Diffuse versus True Coevolution in a Physics-based World

Gregory S. Hornby
Computer Science Dept.
Brandeis University
Waltham, MA 02254-9110
hornby@cs.brandeis.edu

Brian Mirtich
Mitsubishi Electric Research Lab
201 Broadway
Cambridge, MA 02139
mirtich@merl.com

Abstract

We compare two types of coevolutionary tournaments, *true* and *diffuse*, in contests using a general-purpose, physics-based simulator. Previous work in coevolving agents has used true coevolution and found that populations tend to enter mediocre states. One hypothesis for alleviating these problems is to use diffuse coevolution. Our results show that agents evaluated with diffuse tournaments are more generalized than those evaluated with true tournaments.

1 INTRODUCTION

Evolutionary algorithms have been used to optimize the behavior of a population of *agents* through repeated modifications and fitness evaluations. Often the agents fall into distinct populations or *species* that directly compete with each other, as in a predator-prey situation. The individual species may be evolved separately, although not independently, in a strategy termed *coevolution*. A good example of the technique is the work of Sims, in which three-dimensional robots were evolved through competitions that involved controlling a cube [Sims, 1994].

Coevolution has much promise as a means for automatically improving the quality of the fitness landscape. As the populations improve, they provide more difficult challenges for each other, thereby raising the fitness bar. *True* coevolution describes a situation where exactly two species are simultaneously evolved and compete head-to-head in the fitness trials. One problem with true coevolution is that the species may become specialized to one another, each relying on the opposing species' strategies and weaknesses. The species then stabilize in mediocre states, and an agent from one of the species will not perform well when pitted against a new type of opponent. To alleviate this problem, [Bullock, 1995] proposes using *diffuse* coevolution: allowing each species to compete against multiple opposing species, which are all evolved separately. Since each agent is tested against a wider diversity of opponents, more general strategies can evolve. The goal of this research was to compare true and diffuse coevolution, using a relatively sophisticated competition, to determine if diffuse coevolution produces more general strategies. Our results support Bullock's hypothesis that diffuse coevolution produces better agents than true coevolution.

We chose a predator-prey scenario to study coevolution. While one of the simplest forms of coevolution, predator-prey coevolution may lead to a better understanding of coevolution in general. Other reasons for studying predator-prey coevolution are discussed in [Miller and Cliff, 1994], including applications to software agents and robotics and relevance to other fields such as game theory.

A meaningful study of predator-prey coevolution requires evolution of complex strategies. One of the factors that limits the evolution of strategies is the agent's environment. The environment comprises the objects the agent interacts with as well as the rules, such as physical laws, governing these interactions. The objects may be static entities, like floors and walls, or they may be other agents inhabiting the same world.

In simple environments simple behaviors are generally sufficient, and thus complex behaviors do not evolve. As the environment becomes more complex there is room for complex behaviors to evolve. Hence there is a trend in predator-prey coevolution towards richer environments. The environment used by Reynolds to evolve tag players was a simple one: agents were modeled as circles existing in a discrete-time, flat, two-dimensional world without momentum, friction or other objects [Reynolds, 1994]. Reynolds states that a world with more realistic physics could produce more interesting motion and a rich environment for future studies. More complex physics was used in [Cliff and Miller, 1995] and [Cliff and Miller, 1996]. This environment, while two-dimensional, was continuous-time and included momentum. Again, no other objects existed in the flat world. In [Floreano and Nolfi, 1997] a Khepera simulator simulator was used followed by coevolution with actual Kheperas in [Floreano and Nolfi, 1998]. They used the real world—or a good simulation of it—but the environment was simple and essentially flat. Another limitation is that only Kheperas (simple robots) can be used and it cannot be extended to more complex agents or environments.

For our own experiments, we use predator-prey competitions similar in principle to those previously mentioned, but driven by a general-purpose, three-dimensional, rigid-body simulator. The agents are tricycles, with independently controled wheels. Competitions take place in an enclosed ring with an obstacle in the center (Figure 1). The obstacle in the center prevents trivial strategies from developing. The competitions last 90 simulated seconds. The combination of the outer wall and long simulation time forces agents into close quarters, encouraging them to be reactive. Although the environment is basically flat, the three-dimensional nature of the simulator plays a definite role in the evolution of strategies: agents flip over if they cut corners too tightly or experience destabilizing collisions. Using the simulator to drive the competitions, we compare two types of tournaments for predator-prey coevolution, one based on true coevolution and the other based on diffuse coevolution.

