Similarities Between Co-evolution and Learning Classifier Systems and Their Applications

Ramón Alfonso Palacios-Durazo[1] and Manuel Valenzuela-Rendón[2]

[1] Lumina Software,
apd@luminasoftware.com,
http://www.luminasoftware.com/apd
Washington 2825 Pte C.P. 64040,
Monterrey N.L., Mexico
[2] Centro de Sistemas Inteligentes,
Instituto Tecnológico y de Estudios Superiores de Monterrey,
valenzuela@itesm.mx,
http://www-csi.mty.itesm.mx/~mvalenzu
Sucursal de Correos J, C.P. 64849
Monterrey, N.L., Mexico

Abstract. This article describes the similarities between learning classifier systems (LCSs) and coevolutionary algorithm, and exploits these similarities by taking ideas used by LCSs to design a non-generational coevolutionary algorithm that incrementally estimates fitness of individuals. The algorithm solves some of the problems known to exist in coevolutionary algorithms: it does not loose gradient and is successful in generating an arms race. It is tested on MAX 3-SAT problems, and compared to a generational coevolutionary algorithm and a simple genetic algorithm.

1 Introduction

Coevolution refers to the simultaneous evolution of two or more genetically distinct species. This evolution may be such that the species cooperate or compete. The application of competitive coevolution to problem solving has been of interest in the genetic algorithm research community because *competition*, in its most general sense, encourages the generation of better *competitors*. The implementation of this idea, in the form of a competition between possible solutions and instances of a problem, has been reported with varying degrees of success: Hillis [5] used a coevolutionary approach to find sorting nets for sorting arrays of 16 elements. He implemented a coevolutionary algorithm where sorting nets competed against permutations (each permutation is an instance of a sorting problem). Pollack and Rosin [8,10] generated playing strategies for games, Ficici [4] used a coevolutionary algorithm for generating predictors and Cliff [3] for evolving persecution and evasion strategies. Many other applications have been reported.

In the competitive coevolutionary approach to problem solving, the individuals in two populations are made to compete to determine their fitness. The

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 561–572, 2004.

fitness of an individual depends on its performance against opponents in the current generation of the competing population [9]. Since the competition is similar to the relationship between predator and prey, or parasites and their hosts, the populations in a coevolutionary algorithm are usually named as such. For the purpose of this article, *hosts* will refer to a population of possible solutions, and *parasites* will refer to a population of instances of or parts of a problem.

The central idea of coevolution lies in the fact that the fitness of an individual depends on its performance against the current individuals of the opponent population. This is the competition that we hope will generate better solutions. However, this simple idea gives rise to not-so-simple implications: the most simple form of coevolution is inherently unstable, costly in computer effort, and holds no guarantee it will work [12,9,4].

Unnoticed by most researchers are the similarities between coevolutionary algorithms (CEAs) and learning classifier systems (LCSs). The purpose of this article is to highlight these similarities, to address the issue of *loss of gradient*, and to introduce a new paradigm in CEAs based on the way LCSs work: incrementally adjusting fitness of individuals in a non-generational genetic algorithm. We devised a CEA based on these premises and test and compare it with a simple generational CEA and a simple genetic algorithm (SGA).

The remainder of the article is organized as follows: Section 2 defines a simple CEA and describes goals and obstacles CEAs face. Section 3 derives the incremental and non-generational algorithm. Section 4 describes the experimentation and results and Section 5 concludes this article.

2 Goals and Challenges of Coevolutionary Algorithms

In its most simple form, coevolution is implemented in the manner shown in Figure 1 in an algorithm we will call the *simple coevolutionary algorithm* (SCA).

Variations of the SCA are used by researchers [8,4], mostly by changing the form of the competition. The manner in which a competition is performed depends on the application as well as how the "winner" is determined. Some applications may not have an absolute winner, more likely a degree of winning, or a score. The way the fitness of an individual is calculated is also application dependant. The number of wins may be substituted by a sum of scores or other appropriate measures of performance.

Likewise, the opponents chosen for competition are not necessarily the complete population. Many researches use a subset of the opponent population for purposes of fitness calculation, either randomly or specifically selected. The SCA, and the variations found to be used by researchers encourage the selection and reproduction of individuals that perform well against the fitness landscape represented by the opponent population.

