Evolving Cooperative Strategies for UAV Teams

Marc D. Richards
mdr@cs.colostate.edu

Darrell Whitley
whitley@cs.colostate.edu

J. Ross Beveridge
ross@cs.colostate.edu

Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523

ABSTRACT

We present a Genetic Programming approach to evolve cooperative controllers for teams of UAVs. Our focus is a collaborative search mission in an uncertain and/or hostile environment. The controllers are decision trees constructed from a set of low-level functions. Evolved decision trees are robust to changes in initial mission parameters and approach the optimal bound for time-to-completion. We compare results between steady-state and generational approaches, and examine the effects of two common selection operators.

Categories and Subject Descriptors

I.2.2 [**Automatic Programming**]: [Program Synthesis]; I.2.11 [**Distributed Artificial Intelligence**]: [Multiagent Systems]

General Terms

Experimentation, Performance

Keywords

Autonomous control, cooperative agents, genetic programming, simulated robotics

1. INTRODUCTION

Advancements in unmanned aerial vehicle (UAV) technology have made it possible to keep human pilots out of many dangerous aerial situations. UAVs can be used in place of piloted planes for military missions, hazardous search and rescue operations, and a variety of private-sector domains (aerial photography, collection of weather data, etc). Currently, most UAVs are controlled by an individual or team of individuals operating the craft remotely from a control station. This can become difficult if the controller must be stationed in a dangerous area, or when performing a task that requires multiple cooperating UAVs.

Cooperation among UAV team members is important for a variety of reasons. Some missions may be quite dangerous, and it is unlikely that a single agent would survive long enough to complete the task. Some UAV models are inexpensive but prone to failure, in which case it may be more economically feasible to use a large number of cheap UAVs rather than risk a single expensive one. Many tasks, such as searching a particular area, can be completed more quickly using multiple UAVs. Some types of missions may require multiple UAVs, such as the tracking of multiple mobile targets. In all cases, cooperation among the UAVs is required for efficient and/or successful completion of the mission. As the number of UAVs available for missions increases, the task of coordinating between UAVs becomes more difficult. Autonomous computer controllers that are capable of performing complex and cooperative behaviors are desired.

In this paper, we present a method for evolving cooperative strategies[1] for teams of UAVs using Genetic Programming (GP). Our focus is a search task, although we believe the approach can be easily extended to a variety of mission types. Additionally, we examine the differences in using a steady-state approach based on the Genitor algorithm compared to the traditional generational technique more commonly used in GP. Section 2 gives a brief overview of other approaches to cooperative UAV control. Section 3 introduces our simulator environment and the details of the search mission. Section 4 shows how GP can be applied to the cooperative search mission. Section 5 presents our empirical results, and Section 6 concludes the paper with our analysis and a discussion of future work.

2. RELATED WORK

Current approaches for autonomous cooperative UAV control can be separated into several groups. *Behavior-based* control systems use a network of interacting high-level behaviors to perform a task. Cooperation is achieved through the local interactions between UAVs performing the behaviors. Early behavior-based control systems are reviewed in [11]. A sample of more recent work with cooperative ground-based autonomous robots includes [14] and [2].

Deliberative approaches focus on developing a specific flight path for each UAV to follow. Ablavsky et al. [1] present a geometric approach for constructing flight paths for a single agent and give suggestions on how it can be extended to multi-UAV scenarios. Such flight paths are rigid and no ef-

[1]We use the term "strategy" to refer to the decision process that controls the UAV. Some researchers may prefer the terms "policy", "behavior", or "controller".

fort is made to alter them in the event that new information is received, such as the discovery of a hostile element. To achieve some degree of flexibility, many deliberative systems incorporate an element of *adaptive replanning*. In adaptive replanning, a centralized controller generates a specific flight path for each UAV to follow based on the information that is currently available. The UAV follows that flight path, sending sensor information back to the controller as it becomes available. As the controller receives new information, it may generate new flight paths that are communicated to the UAVs. The new plans may, for example, take into account the location of a previously unknown enemy or the fact that a UAV was lost due to hostile fire or mechanical failure. Parunak et al. [13] present a planner based on a pheremonal analogy, similar to ant-colony optimization. Rathbun et al. [17] present a method for choosing new flight paths using an evolutionary search. Other adaptive replanners are considered in [15], [18], and [20].

