Competitive Environments Evolve Better Solutions for Complex Tasks

Peter J. Angeline and Jordan B. Pollack

Laboratory for Artificial Intelligence Research
Computer and Information Science Department
The Ohio State University
Columbus, Ohio 43210

pja@cis.ohio-state.edu
pollack@cis.ohio-state.edu

Appears in:

Genetic Algorithms: Proceedings of the Fifth International Conference (GA93)

Edited by Stephanie Forrest

Competitive Environments Evolve Better Solutions for Complex Tasks

Peter J. Angeline and Jordan B. Pollack
Laboratory for Artificial Intelligence Research
Computer and Information Science Department
The Ohio State University
Columbus, Ohio 43210
pja@cis.ohio-state.edu
pollack@cis.ohio-state.edu

Abstract

In the typical genetic algorithm experiment, the fitness function is constructed to be independent of the contents of the population to provide a consistent objective measure. Such objectivity entails significant knowledge about the environment which suggests either the problem has previously been solved or other non-evolutionary techniques may be more efficient. Furthermore, for many complex tasks an independent fitness function is either impractical or impossible to provide. In this paper, we demonstrate that *competitive fitness functions*, i.e. fitness functions that are dependent on the constituents of the population, can provide a more robust training environment than independent fitness functions. We describe three differing methods for competitive fitness, and discuss their respective advantages.

1 INTRODUCTION

Competitive learning is a long standing topic in machine learning (Samuel, 1959; Tesauro, 1992). Interest for using competition in machine learning tasks stems from a desire for a program to discover the strategic nuances of a complex task directly from the first principles of interaction. Appropriate complex structures arising purely from the "physics" of the task environment would be the ultimate validation of machine learning capability. Such is the essence of emergent computation (Forrest, 1991).

A competitive learning process encourages an evolutionary development such that as new strategies are developed by one learner, its opponent adjusts its abilities and discovers new strategies of its own. This strategic "arms race" ideally increases the overall ability of the learners until they reach near optimal abilities. Unfortunately, there is a possibility of competitive learners falling into local minima where important task configurations are underexplored, thereby leading to immature inductions (Tesauro, 1992; Epstein, 1992).

One manner of forcing the competition into a variety of representative strategic situations is to introduce a nondeterministic element into the competition. For instance, Tesauro (1992) describes a neural network that learns to play backgammon at an expert level purely from self-competition, i.e., the network plays against itself and updates its weights at the end of each game. Such a reflexive environment would usually *maximize* the potential for the network to fall into poor strategic minima if it weren't for the natural non-determinism of the dice roll in the backgammon task (Tesauro, 1992; Epstein, 1992). The roll of the dice occasionally forces play into board configurations that have never been visited in any previous game and consequently provides feedback to refine the network. In tasks where there is no natural source of non-determinism, an artificial random element must be introduced.

In genetic algorithms, the population represents a natural source of diversity that, while not entirely random, can be recruited to create non-deterministic competitive environments. Competitive populations have been underexploited in genetic algorithms; exactly why is unclear. One possibility is that competitive environments are thought to be too unstructured to guide a population toward a particular goal unless the goal is suitably vague or non-existent. Such an attitude could explain the relegation of competitive evolutionary environments to the Artificial Life community (e.g., Ray, 1992; Lindgren, 1992). Another possibility is that competition is considered too expensive for practical problems; that it requires too many evaluations of population members to determine an accurate ranking. Without an accurate enough ranking, the natural dynamics of the evolutionary process might be compromised.

In this paper, we enumerate the advantages of competitive fitness functions and show them to be a powerful unexplored resource in genetic algorithms. We describe three types of competitive fitness functions as examples. The first is a full competition model used in Axelrod (1989) to evolve strategies for the Iterated Prisoner's Dilemma. The second is a bipartite competition in Hillis (1992) used to evolve sorting networks. The third we introduce in this paper and demonstrate its ability to evolve more robust

genetic programs (Koza, 1992; Koza, 1992b) than standard non-competitive fitness functions. We conclude with a discussion of the various beneficial properties of competitive fitness functions.

2 COMPETITIVE FITNESS FUNCTIONS

The standard fitness functions used in genetic algorithms are exemplified by the functions studied in DeJong (1975). Such functions return the same fitness for an individual regardless of what other members are present in the population. Their independence from the population's composition allows these functions to provide an accurate and consistent fitness measure throughout the evolutionary process.