Watson and Pollak[12] clearly stated three interesting and useful goals and potential pitfalls for a coevolutionary algorithm:

– Providing a "hittable" target (gradient). The individuals in both populations should be relatively of the same quality and should improve roughly at the

```
(*Initialize populations*)
Generate random host population
Generate random parasite population

(* Main cycle*)
repeat

  (* Competition cycle*)
  for-each p ∈ parasites
    for-each h ∈ hosts
      h and p compete
    end-for
  end-for
  Fitness of parasites and hosts calculated
    based on competitions won

  One generation of a GA is applied to hosts
    selection, crossover, and mutation

  One generation of a GA is applied to parasites
    selection, crossover, and mutation

until termination criteria met
```

Fig. 1. The simple coevolutionary algorithm

same rate. If one population outperforms the other drastically, the latter will have no opportunity to learn and improve.
- Providing a relevant target (focusing). The opponents must represent something worth beating. Different strengths and weaknesses should emerge. These strengths are also known as *specialization niches* and have been described by Rosin and Belew [10].
- Providing a progressive moving target (open-endedness). The gradual improvement of both populations should be such that individuals *progress*. It is possible for individuals to "forget" strengths and fall into what is known as a mediocre stable state [1].

Methods and techniques have been proposed to compensate the challenges faced by CEAs [9], mostly trying to solve each known problem one at a time.

3 An Incremental and Non-generational Coevolutionary Algorithm

In this article we take lessons learned from the field of learning classifier systems to propose a different approach to implementing coevolution that we call the *incremental coevolutionary algorithm* (ICA).

In ICA, the importance of the coexistence of individuals in the same population is as great as the individuals in the opponent population. This is similar to the problem faced by learning classifier systems (LCSs) [6] and multiobjective optimization as done by Valenzuela-Rendón and Uresti-Charre [11]. We take ideas from these algorithms and put them into the ICA. A non-generational genetic algorithm is used and an incremental approach is taken to estimate the fitness of an individual. With this new design, we are able to avoid some of the pitfalls faced by other CEAs.

The motivation to use ideas from LCSs comes from the similarity between the challenges LCSs and CEAs face:

- The performance of the algorithm depends on the coexistence of individuals in the population.
- A generational algorithm in which each new generation is composed of new individuals is very disruptive.
- There is a need both to explore and to remember.

In a simple genetic algorithm, the objective is to find the individual that has the best possible fitness as defined by the objective function. In a CEA, the fitness landscape depends on the opponent population, therefore it changes over time (every generation, in fact). The evolution of hosts depends on the existence of parasites. The fitness landscape presented by the parasites determines how the population of hosts is formed. Likewise, the population of parasites depends on the fitness landscape of the population of hosts. The individuals selected for reproduction are those more promising to perform better against the fitness landscape represented by the opponent population. However, if the complete population of parasites and hosts are recreated in every generation, the offspring of each new generation face a fitness landscape unlike the one they where bred to defeat. Clearly, a purely generational approach to coevolution can be too disruptive.

As the fitness landscape presented by the opponent population gradually changes, so does the value of an individual. Instead of calculating or estimating the fitness based on competitions against the current opponents, it makes more sense to incrementally adjust the fitness of an individual as it faces each new generation of opponents.

These two ideas define the main approach of the ICA: the use of a non-generational genetic algorithm and the incremental adjustment of the fitness estimation of an individual. The general design of the ICA is presented in Figure 2.

The manner in which the fitness of host and parasite are adjusted must be designed to insure that the fitness of parasites and hosts behave as desired, and can be done in a manner very similar to the adjustment of strengths of classifiers in an LCS.

If we call the result of a competition a *score*, and hosts are set to minimize the score, and parasites are set to maximize it, then the general equation for adjusting the fitness of a parasite should include an increment proportional to the score, but should also include some form of fitness reduction so the value does not grow indefinitely. Therefore, the fitness of a parasite (S_p) at a time

```
(*Initialize populations*)
Generate random host population
Generate random parasite population

(* Main cycle*)
repeat
  (* Competition cycle*)
  for c ← 1 to determined number of cycles
    p ← parasite selected proportionally to fitness
    h ← host selected proportionally to fitness
    h and p compete, their fitness is adjusted
      incrementally, depending on the score
  end-for c

  (* 1 step of a GA in the parasite population*)
  Select parasite parents proportionally to fitness
  Create parasite by doing crossover
    and mutation
  Delete parasite with worst fitness and
    substitute with new parasite

  (* 1 step of a GA in the host population*)
  Select host parents proportionally to fitness
  Create host by doing crossover
    and mutation
  Delete host with worst fitness and
    substitute with new host

until termination criteria met
```

