OPTIMIZATION AS A STEP IN COREWAR PROGRAM ANALYSIS, EVOLUTION AND CATEGORIZATION

Nenad Tomašev, Dragan Mašulović
Department of Mathematics and Informatics
The Faculty of Natural Sciences, University of Novi Sad
Trg Dositeja Obradovića 3, 21000 Novi Sad, Serbia
Tel: +381 21 520 869;
e-mail: nenad.tomasev@gmail.com

ABSTRACT

In this paper, an optimizer for programs written in an assembly-like language called Redcode is presented. Relevance of code optimization in evolutionary program creation strategies and code categorization is discussed. CoreWar Optimizer is the first user-friendly optimization tool for CoreWar programs offering various optimization methods and a carefully picked benchmark. The methods at the user's disposal are: random, modified hill climbing algorithm, simulated annealing, predator-prey particle swarm optimization and genetic algorithms. All these methods use a speed-up trick which drives the value optimization across three fitness landscapes instead of just one.

1 INTRODUCTION

CoreWar was introduced to the scientific community by A.K. Dewdney in 1984 in an article published in Scientific American [1]. It represented an interesting, but not completely unfamiliar concept. It was based on a somewhat similar computer simulation, a game called Darwin, which was developed in Bell Labs some twenty years earlier.

In CoreWar, programs written in an assembly-like language called Redcode compete against each other in a simulated environment that is being controlled by MARS (Memory Array Redcode Simulator). The program that seizes complete control of the process queue is designated a winner of the encounter. CoreWar programs are referred to as *warriors*. Competing programs are being loaded into a looping memory array and are only given access to that part of the memory. There are many differences between Redcode and other assembly-like languages, but providing a full survey of those details is well beyond the scope of this paper.

CoreWar environment bears some peculiarities and as such can not be used to draw general conclusions about any aspect of code analysis. However, its relative syntactic simplicity, as well as clearly presented program goals make it easier to define strategic program categories. With that in mind, research on CoreWar could only suggest potential fruitfulness of such analysis in a more general setting and either encourage or discourage further application of employed methods.

2 EVOLUTION STRATEGIES AND OPTIMIZATION IN COREWAR

Even though CoreWar is a sort of a programming challenge, programs are often generated automatically, usually via genetic programming. This type of reasoning has worked really well, due to the natural mapping between CoreWar programs and corresponding genotypes. Evolutionary pressure is reflected in the competition itself and fitness can be determined according to benchmarking and competition scores [2].

Evolved CoreWar programs are still superior to human-coded programs in some of the basic confrontation environments. However, these *evolvers* have failed to produce programs of higher complexity and there is usually little diversity in the evolved sets [3].

Program optimization does not have to be related to changes in the code structure. Once a good program template is produced, there is usually an issue of choosing values for some variables in the code. Altering these instruction field values can lead to dramatic changes in program fitness.

Prior to CoreWar Optimizer, which will be presented in this paper, there was only one available optimizer for CoreWar – Optimax. Its focus was on simulation speed and it searched the value space at random. It was a command-line application with no graphical user interface [4].

The fitness landscape corresponding to variables in CoreWar programs and the respective average performance is characterized by many local optima. Such structure of the search space calls for methods that are able to cope with target function multimodality.

Some intuition about more general fitness landscapes in CoreWar can be obtained by analyzing the score surface plots in Figure 1. Some negative linear dependencies between the plotted variables are quite apparent. However,

it must be noted that the images were generated for average scores against a single opponent (in both cases). For benchmarks of considerable size and a realistic case of having at least 5-6 variables to optimize, the search space would certainly become much more chaotic.

