Competition, Coevolution and the Game of Tag

Craig W. Reynolds
Electronic Arts
1450 Fashion Island Boulevard
San Mateo, CA 94404 USA
telephone: 415-513-7442, fax: 415-571-1893
creynolds@ea.com
cwr@red.com

Abstract

Tag is a children's game based on symmetrical pursuit and evasion. In the experiments described here, control programs for mobile agents (simulated vehicles) are evolved based on their skill at the game of tag. A player's fitness is determined by how well it performs when placed in competition with several opponents chosen randomly from the coevolving population of players. In the beginning, the quality of play is very poor. Then slightly better strategies begin to exploit the weaknesses of others. Through evolution, guided by competitive fitness, increasingly better strategies emerge over time.

1. Introduction

Many of us remember playing the *game of tag* as children. Tag is played by two or more, one of whom is designated as *it*. The *it* player chases the others, who all try to escape. Tag is a simple contest of pursuit and evasion. These activities are common in the natural world, most predator-prey interactions involve pursuit and evasion. Tag also includes an aspect of role-reversal, both pursuit and evasion skills are required. *It*'s goal is to catch up with another player, to get close enough to reach out and touch the other player. At this point the pursuer shouts "Tag! You're *it*!" and the former evader becomes the new pursuer.

The game of tag serves here as a toy example, to study the use of competitive fitness in the evolution of agent behavior. Tag is intended as a simple model of behavior based on control of locomotion direction, or *steering*. By evolving a vehicular steering controller for the game of tag we have a test case to learn about evolving controllers for related, but more complex tasks.

We seek to automatically discover a controller through evolution based solely on competition between controllers. This approach stands in contrast to evolving controllers by pitting them against a static, predetermined expert strategy. The use of *competitive fitness* has the significant advantage of avoiding the paradoxical need for an expert controller as a prerequisite for evolving an expert controller. Another advantage of competitive fitness is that since each fitness test is unique, there is no danger of overfitting a static expert.

2. Related Work

This research was originally inspired by Pete Angeline's elegant work on coevolution of players for the game of Tic Tac Toe, using competitive fitness [Angeline 1993]. That fitness could be tested by competition alone, even in the absence of an expert player, is a key insight in applying evolutionary techniques to the discovery of complex goal-directed behaviors. The work reported here extends Angeline's paradigm from games of pure strategy to those involving geometric motion. The competition in [Angeline 1993] was in the form of a single elimination tournament tree. The players in each new generation were paired up in competition. The winners of each pairing went on to a second round, and so on for several rounds until only one champion remained. A player's fitness was determined by the number of matches won before a defeat. See Table 1 for a comparison of several competitive architectures described in this section.

A similar approach has recently been used to coevolve strategies for the game of Othello [Smith 1994]. A genetic algorithm was used with competitive fitness to determine time-varying weightings for the static evaluator used in an alpha-beta search of the Othello game tree. In this work, two new players are placed in competition with each other, the score of their game determines the fitness of both.

Another paper on competitive evolution of behavior appears elsewhere in this volume [Sims 1994]. It describes experiments where the morphology and behavior of artificial creatures evolve through competition for control of an inanimate object. The creatures each try to get closest to the object, and if possible to surround it. Each new creature

Table 1:

competitive architecture	matches per generation of n	opponents per individual	reference
new versus all	$(n^2-n)/2$	n-1	[Koza 1992]
new versus several	nk	k	this paper
single elimination tournament tree	n-1	$\log_2 n$	[Angeline 1993]
new versus previous best	n	1	[Sims 1994]
new versus new	n/2	1	[Smith 1994]

To appear in Artificial Life IV.

is placed in competition with the best creature from the previous generation.

John Koza evolved pursuer-evader systems closely related to those reported here (see pages 423-428 of [Koza 1992]). But in that work the pursuers and evaders existed in separate populations. Their fitness was determined by comparison with a pre-existing optimal player. In the same book, Koza first discusses coevolution in Genetic Programming (pages 429-437) in the context of a discrete strategy game.

In the work reported here, the vehicle model, and the noise-tolerant Steady-State Genetic Programming system was taken from [Reynolds 1994a] and [Reynolds 1994c]. The vehicle model draws heavily from [Braitenberg 1984] and is equivalent to the *turtle* of the LOGO programming language.

Competitive fitness and coevolution were first explored in evolutionary computation in the context of the Iterated Prisoner's Dilemma in [Axelrod 1984], [Axelrod 1989], [Miller 1989], [Lindgren 1992] and [Lindgren 1994]. A restricted form of competitive fitness was used in [Hillis 1992] to drive the evolution of sorting networks: antagonistic test cases were coevolved to force generality. Coevolution and competitive fitness are fundamental to a wide class of ecological simulations studied in Artificial Life such as: ECHO [Holland 1992], Tierra [Ray 1992], BioLand [Werner 1993], PolyWorld [Yeager 1994], LEE [Menczer 1994], and the work reported in [D'haeseleer 1994]. Chapter six of [Kauffman 1993] is an authoritative analysis of "The Dynamics of Coevolving Systems."

The reader should understand that this work is *not* about optimal control theory, despite the fact that frequent mention is made of optimal strategies, and that this type of pursuer-evader system has a long history in the optimal control literature [Isaacs 1965]. The work reported here benefits from (but does not depend upon) the ability to compare evolved behavior with optimal behavior. It should be noted that this is practical only because of the extremely simple vehicle model used in these studies. Once slightly more complicated models are used, optimal behavior becomes a much more complicated issue, see for example [Merz 1971].

3. Experimental Design

In this work, Genetic Programming is used to evolve control programs for simulated vehicles, based on their ability to play the game of tag. Each new player's fitness is determined through competition with existing players. The game of tag provides a framework wherein the relative fitness of two players is judged through direct competition.

The vehicles are abstract autonomous agents, moving at constant speed on a two dimensional surface. Each vehicle's evolved control program is executed once per simulation step. Its job is to inspect the environment and to compute a steering angle for the vehicle. Angles are specified in units of *revolutions*, a normalized angle measure: 1 revolution equals 360 degrees.

For each player, at each simulation step: (1) its control program is run to determine a steering angle, (2) the vehicle's heading is altered by this angle, (3) the vehicle is moved a fixed distance along its new heading, and (4) tags are detected and handled. The forward step length is typically 125% longer for *it*. The per-step steering angle is unconstrained: these vehicle can instantaneously turn by any amount. This is an abstract *kinematic* vehicle model, as opposed to a *physically realistic* model. In this work there is no simulation of force, mass, acceleration, or momentum. In contrast, a physically realistic model would serve to limit the turning radius.

