# Re-imagining Cooperative Co-Evolution: Modular Genetic Algorithms

Yaron Strauch
Re-written university coursework from my MSc Artificial Intelligence

***Abstract*—Genetic Algorithms are heuristic computer simulations that find parameter configurations in complex search spaces by modelling evolution by artificial selection. Cooperative co-evolution was adapted from nature in 1994 to improve the speed of finding a solution of genetic algorithms by maintaining sub populations with competing specialised individuals. This work shows that the original paper, claiming that co-evolution speeds up the problem, appears to be mainly faster because it decomposes the problem. A modular genetic algorithm is presented that engineers a genome from multiple parents and has the same performance as its co-evolutionary counterpart without maintaining sub populations on the original fitness landscapes.**

## I. Background

Optimisation problems can be formalised as a space of possible variable configurations and some measure of how good these configurations perform, either an error function subject to minimisation or a fitness function subject to maximisation. When search spaces grow too big to be explored comprehensively, stochastic methods can find configurations that are good enough.

One possible way is doing so through a hill climber, a simple algorithm which ascends the fitness gradient to find maxima, however its greedy ascend may get stuck on local rather than global optima depending on the fitness landscape.

Another way to find good parameter configurations are *Genetic Algorithms* (GAs), which *evolve* parameter configurations by imitating evolution by natural selection. Possible solutions are represented by a population of genomes which code for individuals, each representing one particular configuration of variables. GAs let individuals compete with each other by evaluating them on a fitness function. Individuals are selected proportionally to their fitness score for genetic operations such as mutation and crossover. Mutations are small, random, undirected changes to the genome; crossover re-combines two good parental genomes into a child genome, hopefully re-combining good features from two genomes into one, allowing faster convergence than using a hill climber. Due to reproduction being relative to fitness, the population as a whole converges towards an optimal solution. [1]

A sub type of GAs are co-evolutionary GAs, as introduced by Potter and De Jong introduced in 1994 [2]. Competitive co-evolution models relationships like predator/prey, i.e. a deer that needs to survive predators in order to be more fit than its competitors in its same sub population (i.e. species), while cooperative co-evolution models symbiotic relationships like bees and flowers that need to adapt to each other in order to reproduce. When modelling co-evolutionary systems, the fitness of an individual is not generated objectively, rather it can only be generated together with other species that it needs to adapt to, no matter if competitively or cooperatively. Individuals only compete genetically with individuals from their own sub population, effectively increasing competition within the same role. Potter and De Jong introduced *CCGA1* (Cooperative Co-evolutionary Genetic Algorithm 1) [2], which is the foundation of many co-evolutionary algorithms today. In this context individuals within a cooperative evaluation have exactly one gene, representing one variable on a fitness landscape. Each sub population contains individuals competing for this one variable in their fitness landscape.

To evaluate individuals, the authors use four multi-variable optimisation functions, namely Rastrigin, Schwefel, Griewangk and Ackley. Since they represent errors, the algorithm is supposed to minimize the error and find the global minimum. Figure 1 illustrates the landscapes for two variables each instead of the 20, 10, 10, and 30 variables respectively that the algorithms are evaluated on in this work.

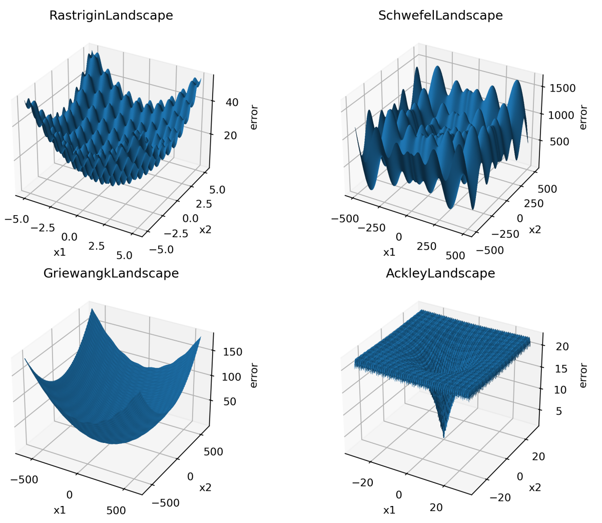

Fig. 1. Error landscapes with 2 variables.