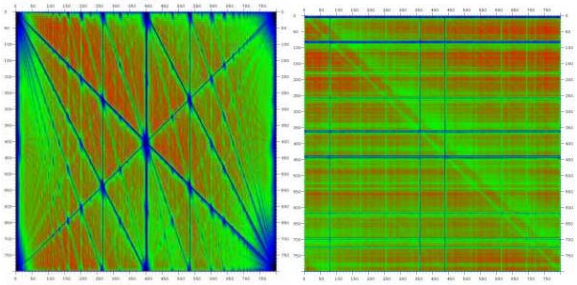

Figure 1: *Some 1-1 score surfaces for YAP (a replicator CoreWar program) for 2-value optimization task plotted across all possible value pairs. Blue colour denotes bad performance, while red areas represent high scores[5] .*

3 COREWAR PROGRAM CATEGORIES

3.1 Basic Strategies

In more than twenty years of competition, many interesting program strategies have been either devised by programmers or generated via evolutionary approach. Programs can rely on brute-force random attacks or implement various algorithms for opponent's code detection and elimination. Most programs can be roughly separated into three large categories – *stones, papers* and *scissors* – implying that there is no dominant approach in CoreWar. Due to the rise in strategic program diversity over the past years and continuous increase in deployment of hybrid programs utilizing more than one basic strategic block, a more flexible and detailed categorization was required. One such categorization has been presented in [6] and comprises 14 program types.

3.2 Automatic Categorization

In case of human-coded warriors, there is no need for automatic categorization, since a program category can easily be determined by looking at the code. It is different with evolved programs. Their code is much less structured, their behavior less straightforward and hundreds or thousands of them are being generated and tested in each iteration of CoreWar evolvers. It has already been mentioned that evolved populations often suffer from lack of diversity. This is precisely why dynamic diversity control is of great importance in such software tools. Accurate insight into the strategic composition of populations would allow for better dynamic changes in mutation weights, population size, parent selection, etc.

Attempts at creating classifiers for CoreWar program categorization and analysis of evolved data sets have been made, with moderate success [6] [7]. Two types of data representation had been used (as well as their combination):

- *Static* representation, based purely on syntactic analysis
- *Dynamic* representation, based on benchmark scores.

Best results were obtained when using the combined representation, achieving 84% accuracy with SVM. Static representation was not sufficient for reliable categorization and some alterations would need to be conducted in the future if it is to be used as a sole factor in predicting a CoreWar program category [6].

3.3 Impact of Optimization on CoreWar Program Categorization

It had been shown that classifiers trained on human-coded data sets were unable to achieve the same level of accuracy when used on evolved data sets. Since the dynamic representation has the most influence, a part of the issue must lie within the differences in benchmark scores of programs of the same class in evolved and human-coded sets. Once those scores were inspected, it became quite apparent what was the reason for such a mismatch. Even though there are good evolved CoreWar programs, these appear in the end of the process, as the populations begin to converge towards some program templates. Most of the CoreWar programs in those sets are not as optimized as their human-coded counterparts. This results in scores differing in a way that makes score-based categorization much more cumbersome, because the distribution of scores over the set of strategies in the opposing benchmark tends to become more uniform. This uniformity is never present in highly optimized programs. Programs from the same category achieve similar average scores against any particular strategic group.

It can be seen that the use of classifiers in determining population diversity in CoreWar evolvers would not be as informative as it might have appeared unless CoreWar program populations are at least slightly optimized before the automatic categorization takes place. Inserting optimization as a step in CoreWar program evolution has other benefits as well. After all, if a good program template is encountered, it would be rejected by the evolver unless it has scored above some static or dynamic threshold. If all the templates were optimized, better templates would be more likely to be passed on to the next generation.

There is also a possibility of extending and improving the syntactic representation of the data, but this would probably not be as helpful in evolved data sets, because majority of the code in most evolved CoreWar programs is junk code that is not used by the programs in the simulation. Including benchmark scores in the representation is necessary in this case.

4 COREWAR OPTIMIZER

4.1 General Description

CoreWar Optimizer is a tool that was developed with intent of allowing for both easier and more effective manual optimization of CoreWar programs and also integration into evolutionary optimization software for CoreWar.
Main features of CoreWar Optimizer are:

- Graphical user interface allowing program editing, parameter setting, optimization statistics overview, etc.
- Several methods for value optimization
- Speed-up achieved by splitting the optimization process into several phases

4.2 Optimization Phases

Speed is always one of the most important issues in CoreWar optimization, since it takes several minutes on a fast PC to benchmark a program instance. On the other hand, it was empirically determined that for some strategic categories, up to several thousand instances need to be examined. This is why there are three phases in all the algorithms in CoreWar Optimizer. In the first phase, an instance is set up against a single opponent (selected by the user). In the second phase, instances are competing against a selected strategic subset of the benchmark. Only in the third phase do programs compete against the entire benchmark. Thresholds for acceptable scores in the first two phases are set by the user. Most users prefer to set up high thresholds, allowing only about 1-3% of generated instances to enter the final optimization phase.