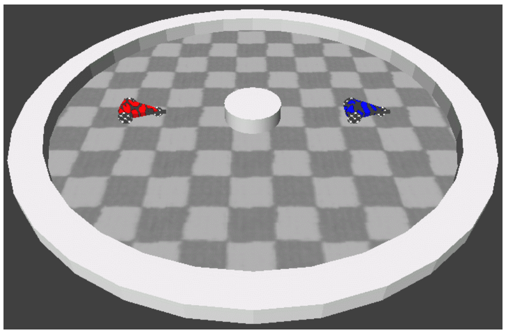

Figure 1: *Pursuer-Evader Environment*

The rest of the paper is organized as follows. We discuss some details of the physical simulator used for our experiments in Section 2. This section includes pointers to source code for some of the publicly available modules used in the simulator; these may be useful to others wishing to use physics-based simulation as a tool for evolving agents. Details of the competitions and computation of fitness scores are described in Section 3. Next we describe our evolutionary algorithm in Section 4 and our neural control system, which drives the agents, in Section 5. We present, and discuss, our results in Section 6. Concluding remarks and directions for future work are in Section 7.

2 SIMULATOR

The contests between the pursuer and evader vehicles were carried out using a general-purpose, dynamic simulator for three-dimensional, rigid bodies. This research tool is a compilation of many algorithms developed in the graphics and physics-based modeling community. The simulator enforces physical laws governing momentum, friction, and collisions; these have a significant bearing on the predator-prey competitions. Simulating the contest could generally be done faster than real time: a 90 second contest takes 30-60 seconds to simulate. Nonetheless, the large number of competitions needed to evolve agents mandated overnight runs. A detailed discussion of the principles underlying the simulator is beyond the current scope, but this section summarizes the salient points. More information can be found in [Mirtich, 1996].

2.1 GEOMETRY AND COLLISION DETECTION

The geometries of all parts of the agents and environment are described by polyhedra; curved surfaces like spheres and cylinders are suitably approximated. For instance, the tricycle wheels are modeled as extruded 30-sided regular polygons. Along with each rigid part, either a mass or a density is specified. The simulator uses this information, along with the geometry to automatically compute the inertial properties needed for simulation: total mass, the coordinates of the center of mass, the principle axes of inertia, and the corresponding moments of inertia. Source code for these computations is publicly available.[1]

The starting point for rigid body simulation is an analysis of what bodies are in contact or colliding. This is what creates interesting motion; without collision or contact, bodies follow the simple ballistic trajectories of Newtonian physics. The collision detection system is responsible for reporting collisions and contacts between different bodies. From a geometric standpoint, each tricycle comprises four individual pieces:

[1] www.merl.com/projects/rigidBodySim/massProps

the chassis and three wheels; each of these must be tested individually for collisions with the environment.

For a simulation with n bodies, the number of possible interactions at any given time is $O(n^2)$. To reduce processing time, the collision detection system has two phases. The *broad phase* culls most of the $O(n^2)$ pairs of bodies from further consideration. It uses simple bounding boxes around the objects to quickly determine which pairs are in no danger of colliding. The bounding boxes are stored in a multi-resolution spatial hash table as described in [Overmars, 1992]. This data structure can efficiently report which bounding boxes overlap a region of three-dimensional space.

The pairs which the broad phase can not eliminate are passed to the slower but more precise *narrow phase* of the collision detection system. The narrow phase considers the exact geometry of the objects and determines if they are intersecting, in contact, or separated. There are many choices for narrow-phase detectors[Lin, 1993, Hubbard, 1994, Gottschalk et al., 1996, Cameron, 1997]; our simulator uses the publicly available *V-Clip* algorithm [Mirtich, 1998].[2] The algorithm, designed for speed and robustness to geometric degeneracies, is well-suited to our task of performing thousands of simulations in overnight runs. Together, the two phases result in a fast collision detection system that does not dominate the computational cost of the simulation.

