Comparing Coevolution, Genetic Algorithms, and Hill-Climbers for Finding Real-Time Strategy Game Plans

Christopher Ballinger
University of Nevada, Reno
Reno, Nevada 89503
caballinger@cse.unr.edu

Sushil Louis
University of Nevada, Reno
Reno, Nevada 89503
sushil@cse.unr.edu

ABSTRACT

This paper evaluates a coevolutionary genetic algorithm's performance at generating competitive strategies in the initial stages of real-time strategy games. Specifically, we evaluate coevolution's performance against an exhaustive search of all possible build orders. Three hand coded strategies outside this exhaustive list provide a quantitative baseline for comparison with other strategy search algorithms. Earlier work had shown that a bit-setting hill-climber only finds the best strategies six percent of the time but takes significantly less time compared to a genetic algorithm that routinely finds the best strategies. Our results here show that coevolved strategies win or tie against hill-climber and genetic algorithm strategies eighty percent of the time but routinely lose to the three hand coded baselines. This work informs our research on improving coevolutionary approaches to real-time strategy game player design.

Categories and Subject Descriptors

I.2.m [**Artificial Intelligence**]: Evolutionary Computation; I.2.1 [**Applications and Expert Systems**]: Games

General Terms

Algorithms

Keywords

Coevolution, Real-Time Strategy Games, Build-orders

1. INTRODUCTION

Finding effective and robust strategies in Real-Time Strategy (RTS) games presents a challenging problem. RTS game players must compete for resources, build up an economy able to support their military force, expand their control over the map, and eventually destroy their opponent's base. Any advances in developing RTS game players will impact planning and execution in competitive industrial settings through the development of smart, realistic opponents.

Our overall goal is to create competent RTS game players and this paper compares CAs to GAs and HCs. However, several issues make such comparisons difficult. First, there is no single optimal strategy. RTS games are designed like rock-paper-scissors games and for every possible good strategy there is a counter strategy that can beat it.

In order to make such comparisons more meaningful, we also need to look at the space of all possible strategies, that is, we need to exhaustively list all possible strategies and evaluate their performance against our three baselines.

Our results show that the CA solutions beat GA generated and HC generated solutions 80% of the the time but fare much worse against the three hand coded baselines. Since the GA generated solutions do relatively well against the baselines, the CA tends to find solutions that beat the GA but not the baselines. These results inform our current research in developing a full game AI for WaterCraft, our open-source RTS clone of StarCraft.

2. METHODOLOGY

We created three hand-tuned "baseline" build-orders to compare against potential solutions generated by the CA, GA and HC. These baselines take 24-39 bits to encode, compared to the 15 bit solutions we develop in coevolution. This allows the baselines to present a challenging opponent that coevolution will have never encountered before, in contrast to the 15 bit solutions produced by the GA and HC, which the CA may still independently discover.

Our fitness calculation encourages players to find ways to destroy enemy units and structures, as shown in Equation 1. Where F_{ij} is the fitness of individual i against individual j, SR_i is the amount of resources spent by individual i, UD_j is the set of units owned by individual j that were destroyed, UC_k is the cost to build unit k, BD_j is the set of buildings owned by individual j that were destroyed, and BC_k is the cost to build building k.

$$F_{ij} = SR_i + 2 \sum_{k \in UD_j} UC_k + 3 \sum_{k \in BD_j} BC_k \qquad (1)$$

We represented a build-order as a sequence of commands. Since GAs prefer a binary encoding, the CA, GA and the HC all worked with the same binary encoded sequence of commands [2]. We encoded the chromosome as a 15-length binary string. Every three bits of the binary string encodes an action and required prerequisites. WaterCraft sequentially decodes the binary string and inserts a request for each encoded action, preceded by any missing prerequisites, into a queue. Our game AI will issue actions from the queue as quickly as possible.

When evaluating an individual during coevolution, we used a teach set, scaled fitness, hall of fame, shared sampling, and fitness sharing as described by Rosin and Belew [3].

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.
ACM 978-1-4503-1964-5/13/07.

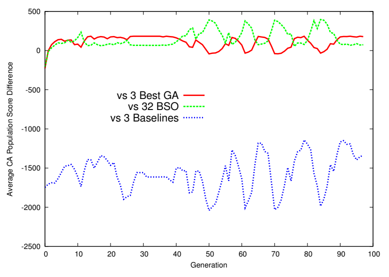

Figure 1: Avg Score Difference of CA Population.

