$
      For each action $x$ in $p'$
        $M1Fit_x = \max_{j \in Archive_p} F_x^j$
        For each action $i$ in $p - Archive_p$
          $M2Fit_x^i = \max_{j \in Archive_p \cup \{i\}} F_x^j$
          For each action $y$ in $p'$
            $M3Fit_{x,y}^i = \begin{cases} M2Fit_x^i & \text{if } \begin{array}{c} M1Fit_x \leq M1Fit_y \\ \text{and} \\ M2Fit_x^i > M2Fit_y^i \end{array} \\ -\infty & \text{otherwise} \end{cases}$
      For each action $i$ in $p - Archive_p$
        $MaxFit_i = \max_{x \in p'} \max_{y \in p'} M3Fit_{x,y}^i$
      Select $a = \text{argmax}_i MaxFit_i$
      If $MaxFit_a = -\infty$
        Break from while loop
      Add $a$ to $Archive_p$
    End while
```

4 Experiments

In this section, we investigate the behavior of four different coevolutionary algorithms, and we compare them in terms of performance and in terms of the number of evaluations they require to achieve that performance. The first algorithm is the pCCEA algorithm introduced in [2]. Second is the cCCEA algorithm, which is a traditional CCEA algorithm that evaluates the fitness of an action as the maximum reward it receives when in combination with any of the actions in the teammate's population. cCCEA is guaranteed to converge to the global optimum if the population size is sufficiently large [14]. Third is the oCCEA algorithm proposed in Section 3. Fourth, rCCEA evaluates the fitness of an action as the maximum when partnered with six actions from the teammate's population: five chosen at random, plus the fittest action from the teammate's *previous-generation* population. We included this algorithm in the comparison because it uses a fixed small number of evaluations per each generation, such as it would be preferable for applications of multiagent learning to real problems.

We will test these algorithms using a class of problem domains called the *maximum of two quadratics* (or MTQ). These problems include a global optimum and a local suboptimum, where the suboptimum covers a much wider range of the search space and is thus difficult to escape. The problems have been used before by [2, 15].

We will assume that each action is a real-valued number from 0 to 1 inclusive. This defines a metric space for actions: in some sense action 0.5 is more similar to action 0.6 than action 0.9 is. While other techniques search for optima among sets of actions that have no "distance" relation among them, EC methods assume a distance relation: when breeding an action to form a new one, they will generally make more small (distance) changes than large changes.

The joint reward function for the MTQ class is defined as:

$$\mathrm{MTQ}(x,y) \leftarrow \max \begin{cases} H_1 * (1 - \frac{16*(x-X_1)^2}{S_1} - \frac{16*(y-Y_1)^2}{S_1}) \\ H_2 * (1 - \frac{16*(x-X_2)^2}{S_2} - \frac{16*(y-Y_2)^2}{S_2}) \end{cases}$$

where x and y may take values (actions) ranging between 0 and 1. Different settings for H_1, H_2, X_1, Y_1, X_2, Y_2, S_1, and S_2 affect the difficulty of the problem domain in one of the following aspects. H_1 and H_2 affect the heights of the two peaks: higher peaks may increase the chances that the algorithm converges there. S_1 and S_2 affect the area that the two peaks cover: a higher value for one of them results in a wider coverage of the specific peak. This makes it more probable that the coevolutionary search algorithm will converge to this peak, even though it may be suboptimal. Different values for X_1, Y_1, X_2, and Y_2 result in changes in the locations of the centers of the two quadratics, which also affect the relatedness of the two peaks: similar values of the x or y coordinates for the two centers imply higher overlaps of the projections along one or both axes (the projections of the joint action space for one or both agents may retain more information about the globally optimal solution even if the other agent's population starts to converge to the suboptimal solution). In these experiments, we set $S_1 = \frac{16}{10}$, $X_1 = \frac{3}{4}$, $Y_1 = \frac{3}{4}$, $H_2 = 150$, $S_2 = \frac{1}{32}$, $X_2 = \frac{1}{4}$, $Y_2 = \frac{1}{4}$; H_1 was varied across experiments, but it was always less than 125.

MTQ using such settings is fairly difficult to optimize: the probability that a random sample exceeds a function value of 149.99 can be computed as $\pi * \left(1 - \frac{149.99}{150}\right) * \frac{S_2}{16} = 0.0000004090615$. Given 51200 random samples (approximately the number of action evaluations performed during a typical evolutionary run), the probability that one of them exceeds a function value of 149.99 is $1 - (1 - 0.0000004090615)^{51200} = 0.02072615$.