Adaptive replanning has a number of known drawbacks: Every time a new set of flight plans is generated, the centralized controller must broadcast them to each UAV in the field. A non-trivial choice must be made as to when is the appropriate time to replan. The replanning process is not instantaneous, and by the time the new plan is sent to the UAVs it may be obsolete.

Our research focuses on generating *reactive* strategies rather than a specific flight path that must be updated during the mission. The strategy is analogous to a single decision tree that controls the UAV for the life of the mission. The decision tree determines changes in the UAV's heading, based on immediate low-level information from sensors. The challenge is finding a strategy that can effectively react to potential hazards and dynamic elements, while still retaining a level of cooperation that allows for efficient completion of the task. Our solution uses Genetic Programming (GP) to evolve cooperative strategies.

Past research has shown that GP can successfully evolve cooperative behaviors in simple predator-prey domains [10] as well as robotic soccer [8]. GP has also been used to evolve control strategies in single-aircraft scenarios. Moore [12] evolved effective behaviors for missile avoidance on manned jets, while Barlow et al. [3] have evolved controllers for locating a radar source in a single-UAV domain. However, there is a lack of research exploring the use of GP for evolving cooperative strategies in realistic multi-UAV domains.

3. ENVIRONMENT

For reasons of cost and time, virtually all uses of GP for robot and UAV control require the use of a simulated environment. The strategies must be evaluated in this simulated environment, as the evolutionary process requires potentially thousands of strategy evaluations to converge on effective solutions. In addition, GP algorithms may progress through many failing solutions on their way to good solutions. In a simulator, these failure accrue no real cost, where repeated failures with real vehicles might.

3.1 Simulator

Our simulator environment is capable of supporting a variety of different mission objectives. It approximates continuous space within double-precision floating point accuracy. Time is modeled discretely, meaning that at each update of the environment a set number of milliseconds have passed in the simulated world. The present research focuses only on navigation in two dimensions: the altitude of the UAVs remains constant throughout the mission.

UAVs have a fixed turning radius that mimics actual flight dynamics. Movement between two points is not always a straight line and depends on the current heading of the UAV. Automated controllers "steer" the UAV by specifying a desired heading at each update. If the new heading is beyond the UAV's turning capabilities, the maximum turn will be taken in that direction.

While locations can be pinpointed with double-precision accuracy within the simulator, real UAVs will rely on sensor readings of lower accuracy. For this reason, all distance and location information that the UAV receives through its simulated sensors is rounded to the nearest meter. Jakobi et al. [6] addressed the general "reality gap" problem of moving specialized controllers from simulated environments to the real-world. Future work will incorporate some of their ideas to reduce the problems associated with such situations.

3.2 Mission

We have chosen to focus on the coordinated search problem. In this domain, a team of UAVs is tasked with exhaustively covering a pre-defined search area and then returning to a base location. The return-to-base requirement significantly increases the complexity of the missions for the evolutionary process. Rather than attempting to evolve effective return strategies, a heuristic was employed that guides the UAVs home at the appropriate time.

Each UAV is equipped with a sensor that can scan a certain portion of the search area as the UAV flies over it, "sweeping" it. The mission ends successfully when the search area is completely swept and all surviving UAVs have returned home.

The search area is represented internally as a discretized grid of virtual beacons. When an agent flies sufficiently close to a beacon, that beacon is removed from the agents' internal maps, and that portion of the search area is considered swept. In our current implementation, agents share the same map, so communication latency for a swept beacon is zero. Future implementations can more accurately reflect real UAV communication issues.

The search mission was chosen because solutions clearly require cooperation to be efficient. While explicit formation flying is not required for this domain, implicit cooperation is required for the UAVs to spread out and minimize redundancy in their flight paths. Similar search tasks are common in the UAV literature already referenced, including [1], [15], and [20].

