

Evolutionary optimization of cooperative heterogeneous teams

Terence Soule and Robert B. Heckendorn

University of Idaho, Computer Science Dept, Moscow, Idaho, USA

ABSTRACT

There is considerable interest in developing teams of autonomous, unmanned vehicles that can function in hostile environments without endangering human lives. However, heterogeneous teams, teams of units with specialized roles and/or specialized capabilities, have received relatively little attention. Specialized roles and capabilities can significantly increase team effectiveness and efficiency. Unfortunately, developing effective cooperation mechanisms is much more difficult in heterogeneous teams. Units with specialized roles or capabilities require specialized software that take into account the role and capabilities of both itself and its neighbors.

Evolutionary algorithms, algorithms modeled on the principles of natural selection, have a proven track record in generating successful teams for a wide variety of problem domains. Using classification problems as a prototype, we have shown that typical evolutionary algorithms either generate highly effective team members that cooperate poorly or poorly performing individuals that cooperate well. To overcome these weaknesses we have developed a novel class of evolutionary algorithms. In this paper we apply these algorithms to the problem of controlling simulated, heterogeneous teams of “scouts” and “investigators”. Our test problem requires producing a map of an area and to further investigate “areas of interest”. We compare several evolutionary algorithms for their ability to generate individually effective members and high levels of cooperation.

Keywords: Evolutionary Algorithms, Genetic Programming, Cooperation, Autonomous Vehicles, Planning, Heterogeneous Teams

1. INTRODUCTION

Many practical problems are too large, too complex or structured inappropriately for single, monolithic intelligent agents to solve successfully or efficiently. For instance, large complex problems may contain specialized subdomains within a larger problem space that a single agent might overlook. A problem may be structured such that a single agent cannot be reasonably expected to have the resources to solve the problem. A problem space may be too large to be efficiently processed by a single agent and a cooperative approach will yield results with limited resources or time. For these problems and many others cooperative multi-agent systems are an obvious, and often a necessary, approach. Cooperative teams have been already been successful at solving a wide range of complex problems ranging such as predicting disordered regions in proteins,¹ learning musical models,² and weather prediction.³ However, for multi-agent systems to outperform single agent systems they must include both highly successful members and members that cooperate well. There is growing evidence that many of the traditional approaches do not meet both of these goals. Particularly when there teams are heterogeneous.

In this paper we test and compare several genetic programming techniques that apply varying amounts of selective pressure on team member and on teams as a whole. Our goal is to maximize team performance by finding the right balance of strong member performance and high levels of cooperation. The test problem requires teams of heterogeneous agents, “scouts” and “investigators”, to explore a previously unknown but bounded space. The agents must work as a team using their individual strengths to find, mark, and investigate as many of the interesting areas as possible in an allotted time.

Further author information: E-mail: heckendo@uidaho.edu, Web: <http://marvin.cs.uidaho.edu/heckendo/>;
E-mail: tsoule@cs.uidaho.edu

2. BACKGROUND

To be successful a multi-agent system must have members that individually perform well and cooperate well as a team. Typically this means that the members *specialize* on distinct, but potentially overlapping, sub-domains of the problem space. This requires heterogeneous control structures, which makes programming the agents difficult. Furthermore there are often multiple ways to divide a problem into unique sub-domains and it is unclear *a priori* what choice of sub-domains will lead to an optimal solution. For these two reasons there is considerable interest in using machine learning techniques, to automatically generate the control structures of the agents in multi-agent systems. For an overview of the field see Paniut and Luke.⁴

Research has shown the evolutionary techniques, including both genetic algorithms (GAs) and genetic programming (GP), are extremely effective at evolving multi-agent teams. Evolutionary approaches for training multi-agent teams have been applied to a wide range of knowledge representations, including teams of: neural networks,⁵ oblique decision trees,⁶ stack based predictors,⁷ and teams of induced functions.⁸ Evolutionary approaches have also been applied to a wide range of problem domains including robot navigation,⁹ sporting strategies,¹⁰ predator strategies,^{11,12} hazard assessment,¹³ and cancer and diabetes diagnosis.^{5,6}