In these experiments there are always two players in a tag game. The playing field is featureless. Each vehicle's environment consists of just one object: the opponent vehicle. The opponent's current heading is not relevant. In the absence of momentum, there is no correlation between the vehicle's current heading and its position in the next time step. For a given controller the entire state of the world consists of: a flag indicating who is *it*, and the relative position of the opponent's vehicle. The position information is presented to the control program in terms of x and y coordinates of the opponent, expressed in the local coordinate system of the controller's own vehicle. A vehicle's local coordinate system is defined with the positive y axis aligned with the vehicle's heading and the positive x axis extending to the vehicle's right. See Figure 1.

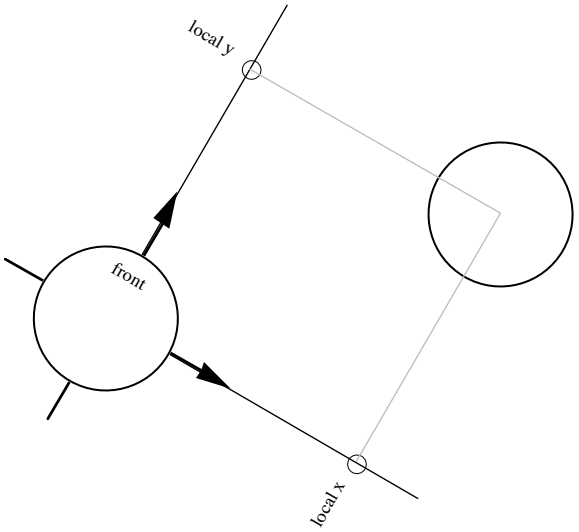

Figure 1: the input to a vehicle's controller is the position of the opponent vehicle, expressed in terms of local x and y coordinates.

While the traditional game of tag has no formal method of scoring, the intuitive goal is to avoid being *it*. Therefore, fitness is defined here to be the portion of time (simulation steps) spent not being *it*. Note that this is a normalized, bigger-is-better fitness metric: zero is worst, one is best. To determine a player's fitness it competes with other players.

To compare two players, a series of four games is played. Before each game the players are given random initial headings and are randomly positioned within a starting box measuring about 3.5 vehicle-body-lengths on a side. The two players alternate starting as *it* for each game of the

series. If *it* tags the opponent, by getting to within one vehicle length, the *it* and non-*it* roles are reversed. Each game consisted of 25 simulation steps. A player's score for a game is the number-of-non-*it*-steps divided by 25. See Figure 2.

To determine a player's fitness, it is pitted against 6 other players. These opponents are chosen by uniform random selection from the existing population. Scores from these 24 games (a series of 4 games against each of 6 opponents) are averaged together to obtain the final fitness value. A value of 50% indicates that the player has an ability comparable to the population average. Fitness values above 50% indicate increasingly better players. Because opponents are randomly selected, and because initial conditions are randomized, these fitness values have significant variance and provide only a rough estimate of a player's actual fitness.

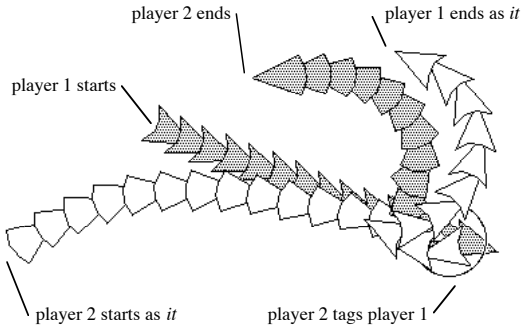

Figure 2: a game of tag between player 1: an evolved controller from run G, and player 2: the optimal strategy. As in all such diagrams in this paper, the pursuer (*it*) is shown in white, the evader is gray, and the site of a tag is indicated by a circle. Player 1 is shown here with a concave back edge and player 2 has a convex back edge. In this game player 1 begins by fleeing in a naive direction. Player 2 chases down and tags player 1. Player 2 then escapes because player 1 is too slow turning around. Player 1 was not it for 14 of the 25 steps so earned a score of 56%. Player 2 was not *it* for 11 steps so scored 44%.

In the discussion of experimental results below, reference will be made to an *optimal player*. Because of the unrealistically simple vehicle model used here, particularly the absence of momentum, the optimal strategy is quite straightforward. For the pursuer, the optimal strategy is to turn toward the evader (so the evader lies on the pursuer's positive y axis). Similarly the optimal strategy for the evader is to turn directly away from the pursuer (so that the pursuer lies on the negative y axis).

The "four quadrant arctangent" implements this optimal strategy. However the fitness criterion in these experiments is performance in tag competition. The optimality of a given controller is irrelevant to its fitness, all that matters is its performance in tag competition with the coevolving population. As will be seen below, controllers which are clearly non-optimal can still produce behavior which is asymptotically close to optimal performance.

4. Genetic Programming and Tag

The technique used to evolve computer programs in this work is known as Genetic Programming ("GP"), invented by John Koza. The best reference on this technique and its application is [Koza 1992]. In its original formulation, GP operated on a population of individuals generation by generation. Alternatively, GP can be combined with Steady State Genetic Algorithms [Syswerda 1991] as described in [Reynolds 1993a].

A very brief description of Steady State Genetic Programming (SSGP) follows. First a population of random programs is created and tested for fitness. In these experiments the population consisted of 1000 to 5000 programs. Thereafter SSGP proceeds by: (1) choosing two *parent* programs from the population, (2) creating a new *offspring* program from them, (3) testing the fitness of the new program as described in the previous section, (4) choosing a program to remove from the population to make room, and (5) adding the new program into the population. The parent programs are chosen in a way that favors the more fit while not totally ignoring the less fit, thus balancing *exploration* of the whole gene pool with *exploitation* of the current champions. In these experiments this choosing is done using *tournament selection*: seven individuals are chosen from the population at random, the most fit of those is selected as the winner. The recombination of two parents to form a new offspring is accomplished by the Genetic Programming *crossover* operator. GP crossover is a bit like "random cut and paste" done on balanced parenthetical expressions to guarantee the new program's syntactic correctness. After crossover, a program might be *mutated* by substituting one function for another, one terminal for another, or by crossover with a new random program fragment.

Selecting a program to remove from the population could be done by using inverse tournament selection: removing

Table 2

function	usage	description	source
`+`	`(+ a b)`	a plus b	Common Lisp
`-`	`(- a b)`	a minus b	Common Lisp
`*`	`(* a b)`	a times b	Common Lisp
`%`	`(% a b)`	if b=0 then 1 else a divided by b	[Koza 1992]
`min`	`(min a b)`	if a<b then a else b	Common Lisp
`max`	`(max a b)`	if a>b then a else b	Common Lisp
`abs`	`(abs a)`	absolute value of a	Common Lisp
`iflte`	`(iflte a b c d)`	if a ≤ b then c else d	[Koza 1992]
`if-it`	`(if-it a b)`	if this player is *it* then a else b	this paper

the least fit of seven randomly chosen programs. However the greedy nature of SSGP, combined with the variance of fitness measures used in these experiments, leads to the possibility of a mediocre-but-lucky program receiving an undeservedly high fitness and going on to dominate the population. To combat this possibility, a modified removal policy was used in these experiments: half the time inverse tournament selection was used, the other half of the time an individual was selected for removal at random (without regard to fitness). All programs, even the best one, has a certain small but non-zero probability of being removed at each SSGP step. This approach reduces the possibility that the population could stagnate with a collection of mediocre-but-lucky programs. To survive, winning strategies must continue to perform well.