In Potter and De Jong's work, it is important to realise that they moved two steps at once when moving from a standard GA to CCGA1: They introduced modularity and co-evolution at the same time. While co-evolution is not possible without decomposing modular problems (how would one split a population into sub populations when the problem is not decomposable?), it is possible to introduce modularity in optimisation algorithms without the need to co-evolve (as will be demonstrated in this work).

*Does the performance gain of CCGA1 over the standard*

*GA on these landscapes rely mainly on modularity, or is its cooperative design of sub populations the main driver?*

## II. Methods

To answer this question, Potter and De Jong's paper is reimplemented and compared to a modular GA which exploits decomposability without cooperative evolution.

Genes on the genome represent one variable on the fitness landscape. Individuals in the non-cooperative GAs have one gene for every variable, individuals in CCGA1 have exactly one gene and are split into sub populations, one for each variable.

Each gene consists of 16 bits and represents a real number $x$ within the applicable ranges of each landscape (details in appendix A). Fitness-proportionate selection with a scaling window of 5 and conversion from bitstring to numbers was implemented as described by Bäck and Schwefel [1]. Mutation of a genome is implemented as a bit-flip of every allele with a probability of 1 / chromlength. Two-point crossover is applied to an individual with a probability of 0.6.

### *A. Standard GA*

The standard GA only exploits modularity when recombining two individuals via crossover. Each individual has $n$ genes corresponding to the number of variables in the fitness landscape.

1) Initialise population $P$ of 100 random individuals, each with $n$ genes
2) Repeat until a solution was found or 100,000 function evaluations reached:
    a) Evaluate fitness for each individual in $P$
    b) Create new generation $P'$ which holds the best individual from $P$ (elitism).
    c) Repeat until $P'$ has 100 individuals:
        i) Select one individual from $P$ according to fitness
        ii) With a probability of 0.6, perform two-point crossover with second selected individual
        iii) Apply mutation (bit-flipping)
        iv) Append to $P'$
    d) Discard old population ($P = G$)

### *B. Cooperative Co-Evolution GA (CCGA1)*

CCGA1 exploits modularity by having $n$ sub-populations, each population representing one variable in the fitness landscape. Individuals are evaluated together with the best individuals from the previous generation.

1) Initialise best population $B$ of one random individual per sub population, each one gene
2) Initialise $n$ sub populations each with 100 random individuals, each one gene, and group them into population $P \supset B$
3) Repeat until a solution was found or 100,000 function evaluations reached:
    a) Evaluate cooperative fitness for each individual in $P$ by combining it with the remainder from $B$
    b) Create new generation $P'$ from $P$ using elitism, two-point crossover and mutation
    c) Update $B$ to contain the best individual per sub population from $P'$
    d) Discard old generation ($P = G$)

### *C. Modular GA*

The Modular GA introduced in this work exploits modularity in the same way CCGA1 does without maintaining a cooperative sub population. 99 of 100 individuals are created by the same process as the Standard GA. The 100th individual is generated by exploiting modularity; its genome is puzzled together from other individuals and therefore exploiting modularity in the problem.

This puzzling process is not done randomly, instead each gene's fitness is estimated. Similar to CCGA1 evaluating one individual by combining it with the best individuals from the other sub populations (called $B$), the 100th individual is created by selecting $n$ good individuals and evaluating one gene at a time by joining it with the remaining genes of the best individual. The score of the combined genome is assumed to be the score of the selected gene. After evaluating all the genes, it builds one combined genome with all the best genes found, maintaining genome ordering.