2.2 COLLISION RESOLUTION AND CONTACT MODELING

After the collision detection is performed, impulses must be applied to bodies that are colliding. These impulses instantaneously change the velocities of the bodies and prevent them from penetrating. Impulses are computed using a practical algebraic formulation described in [Chatterjee and Ruina, 1998]. This method is easy to implement and has several desirable properties. For instance, it is guaranteed not to create energy during a collision, which is a weakness of many algebraic formulations. It also incorporates friction and restitution, each of which is described by a pair of coefficients.

More difficult than resolving collisions is the problem of modeling the interactions between bodies in persistent contact, such as the wheel of a tricycle rolling along the ground. For much of this interaction, our simulator uses a novel approach known as *impulse-based* simulation. In this paradigm, persistent contact is modeled through trains of small, rapid collision impulses. These can approximate a steady contact force well under the right conditions. More details of the method are in [Mirtich, 1996]. The primary advantages of impulse-based simulation are that it is simple to implement, robust, and in many situations fast, however it slows down considerably when there are many persistent contacts. In our tricycle competitions, there were usually six wheels rolling along the ground at any given time. To improve performance in such situations, our simulator also employs a *constraint-based* contact modeler. Constraint-based formulations are described in detail in [Baraff, 1989, Baraff, 1992]. They work by computing the forces exerted between bodies in contacts. Our approach uses a linear programming technique similar to but not as sophisticated as the standard complementarity formulations [Baraff, 1992, Pang and Trinkle, 1996]. Both static and dynamic friction are supported.

The transition between impulse-based contact and constraint-based contact is a smooth one in our simulator. Impulses are used to resolve collisions, and also to prevent penetration between bodies that are in transient contact, as when the chassis of a tricycle momentarily rubs along the outer wall of the arena. As the frequency of impulses between two contact points increases, the contact is switched over to a constraint-based approach, with a genuine contact force applied. This is usually the case with the wheel-ground contacts. Computing the contact forces is the dominant cost in the simulation system, generally accounting for over 75% of the computation time.

2.3 MULTIBODY DYNAMICS AND CONTROL

The simulator supports *multibodies*: collections of rigid bodies connected by idealized joints. Both revolute (rotating) and prismatic (sliding) joints are supported, although only revolute joints are needed for the tricycles. The dynamics of multibodies are more complicated than those of unlinked rigid bodies but can be computed by classical algorithms. Many algorithms for computing the accelerations of an n-link multibody have $O(n^3)$ complexity, however Featherstone invented an elegant $O(n)$ method [Featherstone, 1983]. The simulator uses a publicly available implementation of Featherstone's algorithm.[3]

The multibody dynamics algorithms takes as input the current joint positions and velocities, as well as any external torques applied by simulated joint motors. This is the sole means by which the evolved behavioral systems, described in Section 5, can influence the course

[2] www.merl.com/projects/vclip

[3] www.merl.com/projects/rigidBodySim/multibodyDyn

of the simulation. Beyond the joint torques, everything is governed by physics alone.

While it might have been possible to have the behavioral system learn to control the joint torques directly, this was deemed an unnecessarily difficult task. Real robot controllers are hierarchical: high-level controllers command particular joint positions and velocities to low-level controllers that directly determine motor torque. This division is natural and still admits a wide variety of control strategies. The interaction between the low-level controllers and the actual motors is a standard problem of control theory. The simulator provides standard low-level controllers, such as proportional derivative (PD) controllers, to actuate the multibody joints. In evolving agents, our interest is in the high-level (behavioral) control: what are the appropriate positions and velocities to command? At a 4 Hz rate, the behavioral system uses a neural network to determine the desired velocities for the three wheels and a desired position for the steering joint. These are passed to low-level controllers that attempt to maintain the desired values until new values are computed. External forces and collisions cause discrepancies between the actual and desired values.

3 EXPERIMENTAL METHOD

For our experiments the pursuer-evader game is between two tricycles. The contest takes place in a ring of radius 700 units. In the center of the ring is a circular barrier of radius 100. The pursuing and evading vehicles are physically identical tricycles with rough dimensions of $150 \times 80 \times 50$ units. The two rear wheels spin about a transverse axis; the front wheel is mounted in a fork to provide steering.