Another issue is that competitive fitness values are measured relative to the population at a certain point in time. These fitness value becomes less and less relevant as time passes. The hybrid SSGP removal policy ensures that the population is continually being recycled.

Because steady state genetic computation proceeds individual by individual, there is no demarcation of generations. However it is often convenient to describe the progress or length of a SSGP run in terms of "generation equivalents:" processing as many new individuals as there are programs in the population.

In order to evolve tag-playing control programs with Genetic Programming, we first choose a set of functions and terminals to form the language in which they will be expressed. Certain choices are dictated by the application itself, and some are intended to provide the logical and arithmetic "glue" with which GP will construct control programs. Three functions used here are specific to the underlying application, they provide the three parameters for the control programs. `local-x` and `local-y` are functions of zero arguments which return the x and y coordinates of the opponent player. The `if-it` macro provides conditional execution of its two subexpressions depending on whether this player is currently *it*. In addition to these functions, several others are provided to help form control programs. The choice is rather arbitrary, it was influenced by the author's domain knowledge and by preliminary attempts to write a tag-playing control program by hand. This function set provides for arithmetic, thresholding, sign manipulation, and conditional computation, see Table 2.

In addition to `(local-x)` and `(local-y)` mentioned above, the terminal set includes `:random-0-1`, an ephemeral random constant. After a new program is formed, any occurrences of this terminal are replaced by a pseudo-random floating point number between 0.0 and 1.0. Evolved programs are subject to a size limitation which is measured in term of the total number of functions and/or terminals. When crossover produces a program whose size exceeds this limit, the *hoist* genetic operator [Kinnear 1994] is used to find a smaller (but hopefully still fit) subexpression. In these experiments the size limit was either 50 or 100.

These experiments used the `if-it` conditional in two different ways. In the early runs, a *syntactic constraint* was used to ensure that evolved programs always contained exactly one occurrence of `if-it` and it was always at the top (outermost) level of the program. That is, all programs in those runs had this structure:

`(if-it` <*pursuer-branch*> <*evader-branch*> `)`

The GP crossover operation was modified to exchange code fragments only within branches of the same type. With this constrained structure and crossover, the effect is to create two distinct gene pools, one for pursuit behavior and one for evasion behavior. Thus the frequency of code fragments are allowed to evolve differently in each population. This approach is similar to the Automatically Defined Functions described in [Koza 1992].

In later experiments this syntactic constraint was removed and the `if-it` conditional was treated as just another non-terminal. As a result there could be any number of `if-it` forms in an evolved program. There was no segregation of the gene pool, code fragments could cross over from the pursuit branch to the evasion branch, and vice-versa.

The Genetic Programming substrate used here was originally developed by the author on Symbolics Lisp Machines and subsequently ported to Macintosh Common Lisp (version 2.0p2). These experiments were run on Macintosh Quadra 950 workstations. In this implementation a fitness test consisting of 24 tag games takes 7 to 12 seconds to run, depending on program size.

5. Results

Seven evolution runs were performed using the basic experimental design described above. The runs varied in terms of population, the relative speed of the two players, the program size limit, whether mutation was used, whether the *it*/not-*it* branches were segregated, and the method used for selecting opponents. See Table 3. Runs A, C and G will be discussed in more detail in the following sections.

5.1 Run A

Run A had a population of 5000 individuals. Both players moved at the same speed. This puts *it* at a fundamental

Table 3:

run name	population size	*it* speed ratio	program size limit	mutation used	segregated branches	opponent selection
A	5000	1.00	50	no	yes	uniform
B	2000	1.25	50	no	yes	uniform
C	1000	1.25	50	yes	yes	uniform
D	1000	1.25	50	yes	yes	tournament
E	1000	1.25	50	yes	no	uniform
F	1000	1.25	50	no	no	uniform
G	1000	1.25	100	no	no	uniform

disadvantage: as long as the other player moves away, the best *it* can do is to follow at a constant distance, unable to close the gap.

Most of the initial, random programs in run A used ineffective strategies. A typical behavior was to bumble around aimlessly. Some of the early programs effectively had no steering behavior. These control program always returned zero (or some very small value) and the vehicle would simply travel in a straight line. While this is not a very effective pursuit strategy (for *it*), it is occasionally a good evasion strategy. If *it* happened to start off positioned behind you, running straight ahead is essentially optimal evasion. Since initial position and orientation is random, this non-turning evasion behavior would be effective between one-half and one-quarter of the time, depending on how good *it*s pursuit strategy is. As a result, early in the run these non-turners began to proliferate through the population of evasion strategies.

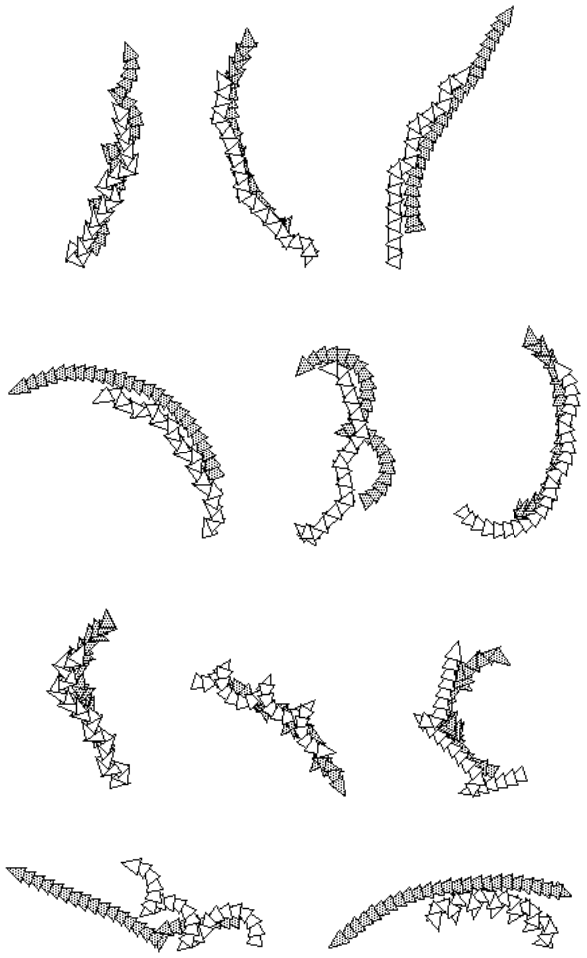

Figure 3: pursuit and evasion from run A in which both players moved at the same speed.