Evolutionary algorithms are optimization algorithms that employ the principles of Darwinian evolution to pressure a population of possible solutions to evolve toward an optimum solution. The algorithms have several features in common. They require that the solution space have a codified **representation** and that a **population** of a variety of solutions be created in that representation. A **fitness function** that evaluates the goodness of the solution is all that is required to apply **selective pressure** to the population. Elements of the population are preferentially selected based on fitness. These elements are blended and altered by functions representing biological **crossover and/or mutation**. The results are evaluated for fitness and inserted into the population if they are sufficiently fit. From this basic outline, a large variety of powerful general purpose algorithms that structure and manipulate populations in various exist.

Research suggests that finding a good evolutionary algorithm for training teams can be difficult. Traditionally, evolutionary approaches have been roughly divided into two approaches: island and team. If a team can be thought of as having distinct labeled positions, in an island approaches, an individual for a given position is evolved in an independent population, i.e. on independent islands. In team approaches the entire team is evolved as a whole. In both cases individuals are trained for a given labeled position on a team. In previous research we have shown that island approaches produce highly fit team members, but there is a high probability that those members have correlated errors resulting in less than optimal team performance. Furthermore we have shown that team approaches can produce team members with inversely correlated errors leading to relatively good team performance, but the members themselves are relatively poor, which limits the team's performance.¹⁴⁻¹⁶

To overcome the weaknesses of the island and team evolutionary approaches, we have developed a novel class of genetic programming algorithms that alternates between treating the evolving population as consisting of independent islands of agents and treating it as single population of teams. These orthogonal views of the population lead to the name Orthogonal Evolution of Teams (OET).¹⁵ In previous work we showed that OET produces teams whose members perform better than those generated with team approaches and who cooperate better than those generated using island approaches.^{15,16} However, this research was limited to classification problems where an expected failure rate model could be directly applied and where agents cooperated via a voting mechanism on classification problems. Many multi-agent problems require more complex interaction between the team members. Thus, in this paper, we apply the OET algorithm to a more traditional multi-agent problem: exploring an unknown environment. We compare those results on the same problem using an the island and a team approach.

3. THE PROBLEM ENVIRONMENT

The environment is divided into a two dimensional grid (40x40). At the beginning of each evaluation each grid square has a twenty percent chance of being labeled as interesting. The interesting squares are determined randomly for each evaluation so that agents cannot memorize where the interesting squares are, instead the agents must learn general search patterns.

There are two agent types: **scouts** and **investigators**. The scout's role is to find interesting squares and mark them with a beacon that is detectable at a distance by the investigators. The investigator's role is to investigate interesting squares and mark them as **investigated**. If a scout is in an interesting square or is next to an interesting square, it automatically places a beacon in the interesting square (unless there is already a beacon there). If an investigator enters an interesting square, regardless of whether the square is marked with a beacon, it changes the square to **investigated** and deactivates any beacons in the square. The investigators work at half the speed of the scouts but can see the beacons at a distance. Therefore the space can be more efficiently explored by fast scouts marking interesting areas and investigators using the beacons to go directly to the areas to be investigated. Furthermore the two groups of agents have different subgoals and must divide up the space to be searched efficiently since the task has a time limit.

This model represents an abstraction of a number of practical problems. For example, scouts and investigators could represent two robot types exploring a minefield. Scouts fly overhead marking locations of potential mines and investigators deactivate the mines. Alternatively, they could represent an automated planetary surveying team. Scouts identify potentially interesting geological formations and investigators follow up by taking soil samples, etc.

We seek to answer several fundamental questions about heterogeneous team training.