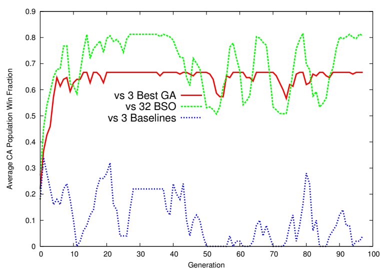

Figure 2: Avg Win Fraction of CA Population.

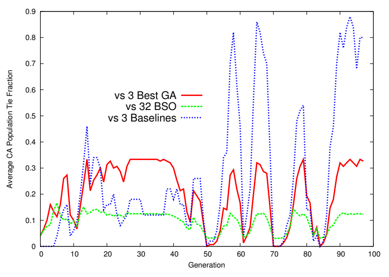

Figure 3: Avg Tie Fraction of CA Population.

The teach set that every chromosome plays against contains eight chromosomes selected from two sources: four chromosomes from the hall of fame and four chromosomes from shared sampling. We limited our teach set size to eight, so that with our population size of 50, our entire cluster of 400 nodes would be full. The hall of fame is simply a list of chromosomes that have performed well in the past. By using fitness sharing, the CA will not only favor strategies that defeat many opponents, but also strategies that defeat only a few opponents that rarely lose, since they contain important new information on how to beat those opponents.

For the basic CA operations and parameters, we used uniform crossover with a 95% chance of crossover occurring and bit-mutation with a .1% chance of each individual bit changing value. The chromosomes are selected for crossover by using standard roulette wheel selection. Once we have created all the children chromosomes, we evaluate them against the same teach set, then use CHC selection to select chromosomes for the next population [1].

Our GA implementation was simple. We modified the concept of the teach set we used for coevolution; instead of creating a new teach set each generation, we populate the teach set solely with our baseline strategies. Other than the change to the teach set, we used all the same parameters and methods that we used for coevolution.

We use the bit-setting optimization HC which attempts to find an effective solution by sequentially flipping each bit and keeping the value with the highest fitness [4].

$$f_i = \sum_{j \in B} F_{ij} - F_{ji} \qquad (2)$$

We determine the fitness by playing a chromosome against all three baselines and taking the sum of the differences in scores, as shown in Equation 2 Where f_i is the fitness of chromosome i, j is a baseline in the set of all baselines B, F_{ij} is the fitness chromosome i got against baseline j, and F_{ji} is the fitness baseline j got against chromosome i.

HC performance depends on the initial seed. We initialize this HC with thirty-two different seeds: a chromosome set to all 0's, a chromosome set to all 1's, and thirty randomly generated chromosomes.

Exhaustive search evaluates all 2^{15} possible build-orders against all three of our baselines. We limited ourselves to 15-bit solutions since that was the maximum build-order length we could exhaustively search in a reasonable amount of time. Exhaustive search enables us to rank all possible 15-bit solutions, and compare the effectiveness of solutions against the baselines found by CAs, GAs, HCs, as well as other search methods in the future.

3. RESULTS

We ran coevolution with a population size of 50 on our 400 node cluster for 98 generations, in order to match the number of generations performed by our GA. We measured coevolution's progress by playing all members of the population at each generation against three sets of opponents, taken from our previous study: The three best solutions found by the GA, 32 solutions found by the HC, and the three baselines used to evaluate the GA and HC solutions.

Figure 1 shows that coevolution very quickly moves to increase the average score of the population, but not by very much. However, this slight increase in average score has a huge affect on the number of wins and ties the solutions achieve, as shown by Figure 2 and Figure 3.

While Figure 1 and Figure 2 shows that our coevolved solutions are not as good against the baselines as the GA results, all three figures seem to indicate that an increase / decrease in performance against the GA and HC solutions correlates to an increase / decrease in performance against the baselines.

These results help specify the trade-offs between injecting human expertise into our evolutionary algorithms. Using a genetic algorithm against a set of hand-tuned baselines produces strategies that are robust against the opponents used in training. The coevolutionary algorithm produced strategies that defeated opponents of the same bit-length, but only made minor improvements against the baselines. These results inform our future work on finding robust strategies that also defeat specific opponents. This research is supported by ONR grant N000014-12-C-0522.

4. REFERENCES

[1] L. J. Eshelman. The chc adaptive search algorithm : How to have safe search when engaging in nontraditional genetic recombination. *Foundations of Genetic Algorithms*, pages 265–283, 1991.

[2] D. E. Goldberg. Genetic algorithms in search, optimization, and machine learning. 1989.

[3] C. D. Rosin and R. K. Belew. New methods for competitive coevolution. *Evol. Comput.*

[4] S. W. Wilson. Ga-easy does not imply steepest-ascent optimizable, 1991.