Each contest lasts 90 seconds with the initial vehicle positions as shown in Figure 1, with the pursuer on the left and the evader on the right. The object of the contest is for the pursuing vehicle to catch the evading vehicle. In practice, a triangle 1.3 times the size of each vehicle is placed around each vehicle, and the pursuing vehicle catches the evading vehicle when the triangles intersect each other.

The maximum distance $d_{\max}$ and minimum distance $d_{\min}$ between the pursuer and evader are monitored during the game. Scores for a game are determined by these distance and whether the pursuer caught the evader. If the pursuer does not catch the evader then the score of the evader is $((d_{\min}+d_{\max})/400)-2$, lying in the range 0 to 7. The pursuer's score is the negative of this value. If the pursuer catches the evader at time t, its score is $(90-t)/30$, lying in the range 0 to 3. The evader's score is the negative of this value.

4 EVOLUTIONARY ALGORITHM

The difference between our diffuse and true coevolutionary systems is the number of species against which individuals compete. In one system individuals compete against only one other species of individuals. We call this system *true-coevolutionary tournaments* (TC-tournaments). With the other system individuals compete against four different species of opponents. We call this system *diffuse-coevolutionary tournaments* (DC-tournaments). We now describe our coevolutionary algorithm.

In both cases two populations of 96 individuals are used. Each population is split into four independent subpopulations of 24 individuals. We call each of these subpopulations a *species*. There is no migration between the species. Each species is decomposed into 4 demes of 6 individuals. The demes are arranged in a ring. In this way each species has a spatial structure. At the end of a generation each deme passes its second best individual to the next deme in the ring (Figure 2). Evolution is performed for 80 generations.

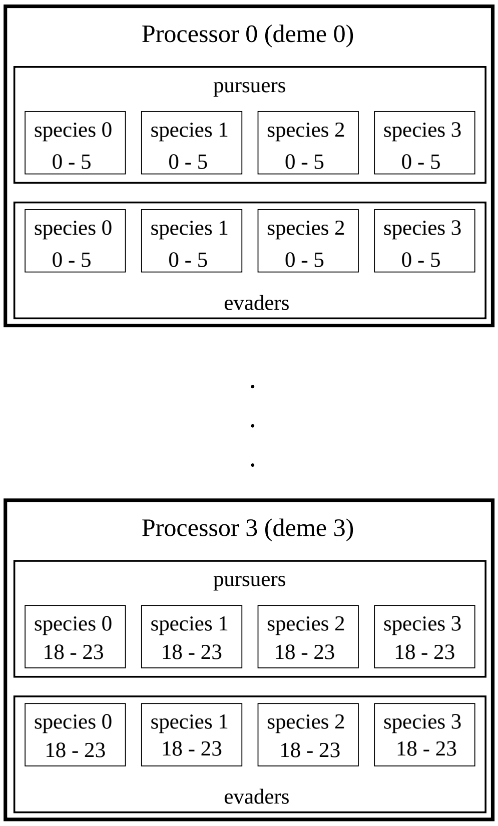

Figure 2: *Population Structure*

We are interested in the effects of varying the number of species an individual competes against. To eliminate dependencies on population sizes, we coevolve four independent pairs of pursuer and evader species in the TC-tournaments. Future work might compare the trade-offs between using single populations coevolving against each other with multi-species populations coevolving against each other.

To reduce the amount of time for processing the population is distributed amongst four processors. Each processor contains one deme from each species and operates asynchronously. As with demes, processors are treated as being in a ring. At the end of a generation the processor writes its population to file for the next processor to read and it reads the population file of the previous processor. Using four R1000 processors of an SGI Reality Engine, 96 generations of evolution took approximately 100 hours.

Selection and reproduction are similar to other work. In each deme of 6 individuals the best two individuals are copied to the next generation. Parents for creating individuals for the next four slots are selected stochastically based on rank. The probability of creating new individuals through mutation and recombination are 0.4 and 0.6, respectively.

The DC-tournaments and TC-tournaments differ in how opponents are paired. In both algorithms individuals are evaluated by their total fitness over four games. In TC-tournaments individuals are evaluated against four individuals from the same opponent species (i.e. individuals from species 0 only play opponents from species 0; individuals from species 1 only play opponents from species 1; etc.). In DC-tournaments individuals are played against one individual from each opponent species.