1. Given a team of agents with different capabilities can evolutionary methods evolve appropriate programming for each agent type?
2. Given a team of agents with different capabilities what level of selective pressure on individuals versus teams optimizes the behaviors of the different agent types?
3. Given a team with multiple agents of the same type (e.g. multiple scouts and/or multiple investigators) will evolutionary algorithms evolve appropriately programming for agents of the same type? For example, will the evolutionary algorithms evolve control structures that cause the scouts to explore different regions of the space.
4. Given a team with multiple agents of the same type what level of selective pressure on individuals versus teams optimizes appropriately heterogeneous control mechanisms for different agents of the same type?
5. How well can evolutionary techniques use cooperation to overcome limited resources? E.g. if no single agent has sufficient time/energy to explore the entire space can the agents learn to partition the search effectively? Does this improve or hinder evolution of teams?

3.1. A vector model for agent movement

Space is a two dimensional real valued space. Agent movement is determined by a vector expression, represented by an expression tree, that calculates the next move. This expression calculates and returns a vector, based on current input data, and the agent moves in that direction. Investigators are limited to moves of length one and scouts are limited to moves of length two. It is this expression tree that is the representation of the solution space in the form of a program for each agent to determine its behavior.

Input vectors (terminal nodes in an agent's evaluation tree):

- North - a unit vector pointing North, 0 radians.
- Constant - a vector that is initially generated randomly at the time an agent's program is initialized. The constant remains so throughout the lifetime of agent but can be overwritten during mutation for instance.
- Random - a vector that is randomized at each timestep.
- Nearest scout - a vector to the nearest scout.
- Nearest investigator - a vector to the nearest investigator.

- Nearest beacon - a vector to the nearest beacon.
- Last move - a vector representing the agent’s last move.
- Nearest edge - a vector to the nearest boundary of the search space.

In the current implementation there is no limit on an agent’s vision e.g., an agent can “see” the nearest beacon regardless of its distance. If an input is meaningless, e.g. nearest beacon when no beacons are present, the zero vector (direction = 0, magnitude = 0) is returned.

Vector operations (non-terminal nodes in an agent’s evaluation tree):

- Invert - takes a single vector argument and inverts it (rotates π radians).
- Sum - takes two vector arguments, returns the vector sum.
- IfLTEMagnitude - takes four vector arguments, if the magnitude of the first argument is less than the magnitude of the second argument it returns the third argument, otherwise it returns the fourth argument.
- IfLTEDirection - takes four vector arguments, if the angle of the first argument is less than the angle of the second argument it returns the third argument, otherwise it returns the fourth argument.

Clearly the choice of inputs and operations has a significant influence on how the agents can evolve. We chose a fairly extensive set of input vectors, moving these experiments in the direction of global information zzz(??). Future research will look at the effect of changing this set: reducing the range of agent’s vision, allowing agents to sense each other by “name”, etc.

4. FITNESS EVALUATION

A significant difficulty in many multi-agent systems is how to assign credit (fitness) to individual team members. In this research, we chose to begin with a simple, basically greedy, approach. Each evaluation of fitness involves a new random problem space of beacons being setup and the bots placed dead stop in the center pointing “North”. Then they are given a fixed amount of time and their fitnesses for the scouts and investigators are as follows:

$$fitness_s = 3\beta - .1b \qquad fitness_i = 3I - .1b$$

Where β is the number of beacons placed, b is the number of time steps outside of the bounded problem area, I is the number of interesting areas investigated. Thus, scouts and investigators are rewarded for finding interesting squares and investigating them respectively and penalized for leaving the boundaries. Interestingly we have observed that if the boundary penalty is too large agents will evolve that simply sit still to avoid incurring a penalty.

The fitness of a team is the sum of the team members’ fitnesses. The fitness of a particular member or team will vary somewhat between evaluations because for each evaluation the environment is randomly generated.

5. EVOLUTIONARY ALGORITHMS

A steady-state evolutionary algorithm is used. The basic algorithm parameters are given in Table 1. **Number of trials** is the number of times the algorithms is run for statistical purposes. **Number of time steps** is the virtual time allowed the team to perform on a given problem. **Run time** is the number of teams evaluated for their fitness during a evolutionary algorithm’s run.