Fig. 2. General form of the incremental coevolutionary algorithm

$t+1$ depends on the fitness at time t, plus some form of reward for obtaining the *score*, minus a cost or taxation for competing:

$$S_p(t+1) = S_p(t) + \text{Reward} - \text{Taxation} \tag{1}$$

The reward should be in some way proportional to the score and the term for taxation is defined as a portion of an individuals fitness, similar to the manner done by LCSs:

$$\text{Reward} = C_1(\text{score}) \tag{2}$$

$$\text{Taxation} = C_2 S_p(t) \tag{3}$$

The value of the score should be limited in magnitude to avoid unstable behavior. We define a function $A(\text{score})$ that will limit the values of score that can be used in equation 2. We use $\tanh(x)$, because $\tanh(0) = 0$ and it grows

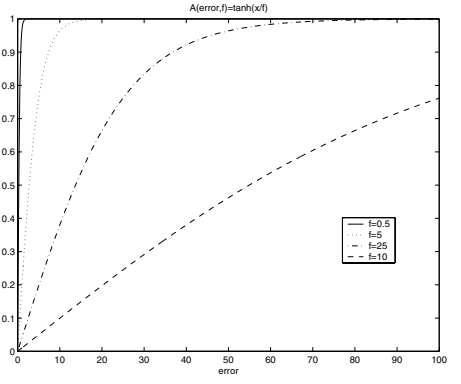

Fig. 3. Effect of the scaling factor

asymptotically to 1. Furthermore, it is possible to change the scale of $\tanh(x)$ adding a scaling factor h:

$$A(\text{score}, h) = \tanh\left(\frac{\text{score}}{h}\right) \tag{4}$$

The effect of the scaling factor can be seen in Figure 3. Equation 1 can now be fully written as:

$$S_p(t+1) = S_p(t) + C_1 A(\text{score}, H) - C_2 S_p(t) \tag{5}$$

which can be rewritten as:

$$S_p(t+1) = (1 - C_2) S_p(t) + C_1 A(\text{score}, h) \tag{6}$$

The equation for adjusting the fitness of a host is very similar.

$$S_h(t+1) = S_h(t) + \text{Reward} - \text{Taxation} \tag{7}$$

Taxation is also defined in a similar manner to parasites and the host's reward should be inversely related to the score:

$$\text{Reward} = C_3(1 - A(\text{score}, h)) \tag{8}$$

$$\text{Taxation} = C_4 S_h(t) \tag{9}$$

The complete equation for adjusting the fitness of a host is:

$$S_h(t+1) = S_h(t) + C_3(1 - A(\text{score}, h)) - C_4 S_h(t) \tag{10}$$

also expressed as:

$$S_h(t+1) = (1 - C_4) S_h(t) + C_3(1 - A(\text{score}, h)) \tag{11}$$

Do these equations accomplish what is wanted? A simple steady state analysis can show they do. When the algorithm stabilizes, the fitness of an individual should not change, therefore making $S(t+1) = S(t)$ for both hosts and parasites in equations 6 and 11. Doing trivial algebraic manipulation, we have that

$$S_p = \frac{C_1}{C_2} A(\text{score}, h) \tag{12}$$

and

$$S_h = \frac{C_3}{C_4}(1 - A(\text{score}, h)) \tag{13}$$

The fitness of parasites that cause low values of score will tend to 0, but parasites that cause high values of score will tend to C_1/C_2. Likewise, hosts that have high values of score will tend to 0, but hosts that have low score will tend to C_3/C_4. We can make $C_1 = C_2 = \alpha$ and $C_3 = C_4 = \beta$ for formula simplification:

$$S_p(t+1) = (1-\alpha)S_p(t) + \alpha A(\text{score}, h) \tag{14}$$

$$S_h(t+1) = (1-\beta)S_h(t) + \beta(1 - A(\text{score}, h)) \tag{15}$$

The equations will make the fitness of hosts and parasites behave as desired.