While global accuracy is easily computed when evolving solutions for many simple optimization problems, it is often impractical for problems with higher complexity. The difficulty stems directly from the objectivity of the fitness function, since objective accuracy often comes only at the cost of significant knowledge about the search space. For instance, consider the expense of a standard fitness function for evolving an optimal strategy for a particular game. Such a function would need to test members of the population against all possible strategic situations to garner an objectively accurate measure. For anything but a trivial game such a computation is immense. If a suitable "expert" strategy were available, an independent fitness function could still be constructed, however, the evolved solutions would only be "optimal" with respect to this "expert" rather than the original task.

In contrast, a *competitive fitness function* is any calculation for fitness that is *dependent* on the current population to any degree. The dependency could be relatively minimal, such as on a single population member, or fairly comprehensive in functions that use the entire population to determine a single strategy's fitness. In essence, competitive fitness is the original intention behind a fitness function since it provides a measure of an individual's ability relative to the current population rather than relative to the global optimum.

Axelrod (1989) experiments with both an independent and a competitive fitness function to evolve strategies for the Iterated Prisoner's Dilemma. For the independent fitness function, a weighted sum of the scores against eight preselected strategies was used, where the weights and representative strategies were selected on the basis of knowledge gained from previous experiments (Axelrod, 1984). In the competitive fitness function, Axelrod (1989) tests every population member against every other population member, which presents an adaptive developmental environment for the population. However, there is no mention in the study about the relative abilities of the evolved strategies. A schematic of the competitive pairings associated

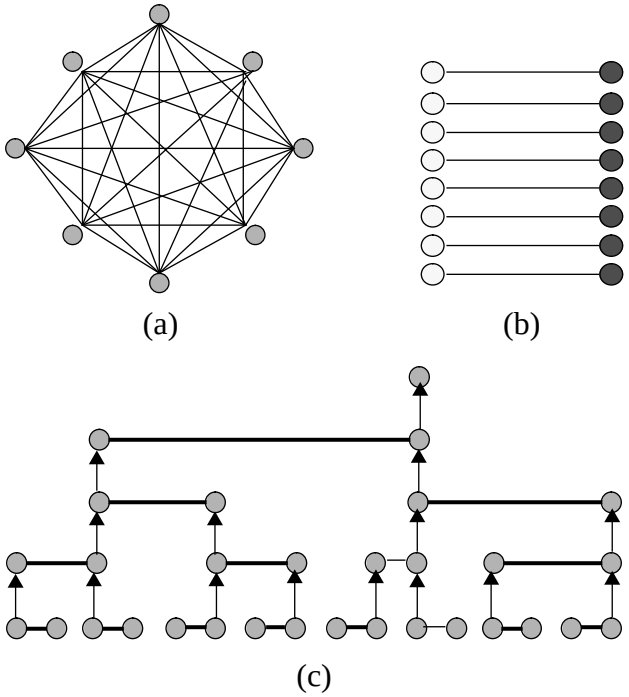

Figure 1: Three types of competitive fitness functions. (a) Full competition used in Axelrod (1989); (b) Bipartite competition used in Hillis (1992); and (c) Tournament fitness with each horizontal line designating a competition and each upward arrow designating the winner progressing in the tournament.

with this method appears in Figure 1a. Assuming the size of the population is n, the number of competitions executed in a generation is n^2. When the task to be solved is quite complex and requires a large population or a significant number of generations, this number of competitions per generation may be prohibitive.

Hillis (1992) demonstrates a dependent fitness function with an interesting competitive approach. The problem explored in Hillis (1992) is to evolve a sorting network for any arrangement of 16 integers with as few position exchanges as possible. Notice that this task is not so different from a game. The sorting networks represent various strategies and the 16! potential arrangements of integers represent the various board configurations. Clearly, using a fitness function which tests all possible permutations on each sorting network is impractical. Additionally, a static subset of permutations would clearly encourage solutions that sort only the chosen subset. Hillis (1992) reports that even using a randomly selected subset of permutations that changes every generation does not provide a sufficient environment to evolve adequate sorting networks.