Three variants of this mission are presented here. Each mission takes place on an approximately 50 by 40 km map. *SweeperA*, and *SweeperB* each contain a 12.5 by 5 km rectangular search area with two groups of five agents starting at fixed locations to the northwest and southwest of the sweep area. The base area (to which the agents must return when the sweep is complete) is located south of the search area. The relevant portion of the map for these missions is shown in Figure 1.

SweeperA contains a stochastic element of danger. UAVs are destroyed with a small but nontrivial probability as they fly over the search area. This is to mimic possible hostile fire or mechanical failure. (They can also be destroyed if they collide with one another). As UAVs are inevitably lost,

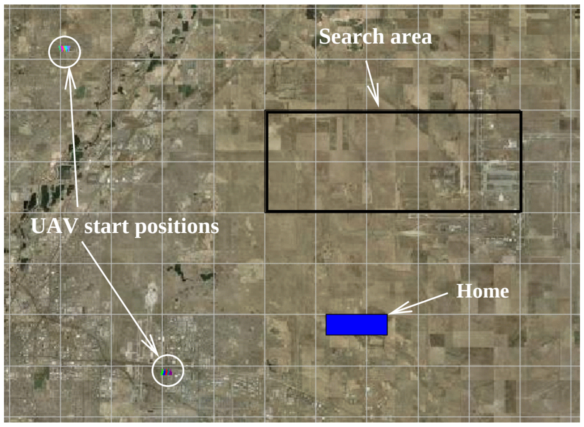

Figure 1: Initial map configuration for *SweeperA* and *SweeperB*

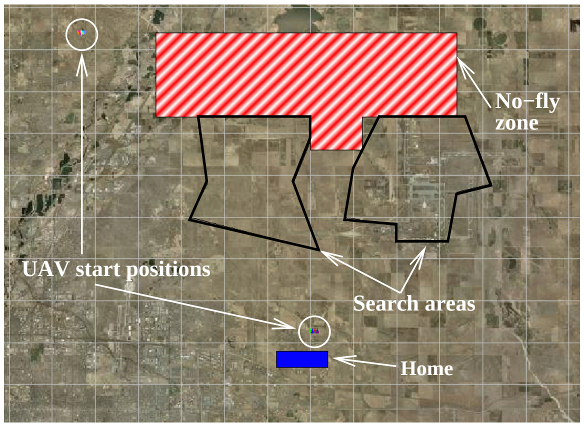

Figure 2: Initial map configuration for *SweeperC*

the remaining UAVs must continue sweeping until the entire search area is swept.

SweeperB includes a hostile agent of fixed but initially unknown location within the search area. When a UAV flies within range of the hostile agent, it is destroyed and all remaining UAVs are notified of the hostile agent's location. The UAVs must navigate around the hostile agent while continuing to sweep the remaining uncompromised search area.

SweeperC adds a no-fly zone to the map that borders the search area. This no-fly zone may represent a politically sensitive or excessively hostile region where the UAVs should not travel. Any UAV entering the no-fly zone is immediately destroyed. Additionally, the search area in *SweeperC* is irregular and non-continuous. The relevant portion of the map for this mission is shown in Figure 2.

The first two missions are relatively easy due to the rectangular geometry of the search area. If the UAVs are sufficiently spread out, they can cover the search area in a single horizontal flyover. *SweeperA* is non-trivial because some of the UAVs will be destroyed. However, the uniform probability of destruction means that a general strategy is still fairly easy to find. Strategies for *SweeperB* can be similar to the strategy used for the previous mission, with a contingency to fly around the location of known enemies. While preplanning an optimal path is not possible, it is clear that efficient strategies do not have to be extremely complicated. These missions are nevertheless interesting for testing our system, as it is fairly easy to determine that the resulting solution is indeed a good one.

SweeperC is considerably more difficult due to the addition of the no-fly zone and the irregularities of the search area. An optimal strategy is not at all obvious from looking at the initial map configuration. Simple back-and-forth flying will not result in the fastest search times, as the agents will have to continually fly over areas that are not part of the search area. This is the most challenging of the three missions.

For all missions, virtual beacons were placed at regular intervals in the search area, representing a UAV sensor width of 500 m. UAVs travel at a constant velocity of 150 km/hr with a fixed turning radius of 250 m. Simulator time was discretized to 1 second intervals. The maximum time allowable, t_{stop}, varied between 25 and 40 minutes of simulator time, depending on the mission. It is worth emphasizing that these virtual beacons are not intended to represent any real object on the ground, but instead are a useful construct by which agents can communicate to each other when a location has been swept.