5 NETWORK ARCHITECTURE

The control system of each agent is a recurrent, artificial neural network. The network architecture is based on the *GNARL* system ([Angeline et al., 1994]) with the main modification being individuals are evolved under a standard evolutionary algorithm with both recombination and mutation. Our networks also have some influence from continuous-time networks ([Beer and Gallagher, 1992]).

5.1 DATA STRUCTURE

Each vehicle uses its own neural network to drive the inputs to its control system. A network consists of twenty processing units. Eight processing units receive the following inputs: the actual wheel velocity of one of the rear wheels; the actual steering angle; the heading to the other vehicle; the distance to the other vehicle; and four inputs giving the nearest distance to a wall at angles 30, 0, -30 and 180 degrees. Two of the processing units are used as outputs, giving desired values for wheel velocity and steering angle. The maximum desired wheel velocity is 240 degrees per second for the pursuer, and 200 degrees per second for the evader, giving the pursuer a 20% speed advantage. For both vehicles, the maximum desired steering angle is 40 degrees in the left or right direction. The remaining ten processing units are hidden units.

Networks have a fixed number of processing units allowing a network data structure to be a fixed size. A network data structure consists of: w, a matrix of real-valued weights; f, a vector of processing functions; θ, a real-valued vector of input biases; σ, a real-valued vector of output biases; τ, a real-valued vector of time constants; and s, a vector of boolean values signifying if a processing unit is a stepping unit. The value of the ith row and jth column of the weight matrix is the value of the weight from the ith processing unit to the j processing unit. The ith value of the vectors is the attribute for the ith processing unit. A processing unit functions as follows:

$$a'_i = \tau_i a_{i-old} + (1-\tau)[f(\sum_j w_{ji} a_j + \theta_i) + \sigma_i] \quad (1)$$

If s_i is true, then the unit is a stepping unit. In which case if a'_i is greater than 0 then a_i is set to 1, otherwise a_i is set to 0. If the processing unit is not a stepping unit then a_i is set to a'_i.

5.2 CREATION AND REPRODUCTION

The initial population of networks consists of randomly generated networks. The weight matrix is created by setting an initial connectivity of 4 to 14 input links for each processing unit. Weights and real-valued vectors are assigned a random value with uniform distribution in the range [-1, 1]—except for τ, the vector of time constants, which has values assigned with a uniform distribution in the range [0, 1]. Processing functions are randomly selected to be one of $\{x, sin(x), \frac{2atan(x)}{\pi}, e^x, \frac{1}{1+e^{-x}}\}$. The values of s are randomly selected to either true or false.

The mutation operator has a 0.6 probability of mutating weights in the weight matrix, a 0.15 probability of adding or deleting links (setting to 0 or setting to a random value an element in the weight matrix) and a probability of 0.25 of mutating the other properties of each processing unit. In mutating a real value a Gaussian random number ($\sigma = 0.01$) is added to it. In mutating the processing function vector a new function is randomly selected. In mutating the boolean value signifying whether or not a unit is a stepping unit, true and false are assigned with equal probability.

Recombination between two networks consists of making a copy of the first parent then selecting one attribute and making that a recombination of the values of both parents. There is a probability of 0.6 of recombining the weight matrix with the remaining 0.4 divided amongst the other attributes. When both parents genes have non-zero values, real-valued recombination uses the equation,

$$v_c = v_{p1} + \alpha(v_{p1} - v_{p2}) \tag{2}$$

where α is a number randomly generated with a uniform distribution in the range [-1, 1]. Otherwise the child's value is randomly selected to be the value of one of the parents. The weight matrix is recombined by applying the real-valued recombination to each corresponding pairs of elements in the parents' weight matrices. Vector recombination consists of applying the real-valued recombination operator to the corresponding values in the parents' vectors to generate the values of the child's vector. The exception is the s vector which is created through uniform crossover of the parents' s vectors.

6 RESULTS

The first few seconds of a contest set the stage for the rest of the match. Moving forward was the initial action evolved by both pursuers evaders. With a wall ahead both agents needed to choose between turning left or right. If the pursuer guessed correctly and turned in the same direction as the evader it had the evader in front of it in a favorable position for catching. By steering directly towards the evader and moving ahead at full speed a pursuer was almost guaranteed a win. At this point some pursuers were more successful than others with proficiency improving over the generations. It should be mentioned that pursuers also had a tendency, especially in early generations, to go straight towards the evader, without turning to avoid the intervening obstacle.