At this early stage, the most effective pursuit strategies appear to have been looping (constant steering angle) and "stumblers" that seemed to move erratically, but managed to creep slowly towards their target. The looping behavior was successful because it allowed the pursuer to cover more ground, increasing its chance of blindly tagging the evader. The stumblers, while quite inefficient when compared to optimal pursuit behavior, were able to survive by *preying* on incompetent, aimless evaders. The large number of bad strategies in the population provided fertile ground for innovation: new strategies, even inefficient ones, could prosper by exploiting the weaknesses of others.

Later in run A, an improved evasion strategy appeared. The gist of it was: if the pursuer is behind you, go straight ahead, otherwise turn randomly. The effect is that the evader would twitch randomly for a few steps, then it would find itself pointing away from the pursuer and head straight away. Not only is this a fairly robust strategy, but it was easily implemented (hence easily evolved) in the programming language used in these experiments:

`(if-it <`*pursuer-branch*`> (max 0 (local-y)))`

This code fragment, and many variations of it, appeared in the population. Most of the variations were larger expressions which were functionally equivalent. Once this simple but effective evader appeared, the pursuers were in trouble. Because of the advantage mentioned earlier, that the evader need only move away from the pursuer in order to win, the pursuers could only achieve mediocre performance. In general the quality of pursuit in run A improved. The pursuers got to be very good at picking off the easy targets, the inefficient evaders. But this only served to improve the gene pool of the evaders. Soon only fairly good evaders remained, and they could easily escape from the best of the pursuers.

This sounds like it might have been the end of the story, but something interesting happened. In the interval between individuals 57000 and 76000, the pursuers developed several different techniques for following their targets, each with a characteristic and visually distinct pattern of motion (see Figure 3). Each of these patterns of motion had to have a certain measure of success to survive, but none could overcome the built-in advantage of the evaders in run A. There is a temptation (unsupported by any data) to see these distinct patterns of motion as something akin to evolutionary stable strategies (or perhaps species in environmental niches). It is also possible that these classes of motion arose solely as artifacts of the function set used in the GP representation.

One pursuit strategy seen in this run (and others) used a competent but inefficient "three phase" technique. These players would always do one of three things: turn left, turn right, or turn around. It is inefficient because turning left or turning right is a good idea only if the quarry is off to the side. If your opponent is directly ahead, a zig-zag path will slow your rate of progress. The code below is the pursuit branch of individual number 179120, unedited except to add comments:

```
(iflte 0.029628         ; if y > 0  (quarry ahead)
       (local-y)        ;
       (iflte 0.212021  ;   if x > 0.2  (quarry on right)
              (local-x) ;
              0.862946  ;      turn 49 degrees to right
              0.134561) ;      turn 48 degrees to left
       0.541760)        ;   turn 195 degrees (about face)
```

As evolution in run A proceeded, the evaders developed an unexpected behavior. For the majority of them life was easy, they could usually escape the pursuers. (Those who could not were quickly killed off.) One might guess that the evader population would settle down into using a simple, efficient strategy. Instead it seems that they had an overabundance of genetic material and were determined to use it! They adopted a complicated, elaborate strategy. Because both players move at the same speed, the pursuer is not a threat until it moves close to the evader. This defines a certain "threat radius" around the evader. If the pursuer is outside this radius it doesn't matter what the evader does, it could just as well bumble around aimlessly. Once the pursuer crosses the threat radius the evader must snap to attention and run away efficiently. These elaborate evaders are very large programs and hard to analyze in detail. One possible explanation is that an effective evasion strategy was discovered, but its implementation depended on the values of local-x and local-y being limited to certain small values. The result was an evader that would flee very efficiently when closely pursued, but otherwise would appear to randomly flip around. Subjectively this behavior looked for all the world like the evader was "saving its strength" -- moving slowly until the pursuer got close enough to be a threat. Yet this could not have been what actually happened, since there was no modeling of expended energy, nor did the fitness function reward conservation of energy.

A note about the selection of games shown in, for example, Figure 3. The choice of images for this paper was a biased, subjective process. An unedited random sampling of images from the run might have been more representative, but would have been much less visually interesting. These "photogenic" images were chosen because of their clarity, simplicity, and visual appeal. Several were typical, a few were unusual. Collecting these images was a process somewhat like nature photography. Lots of pictures were taken, many were discarded, and a few were selected that captured the spirit of the behavior.

5.2 Run C

Evolution of pursuit in run A was stymied by the evader's advantage. Run C sought to even out the competition by giving *it* a 25% speed advantage. Now a good pursuer could close in on and tag a good evader. This served to encourage efficient pursuit, which in turn encouraged improved evasion.

Run C used a population of 1000. Mutation was added in an attempt to help prevent the loss of diversity observed in earlier runs. Run C was evolved for 215 generation equivalents. The quality of pursuit and evasion improved steadily, becoming skillful and well matched. Many games consisted of a chase featuring near-optimal pursuit and evasion followed by a series of rapid tags with *it* alternating back and forth each step. After each tag the new *it* would just turn around, take one big step, and tag the other player. This was reminiscent of the games seen when two optimal players were pitted against each other.

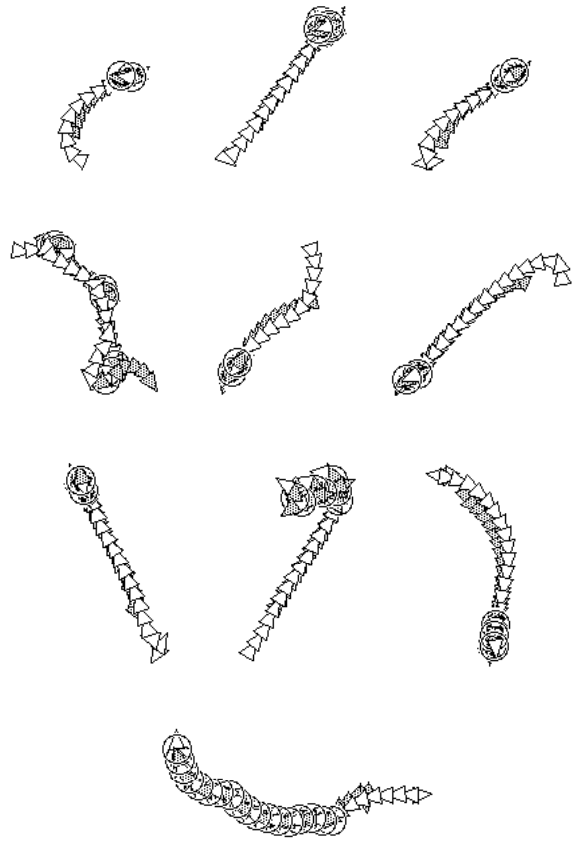

Figure 4: tag games from run C. Most consist of a successful chase followed by a series of tags.