4. GENETIC PROGRAMMING

The basic concept of GP is closely related to genetic algorithms [5]. A population of random solutions is generated, and each solution is evaluated for fitness. New members of the population are created by performing reproductive operations on individuals with high fitness values. Some of the reproduction operations will result in improved fitness values. Over successive generations, the population is driven towards optimal or near-optimal solutions.

The primary difference between GP and genetic algorithms is the representation of the individual. Instead of representing a scalar or set of numerical values, the individual represents an algorithm. Our individuals take the form popularized by Koza [7], where algorithms are expressed as Lisp-like code strings. The Lisp code implies a decision tree constructed from an initial set of functions and terminals. Reproductive operators combine or mutate the subtrees to create new individuals. In subtree crossover, a randomly-selected subtree of one individual is replaced by a randomly-selected subtree of another individual. In mutation, a randomly-selected subtree is replaced by a randomly-generated subtree of similar depth.

4.1 GP for Cooperative UAV Control

In our case, the algorithm defines a decision process for changing the heading of a UAV. Terminals represent flight vectors and all functions accept flight vectors as arguments. Other forms of GP allow mixed-type operators and terminals, but we have found that using vectors alone meets our current needs.

A simple example of a flight strategy that can be defined by our system is:

```
(neg (closest-enemy))
```

This defines a strategy which states the UAV should fly in the opposite direction of the nearest known hostile agent. A somewhat more complex example is:

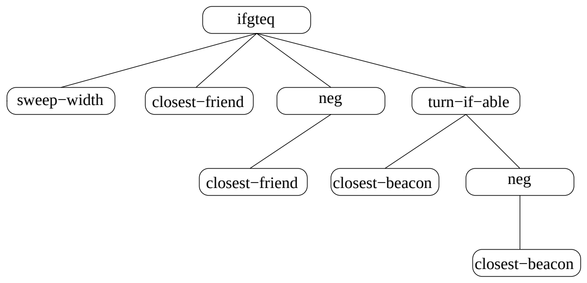

Figure 3: Example strategy as a decision tree

```
(ifgteq (sweep-width) (closest-friend)
  (neg (closest-friend))
  (turn-if-able
    (closest-beacon)
    (neg (closest-beacon))))
```

This is perhaps easier to understand when represented as a decision tree, as in Figure 3. A human can interpret this strategy as: "If the distance to the closest friendly UAV is less than my sweep width, move away from that teammate. Otherwise, if my turning radius makes it possible for me to reach my closest beacon, go towards the closest beacon. Otherwise, turn away from the closest beacon." This illustrates how strategies are capable of producing behaviors that allow the UAVs to spread out and avoid circling a fixed point. A list of available functions and terminals, many adapted from [10], is given in Table 1.

For the current research, all UAVs use the same decision tree. Homogeneous strategies do not allow the creation of explicit specialists [16], but because only one strategy is being evolved, fewer evaluations are required for convergence. Also, the single decision tree is capable of specialized behavior by branching on certain conditions, such as how far one is from the search zone.

4.2 Fitness

Fitness is evaluated in two different ways, depending on if the mission has failed or succeeded. A mission is considered successful if the search area is completely swept when time expires (or when all agents return to the base). Fitness of a successful mission is computed as a weighted average of the time each agent spent in the field and the overall time-to-completion. Formally, this is represented as:

$$\frac{\alpha}{n}\sum_{i=1}^{n} t_i + \beta t_{\max} \tag{1}$$

where n is the number of agents, t_i is the time elapsed when agent i returns to the base, $t_{\max}$ is the maximum of all t_i values, and $\alpha + \beta = 1$. t_i is set to t_{stop} for any agent that has not returned to the base when time expires. This fitness function forces UAVs to work together to minimize both the time that individuals spend in the field as well as the finishing time for the entire team. Fitness of a failed mission is computed as the maximum allowable time plus the proportion of the sweep area left unswept:

$$t_{\text{stop}} + \frac{\text{area}_{\text{unswept}}}{\text{area}_{\text{total}}} \tag{2}$$

Note that by using these equations, the fitness of a failed mission is always greater than a successful mission.