As we will see in the next sections, the median of the results for the proposed oCCEA method is significantly higher than 149.99, which implies that oCCEA finds better approximations of the global optimum in more than 50% of the runs. This shows that oCCEA significantly outperforms random search in this domain, and implicitly the algorithms proposed in [1].

To further increase the difficulty of the problem domains with respect to the algorithms we analyze, we created a second class of problem domains, SMTQ, which is defined as:

$$\mathrm{SMTQ}(x,y) \leftarrow \max \begin{cases} H_1 * (1 - \frac{16*(x_1^r-X_1)^2}{S_1} - \frac{12*(y_1^r-Y_1)^2}{S_1}) \\ H_2 * (1 - \frac{16*(x_2^r-X_2)^2}{S_2} - \frac{12*(y_2^r-Y_2)^2}{S_2}) \end{cases}$$

where x_1^r, y_1^r, x_2^r, and y_2^r are the original x and y values (which ranged between 0 and 1) rotated around the centers of the two peaks by $\frac{\pi}{4}$:

$$x_1^r = (x - X_1) * \cos\frac{\pi}{4} + (y - Y_1) * \sin\frac{\pi}{4} + X_1$$

$$y_1^r = (x - X_1) * \cos\frac{\pi}{4} - (y - Y_1) * \sin\frac{\pi}{4} + Y_1$$

$$x_2^r = (x - X_2) * \cos\frac{\pi}{4} + (y - Y_2) * \sin\frac{\pi}{4} + X_2$$

$$y_2^r = (x - X_2) * \cos\frac{\pi}{4} - (y - Y_2) * \sin\frac{\pi}{4} + Y_2$$

Observe that the two peaks have ellipsoid shapes aligned diagonally with the axes, as opposed to circular shapes in the MTQ problem domains. The two Nash equilibria from the MTQ class have now become an infinity of Nash equilibria in the SMTQ class. This creates an additional difficulty for the coevolutionary search. We used the same values for H_2, X_1, Y_1, X_2, Y_2, S_1, and S_2 as for the MTQ class.

The experiments used the ECJ package [11]. Each population contained 32 actions. cCCEA and rCCEA used elitism of size 1, meaning that the fittest individual in each population in the previous generation is automatically copied into the next generation's population. The entire archive survived automatically from one generation to the next for oCCEA and pCCEA. Unless stated otherwise, oCCEA used $MaxEvals = 5$ and $MaxArchiveSize = \infty$. The EC breeding method created children by selecting a parent, copying it, then "mutating" the copy by adding a gaussian random variable from a distribution with mean 0 and standard deviation 0.01, bounding the value to between 0 and 1. Parents were selected using "tournament selection" whereby two random parents are picked with replacement from the population, and then the fitter of the two is selected. Runs lasted 50 generations.

The quality of a technique was defined as the average, over 250 independent runs, of the fitness of the best action (one per population) in the last generation of that run. The results usually have a peculiar bimodal distribution, with many values close to one of the two peaks. For this reason, we report information on the quartiles, as opposed to mean and standard deviation. For the same reason, we verify statistical significance via non-parametric t-tests combined using the Bonferroni correction. Sometimes these non-parametric tests will return the opposite result than a regular parametric test would, and although some conclusions are different from the ones reported in [2], we believe our comparison methodology is well-founded.

4.1 Experiment 1: MTQ and SMTQ with $\mathbf{H_1 = 50}$

In the first experiment, we set H_1 to 50 to have a wide difference between the height of the two peaks. In this case, coevolution may have difficulties finding the global optimum primarily because its coverage is significantly smaller than that of the suboptimal peak.