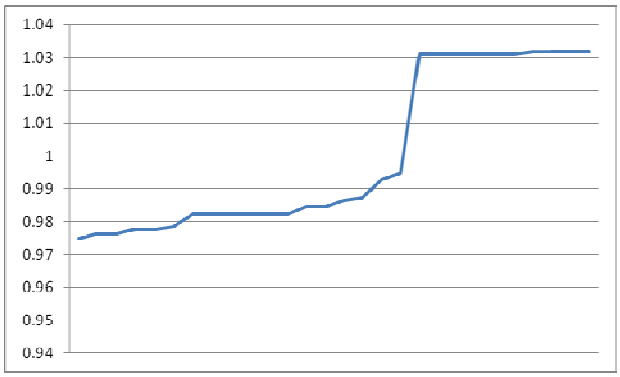

Figure 2: *Normalized plot of the highest achieved score of one tested replicator with GA being used.*

Selection of potentially good instances in the early phases leads to lower variance in the benchmarked set. This is illustrated in Figure 2. Even though the best achieved score in the end was only 6% better than the first one in terms of average highest score over the optimization run, these few score points are not insignificant. On the contrary, in most CoreWar competitions, competing instances are usually a fraction of a point apart.

4.3 Optimization Methods

Optimization in CoreWar depends on the program strategic category and the nature of the variables being optimized. This is why several methods have been included in CoreWar Optimizer, so that the user might select one of them according to his own judgement.

- Random search was included purely for reasons of comparison with the previous optimization tool, Optimax [4].
- Hill climbing is the simplest of the implemented algorithms. Steepest ascent is performed on a landscape until a point is reached where the threshold condition is satisfied. At that point, the ascent continues on the landscape of the next optimization phase. However, threshold conditions are constantly being checked for the previous phases. If none of the neighboring instances satisfy those conditions, optimization is reset to one of the earlier phases. Also, mutation amplitude depends on the time that has passed since the last instance entered the final optimization phase. This has been done to enable the algorithm to quickly leave depressions on the fitness landscape.
- Simulated annealing differs from hill climbing in an important way, since it offers a better trade-off between exploration and exploitation of the search space. The update formula that was used for calculating the probability of choosing a state if it offers no improvement is given below:

$$p(s_{i+1} = o_i \mid f(o_i) < f(s_i)) = e^{(f(o_i)-f(s_i))/Td}$$

  The cooling schedule was implemented according to [8]. This method optimizes over three fitness landscapes in the same way as the hill climbing method.
- Genetic algorithm implementation includes modules for small and big mutations, cross-over by swapping or affine combinations, etc. Mutation rates also evolve over time and are instance-specific – in other words, a part of the genotype [9]. The locality of mutation rates is motivated by the fact that each candidate solution serves as a representative of its neighborhood in the search space and the local properties of the fitness landscape vary accordingly. Thus, different mutation rates might be well suited for different areas in the domain. The mutation update rules are:

$$allele_i(t+1) = allele_i(t) + \lfloor \sigma_i(t+1) \cdot N(0,1) \rfloor$$
$$\sigma_i(t+1) = \sigma_i(t) e^{\tau \cdot N(0,1)}$$

- Particle swarm optimization is based on observations from social animal behavior – flocking of birds, schooling of fish, etc. Animals form groups in order to find more food and water. In particle swarm methods, a population of instances is initialized to some random points in the search space. These particles then share information about the most promising areas of the search space, thus driving the search towards both local and global high-fitness areas. In practice, this usually results in better search than in cases of hill-climbing and simulated annealing which are one-particle models [10]. One of the main problems with particle swarm methods is premature convergence. This is partially prevented by setting suitable inertia weights in the update rules. Method employed in CoreWar Optimizer follows the idea of introducing predator-prey relationships in the model. One predator particle is introduced and it acts as a repellant which follows the best particle in the swarm and forces other particles away from local optima. It has been shown that this has a positive influence on the quality of the search. This model has first been described in [11].