With TC-tournaments the four pursuer species each coevolved against a different evader species. In early generations vehicles moved about randomly and were unresponsive to each other. By generation 30 contests settled into two types: where both pursuer and evader turned left or both turned right; and where one turned right and the other left. After this point agents did not change their steering much. Evaders would continue going around the ring in varying sized arcs. Pursuers would drive in the general direction of the evader. Figure 3 plots the win ratios for the different species of the pursuer population.

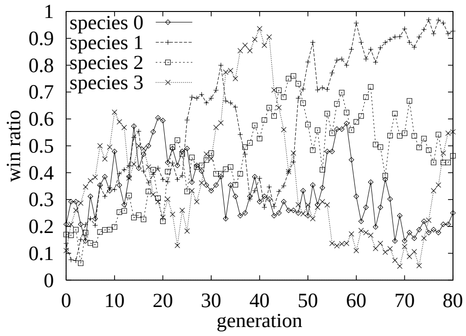

Figure 3: *TC-tournament Results*

Agents evolved with DC-tournaments were more interesting. Three of the four species of evaders evolved strategies similar to those of the previous system—although later investigation showed them to be more reactive. Evaders of the fourth species came up with a novel strategy of starting by going in reverse for a few seconds before going forward. This began in generation 30 with an individual that started going backwards a little, then stopped and went forwards and to the right around the ring. By generation 60 this strategy dominated that species. In one case, after its initial turn-around the evader would go straight for the middle barrier in such a way that the pursuer had to go on the other side to catch him. The evader then stopped with the barrier in between and then went backward as the pursuer passed. Similarly, in other games agents from this species would stop on the outside of the wall if the pursuer was far away. They would sit and do nothing until the pursuer started getting close. And they would move in reverse to get away. Figure 4 plots the win ratios for the different species of the pursuer population.

Figure 5 depicts two 80th-generation contests: one from the TC-tournament and one from the DC-tournament. The top row depicts the TC contest: (1) the pursuer discovers the center obstacle as the evader hugs the perimeter; (2) a velocity reversal looks like a winning pursuer strategy until (3) reckless turning speeds cause the pursuer to flip over while the unimaginative evader sneaks by. The bottom row depicts the DC contest: (1) the pursuer closes in on the retreating evader; (2) a sneaky evader spin leads to a momen-

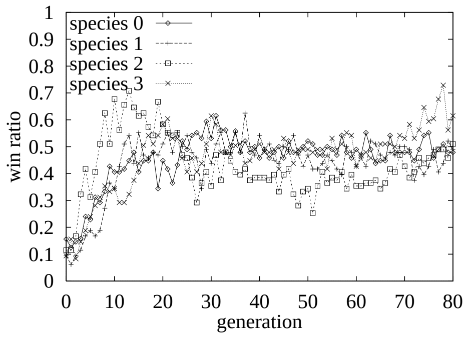

Figure 4: *DC-tournament Results*

tary escape; and (3) the pursuer reacts, reverses, and eventually tags the evader.

In comparing the two systems, agents using TC-tournaments evolved simpler strategies. Prey almost never went backwards. In competitions between agents from different species, the agents played against *phantom* opponents from the species they were coevolved against. That is, agents moved along predetermined paths and were not reactive to their actual opponent. Sometimes the predator would drive right past the prey, ignoring it as it tried to catch its phantom opponent. Also, even in the last five generations predators drove straight into the barrier in the pursuit of the prey. Strategies consisted of prey oscillating over the generations between going left or going right with the predators following suit. This oscillation between two mediocre (non-reactive) strategies may explain the large difference in a species' performance variation between the two systems. In contrast, agents evolved in DC-tournaments were more responsive to their inputs. Prey would stop and go backward when it was the better option.