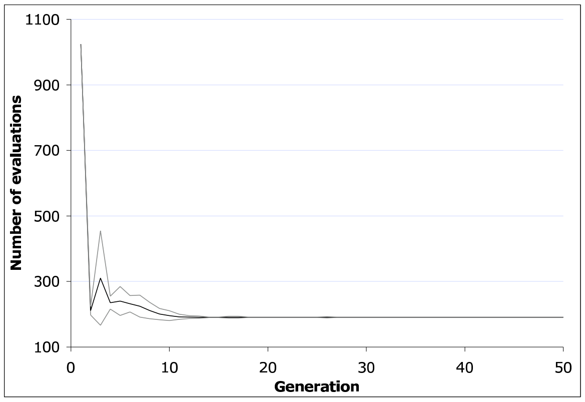

Figure 1: Average number of evaluations for the oCCEA algorithm in the MTQ domain instance with $\mathbf{H_1 = 50}$

Table 1: Results of the four methods in the MTQ domain instance with $\mathbf{H_1 = 50}$

Method	First Quartile	Median	Third Quartile	Average # Evals
pCCEA	148.62776	149.74876	149.94931	51200
cCCEA	149.99971	149.99995	149.99998	51200
oCCEA	149.99990	149.99997	149.99998	10676.5
rCCEA	50	50	149.99998	19200

Table 1 presents the performance of the four methods in the MTQ instance ($H_1 = 50$), as well as the number of evaluations required to achieve that performance. Although the differences appear small, the large number of observations (250) leads to statistically significant differences among the methods with confidence 95%. The results indicate that rCCEA performs worst, followed by pCCEA, cCCEA and oCCEA. There are statistically significant differences between all pairs of methods. And importantly, oCCEA achieves a significant reduction in the number of evaluations as compared to the other methods.

Figure 1 plots the number of evaluations required by the oCCEA algorithm at each generation. The algorithm starts with a complete round-robin evaluation (requiring 32×32 evaluations), followed by a drastic decrease in the number of evaluations. This is due to the fact that the number of actions needed to accurately rank the other population (hence the archive size) decreases significantly. In contrast, in the pCCEA algorithm the Pareto front in the MTQ domain is not discrete, and thus the archive grows rapidly in size until it occupies the whole population (Figure 2). At this point, learning stagnates.

Table 2 presents the results of the four methods in the SMTQ domain, as well as the number of evaluations required to achieve that performance. The rCCEA and pCCEA methods perform worst (with no statistically significant difference among them). cCCEA is better than both rCCEA and pCCEA. oCCEA significantly outperforms all three other methods. The dynamics of the archive size for pCCEA and oCCEA are very similar to

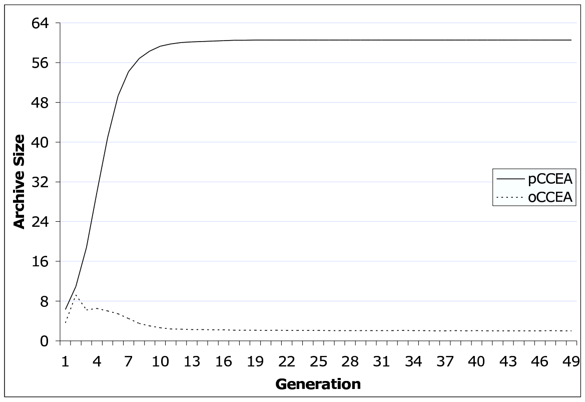

Figure 2: Average archive size for the oCCEA and the pCCEA algorithms in the MTQ domain instance with $\mathbf{H_1 = 50}$

Table 2: Results of the four methods in the SMTQ domain instance with $\mathbf{H_1 = 50}$

Method	First Quartile	Median	Third Quartile	Average # Evals
pCCEA	133.21958	146.85126	149.49486	51200
cCCEA	149.99975	149.99995	149.99998	51200
oCCEA	149.99991	149.99997	149.99998	10982.3
rCCEA	50	149.99986	149.99998	19200

those in the MTQ domain: pCCEA's archive again rises to consume most of the population, while oCCEA's archive size rises to 8 early but converges to approximately 5. This is summed over both populations. The average archive size for each population is half the value: an average size of 4 early, and 2.5 later.

4.2 Experiment 2: MTQ and SMTQ with $\mathbf{H_1 = 125}$

Similar to the experiments in [2], we set H_1 to 125 to create a more deceiving domain instance: the actions on the suboptimal peak have higher fitness and they are thus more likely to be selected.

Tables 3–4 present the results of the four methods in the MTQ and SMTQ domain instances with $H_1 = 125$. The results are consistent with the ones in Section 4.1: oCCEA is always better than cCCEA, which is in turn always better than pCCEA (with confidence 95%). The rCCEA method is worst: it finds the global optimum in only 16% of the runs in both the MTQ and the SMTQ domain ($H_1 = 125$). As before, oCCEA requires significantly fewer evaluations than the other methods. The average archive size is slightly higher (statistically significantly higher for oCCEA) than in the case of $H_1 = 50$, but it follows the same trend as that shown in Figure 2.