In order to maintain a consistently difficult set of permutations to evaluate the sorting networks, Hillis (1992) creates a second population for the experiment. Each member

of the second population encodes a small set of permutations to be sorted by one of the sorting networks of the first population with *both* populations evolving from generation to generation. Fitness for the population of sorting networks is defined to be how well the member sorts the various permutations within the associated member of the second population. The fitness of a member in the second population is a measure of how *poorly* the sorting network sorts the set of permutations it contains. This bipartite competition is illustrated in Figure 1b. With this fitness function the system evolved a sorting network with only 61 position exchanges, which is a single exchange worse than the best known sorting network for 16 numbers.

Assuming the sizes of the populations are the same and when combined equal *n*, the bipartite competition in Hillis (1992) uses a total of *n*/2 competitions each generation. This is far fewer than a full competition, as in Axelrod (1989). However, while the fitness function used in Hillis (1992) is an example of a competitive fitness function, there is no method for determining which member of the population is the best sorter. Because each sorting network competes against a single member of the second population there is no basis of comparison *between* sorting networks. The score received by a sorting network is relative to the difficulty of permutations it attempted and each sorting network sees distinct sets of permutations. In addition, the bipartite nature of the competition model used in Hillis (1992) may be unnatural for some problems.

Pitting evolving members of a population against each other to determine fitness creates an interesting tension in the genetic algorithm. For instance, while the population of sorting networks in Hillis (1992) is adapting to the specific permutations it is being tested against, the population of permutations is searching for the set that forces the sorting networks to perform as badly as possible. In order for the sorting networks to reproduce from generation to generation consistently, they must generalize their sorting ability rather than encode for a specific subset of permutations. The need to compensate for the continuing diversity in the permutations inspires generalization in the sorting networks. A similar dynamic occurs in the competitive single population of Axelrod (1989). In the following section, we describe a third type of competitive fitness function that uses a single homogenous population with fewer competitions than full competition and still permits a best member to be identified.

3 TOURNAMENT FITNESS

Rather than exhaustively testing each member against the rest of the population, in *tournament fitness* a single elimination, binary tournament is run to determine a relative fitness ranking. Initially, the entire population is in the tournament. Two members are selected at random to compete against each other with only the winner of the competition progressing to the next level of the tournament. Once all the first level competitions are completed, the winners are randomly paired to determine the next level winners. The tournament continues until a single winner remains The fitness of a member of the population is its height in the playoff tree, the player at the top is then the best player of the generation. The competitive parings for tournament fitness are illustrated in Figure 1c. The hierarchical nature of the ranking is strictly enforced, ties being broken by random selection. In the case that the number of competitors at a level is odd, a single population member is passed to the next level of the tournament without a competition, effectively receiving a "bye" for that round. The total number of competitions for a population of size *n* is:

$$\sum_{i=1}^{\lceil \log(n) \rceil} \left\lceil \frac{n}{2^i} \right\rceil = n - 1 \qquad \text{(EQ 1)}$$

which is one fewer comparison than required to play each member of the population against a single "expert" strategy in a comparable independent fitness function.

Quantification of performance on the task is unimportant when using tournament fitness; all that is required is a concept of "better" to compare two strategies. This removes all need for determining exactly how much better one player is than another - the resulting tournament hierarchy is sufficient information for reproduction. Unless the competition, i.e., the measure of "better", is noisy, an optimal player will always win the tournament. However, if the environment is suitably complex and an optimal strategy is not in the population, it is possible for an average or even a comparatively poor strategy to win the tournament for a particular generation. Thus this competitive fitness function can contain a level of noise associated with its ability to rank any given population. How accurately the tournament ranks the population is dependent upon the set of competitors met. For instance, if the best player in the population competes in the initial round of the tournament with the second best member, only the best player will move up the hierarchy with the second best player being assigned the minimal fitness.

Fortunately, the inherent noise of tournament fitness functions is not a serious problem given that the fitness ranking is being created to decide the proportions for reproduction. Consider that the worth of a single competition, in terms of reproduction, is inversely proportional to how high in the tournament the competition occurs. In other words, the higher the level of the competition, the less it is worth in terms of reproductive advantage. For example, consider a single competition in the initial round of the tournament. The first competition determines which half of the rankings the two competitors will reside. The loser will have a fitness that will place it in the lowest 50% of the population's ranking. The winner will at least be in the upper 50% giving it a reasonable chance for reproducing. With each successive round of competitions, exponentially decreasing subsets of the population are divided into winners and losers until the last competition decides between

best and second best for the generation. At this level, the difference in the probability of reproducing is negligeable with any reasonably sized population.