Because the missions can contain stochastic elements, each strategy must be evaluated multiple times to ensure that the fitness value is an accurate predictor of performance. The ultimate fitness value assigned to an individual, i.e. a strategy, is averaged over several attempts at the mission.

4.3 Steady-state GP

There are two predominant types of genetic algorithms. Canonical GAs use a generational approach, meaning that successive generations consist entirely of new individuals created via reproductive operations. The previous population is discarded once the next generation has been created. A long-acknowledged problem with this approach is that it is possible for the individuals in a following generation to be less fit than the generation that preceded it. As a result, many current GA applications use a steady-state algorithm, whereby new individuals are added one at a time to the population if their fitness is suitable. Not only does this keep the best individuals from generation to generation, it also allows future selection operations immediate access to improved individuals. Our GP system is based on Genitor [22], which is a steady-state GA implementation that has been shown to converge to optimal and near-optimal solutions faster than generational approaches for a variety of real-world problems [21].

Although steady-state algorithms are widely used in the GA community, most current GP research uses generational implementations (exceptions include [19] and [4]). For our experiments, we have implemented a variant of Genitor for GP and compare it to the canonical algorithm.

5. EXPERIMENTS

To evaluate our GP approach, strategies were evolved for the three missions described in the previous section. The well-known `GROW` algorithm [7] was used to create initial random solutions. The tree is constructed recursively by uniformly drawing from the set of functions and terminals in Table 1[2]. If a function was chosen, the arguments were chosen by again drawing uniformly from the initial set. The depth of the trees was arbitrarily capped at 10, so the random drawing process was limited to terminals only if the subtree reached depth 9.

Two selection operators were compared: rank-based and tournament. Rank-based selection, described in Appendix 1 of [21], is commonly used in steady-state GA implementations. The bias parameter was fixed at 1.5, which gives relatively low selective pressure. For tournament selection the tournament size was set to 7, which has high selective pressure and is popular in the GP literature.

Trials were performed using population sizes of 50, 100, and 200. Each trial consisted of 16 independent GP runs for each population size. During evolution, new individuals were generated using a crossover rate of 0.7 and a mutation rate of 0.3. In the fitness function, equal weight was given to the mean individual completion time and the maximum completion time ($\alpha = \beta = 0.5$ for Equation 1.)

The crossover operation can lead to trees of exponentially-increasing size, if a root node of one individual is grafted

[2]`closest-enemy` was not used for *SweeperA* because the hostile element does not have a specific location.

Table 1: GP functions and terminals

Name	nArgs	Description
last	0	The direction of this UAV's previous move
closest-beacon	0	Vector from this UAV to the closet unswept patch of the search area
closest-unique-beacon	0	Vector from this UAV to the closet unswept patch of the search area that is not closer to any other UAV. Returns zero vector if no such beacon exists.
closest-friend	0	Vector from this agent to the closest friendly UAV
{right, left}-friend	0	Vector from this agent to the first friendly agent in a clockwise (right) or counter-clockwise (left) sweep
closest-enemy	0	Vector from this agent to the closest non-friendly UAV. In *SweeperC*, it is the shortest vector between the UAV and the no-fly zone.
sweep-north	0	Vector from this UAV to the north-most extent of the current search area.
sweep-{south, east, west}	0	See sweep-north
sweep-width	0	Vector with magnitude equal to the sweep width of the UAV's sensor
unit	1	Normalize the vector to unit length
neg	1	Negate the vector
rot90	1	Rotate the vector by 90 degrees
mul2	1	Double the magnitude of the vector
div2	1	Divide the magnitude of the vector by 2
add	2	Return the sum of both vectors
sub	2	Return the difference between the vectors
turn-if-able	2	If the first argument points to a location that the UAV can fly directly to, return the first argument. Otherwise, return the second argument.
ifzero	3	If the first argument has zero magnitude, return the second argument. Otherwise, return the third argument.
ifgteq	4	If the magnitude of the first argument is greater than or equal to the magnitude of the second, return the third argument. Otherwise, return the fourth argument.
ifdot	4	If the dot product of the first two arguments is greater than or equal to zero, return the third argument. Otherwise, return the fourth argument.