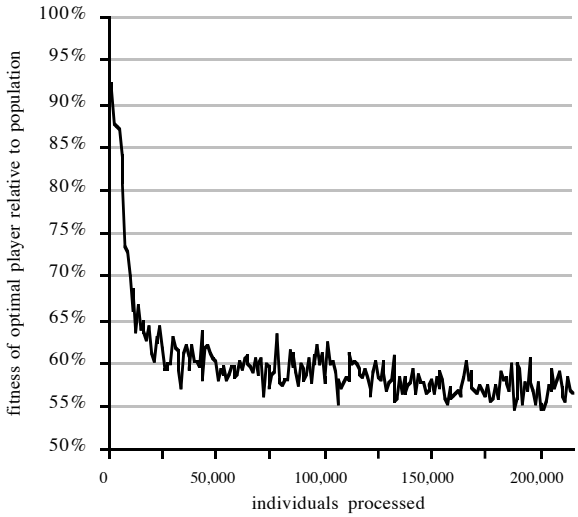

Figure 5: plot of the fitness of the optimal player placed in competition with the evolving population of run C. Since fitness is measured relative to the existing population, the decline in fitness shown here is indicative of the increasing average fitness of the population. Initially the optimal player avoided being *it* 93% of the time. After 22000 individuals (22 generation equivalents), the optimal player is *it* about 60% of the time. Thereafter the value drifts slowly down into the range between 55 and 60%.

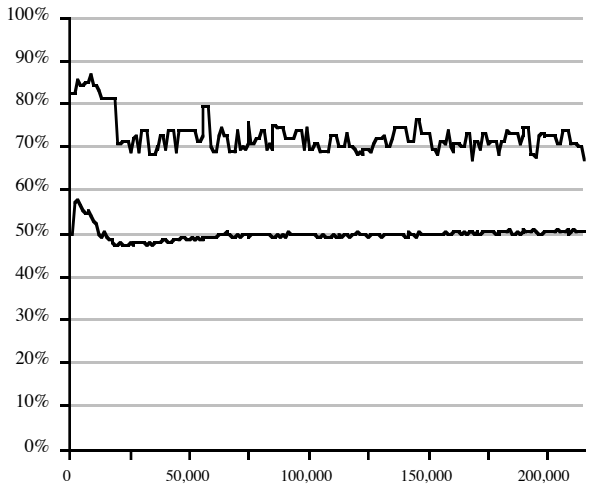

Figure 6: plots of best-of-population fitness (top) and average-of-population fitness (bottom) for 215 generations of run C.

After 215415 individuals were processed in run C there were four individuals with the same best fitness value. One of those was selected for further analysis. It was compared to the optimal player in a series of 100 games. The evolved player got a score of 49.3% indicating that was holding its own against the optimal player, being not-*it* just less than half of the time. The evolved program's size was 48. A few simplifications were made by hand to produce this size 38 version:

```
(if-it (max (% (local-y)
               (+ (min (local-y)
                       (min (abs (local-y))
                            (iflte 0.25235322
                                   (local-y)
                                   (* 0.09556236
                                      (local-x))
                                   (local-x))))
                  (local-y)))
           0.25235322)
       (iflte (local-y)
              0.0055459295
              (iflte (local-y)
                     (* (local-y)
                        (min (* (local-x) (local-x))
                             0.47677472))
                     (* 0.051421385 (local-x))
                     (* 0.107852586 (local-x)))
              0.47677472))
```

To visualize the overall behavior of a player's evolved control program, the two inputs can be used to define a perceptual field embedded in the local space surrounding the vehicle. Mapping the control program over a mesh of points in this field yields an array of steering angles. This map describes the stimulus-response relationship implemented by the control program as it maps an opponent's position into a steering angle.

Figure 7 shows plots of this relationship for the optimal player. Each arrow in this diagram indicates the player's steering angle response to an opponent in the vicinity of the arrow's tail. Figure 9 summarizes the behavioral mapping for an evolved player from run C. This useful visualization tool has long been applied to the study of reactive robotic control system in the work of Ronald Arkin, see for example [Arkin 1987].

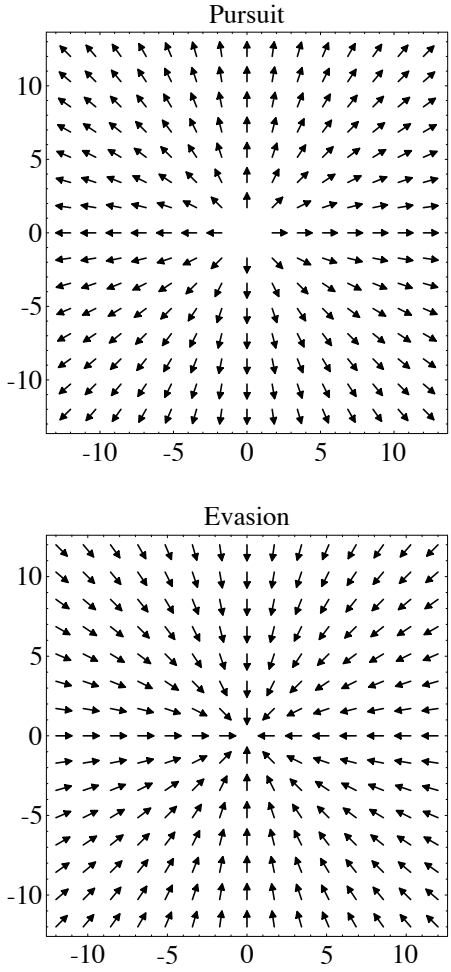

Figure 7: Plots of steering angle versus opponent position for the optimal player. These stimulus-response diagrams summarize the player's behavior.

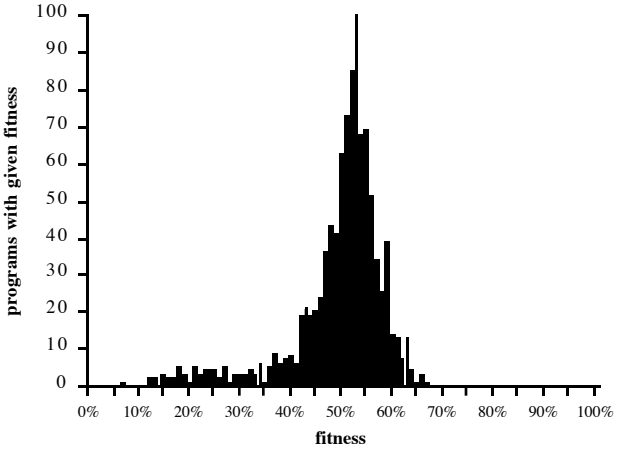

Figure 8: histogram of fitness distribution in population of run C after 215 generation equivalents.