During the mutation step, only leaf nodes (terminals) are mutated. Non-terminals can change through crossover. Crossover involves choosing, at random, a single subtree in each of the parent evaluation trees and exchanging them. Terminal and non-terminal nodes were chosen with equal probability.

To vary the amount of selective pressure for team versus member performance we tested three evolutionary algorithms: team, OET, and an island variation. Each algorithm varies in how teams and team members are selected. We then compare the fitness of the teams and the team members under the three algorithms.

Table 1. Summary of the evolutionary algorithm parameters.

	Standard	Restricted Resources
Population Size	100	100
Crossover Probability	1.0	1.0
Mutation Probability	2/tree size	2/tree size
Selection	3 member tournament	3 member tournament
Run Time	1500 evaluations	1500 evaluations
Maximum Size	None	None
Initial Population	Full trees of depth 4	Full trees of depth 4
Number of trials	30	30
Grid Size	40 by 40	40 by 40
Number of time steps	534	200

5.1. Team Algorithm

In the team algorithms all evolutionary pressure is applied to teams as a whole. In each iteration, tournament selection (tournament size of 3) is used to select two “parent” teams. The teams undergo crossover with each of the team members in the first parent being crossed with the equivalent team member in the second parent (e.g. member 1 of parent team 1 is crossed with member 1 of parent team 2; member 2 of parent team 1 is crossed with member 2 of parent team 2; etc.). This produces two new “offspring” teams. Each member of the new offspring teams is subjected to a mutation operation. During mutation each terminal node has a 1 in 10 chance of being randomly mutated into a new leaf node. Internal nodes are not effected by mutation. Two reverse tournaments, tournaments in which the individual with the lowest fitness “wins”, are used to select two poorer teams. These teams are replaced by the offspring teams.

In the team algorithm all of the selective pressure applies to the teams. Parents are selected based on the performance of the team as a whole. Teams are selected for replacement based on the performance of the team as a whole compared to members of a tournament of teams.

5.2. OET Algorithm

In the OET algorithm pressure is applied to both members and teams. In each iteration tournament selection (tournament size of 3) is used to select team members to generate, in piecwise fashion, two parent teams. E.g. tournament selection is applied to the first members of the teams to pick an “winning” team member 1, then tournament selection is applied to the second members of the teams to pick an above average team member 2, and so forth, until a team consisting of above average members is created. Thus, if teams consist of three scouts and three investigators the process is repeated six times to create a new “all star” team.

This process is repeated to create two “parent” teams. These two parent teams undergo crossover and mutation as in the team algorithm. And, also as in the team algorithm, replace two below average teams.

Thus, team members must individually be at or above average in a tournament to be selected for the parent teams. Teams must also be above average in a team tournament or they may be replaced by the offspring teams. Thus, there is direct pressure on both the team members and the teams as a whole.

5.3. Island Algorithm

In the Island algorithm, pressure is applied to members only. In each iteration, for each team member, four tournaments of size 3 are performed. Tournament selection is used to select two winning members in the tournaments and two losing members. The losing members are replaced by the winning members. The two replacement members then undergo crossover and mutation. The two teams containing the newly crossed and mutated members are then evaluated. In this algorithm the fitness of a team as a whole is never considered during the evolutionary process for the individual. Thus, evolution will emphasize individuals performance.

6. EXPERIMENTS

For each of the three evolutionary algorithms we used two experimental designs. In the first design, each evaluation of a team used 534 (1600/3) time steps, i.e. each agent could take up to 534 steps. On a 40 by 40 grid there are 1600 squares, so it was just possible for the investigators, which only move one square at a time, to completely cover the space. Although clearly a better strategy would be for the faster scouts to place beacons and the investigators to only visit the squares marked with beacons. In fact, the data shows that this better strategy did evolve.