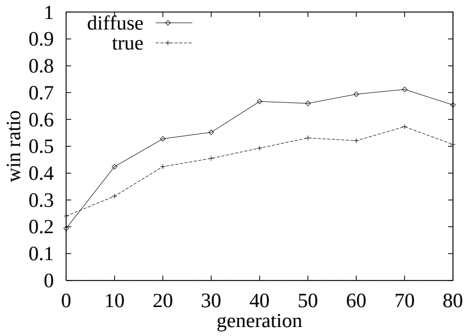

Figure 6: *Diffuse Vs. True*

Finally we compared agents from the two systems against each other. Figure 6 is a graph of win ratios for evaders from one system playing against pursuers from the other system. Each point represents the average win-ratios for the best individuals from each pursuer species competing against the best evader agents from each species from every 10 generations. From this graph it can be seen that pursuer agents evolved using DC-tournaments maintained a 10% to 14% higher win-ratio than did pursuer agents evolved using TC-tournaments. This result confirms that diversity in competitors is important.

7 CONCLUSION

In this paper we compared two types of tournaments, TC-tournaments and DC-tournaments, for coevolution of pursuers and evaders. Experiments used a general-purpose, physics-based simulator for our contests. With TC-tournaments agents coevolve against only one other species. With DC-tournaments agents coevolve against multiple species. Our results show that coevolving against a diversity of opponent species, as with DC-tournaments, produces agents with more general strategies.

One of our beliefs is that the environment has a large influence on the types of behaviors that can be evolved. To date, predator-prey experiments have used simple worlds consisting of just a predator and a prey agent. While results have been interesting none of have been truly amazing. In the real world, interaction between agents occurs in a much richer environment. Increasing the complexity of experimental environment, whether by adding more features to the landscape or including more agents, may lead to the evolution of much more interesting agents.

References

[Angeline et al., 1994] Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65.

[Baraff, 1989] Baraff, D. (1989). Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics*, 23(3):223–232.

[Baraff, 1992] Baraff, D. (1992). *Dynamic Simulation of Non-Penetrating Rigid Bodies*. PhD thesis, Department of Computer Science, Cornell University.

[Beer and Gallagher, 1992] Beer, R. D. and Gallagher, J. G. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122.

[Bullock, 1995] Bullock, S. (1995). Co-evolutionary design: Implications for evolutionary robotics. In *CSRP 384*, University of Sussex.