An examination of Figure 9 suggests that a "foveal region" has developed in the perceptual field of both pursuer and evader. In the pursuer the forward facing quadrant has a near-optimal structure. In the evader the backward facing quadrant (particularly in the region close to the player and along its negative y axis) has a near-optimal structure.

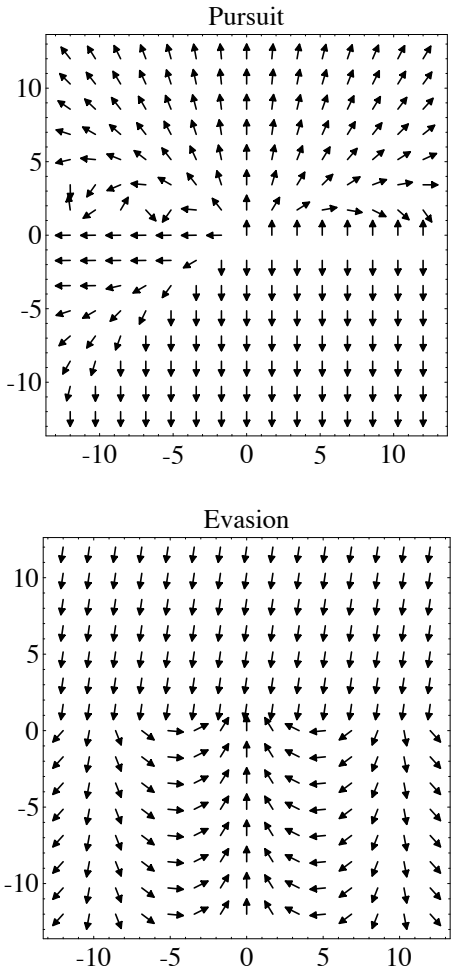

Figure 9: Plots of steering angle versus opponent position for the best-of-population player from run C after 215 generation equivalents

Figure 9 also indicates that in the "anti-foveal" direction the response is simply to turn halfway around. The pursuer will turn around if the evader is behind it, and the evader will turn around if the pursuer is in front of it. The general strategy appears to be to turn "quickly" toward (or away from) the opponent, to place them in your foveal region, wherein your response is close to optimal. The behavior is less organized in the direction perpendicular to the foveal axis, but the effect seems to be the same: after a step or two the vehicle is heading in the right direction.

Since the vast majority of the player's time is spent in this nearly optimal foveal region, there is correspondingly less selection pressure to "optimize" the behavior in other regions. This is particularly true given the relative small number of steps executed in these simulations. For example, sometimes an evader escapes because its behavior is more efficient, but sometimes it escapes because the 25 steps of the simulation have expired. There are very few tag games where non-optimal behavior in the off-foveal regions makes a measurable difference in fitness.

5.3 Run G

Run G did not segregate the pursuer and evader code and used a larger limit on program size. One of the changes seemed to make the problem harder to solve. The syntactic constraint used in earlier runs is a user-provided “hint” that there are two distinct cases to solve. This may lower the difficulty of the problem. Further comparisons of more runs would provide more reliable information about difficulty. Figure 10 shows the performance of the G population against the optimal player. Comparing this to Figure 5 shows that run C generally did better sooner.

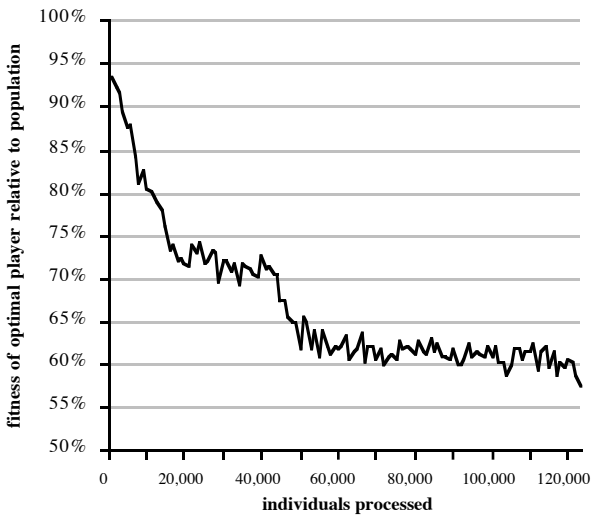

Figure 10: plot of the fitness of the optimal player placed in competition with the evolving population of run G over 123 generation equivalents.

Individual 113520 of run G was the best of population when it was created. The size 98 program is listed below. Its stimulus-response map is shown in Figure 11. This individual has many strange behavioral traits, for example pursuit behavior has a reasonable two phase strategy for opponents up to 5 units ahead but is very inept for opponents further away. The evasion behavior is strongly asymmetrical.

```
(% (% (if-it (abs (local-x)) (iflte (iflte (local-x)
0.57168305 (local-x) (+ (iflte (local-y) (iflte (local-y)
(if-it (local-x) (abs (local-x))) (iflte 0.40530929
0.26004231 (abs (local-x)) (local-y)) (if-it 0.40530929
0.57168305)) (min (abs (local-x)) (+ (local-x) (local-
x))) (local-x)) (local-x))) 0.57168305 (local-x) (+
(iflte (local-y) (iflte (local-y) (if-it (local-x)
(local-x)) (iflte 0.40530929 (local-x) (abs (iflte
(local-x) 0.37254661 0.32281655 (local-x))) (local-x))
(if-it 0.40530929 (abs (local-x)))) (min 0.1637349 (iflte
(local-x) (local-y) (abs (iflte (abs (local-x)) (max (if-
it (local-y) (abs 0.53183758)) (local-x)) 0.32281655
```

```
(local-x))) 0.53183758)) (local-x)) (local-x)))) (+
(local-x) (local-x))) (iflte (- (abs 0.53183758) (if-it
(% 0.57168305 (local-y)) (- 0.1637349 (local-y))))
0.40530929 (abs 0.53183758) 0.83426005))
```

6. Conclusions

Using the game of tag to test relative fitness, artificial evolution was able to discover skillful tag players in the form of vehicle-steering control programs.

These near-optimal players arose solely from direct competition with each other. Good results were obtained despite the considerable variance of fitness values inherent in the staging of individual games and the random selection of opponents. Fitness was based only on relative performance and did not require knowledge of the optimal strategy.