Once a tournament has been run, any standard selection method can be used to designate parents for the next generation. Because all the population members that lost at the same level of the tournament will have the same fitness values, tournament fitness naturally de-emphasizes their worth relative to each other. This is more beneficial than over-committing to an erroneous complete ordering of the population. Selecting between members with the same fitness must be at random which promotes better mixing of the alleles and discourages premature convergence. In fact, in many situations tournament fitness will naturally discourage convergence since as a particular strategy becomes too numerous it will be forced to literally compete against copies of itself. This is akin to a predator/prey system where the prey has been hunted to such low population levels that the predators are forced to feed on each other.

Tournament fitness sets up the same oppositional tension as in Hillis (1992), but in a more comprehensive manner and requires only a single population. Because it is unlikely that all strategies have been represented in past populations, complex developing strategies may contain flaws that may be exploited by a variety of simpler players. This is much like a relatively good chess player being beaten by a novice who only knows the "fool's mate" strategy. Since for any complex strategy there may be numerous ways for it to be beaten, a complex ecology of strategies for the task can develop during the course of the run with only the most robust strategies consistently appearing at the top of the tournament. These complex strategic ecologies are similar to those described in Ray (1992) and Lindgren (1992) but are relative to the specific task being solved.

4 EXPERIMENTS

To test the tournament fitness function, we ran several experiments using our Genetic Library Builder (GLiB) (Angeline and Pollack, 1993; Angeline and Pollack, 1992) modified to perform a hierarchical tournament as described in the previous section. GLiB is based on Koza's genetic programming paradigm (GPP) (Koza, 1992; Koza, 1992b) which uses a primitive programming language arranged in expression trees for the representation of population members. The primitive language relies on a simple and uniform syntax to remove the possibility of generating a non-viable expression trees during recombination. Crossover in GPP simply swaps randomly selected subtrees between the expression trees. Koza has demonstrated the ability of GPP to evolve solutions for a significant number of engineering problems (Koza, 1992). Our system, GLiB, is an extension to GPP that induces new language elements by non-deterministically creating subroutines that are protected from further alteration by recombination. New subroutines are formed with a mutation operator called *compression*, as shown in Figure 2. The result of evolution in GLiB is a modular program to perform the task. For additional information on GLiB, see Angeline and Pollack (1993).

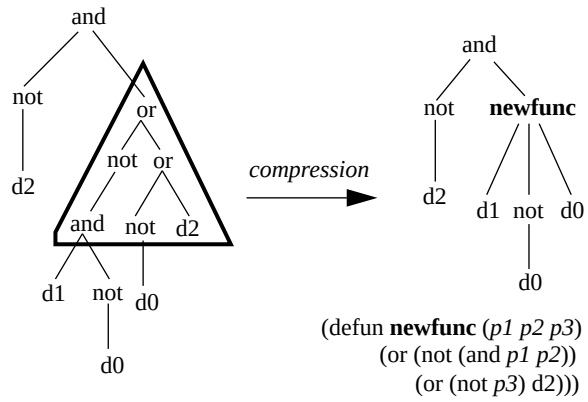

Figure 2: Compression of tree representation used in GLiB. The subtree is removed from the individual and replaced by a new function call defined with the removed subtree. The expansion of a compressed function reverses the process by replacing the compressed function name with the original subtree.