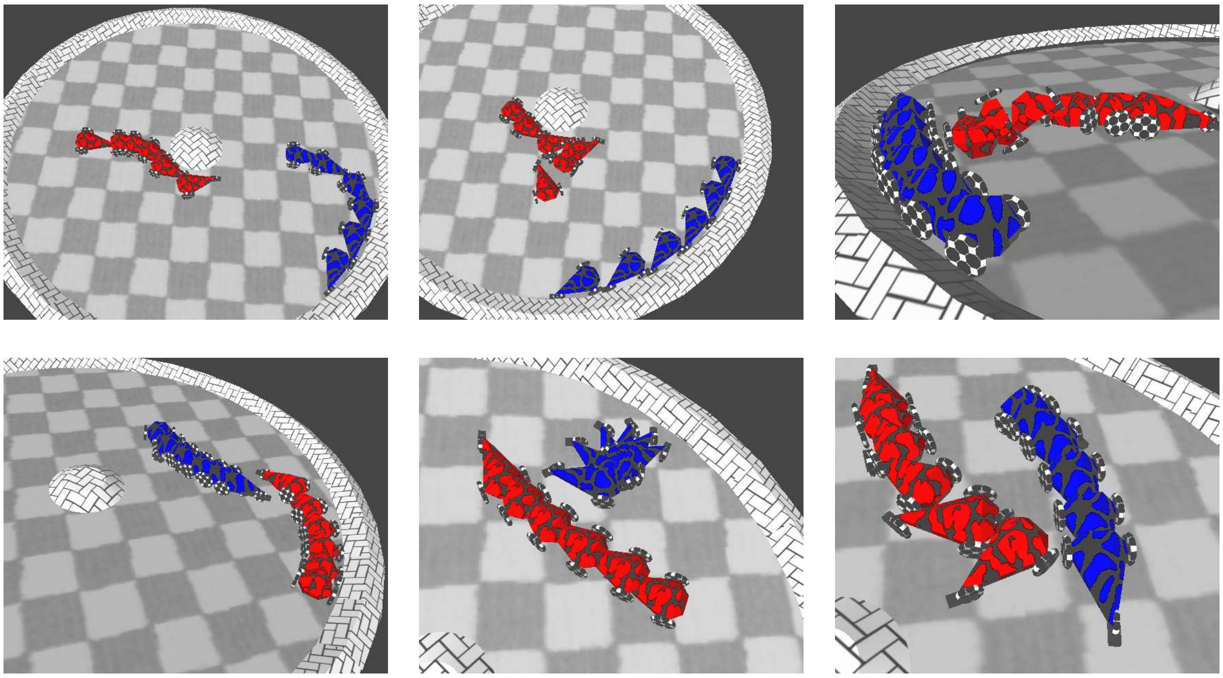

Figure 5: TC And DC Contest Snapshots

[Cameron, 1997] Cameron, S. (1997). Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proceedings of International Conference on Robotics and Automation*. IEEE.

[Chatterjee and Ruina, 1998] Chatterjee, A. and Ruina, A. (1998). A new algebraic rigid body collision law based on impulse space considerations. *Journal of Applied Mechanics*, 65:939–951.

[Cliff and Miller, 1995] Cliff, D. and Miller, G. F. (1995). Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In Morán, F., Moreno, A., Merelo, J. J., and Chacón, P., editors, *Advances in Artificial Life: Proc. of the Third European Conf. on Artificial Life*, pages 200–218, Berlin. Springer Verlag.

[Cliff and Miller, 1996] Cliff, D. and Miller, G. F. (1996). Co-evolution of pursuit and evasion II: Simulation methods and results. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors, *From Animals to Animats 4*, pages 506–515, Cambridge, MA. MIT Press Bradford Books.

[Featherstone, 1983] Featherstone, R. (1983). The calculation of robot dynamics using articulated-body inertias. *International Journal of Robotics Research*, 2(1):13–30.

[Floreano and Nolfi, 1997] Floreano, D. and Nolfi, S. (1997). Adaptive behavior in competing co-evolving species. In Husbands, P. and Harvey, I., editors, *Proceedings of the Fourth European Conference on Artificial Life*, Cambridge, MA. MIT Press.

[Floreano and Nolfi, 1998] Floreano, D. and Nolfi, S. (1998). Competitive co-evolutionary robotics: From theory to practice. In Pfeifer, R., editor, *From Animals to Animats V*. MIT Press.

[Gottschalk et al., 1996] Gottschalk, S., Lin, M. C., and Manocha, D. (1996). Obb-tree: A hierarchical structure for rapid interference detection. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 96. ACM SIGGRAPH.

[Hubbard, 1994] Hubbard, P. M. (1994). *Collision Detection for Interactive Graphics Applications*. PhD thesis, Department of Computer Science, Brown University.

[Lin, 1993] Lin, M. C. (1993). *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley.

[Miller and Cliff, 1994] Miller, G. F. and Cliff, D. (1994). Protean behavior in dynamic games: Arguments for the co-evolution of pursuit-evasion tactics. In Cliff, D., Husbands, P., Meyer, J.-A., Pollack, J., and Wilson, S. W., editors, *From Animals to Animats 3*, pages 411–420, Cambridge, MA. MIT Press Bradford Books.

[Mirtich, 1996] Mirtich, B. (1996). *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley.

[Mirtich, 1998] Mirtich, B. (1998). *V-Clip:* fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208. Mitsubishi Electric Research Lab Technical Report TR97-05.

[Overmars, 1992] Overmars, M. (1992). Point location in fat subdivisions. *Information Processing Letters*, 44:261–265.

[Pang and Trinkle, 1996] Pang, J. and Trinkle, J. (1996). Complementarity formulations and existence of solutions of dynamic multi-rigid-body contact problems with coulomb friction. *Mathematical Programming*.

[Reynolds, 1994] Reynolds, C. W. (1994). Competition, coevolution and the game of tag. In Brooks, R. and Maes, P., editors, *Proceedings of the Fourth Workshop on Artificial Life*, pages 59–69, Boston, MA. MIT Press.

[Sims, 1994] Sims, K. (1994). Evolving 3d morphology and behavior by competition. In Brooks, R. and Maes, P., editors, *Proceedings of the Fourth Workshop on Artificial Life*, pages 28–39, Boston, MA. MIT Press.