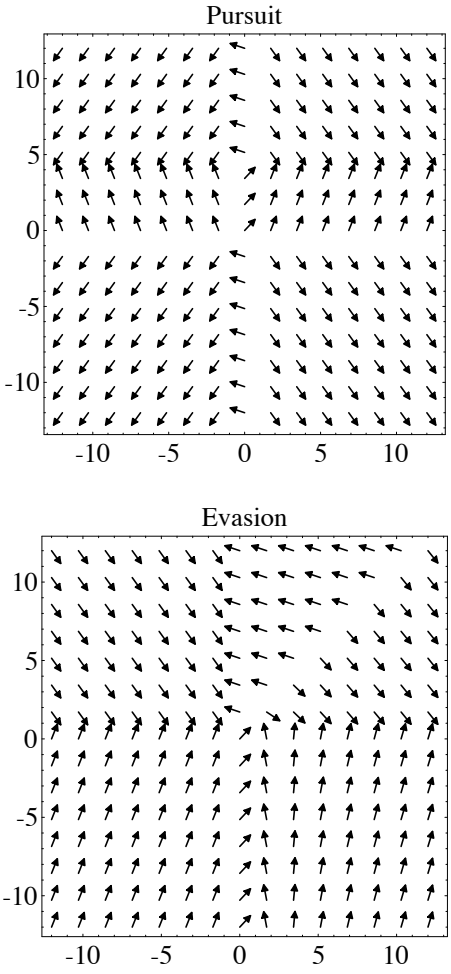

Figure 11: Plots of steering angle versus opponent position for the best-of-population player from run g after 113 generation equivalents

Since the optimal strategy *was* known for the problem studied here, it could be used as a benchmark. The player population evolved according to relative competitive fitness. Its progress in absolute terms could be measured by placing the evolved players in competition with the optimal player.

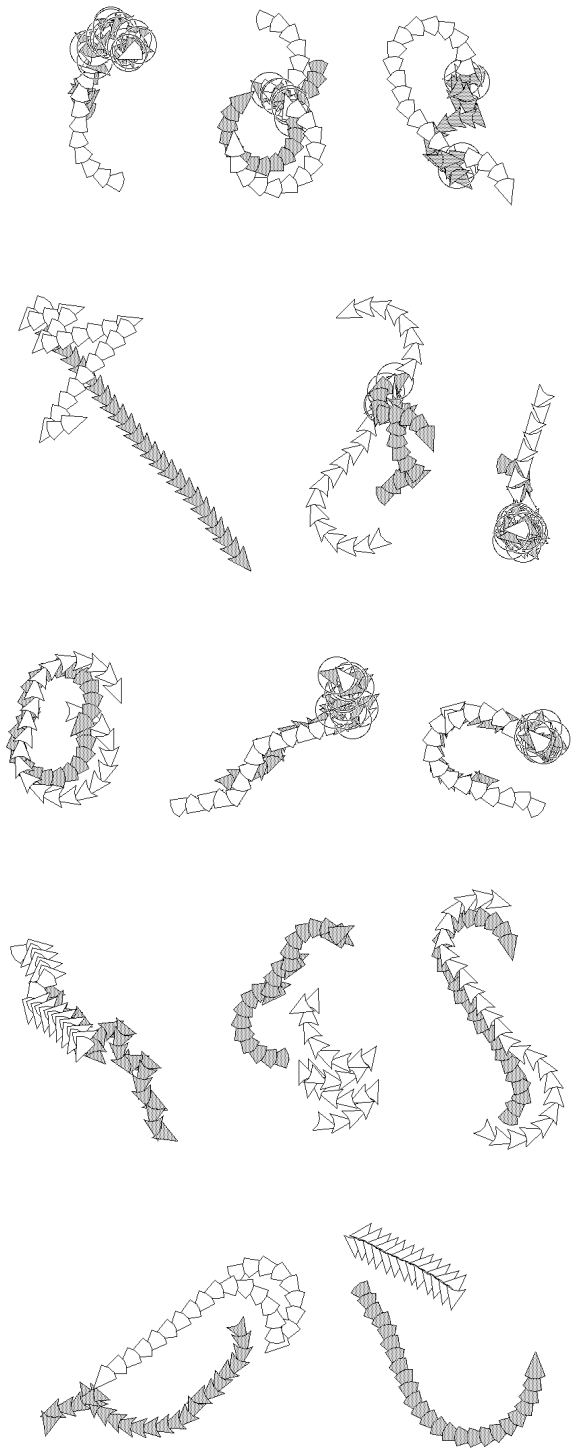

Figure 12: tag games from run G. A common flaw with all of these players is that they tend to have only two forward steering directions. Efficient pursuit and evasion require the ability to modulate turning angle. Some of these strategies look nothing like optimal behavior. They survive because they have adapted to their environment. For example the pursuer (in white) in the lower right image could never catch a good evader, but it is well suited to mowing down incompetent evaders.

Attempts to characterize the quality of evolved players are subject to differences of interpretation, but in at least one experimental configuration (run C) the population's average performance was within 10% of the optimal player, and the best of population individual performed within a few percentage points of optimal. Note that these quality metrics are only as meaningful as the somewhat *ad hoc* fitness measure used in these experiments.

In other runs described here, the quality of evolved players approached, but did not reach, that of the optimal player. It is not clear if this is because of a fundamental limitation of competitive fitness, a flaw in the experimental design, or simply because of limitations of genetic population size and length of runs.

7. Future Work

These experiments used a simple competitive behavior to test the concept of self organizing development of goal-directed behavioral control programs. The generally positive results found here encourage plans for more ambitious future projects.

Eventually the goal is to apply this technique to more complex games. But first, a more elaborate version of the game of tag is an obvious next step. A more complicated control problem, and a more visually interesting class of motion, would result from adding momentum to create a physically realistic model of vehicle motion in place of the current kinematic model. In a physically based model, the angle returned from evolved control programs would be interpreted as the direction of *steering force* to be applied to the vehicle's current momentum. If tag playing vehicles have momentum, their velocity and acceleration become relevant to the control problem. In the real world, the game of tag is usually played with multiple players and in the presence of obstacles. Adding these features, and sensory facilities for dealing with them, would create a rich environment for future studies. In place of the isolated, episodic, pair-wise tag games used in this work, it would be interesting to investigate an ongoing ecological simulation where a large population of agents interact with each other through the game of tag.

In [Angeline 1993] a individual's competitive fitness is determined in a single elimination tournament involving a generation's entire population. In the current work competitive fitness is tested against a randomly selected subset of the existing steady-state population. Several arbitrary choices where made when this selection mechanism was designed. Six opponents are selected with uniform random distribution. Is six too many or too few? Should the selection be skewed toward higher fitness opponents (say by tournament selection) as when parents are selected for crossover? Would judging fitness against the current leaders tend to improve evolvability or stifle it? This approach was attempted in run D but the results were inconclusive. On the other hand this approach is similar to that used in [Sims 1994] which produced good results.