The subject of our experiments is the game of Tic Tac Toe (TTT) also called Noughts and Crosses. Figure 3 outlines the primitive language we use for evolving modular TTT programs. The primitives ***pos00*** to ***pos22*** are the data points representing the nine positions on the TTT board. For the remaining primitives, the return value is either one of these positions or NIL. The binary operators ***and*** and ***or*** each take two arguments. When both arguments are non-NIL, ***and*** returns the second. If either argument is NIL then

pos00	***pos01***	***pos02***
pos10	***pos11***	***pos12***
pos20	***pos21***	***pos22***

pos00 .. pos22 - board positions
and - binary LISP "and"
or - binary LISP "or"
if - if <test> then <arg1> else <arg2>
open - returns <arg> if unplayed else NIL
mine - returns <arg> if player's else NIL
yours - returns <arg> if opponent's else NIL
play-at - places player's mark at <arg>

Figure 3: Primitives used to evolve modular programs to play Tic Tac Toe.

it returns NIL. ***Or*** returns the first non-NIL argument and returns NIL otherwise. The ***if*** primitive is the standard conditional statement. It returns the value returned by its second argument when the <test> is non-NIL otherwise it returns the value returned by the third argument. The ***play-at*** primitive takes a single argument. If the argument is a position and no player has placed a mark there, then the current player's mark is placed at that position and the turn is halted. Otherwise, ***play-at*** will return whatever it is passed. Finally, the operators ***mine***, ***yours*** and ***open*** take a position and return it when the mark in that position fits the test. Otherwise, they too return NIL.

The language outlined above is general enough to cover any number of two player games on a nine position board. Consequently, there is no guarantee that a random program in this language will observe the rules of TTT or even place a single mark on a TTT board. If the program does not make a valid move during a game, then its turn is forfeited, providing a significant advantage for its opponent. We consider legal moves to be a part of the environment's complexity and consequently should be induced by our learning method.

Obviously, the choice of primitive language in GLiB and GPP dictates how difficult a given concept is to learn. The primitive language for TTT above is more general than necessary to solve the task for two reasons. First, the typical approach in machine learning is to separate the learning of the control task from the learning of the evaluation task, often with the control task being assumed (e.g., Tesauro, 1992; Berliner, 1977). We feel that such a separation inhibits the complete learning process. Our learners must acquire both control and evaluation abilities within the same structure at the same time. Consequently, we do not expect our programs to induce the complete concept but only portions. Which portions of the complete task they do acquire and how the task is generalized often illuminates much more about the learning process than complete acquisition. Second, we wish to study the acquisition of higher-level features associated with the tasks rather than provide them a priori as in most learning systems (e.g., Samuel, 1966; Rumelhart et. al., 1986; Tesauro, 1990). This goal requires a representation in which these features can be discovered by the learner during the learning process, such as the language described above.

To compare the ability of the programs evolved by standard independent fitness functions and tournament fitness, we created a collection of "expert" TTT players of varying strategic ability. The three experts used in this experiment were RAND, NEAR and BEST. RAND simply chooses a legal position at random to determine its move for a given board configuration. At the other extreme is BEST which chooses the optimal position to play on each move. No strategy, evolved or otherwise, can win a game against BEST. NEAR, the third algorithm, performs near optimally except that it can be forked by its opponent. A fork is any TTT board configuration where a player has more than one winning move, guaranteeing an opportunity to win on its next turn. Unless forked, NEAR will either draw or beat its opponent. Both NEAR and BEST nondeterministically choose between equal moves in order to force more robust play from the developing programs.

Our experiments cover four different learning situations. For the first three, we use an independent fitness function consisting of one of the above experts. A single competition against an expert is scored by the number of moves the evolved program makes with a 5 point bonus for a draw and a 20 point bonus for a win. A program's average score over four games is its fitness. These runs represent a range of independent fitness functions that might be used for this task. In fact, NEAR is very similar to the expert we used in earlier experiments with GLiB to induce a modular genetic program that could fork (Angeline and Pollack, 1993).

The final experiment uses the tournament fitness function as described in the previous section and is labeled POP in the results. Scoring a single competition between two programs was as described above. A program was deemed "better" than its opponent if it had the greater score after two games, with each player taking the first move in one game. As described in the last section, if the scores were equal the winner of the competition was chosen at random.

Each of the experiments used a population size of 256 and ran for a total of 150 generations. All experiments used roulette wheel selection with linear scaling (Goldberg, 1989) and a scaled fitness maximum of two. Other than the method of training, all other factors were equal. The parameter settings used were as listed in Angeline and Pollack (1993).