The whole algorithm therefore works like this:

1) Initialise population $P$ of 100 random individuals, each with $n$ genes
2) Repeat until a solution was found or 100,000 function evaluations reached:
    a) Evaluate fitness for each individual in $P$
    b) Create a new generation $P'$ of 99 individuals using elitism, two-point crossover and mutation
    c) Select the genome $G_b$ from the best individual of $P$
    d) Select $n$ parent candidates $C$ from $P'$ according to fitness
    e) For every parent candidate $c \in C$ with genome $G_c$:
        i) Concatenate an evaluation genome per allele $a$: $G_e[c, a] = G_b[1 : a-1] + G_c[a] + G_b[a+1 : n]; 1 <= a <= n$
        ii) Evaluate $G_e$ on the fitness function $f$
    f) Combine the best gene found per allele: $G'_b[a] = \arg\max(f(G_e[c, a])); 1 <= a <= n$
    g) Add an individual with the combined genome $G'_b$ to $P'$
    h) Discard old generation ($P = G$)

The authors from CCGA1 defined the mutation probability $p$ to be 1 / chromlength. For 16 bits, this means that the Standard GA mutates with $p = \frac{1}{16n}$ and CCGA1 uses $p = \frac{1}{16}$ because the cooperative individuals only have one gene. If one tries to compare an algorithm without sub populations to CCGA1, they should adapt this mutation rate to have comparable results. The Modular GA therefore uses a mutation rate of $p = \frac{1}{16}$. All other parameters remain the same.

## III. Results

Figure 2 compares the Standard GA, CCGA1 and the Modular GA averaged over 50 runs. As expected from their

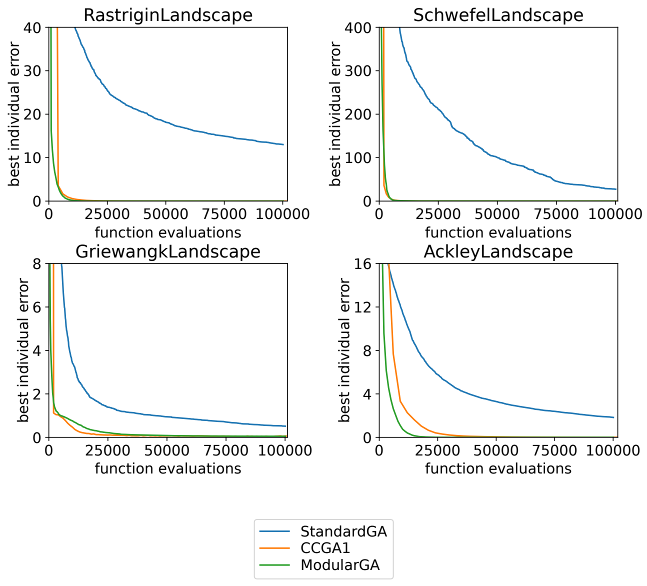

Fig. 2. The Modular GA is almost indistinguishable from the Cooperative GA

publication, CCGA1 performs considerably better than the standard GA. The Cooperative GA and the Modular GA are almost indistinguishable from each other, except for Ackley where the Modular GA performs even better than CCGA1.

For Rastrigin and Schwefel there is no clear winner, on Griewangk the CCGA1 and on Ackley the Modular GA is a bit faster. The differences are minimal.

## IV. Discussion

### A. Reimplementation

The pseudo code in the original paper specifies the GA to first generate a new generation, and then to perform genetic operators (which includes two-point crossover) on the new generation. While for mutation this description makes sense, performing crossover on a population needs more explaining. The paper specifies that the population is fixed at size 100 and does not describe replacement strategies for individuals that result from crossover. The Schwefel paper does specify to replace one individual at random [1], but this would mean that the elitist individual could be replaced. From the graphs it becomes obvious that the error never goes up and therefore the elitist individual must not be replaced. This is why the reimplementation applies crossover during the creation of the new generation and not after.

The reimplementation of both the standard GA and CCGA1 (figure 2) comes very close to the original paper [2, Fig. 3]. The Standard GA converges slightly slower in the reimplementation, this might have to with subtle differences in the implementations such as bitstring to float conversion or float precision, or the aforementioned population management issues.

### B. Technical Comparison

The general idea of the Cooperative GA (evaluating genes by slicing them out and evaluating them in the context of the best individual) has been adapted. But there is an important difference: CCGA1 works with sub populations and directly competing individuals, the Modular GA works on one population with indirectly competing genes. There is no sub population, so every individual competes with everybody. There is therefore no notion of co-evolution.