Two final modifications make the ICA work better: each parasite and host is initialized with a fitness equal to 1. This makes each individual start with a perfect fitness (which is obviously not true). This has the effect distributing the initial competitions evenly among the first generation. Second, each new individual is given a fitness equal to the average of the fitness of its parents. This is considered an initial estimate of the fitness of the new individual.

The ICA has some interesting properties. First of all, it is not generational. Each new individual faces a similar fitness landscape than its parents. The fitness landscape changes gradually, allowing an arms race to occur, avoiding loss of gradient. Second, each population works as a memory, deterring a fall into a mediocre stable state. Third, the fitness landscape each population sees is somewhat distorted in an interesting way: the actual score is modified by the function $A(\text{score}, h)$, but also, since opponents are chosen proportional to their fitness, an individual has a greater chance of facing good opponents. If a particular strength is found in a population, individuals that have it will propagate and will have a greater probability of coming into competition (both because more individuals carry the strength, and because a greater fitness produces a higher probability of being selected for competition). If the population overspecializes, another strength will propagate to maintain balance. Thus, a natural sharing occurs. The formal definition of the ICA can be seen in Figure 4 with the list of parameters defined in Table 1.

The ICA was tested on three different MAX 3-SAT problems, and its performance compared to that of a generational coevolutionary algorithm and a simple genetic algorithm.

```
(*Define A(x,h)*)
A(x, h) ← tanh(x/h)

(*Initialize populations*)
Generate random host population
Generate random parasite population

(* Initialize fitness*)
for-each  p ∈ parasites
  S_p ← 1
for-each  h ∈ hosts
  S_h ← 1

(* Main cycle*)
repeat
  (* Competition cycle*)
  for c ← 1 to N_c
    p ← parasite selected proportionally to S_p
    h ← host selected proportionally to S_h
    score ← competition between h and p
    S_p ← (1 − α)S_p + αA(score, h)
    S_h ← (1 − β)S_h + β(1 − A(score, h))
  end-for c

  (* 1 step of a GA in the parasite population*)
  Select parasite parents (p_1 and p_2) proportionally to S_p
  Create parasite p_0 crossover and mutation
  S_p0 ← (S_p1 + S_p2)/2
  Delete parasite with worst fitness and substitute with p_0

  (* 1 step of a GA in the host population*)
  Select host parents (h_1 and h_2) proportionally to S_h
  Create host h_0 by crossover and mutation
  S_h0 ← (S_h1 + S_h2)/2
  Delete host with worst fitness and substitute with h_0

until termination criteria met
```

Fig. 4. Incremental coevolutionary algorithm

4 Experimentation and Results

The ICA was run five times on each of three 3-SAT benchmark problems from SATLIB[7] (a total of fifteen times). All three problems have 960 clauses and 225 variables and have solutions. The problems are considered hard due to phase transition as described by Cheeseman, Kanefsky, and Taylor [2].

The hosts were coded as a direct variable instantiation (1 bit per variable). The parasites consisted of 5 segment chromosome, each segment representing one

Table 1. Incremental coevolutionary algorithm parameters

Parameter	Description
h	Score scaling factor.
α	Maximum increment ratio in host fitness.
β	Maximum increment ratio in parasite fitness.
N_c	Number of competitions between each step of the GA.
P_{cp}	Crossover probability of the parasite population.
P_{mp}	Mutation probability of the parasite population.
T_p	Parasite population size.
P_{ca}	Crossover probability of the host population.
P_{ma}	Mutation probability of the host population.
T_a	Host population size.

of the 225 variables. The score of the competition between hosts and parasites was the percentage of clauses that contained any one of the variables represented by the parasite that were solved by the variable instantiation represented by the host.

The same representation and competition was used in a generational approach to coevolution. En each step of the generational coevolutionary algorithm, each host was made to compete with each parasite. Fitness was calculated as the percentage of clauses solved by each host (and the reciprocal for the parasite).

Finally, the same number of tests were applied to a simple genetic algorithm. In this case, the fitness of each individual was directly the number of clauses solved. A few pilot runs were done with each algorithm to help determine the best configuration. The parameters used in all three algorithms can be seen in Tables 2, 3, and 4.

Figure 5 shows the results of these experiments. The graph shows the average and standard deviation of the best solution found for all problems. For both coevolutionary algorithms, at each generation step, the individual with best fitness was evaluated to determine how many clauses of the complete 3-SAT problem it solved. It is possible that a better solution existed in the population, but only the best host was evaluated.