In order to observe the training ability of each of the various fitness functions, the evolved program with the best fitness from each experiment played a total of 2000 games against each of the three experts to evaluate its abilities. Results for the four experiments are shown in Table 1. As can be seen from the table, none of the evolved modular programs induced the optimal TTT program. This is due in part to our low level primitive language and our insistence that GLiB acquire both control and evaluation in the same program. More interesting is the difference in ability of the programs induced using the three expert algorithms. The program evolved using RAND is fairly poor, only able to beat the RAND expert a little more than half the time while appearing totally incompetent against the better experts. NEAR's evolved protege embodies a more able concept that displays a broader ability to compete. The program acquired using BEST appears to have induced the ability to draw opponents in many situations but is weak in its ability to win, even against RAND.

Of even more interest is the final experiment which used tournament fitness. The program induced in this experiment is clearly a more robust player if not more proficient than any of those induced using one of the experts. The fact that the program evolved by tournament fitness was the only evolved program that could draw BEST, and did

Table 1: Performance of best evolved program from each experiment against the various "experts."

Fitness Function Used	*Evolved Program vs. RAND*			*Evolved Program vs. NEAR*			*Program vs. BEST*	
	Wins	Draws	Losses	Wins	Draws	Losses	Draws	Losses
RAND	1125	0	875	0	0	2000	0	2000
NEAR	802	104	1094	144	123	1733	0	2000
BEST	310	535	1155	0	360	1640	0	2000
POP	781	471	748	61	588	1351	481	1519

so fairly frequently, shows it acquired a more sophisticated algorithm for TTT than those induced by the independent fitness functions.

An important question raised by these experiments is why none of the programs evolved using the independent fitness functions could draw BEST. This is straightforward if the quality of environment presented by each fitness function is considered in turn. First, RAND provides no pressure for a program to induce a complex winning strategy. Simply hard coding only three positions in a row will guarantee the program will win a few games against RAND. But such a strategy is easily thwarted by NEAR and BEST. When using BEST as the strategy in the independent fitness function, no program ever receives positive feedback for making three in a row, and consequently none of the evolved programs induce this ability very broadly. Programs perform well against BEST when they can play almost anywhere on the board and maximize the number of moves they make before being beaten. This translates into a program that plays several moves but can't put those moves together to form a winning combination. When NEAR is used to evolve solutions, the programs emphasize setting up forks, since this is NEAR's only flaw and the scoring function emphasizes wins strongly. Once a program evolved using NEAR is able to fork it and win, there is no reason to improve and little pressure to develop the ability to consistently draw an opponent. This type of strategy is sufficient to perform well against RAND, but is easily defeated by BEST which protects itself from being forked.

On the other hand, the program evolved by tournament fitness is forced to play against a number of differing strategies from the collection of strategies developing in the population. Some of these strategies will be simplistic and provide little difficulty while others will be equally competent on the task. We explore this and other advantages of competitive fitness functions in the next section.

5 DISCUSSION

One of the primary advantages of competitive fitness functions is their ability to adapt to the level of complexity of the population. In the experiment above, NEAR and BEST present difficult adversaries from the beginning and do nothing to identify preferred performance in the early populations. Such a large difference between the ability of the initial population and the strategy in the independent fitness function can inhibit the evolution of solutions if not compensated for in the function explicitly. For instance, in previous experiments with NEAR, we awarded a fitness bonus for evolved programs that successfully blocked a win (Angeline and Pollack, 1993). With a competitive fitness function, this problem is removed since the population is its own measure. As the ability of the individual members of the population increases on the task, the difficulty of the fitness function evolves with them. Since the function is dependent on the population, it tracks through the population's non-linear development without the need for measuring the average member's ability explicitly. Additionally, the open-endedness of a competitive fitness function is dependent on the open-endedness of the representation for the population members.

As we stated above, ecologies of strategies develop in competitive fitness functions that provide a more consistently difficult environment than independent fitness functions. As evolution continues, the ecological balance will shift in the population to take advantage of exploitable strategic niches. The question still remains as to what prevents the population from wandering aimlessly through the space of strategies rather than moving towards more complex solutions. Given that the population maintains some level of diversity, this is straightforward. Because there are many differing strategies that could be met in any generation, only solutions that can perform well against a number of them will consistently be in the upper tiers of the tournament and be able to continually reproduce. The constantly changing competitive environment forces the developing programs to generalize their abilities, as in Hillis (1992).