Extensive testing of all the implemented methods on representatives of all strategic categories with various number of parameters has not been performed, since such a testing would require too much time if any relevant results were to be obtained. All of the mentioned methods are equally efficient on average basis, considering all possible search spaces [12].

4.4 Benchmark

The benchmark used by the application has been selected to represent all the strategic categories with proportions relative to their frequency in existing data sets. This is shown in Figure 3.

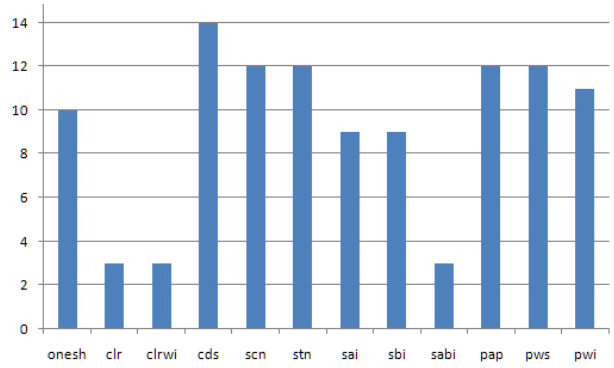

Figure 3: *Strategic distribution of the used benchmark*

5 CONCLUSION

Accurate automatic CoreWar program categorization requires benchmark testing. The results obtained this way can be quite ambiguous unless optimized CoreWar programs are involved. In order to allow for integration of the categorization modules in evolutionary CoreWar software, an optimizer has been implemented. CoreWar Optimizer provides an interface for use of several optimization methods. Initial tests have shown promising results and there has been no negative feedback from the community.

References

[1] A. K. Dewdney. Computer recreations: In the game called Core War hostile programs engage in a battle of bits. *Sci. Am.*, 250(5):14-22, 1984.

[2] Bryan Blumenkopf and Ashley Holtgraver. Artificial life through evolutionary computation. In *Computing Beyond Silicon Summer School,* California Institute of Technology, 2002.

[3] F. Corno, E. Sanchez and G. Squillero. Exploiting co-evolution and a modified island model to climb the CoreWar hill. In *Proc. CEC'03 Congress on Evolutionary Computation,* pages 2222-2229, 2003.

[4] S. Zapf. Optimax. www.corewar.info/optimax/

[5] J. Gutzeit. CoreWar Score Surfaces. http://corewars.jgutzeit.de/score_surfaces/index.en.html

[6] N. Tomašev, D. Pracner, M. Radovanović and M. Ivanović. Automatic Categorization of human-coded and evolved CoreWar warriors. In *Proc. PKDD'07, 18th European Conference on Machine Learning,* Warsaw, Poland, 2007.

[7] D. Pracner, N. Tomašev, M. Radovanović and M. Ivanović. Categorizing evolved CoreWar warriors using EM and attribute evaluation. In *Proc. MLDM'07, 5th Int. Conf. on Machine Learning and Data Mining in Pattern Recognition,* Leipzig, Germany, 2007.

[8] Franco Busetti. Simulated Annealing overview. www.geocities.com/francoburseti/saweb.pdf

[9] A.E. Eiben and J.E. Smith *Introduction to Evolutionary Computing*. Springer Verlag, second edition, 2003.

[10] James Kennedy and Rusell Eberhart. Particle Swarm Optimization. In *Proc. Of the IEEE Int. Conf. On Neural Networks,* pages 1942-1948, Picataway, NJ, 1995.

[11] Ernesto Costa Arlindo Silva, Ana Neves. Chasing the swarm: A predator-prey approach to function optimisation. In *MENDEL2002 8th Internacional Conference on Soft Computing,* Brno, Czech Republic, 2002.

[12] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation,* 1(1):67-82, April 1997.