Table 3: Results of the four methods in the MTQ domain instance with $\mathbf{H_1 = 125}$

Method	First Quartile	Median	Third Quartile	Average # Evals
pCCEA	142.31862	149.03314	149.83407	51200
cCCEA	125	149.99974	149.99998	51200
oCCEA	125	149.99994	149.99998	11277.4
rCCEA	125	125	125	19200

Table 4: Results of the four methods in the SMTQ domain instance with $\mathbf{H_1 = 125}$

Method	First Quartile	Median	Third Quartile	Average # Evals
pCCEA	125	145.66753	149.38516	51200
cCCEA	125	149.99977	149.99998	51200
oCCEA	125	149.99995	149.99998	11406.7
rCCEA	125	125	125	19200

4.3 Experiment 3: The OneRidge Domain

Our last experiment examined the performance of the search methods in the OneRidge domain proposed in [17]. The OneRidge domain is defined as:

$$OneRidge(x,y) \leftarrow 1 + 2 * \min(x,y) - \max(x,y)$$

where x and y range between 0 and 1. OneRidge is particularly difficult for concurrent learners because it contains a very large number of Nash equilibria: for any value v between 0 and 1, (v,v) is a Nash equilibrium. This implies that for almost any Nash equilibrium (except for the global optimum $(1,1)$) there are an infinite number of better Nash equilibria that are infinitesimally close; unfortunately, both agents need to concurrently change their actions for the team to advance to better solutions. To better study the algorithms' capacity to follow this ridge to the global optimum, we randomly initialized the populations of actions for each agent to only values smaller than 0.5.

If SMTQ adds more non-linear interactions among the agents, OneRidge goes even further. As a consequence, the methods have a very different ranking based on their performance in this domain (as shown in Table 5). cCCEA performs best, followed in order by rCCEA, oCCEA, and finally pCCEA; there are statistically significant differences among all the methods.

This poor performance of pCCEA and oCCEA seems unexpected at first. To shed more light onto the behavior of the two algorithms, we plotted the average archive size for pCCEA and oCCEA in Figure 3. As expected, the Pareto frontier in the OneRidge domain makes pCCEA think that *every possible action* is interesting and needs to be added to the archive; consequently, the size of pCCEA's archive is close to 64 even in the very first generations, and so learning stagnates. Unlike MTQ and SMTQ, it is relatively easy in the OneRidge domain to improve upon a joint action by small variations in the actions chosen by each agents. As a consequence, both cCCEA and rCCEA are able to improve until outperforming pCCEA.

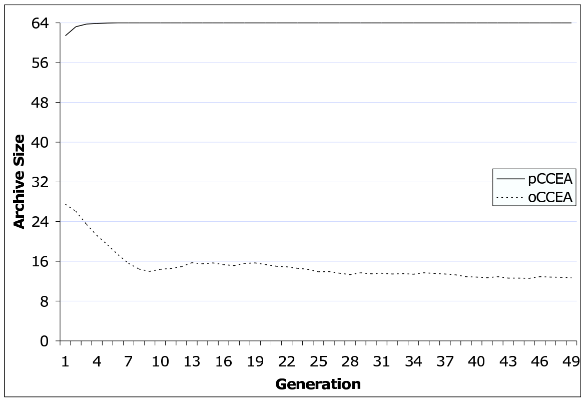

Figure 3: Average archive size for the oCCEA and the pCCEA algorithms in the OneRidge domain

Table 5: Results of the four methods in the OneRidge problem domain

Method	First Quartile	Median	Third Quartile	Average # Evals
pCCEA	1.45762	1.47224	1.48298	51200
cCCEA	1.89700	1.91584	1.93125	51200
oCCEA	1.50217	1.51778	1.53398	21545.3
rCCEA	1.82785	1.84294	1.86300	19200

The poor performance of oCCEA in this domain has a slightly different cause: the archive mechanism was designed to inform the concurrent learning processes of multiple Nash equilibria that are surrounded by large basins of attraction which cannot be avoided by small variations in actions. Given that OneRidge has no such equilibria, the archives serve little purpose, and they instead act to slow the optimization process by reducing the random exploration of the space. As shown in Figure 3, the average archive size of oCCEA is higher than $MaxEvals$ which was set to 5, and thus actions are not evaluated when in combination with random actions from the other population.

Revised Experiment To test this hypothesis, we restricted the maximum archive size of oCCEA to only one action ($MaxArchiveSize = 1$ in oCCEA-Archive-Selection), and we performed another 250 runs in the OneRidge domain. The median performance of the algorithm was 1.84439, with a first quartile of 1.82448 and a third quartile of 1.86095. This is indistinguishable from the performance of the rCCEA method in Table 5. We further doubled the maximum number of generations for the oCCEA algorithm with a maximum archive size of 1 (this setting still involved only around 40% of the budget used by cCCEA and pCCEA), and we ran it another 250 times. The global optimum was consistently and precisely found in all of them. This is significantly better than all other algorithms we tested in this domain.