One advantage for tournament fitness over the bipartite competition used in Hillis (1992) is that proper counterexamples of various strategic difficulty are evolved within a single population removing the need for distinct populations to be maintained and separate fitness functions to be

constructed. The adaptability of the content of the single population may be more beneficial to the evolutionary development of solutions than a predetermined bipartite population. Of course, which form of competition is appropriate for a given task will be somewhat problem dependent.

6 CONCLUSIONS

In this paper, we argue that competitive fitness functions have many advantages over the independent functions that are typically used in genetic algorithms. Not the least of these advantages is that a competitive fitness function requires only a minimal understanding of the search space for a complex task. This removes the need for task knowledge that may be extremely difficult to engineer out of the problem. Furthermore, by employing all strategies represented in the population as potential counterexamples, the fitness function automatically adapts to the nuances of both the individual problem and the specific progression of populations in a particular run. Finally, the experiments above and those in Hillis (1992) demonstrate that using the population as the reservoir for comparison is preferable to using an exemplar for the task when an objective measure of fitness is unavailable. Using a competitive environment permits the evolutionary process to acquire a more general solution that approximates global optimality relative to the task rather than relative to the provided exemplar.

Acknowledgments

This research is supported by the Office of Naval Research under contract #N00014-92-J-1195. We are indebted to David Fogel and two anonymous reviewers who made informed comments on this work. We also thank Greg Saunders and Ed Large for additional feedback and proofreading help.

References

Angeline, P. and J. Pollack, (1993) "Coevolving high-level representations." To Appear in *Artificial Life III*.

Angeline, P. and J. Pollack, (1992) "The evolutionary induction of subroutines." *The Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington Indiana.

Axelrod, R., (1989), "Evolution of strategies in the iterated prisoner's dilemma." *Genetic Algorithms and Simulated Annealing*, L. Davis editor, Morgan Kaufman

Axelrod, R., (1984), *The Evolution of Cooperation*, Basic Books.

Berliner, H., (1977), "Experiences in evaluation with BKG- a program that plays backgammon", *Proceedings of IJCAI*, 1977.

DeJong, K., (1975), *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral dissertation, University of Michigan.

Epstein, S., (1992), "Learning expertise from the opposition: the role of the trainer in a competitive environment.", *The Proceedings of AI 92*, 236-243.

Forrest, S., (1991), "Emergent computation: self-organizing, collective, and cooperative phenomena in natural and artificial computing networks", In *Emergent Computation*, S. Forrest editor, Cambridge, MA: MIT Press.

Goldberg, D., (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley Publishing Company, Inc.

Hillis, D., (1992), "Co-evolving parasites improves simulated evolution as an optimization procedure", In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer and S. Rasmussen. Reading, MA: Addison-Wesley Publishing Company, Inc.

Holland, J., (1975), *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: The University of Michigan Press.

Koza, J., (1992), *Genetic Programming*, Cambridge, MA: MIT Press.

Koza, J., (1992b), "Genetic Evolution and Co-Evolution of Computer Programs." In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer and S. Rasmussen. Reading, MA: Addison-Wesley Publishing Company, Inc.

Lindgren, K., (1992), "Evolutionary Phenomena in Simple Dynamics", In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer and S. Rasmussen. Reading, MA: Addison-Wesley Publishing Company, Inc.

Ray, T., (1992), "An Approach to the Synthesis of Life." In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer and S. Rasmussen. Reading, MA: Addison-Wesley Publishing Company, Inc.

Rumelhart, D., Smolensky, J., McClelland, J., and Hinton, G., (1986), "Schemata and sequential thought processes in PDP models." In *Parallel Distributed Processing: Explorations into the Microstructure of Cognition, Volume 2*, D. Rumelhart, J. McClelland and the PDP Research Group eds., Cambridge, MA: MIT Press.

Samuel, A., (1966), "Some studies in machine learning using the game of checkers, II - recent progress." *IBM Journal of Research and Development 11*, 601-617.

Samuel, A., (1959), "Some studies in machine learning using the game of checkers.", *IBM Journal of Research and Development 3*, 210-229.

Tesauro, G., (1992), "Practical issues in temporal difference learning", *Machine Learning* 8, 257-277.

Tesauro, G., (1990),"Neurogammon: a neural network backgammon program." *IJCNN Proceedings III*, 33-39.