It can be seen than the ICA was able to find better solutions and maintain progress. The gradual change provided by the non-generational nature of the algorithm, as well as the incremental adjustment of fitness allows a gradient to be maintained. In contrast, the generational CEA was unable to provide such an environment, so no progress occurs. It can be seen that very few competitions are required between steps of the genetic algorithm in the ICA. Computational effort is saved by using these competitions with good individuals.

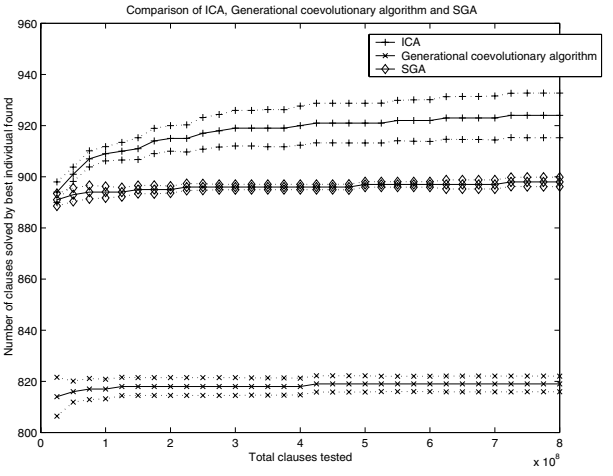

Fig. 5. Results of experimentation

Table 2. The ICA parameters

Parameter	Value
α	0.01
β	0.01
N_c	25
T_a	500
P_{ca}	0.95
P_{ma}	0.05
T_p	100
P_{cp}	0.65
P_{mp}	0.1
h	1

5 Conclusions

CEAs and LCSs have important similarities that have been unnoticed. A successful CEA can be designed following the guidelines of LCSs. The gradual change provided by a non-generational algorithm with incremental adjustment to fitness estimation provide an environment where a true competition can occur, with the desired benefits.

The incremental coevolutionary algorithm is quite robust, it does not fall into a stable mediocre state, it generates a successful arms race and niche specialization. It is simple and elegant and solves some of the problems known to happen in coevolutionary algorithms.

Table 3. Generational CEA parameters

Parameter	Value
T_a	500
P_{ca}	0.95
P_{ma}	0.05
T_p	100
P_{cp}	0.95
P_{mp}	0.5
Selection mechanism	Tournament (size 2)

Table 4. SGA parameters

Parameter	Value
Population size	500
P_c	0.9
P_m	0.05
Selection mechanism	Tournament (size 2)

References

1. Peter J. Angeline and Jordan B. Pollack. Competitive environments evolve better solutions for complex tasks. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 264–270, 1994.
2. Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, pages 331–337, 1991.
3. Dave Cliff and Geoffrey F. Miller. Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *European Conference on Artificial Life*, pages 200–218, 1995.
4. Sevan G. Ficici and Jordan B. Pollack. Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states. In Christoph Adami, Richard K. Belew, Hiroaki Kitano, and Charles Taylor, editors, *Artificial Life VI: Proceedings of the Sixth International Conference on Artificial Life*, pages 238–247, Cambridge, MA, 1998. The MIT Press.
5. W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, volume X, pages 313–324. Addison-Wesley, Santa Fe Institute, NM, 1992.
6. John Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. *Machine Learning: An Artificial Intelligence Approach*, 2, 1986.
7. Holgar H. Hoos and Thomas Stützle. Satisfiability library. http://www.satlib.org, May 2001. Version 1.4.4.
8. Jordan B. Pollack, Alan D. Blair, and Mark Land. Coevolution of a backgammon player. In C. G. Langton, editor, *Proceedings of Artificial Life V*, Cambridge, MA, 1996. MIT Press.
9. Christopher D. Rosin. *Coevolutionary search among adversaries*. PhD thesis, University of California, San Diego, San Diego, CA, 1997.

10. Christopher D. Rosin and Richard K. Belew. Methods for competitive co-evolution: Finding opponents worth beating. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 373–380, San Francisco, CA, 1995. Morgan Kaufmann.
11. Manuel Valenzuela-Rendón and Eduardo Uresti-Charre. A non generational genetic algorithm for multiobjective optimization. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 658–665. Morgan Kaufmann, 1997.
12. Richard A. Watson and Jordan B. Pollack. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 702–709, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.