In the second design each evaluation of a team used 200 time steps. This was not sufficient to fully explore the search space and thus should result in lower overall fitness scores. This puts more pressure on the teams to find was to subdivide the space efficiently. We limited the available time (which could also represent, for example, a fuel limitation) to compare the algorithms' performance with a more highly constrained version of the problem.

For each of the experimental designs and each of the algorithms 30 independent trials were run in order to determine the statistical significance of the difference in performance between the algorithms.

ACKNOWLEDGMENTS

This unnumbered section is used to identify those who have aided the authors in understanding or accomplishing the work presented and to acknowledge sources of funding.

REFERENCES

1. K. Peng, S. Vucetic, P. Radivojac, C. Brown, A. Dunker, and Z. Obradovic, "Optimizing long intrinsic disorder predictors with protein evolutionary information," *Journal of Bioinformatics and Computational Biology* **3**(1), pp. 1–26, 2004.
2. G. Widmer, "Discovering simple rules in complex data: a meta-learning algorithm and some surprising musical discoveries," *Artificial Intelligence* **146**, pp. 129–148, 2003.
3. I. Maqsood, M. R. Khan, and A. Abraham, "An ensemble of neural networks for weather prediction," *Neural Computing and Applications* **13**(2), pp. 112–123, 2004.
4. L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems* **11**(3), pp. 387–434, 2005.
5. Y. Liu, X. Yao, and T. Higuchi, "Evolutionary ensembles with negative correlation learning," *IEEE Transactions on Evolutionary Computation* **4**(4), pp. 380–387, 2000.
6. E. Cantu-Paz and C. Kamath, "Inducing oblique decision trees with evolutionary algorithms," *IEEE Transactions on Evolutionary Computation* **7**(1), pp. 54–68, 2003.
7. M. D. Platel, M. Chami, M. Clergue, and P. Collard, "Teams of genetic predictors for inverse problem solving," in *Proceeding of the 8th European Conference on Genetic Programming - EuroGP 2005*, 2005.
8. T. Soule, "Voting teams: A cooperative approach to non-typical problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, eds., pp. 916–922, Morgan Kaufmann, (Orlando, Florida, USA), 13-17 July 1999.
9. H. Iba, "Multiple-agent learning for a robot navigation task by genetic programming," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. R. Riolo, eds., pp. 195–200, San Francisco, CA: Morgan Kaufmann, 1997.
10. S. Raik and B. Durnota, "The evolution of sporting strategies," in *Complex Systems: Mechanisms of Adaptation*, R. J. Stonier and X. H. Yu, eds., pp. 85–92, IOS Press, 1994.
11. T. Haynes, S. Sen, D. Schoenefeld, and R. Wainwright, "Evolving a team," in *Working Notes of the AAAI-95 Fall Symposium on GP*, E. V. Siegel and J. Koza, eds., pp. 23–30, AAAI Press, 1995.
12. S. Luke and L. Spector, "Evolving teamwork and coordination with genetic programming," in *Genetic Programming 1996: Proceedings of the First Annual Conference on Genetic Programming*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. R. Riolo, eds., pp. 150–156, Cambridge, MA: MIT Press, 1996.

13. D. W. Obitz, S. C. Basak, and B. D. Gute, "Hazard assessment modeling: An evolutionary ensemble approach," in *Proceedings of the Genetic and Evolutionary Computation Conference: GECCO-1999*, pp. 1543–1650, Morgan Kaufmann, 1999.
14. T. Soule, "Cooperative evolution on the intertwined spirals problem," in *Genetic Programming: Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003*, pp. 434–442, Springer-Verlag, 2003.
15. T. Soule and P. Komireddy, *Orthogonal Evolution of Teams: A Class of Algorithms for Evolving Teams with Inversely Correlated Errors*, Springer, 2006.
16. R. Thomason and T. Soule, "Novel ways of improving cooperation and performance in ensemble classifiers," in *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2007)*, Morgan Kaufmann, 2007.