5 Discussion and Conclusions

It is rational for learning agents to explore those actions that are rewarded better. However, each agent's experimentation with a particular action directly affects the way the other agents perceive the search space. Therefore, we argue that agents may benefit from also exploring those actions that inform their teammates about the structure of the search space. We suggest that cooperating agents should be altruistic: the entire team may benefit if each agent helps its teammates to rank their actions better. We demonstrated this idea in a new cooperative coevolutionary algorithm, oCCEA, which requires significantly fewer evaluations to outperform other cooperative multiagent learning methods on our test problems. We also noticed that too much altruism may hurt performance: an agent may waste resources when attempting to inform its teammates about the many Nash equilibria in the space. Restricting the number of informative actions an agent may choose solved the problem in our simple experiments, but we are exploring alternatives that can automatically balance the information provided to other learning agents with the desirability of searching for optimal joint actions. Future work will also examine formal models and guarantees for concurrent learners employing informative actions, as well as demonstrate such methods in complex multiagent domains.

References

[1] R. Brafman and M. Tennenholtz. Efficient learning equilibrium. In *Advances in Neural Information Processing Systems (NIPS-2002)*, 2002.

[2] A. Bucci and J. Pollack. On identifying global optima in cooperative coevolution. In Hans-Georg Beyer et al. [6], pages 539–544.

[3] L. Bull. Evolutionary computing in multi-agent environments: Partners. In T. Back, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 370–377. Morgan Kaufmann, 1997.

[4] L. Bull. Evolutionary computing in multi-agent environments: Operators. In D. W. V W Porto, N Saravanan and A. E. Eiben, editors, *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 43–52. Springer Verlag, 1998.

[5] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of National Conference on Artificial IntelligenceAAAI/IAAI*, pages 746–752, 1998.

[6] Hans-Georg Beyer et al., editor. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2005*. ACM, 2005.

[7] P. Husbands and F. Mill. Simulated coevolution as the mechanism for emergent planning and scheduling. In R. Belew and L. Booker, editors, *Proceedings of the Fourch International Conference on Genetic Algorithms*, pages 264–270. Morgan Kaufmann, 1991.

[8] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI02)*, 2002.

[9] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, San Francisco, CA, 2000.

[10] M. I. Lichbach. *The cooperator's dilemma*. University of Michigan Press, 1996.

[11] S. Luke. ECJ 13: A Java EC research system. Available at http://cs.gmu.edu/∼eclab/projects/ecj/, 2005.

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

[13] L. Panait and S. Luke. Time-dependent collaboration schemes for cooperative coevolutionary algorithms. In *Proceedings of the 2005 AAAI Fall Symposium on Coevolutionary and Coadaptive Systems*, 2005.

[14] L. Panait, R. P. Wiegand, and S. Luke. Improving coevolutionary search for optimal multiagent behaviors. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 653–658, Acapulco, Mexico, 2003. Morgan Kaufmann.

[15] L. Panait, R. P. Wiegand, and S. Luke. A sensitivity analysis of a cooperative coevolutionary algorithm biased for optimization. In Kalyanmoy Deb et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2004*, page (to appear), Berlin, Germany, 2004. Springer.

[16] L. Panait, R. P. Wiegand, and S. Luke. A visual demonstration of convergence properties of cooperative coevolution. In *Parallel Problem Solving from Nature — PPSN-2004*. Springer, 2004.

[17] E. Popovici and K. D. Jong. Understanding cooperative co-evolutionary dynamics via simple fitness landscapes. In Hans-Georg Beyer et al. [6], pages 507–514.

[18] M. Potter. *The Design and Analysis of a Computational Model of Cooperative CoEvolution*. PhD thesis, George Mason University, Fairfax, Virginia, 1997.

[19] Y. Shoham, R. Powers, and T. Grenager. On the agenda(s) of research on multi-agent learning. In *Proceedings of Artificial Multiagent Learning. Papers from the 2004 AAAI Fall Symposium. Technical Report FS-04-02*, 2004.

[20] R. P. Wiegand. *Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, Department of Computer Science, George Mason University, 2003.

[21] R. P. Wiegand, W. Liles, and K. De Jong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In L. Spector, E. D. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2001*, pages 1235–1242. Morgan Kaufmann, 2001.