Looking at the score, both CCGA1 and the Modular GA outperform the Standard GA. The performance gain of exploiting modularity by using decomposability on function optimising algorithms is therefore clear, at least for the landscapes investigated: By exploitation of the decomposability of the problem, it got easier to solve. Between CCGA1 and the Modular GA there is not a clear winner, two experiments are nearly identical, and the other two are split 50/50.

Can we therefore say that CCGA1 performs well not because it uses sub populations but because it exploits modularity? Well certainly on these theoretical error landscapes.

Technologically, the recombination process in the Modular GA is more like a directed multi-point crossover. Directed because each gene is selected according to fitness from a pool of $n$ parent candidates, and multi-crossover because each gene might come from a separate individual. This is conceptually very different to random mutation or random crossover. Is it cheating because 1) this one individual takes many parent candidate evaluations and 2) only the best genes are cherry-picked? I would argue no, because 1) the comparison uses function evaluations on the X axis which will cover any additional candidate evaluations, and 2) the cherry-picking is equal to the artificial selection performed in any GA. The selective pressure per-gene is higher in this version, but that is not necessarily a good or bad thing.

### C. Semantic Comparison

The main appeal of GAs, and by extend cooperative co-evolution, is the simulation of what we observe in nature. By modelling this as true to what we assume to be reality, one might hope to achieve similar complex results, and any deviation from what we know (i.e. having more than two parents) might feel wrong. The approach here instead genetically engineers the genome by re-combining it surgically, decomposing it into genes, estimating fitness per gene, and re-engineering it together. One might judge the CCGA1 approach as more ”natural” and in contrast this new approach ”intrusive” and ”artificial”. These sentiments are obviously misplaced as the whole system is artificially engineered and synthetic. The notion of evaluating one individual together with the best individuals from a previous generation is equally absurd, if not more.

The notion of cooperation of CCGA1 versus the re-combination of multiple parental genomes in the Modular GA is more related than it might seem. If a number of parent candidates are being re-combined by evaluating and re-combining their best treats, they are also cooperating as the offspring contains the best features of its parents. Co-evolution in the real world is highly complex, and the borders between cooperative and competitive co-evolution are not always clear to draw. As individuals in a GA have no intent or will, in the end it doesn't really matter if they are grouped into some subpopulation to compete with another, or if they are surgically recombined.

The big emerging question is now, how many other processes of GAs have we engineered to be ”true” to nature, which simulations could have been improved, and did we miss any shortcuts?

## Appendix A
## Landscapes

### A. Rastrigin Landscape

Rastrigin has local minima that get higher the further away they are from the global minimum.

$$f(x) = 3n + \sum_{i=1}^{n} x_i^2 - 3cos(2\pi x_i)$$
$$n = 20; -5.12 <= x_i <= 5.12$$
$$x_{min} = (0, 0, ...)$$
$$f(x_{min}) = 0$$

### B. Schwefel Landscape

Schwefel has a second-best minimum far away from the global minimum. Note that the original paper incorrectly specified $x_{min} = (420.9687, 420.9687, ...)$ which is actually the maximum.

$$f(x) = 418.9829n + \sum_{i=1}^{n} x_i sin(\sqrt{|x_i|})$$
$$n = 10; -500 <= x_i <= 500$$
$$x_{min} = (-420.9687, -420.9687, ...)$$
$$f(x_{min}) = 0$$

### C. Griewangk Landscape

Griewangk has a product term to introduce interdependencies between variables.

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$$
$$n = 10; -600 <= x_i <= 600$$
$$x_{min} = (0, 0, ...)$$
$$f(x_{min}) = 0$$

### D. Ackley Landscape

Ackley is very rugged (ruggedness increases with $n$).

$$f(x) = 20 + e - 20exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)\right)$$
$$n = 30; -30 <= x_i <= 30$$
$$x_{min} = (0, 0, ...)$$
$$f(x_{min}) = 0$$

## References

[1] T. Bäck and H.-P. Schwefel, “An overview of evolutionary algorithms for parameter optimization,” *Evolutionary computation*, vol. 1, no. 1, pp. 1–23, 1993.

[2] M. A. Potter and K. A. De Jong, “A cooperative coevolutionary approach to function optimization,” in *International Conference on Parallel Problem Solving from Nature*. Springer, 1994, pp. 249–257.