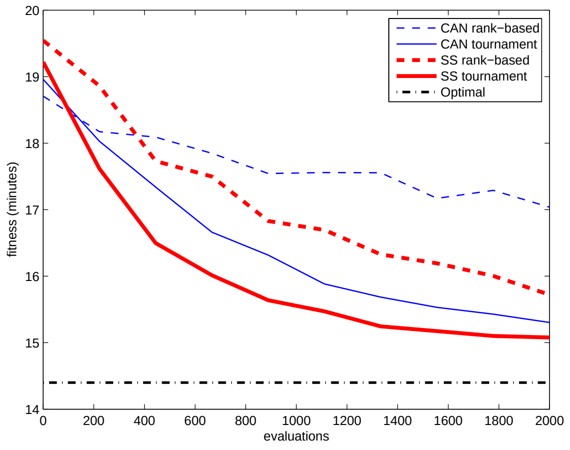

Figure 4: Mean best fitness for *SweeperA* (P=200)

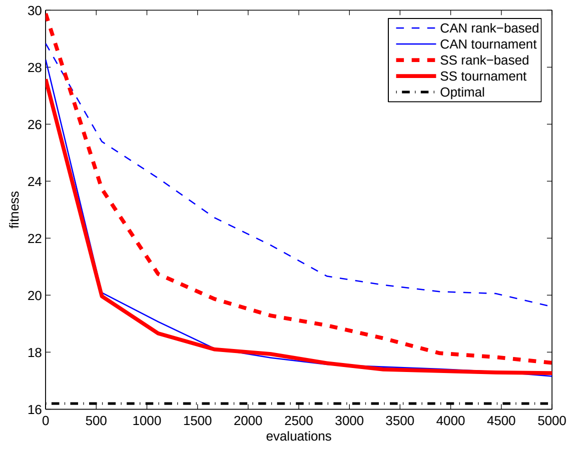

Figure 5: Mean best fitness for *SweeperB* (P=200)

onto the leaf of another. To prevent this so-called "code bloat", any newly-created individual whose tree depth was greater than 10 was replaced with a randomly-generated individual. While this is a somewhat simplistic approach to dealing with code bloat, it has the desirable property of maintaining diversity by occasionally introducing new individuals into the population. A detailed examination of code bloat can be found in [9].

5.1 Steady-state vs. the Canonical Algorithm

Experiments for *SweeperA* were run for 2,000 evaluations each. For population sizes of 50, 100, and 200, this translates to 40, 20, and 10 generations, respectively. While 10 generations may seem unfairly short, it is the same number of new individual evaluations as the steady-state implementation, so similar running times are expected. Independent pairwise t-tests ($p = 0.05$) were used to compare the means of the best individual after evaluation 2,000. Differences due to population size were not found to be significant.

Figure 4 shows results for population size of 200. There was no significant difference between the top three performers. The canonical algorithm using rank-based selection was significantly worse than both steady-state algorithms and the canonical implementation using tournament selection. This was also true for populations of size 100 and 50.

Due to the stochastic adversarial component, it is not possible to determine the best possible fitness score that can be achieved. However, the dotted line in Figure 4 shows the theoretical optimum fitness score *if no UAVs are destroyed*. That value is based on the minimum distance that the UAVs must fly from their starting position to cover the sweep area and return to base. The fitness values for the evolved strategies approach this theoretical optimum under ideal conditions, even though UAVs have been lost. This shows that it is possible to evolve strategies that will perform very close to optimal, even in uncertain conditions.