Another approach to steady-state competitive fitness would be to use Angeline's technique of determining fitness based on standings in a single elimination tournament tree, but to do it in small batches of newly created individuals. A batch of 16 or 32 new individuals would be created, then pitted against each other in a single elimination tournament tree. Their standings in competition with their own littermates would determine their fitness. This approach is similar to that used in [Smith 1994] but would allow fitness values to reflect competition with a wider range of opponents.

The delightful variety of strategies that appeared in these experiments demonstrate that there are many, many way to approach a given task. The road to successful behavior may lie along any of those variations. The most robust evolutionary systems are those that can pursue many approaches at once. Geographically distributed, deme-based systems promote diversity and parallelism. They seem well-suited to competitive evolution of behavior [Ngo 1993] because the quality of competition is better when diversity is preserved.

Acknowledgments

This work is supported by Electronic Arts. The author wishes to thank Luc Barthelet, Vice President of Technology, for allowing research to coexist with product development. Additional thanks to Luc for help preparing the stimulus-response arrow diagrams. Thanks to John Koza and Pete Angeline for inspiration, encouragement, and critique. Thanks to James Rice for a detailed review. Special thanks to my wife Lisa and to our first child Eric, who was born at just about the same time as individual 15653 of run C.

References

[Angeline 1993] Angeline, P. J. (1993) and Pollack, J. B., Competitive Environments Evolve Better Solutions for Complex Tasks, in *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed San Mateo, California: Morgan Kaufmann, pages 264-270.

[Arkin 1987] Arkin, R. C. (1987) Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior", *Proceedings of the 1987 IEEE Conference on Robotics and Automation*, Raleigh, North Carolina, pages 264-271.

[Axelrod 1984] Axelrod, R. (1984) *The Evolution of Cooperation*, Basic Books, New York.

[Axelrod 1989] Axelrod, R. (1989) Evolution of Strategies in the Iterated Prisoner's Dilemma, in *Genetic Algorithms and Simulated Annealing,* L. Davis editor, Morgan Kaufmann.

[Braitenberg 1984] Braitenberg, V. (1984) *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, Massachusetts.

[D'haeseleer 1994] D'haeseleer, P (1994) and Bluming, J., Effects of Locality in Individual and Population Evolution, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, Massachusetts: MIT Press.

[Hillis 1992] Hillis, W. D. (1992) Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure, in *Artificial Life II*, Santa Fe Institute Studies

in the Sciences of Complexity, Proceedings Volume X, C. Langton, Ed., Addison-Wesley, Redwood City, California, pages 313-324.

[Holland 1992] Holland, J. H. (1992) *Adaptation in Natural and Artificial Systems*, second edition, MIT Press, Cambridge, Massachusetts.

[Isaacs 1965] Isaacs, R. (1965) Differential Games, John Wiley and Sons, New York.

[Kauffman 1993] Kauffman, S. A. (1993) *The Origins of Order*, Oxford University Press, New York.

[Kinnear 1994] Kinnear, K. E., Jr. (1994) Alternatives in Automatic Function Definition: A Comparison of Performance, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, Massachusetts: MIT Press.

[Koza 1992] Koza, J. R. (1992) *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, ISBN 0-262-11170-5, MIT Press, Cambridge, Massachusetts.

[Koza 1994] Koza, J. R. (1994) *Genetic Programming II: Automatic Discovery of Reusable Programs*, ISBN 0-262-11189-6, MIT Press, Cambridge, Massachusetts.

[Lindgren 1992] Lindgren, K. (1992) Evolutionary Phenomena in Simple Dynamics, in *Artificial Life II*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume X, C. Langton, Ed., Addison-Wesley, Redwood City, California, pages 295-312.

[Lindgren 1994] Lindgren, K. (1994) and Nordahl, M., Artificial Food Webs, in *Artificial Life III*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XVI, C. Langton, Ed., ISBN 0-201-62494-X, Addison-Wesley, Redwood City, California, pages 73-103.

[Merz 1971] Merz, A. W. (1971) *The Homicidal Chauffeur: A Differential Game*, (PhD Thesis) Stanford University Center for Systems Research, Report SUDAAR 418.

[Menczer 1994] Menczer F (1994?) and Belew R. K., Latent Energy Environments, to appear in *Plastic Individuals in Evolving Populations*, Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley (in press).

[Miller 1989] Miller, J. H. (1989) The Co-evolution of Automata in the Repeated Prisoner's Dilemma, Santa Fe Institute Report 89-003.

[Ngo 1993] Ngo, J. T. (1993) and Marks, J., Spacetime Constraints Revisited, Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993), in *Computer Graphics Proceedings*, Annual Conference Series, 1993, ACM SIGGRAPH, New York, pages 343-350.

[Ray 1992] Ray, T. S. (1992) An Approach to the Synthesis of Life, in *Artificial Life II*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume X, C. Langton, Ed., Addison-Wesley, Redwood City, California, pages 371-408.

[Reynolds 1994a] Reynolds, C. W. (1994) An Evolved, Vision-Based Model of Obstacle Avoidance Behavior, in *Artificial Life III*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XVI, C. Langton, Ed., ISBN 0-201-62494-X, Addison-Wesley, Redwood City, California, pages 327-346.

[Reynolds 1994b] Reynolds, C. W. (1994) Evolution of Obstacle Avoidance Behavior: Using Noise to Promote Robust Solutions, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, Massachusetts: MIT Press.

[Reynolds 1994c] Reynolds, C. W. (1994) Evolution of Corridor Following Behavior in a Noisy World, to appear in *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (SAB94), in press.

[Sims 1994] Sims, K. (1994) *Evolving 3D Morphology and Behavior by Competition*, Artificial Life IV (in this volume), R. Brooks and Pattie Maes, Eds., MIT Press, Cambridge, Massachusetts.

[Smith 1994] Smith, R. E. (1994) and Gray, B., Co-Adaptive Genetic Algorithms: An Example in Othello Strategy, to appear in *The Proceedings of The Florida Artificial Intelligence Research Symposium 1994*, and available as *TCGA Report Number 94002*, The University of Alabama, Tuscaloosa.

[Syswerda 1991] Syswerda, G. (1991) A Study of Reproduction in Generational and Steady-State Genetic Algorithms, in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo, California: Morgan Kaufmann, pages 94-101.

[Werner 1993] Werner, G. M. (1993) and Dyer, M. G., Evolution of Herding Behavior in Artificial Animals, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (SAB92), Meyer, Roitblat and Wilson editors, ISBN 0-262-63149-0, MIT Press, Cambridge, Massachusetts, pages 393-399.

[Yeager 1994] Yeager, L. (1994) Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context, in *Artificial Life III*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XVI, C. Langton, Ed., ISBN 0-201-62494-X, Addison-Wesley, Redwood City, California, pages 263-298.