Figure 5 shows similar results for *SweeperB*. A total of 5,000 evaluations were allowed, as this mission is more difficult. The canonical algorithm using rank-based selection is again significantly worse than the other three implementations. The two algorithms using tournament tournament selection appear to initially outperform the other imple-

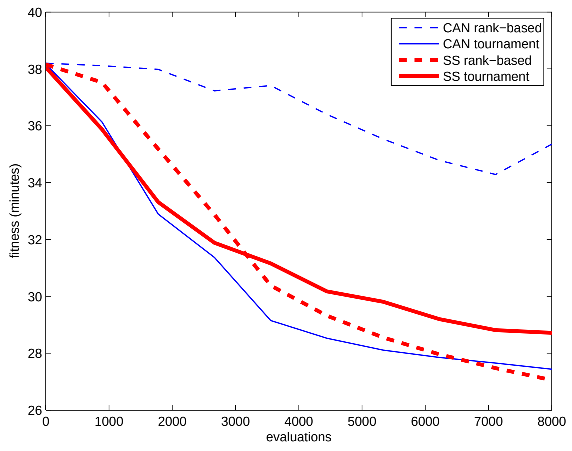

Figure 6: Mean best fitness for *SweeperC* (P=200)

mentations, although the results after 5,000 evaluations are not significantly better than the steady-state algorithm with rank-based selection. As with *SweeperA*, the best results are close to the optimal fitness value under ideal circumstances.

Figure 6 shows the results for *SweeperC* for 8,000 evaluations. Again we find that the canonical algorithm using rank-based selection has been outperformed by the other implementations, this time dramatically so. The theoretical optimal fitness is not shown for Figure 6 because the irregular geometry of the search area makes calculating the optimal fitness a significant challenge unto itself. The minimum amount of time required to complete the mission is about 20 (simulated) minutes, although the final fitness score could be somewhat lower than this. The best fitness variance was relatively high compared to the previous two missions, with values ranging from 21.16 to 29.08 minutes for population size 200.

5.2 Observations

The evolved strategies resulted in behaviors that appear cooperative. UAVs spread out to minimize redundant flight over previously-swept areas. Collisions between UAVs are non-existent in the final evolved behaviors. When UAVs are lost due to the adversaries in the field, the remaining UAVs effectively fill the resulting gaps in the search space.

In *SweeperB*, flight patterns will change when the hostile agent is discovered. UAVs will fly as close as possible to the hostile agent without entering its firing range. The UAVs have evolved an avoidance instinct that successfully keeps them out harm's way once the threat is located.

The strategies that evolved were found to be fairly robust to changes in the starting parameters of the mission. For example, changing the initial positions of the UAVs had no noticeable negative impact on their cooperation and sweeping ability. Even though the UAVs were following a strategy evolved for different initial conditions, they were still able to complete the mission effectively. This can be explained by the fact that the primitives deal entirely with relative positions and spacial information that is local to each individual UAV, which allows the strategy to be flexible in the face of global changes to the mission parameters. Similar results were observed when changing the initial number of UAVs and making small changes in the size of the search area.

This flexibility can also be used to speed up the evolutionary process. For example, *SweeperB* is simply a harder version of *SweeperA*. Instead of starting the evolutionary process for *SweeperB* from scratch, we can seed the population with some of the best individuals found for *SweeperA*. So rather than having an initial population of completely random individuals, the population is bootstrapped with a few decent (although probably not great) individuals. Preliminary results indicate that this will help reduce the time required for evolution. Exactly how beneficial this can be remains to be seen.

6. CONCLUSIONS AND FUTURE WORK

Our results indicate that GP can be an effective technique for generating efficient strategies for cooperative UAV control in simulation. The best evolved strategies approach the theoretical optimum fitness bound, and will therefore perform competitively against other approaches. Although the experiments presented were specific to a coordinated search task with constant probability threats and fixed location threats only, we believe the concepts can be easily extended to work with a broad variety of multi-UAV missions.

Empirical evidence indicates that there is not a significant difference between generational and steady-state algorithms for GP in this domain, provided an appropriate selection operator is used. It is clear, however, that rank-based selection (bias $= 1.5$) does not work well with the generational algorithm. It is likely that the best individuals are not being selected often enough to effectively drive the evolutionary process. In contrast, the steady-state GP quickly removes poor solutions from the population, so evolution is successful even with relatively low selective pressure. Further study is required to determine how these findings apply to GP in general.

Further exploration of the GP parameter space could reveal more information about the optimal settings for this type of problem. Due to the significant run-time requirements of the simulator, we had to fix many parameter settings (such as mutation rate, use of elitism, tournament size, etc). In future work, we would like to be able to examine the effects of these decisions more closely.

7. ACKNOWLEDGMENTS

This work was supported by a grant from Raytheon. D. Whitley was also supported by the National Science Foundation under Grant No. 0117209.

8. ADDITIONAL AUTHORS

Additional authors: Todd Mytkowicz (Department of Computer Science, University of Colorado); Duong Nguyen and David Rome (Raytheon/IIS/Space Systems, 16800 E. CentreTech Parkway, DN, Bldg 77 M/S 3026, Aurora, Colorado 80011-9046).

9. REFERENCES

[1] V. Ablavsky, D. Stouch, and M. Snorrason. Search path optimization for UAVs using stochastic sampling with abstract pattern descriptors. In *Proceedings of the AIAA Guidance Navigation and Control Conference*, Austin, TX, August 2003.

[2] T. Balch and R. C. Arkin. Behavior-based formation control for multi-robot teams. In *IEEE Transactions on Robotics and Automation*, 1999.

[3] G. J. Barlow, C. K. Oh, and E. Grant. Incremental evolution of autonomous controllers for unmanned aerial vehicles using multi-objective genetic programming. In *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems (CIS)*, Singapore, December 2004.

[4] C. Clack and T. Yu. Performance-enhanced genetic programming. In *Proceedings of the 6th International Conference on Evolutionary Programming VI*, pages 87–100. Springer-Verlag, 1997.

[5] J. H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.

[6] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *Proc. of the Third European Conference on Artificial Life (ECAL'95)*, pages 704–720, Granada, Spain, 1995.

[7] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, MA, 1992.

[8] S. Luke. Genetic programming produced competitive soccer softbot teams for RoboCup97. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 214–222, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann.

[9] S. Luke. *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat*. PhD thesis, Department of Computer Science, University of Maryland, 2000.

[10] S. Luke and L. Spector. Evolving teamwork and coordination with genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 150–156, Stanford University, CA, USA, 28–31 1996. MIT Press.

[11] M. Mataric. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems*, 16:321–331, December 1995.

[12] F. W. Moore. A methodology for missile countermeasures optimization under uncertainty. *Evol. Comput.*, 10(2):129–149, 2002.

[13] H. V. D. Parunak, M. Purcell, and R. O'Connell. Digital pheromones for autonomous coordination of swarming UAVs. In *In Proceedings of First AIAA Unmanned Aerospace Vehicles, Systems, Technologies, and Operations Conference*, 2002.

[14] P. Pirjanian and M. J. Mataric. Multi-robot target acquisition using multiple objective behavior coordination. In *International Conference on Robotics and Automation (ICRA 2000)*, San Francisco, CA, April 2000.

[15] M. M. Polycarpou, Y. Yang, and K. M. Passino. A cooperative search framework for distributed agents. In *Proceedings of the 2001 IEEE International Symposium on Intelligent Control*, Mexico City, Mexico, 2001.

[16] M. A. Potter, L. Meeden, and A. C. Schultz. Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 1337–1343, Seattle, Washington, August 2001.

[17] D. Rathbun, S. Kragelund, A. Pongpunwattana, and B. Capozzi. An evolution based path planning algorithm for autonomous motion of a UAV through uncertain environments. In *AIAA 21st Digital Avionics Systems Conference*, Irvine, CA, October 2003.

[18] S. Rathinam, M. Zennaro, T. Mak, and R. Sengupta. An architecture for UAV team control. In *IAV2004: Fifth IFAC symposium on intelligent autonomous vehicles*, Lisbon, Portugal, 2004.

[19] L. Spector, H. Barnum, H. J. Bernstein, and N. Swami. Finding a better-than-classical quantum AND/OR algorithm using genetic programming. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 2239–2246, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.

[20] P. Vincent and I. Rubin. A framework and analysis for cooperative search using UAV swarms. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 79–86. ACM Press, 2004.

[21] D. Whitley. The genitor algorithm and selective pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 116–121, 1989.

[22] D. Whitley and J. Kauth. Genitor: a different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, pages 118–130, Denver, CO, 1988.