# CS4205 - Evolutionary Algorithms
## Assignment 2

May 10, 2019

This document contains descriptions of the 4 possible exercises for assignment 2 of the TU Delft course Evolutionary Algorithms (CS4205). Prior to the start of this assignment, each group of **4 students** is required to denote their relative preference for each of the 4 possible exercises in the Brightspace survey, and will be assigned one exercise based on their preference.

Each group is required to hand in the following deliverables on Brightspace in a single **zip file** prior to the deadline **June 30 2019, 23:59**:

- A report (pdf file) of up to 9 pages, excluding figures.
- All source code used to produce experimental results described in the report.

For questions about any specific exercise, please email the intended supervisor of the exercise.

# Contents

# 1 Genetic Local Search

Intended supervisor: Peter Bosman (Peter.Bosman@cwi.nl)

## Background

Weighted maximum cut (MAXCUT) is a well-known combinatorial optimization problem. It is defined given a weighted undirected graph. The goal in weighted MAXCUT is to split the node set into two sets such that the combined weight of all edges that are thereby cut, i.e. running between vertices in different sets, is maximized.

Many different approaches have been developed over the years to solve this problem. In general, one of the most efficient types of metaheuristics for solving combinatorial optimization problems is genetic local search (GLS). In GLS, a combination is made between evolutionary algorithms (EAs) and local search. Basically, instead of using an EA to optimize the problem directly, for every solution that is ever generated, local search is applied to it first. As a result, the EA operates in the space of local optima, rather than the complete search space.

The structure of the space of local optima is typically not crystal clear and hence, this is an ideal case to apply advanced black-box optimization techniques that attempt to automatically detect and exploit a problems' structure.

## Assignment

In this assignment the task is to design, implement and test GLS algorithms. As a basis, the simple GA and the LT-GOMEA can be used, to see what the added value is of automated problem-structure exploitation in solving weighted MAXCUT. Basic implementations of the EAs in C will be made available, as well as a set of benchmark problems. Local search and how to best combine local search with the EAs will have to be implemented.

## Literature


- Rafael Martí, Abraham Duarte, and Manuel Laguna. 2009. Advanced Scatter Search for the Max-Cut Problem. INFORMS J. on Computing 21, 1 (January 2009), 26-38. DOI=10.1287/ijoc.1080.0275 http://dx.doi.org/10.1287/ijoc.1080.0275

- P.A.N. Bosman and D. Thierens. Linkage Neighbors, Optimal Mixing and Forced Improvements in Genetic Algorithms. In T. Soule and J.H. Moore, editors, Proceedings of the Genetic and Evolutionary Computation Conference - GECCO-2012, pages 585-592, ACM Press, New York, New York, 2012.

- Wu, Q., Wang, Y., & L, Z. (2015). A tabu search based hybrid evolutionary algorithm for the max-cut problem. Applied Soft Computing, 34, 827-837.

# 2 Multi-Objective Permutation Problems

Intended supervisor: Hoang Luong (Hoang.Luong@cwi.nl)

## Background

There are combinatorial optimization problems, especially the ones involving scheduling or routing, that have permutation search spaces. For example, given a list of n cities, the traveling salesman problem (TSP) involves finding the best possible route to visit every city and return to the origin city such that the total traveling distance is minimized. Each permutation of n cities thus constitutes a TSP solution. Or, given a list of J jobs and each job requires M uninterrupted operations on M machines following the same order 1, 2,..., M, the permutation flowshop scheduling problem (PFSP) involves finding the job- processing schedule that minimizes the total time. Each permutation of J jobs thus yields a PFSP solution.

Real-world optimization problems typically require multiple objectives to be optimized. For example, solving the TSP can involve both minimizing the total traveling time and minimizing the total road toll. We have thus multi-objective permutation problems (MOPPs).

Employing evolutionary algorithms with straightforward solution representations and traditional variation operators (i.e., 1-point/2-point/uniform crossover and bit-flip mutation) of discrete optimization is often ineffective for solving permutation problems. Thus, suitable solution representations and variation operators are especially important for permutation optimization. Furthermore, it is also highly research-worthy to investigate the potential of applying linkage learning techniques to guide the search.

## Assignment

1. Study well-known solution representations and variation operators for a chosen MOPP.
2. Investigate the performance of traditional algorithms, such as NSGA-II and MOEA/D, for the chosen MOPP.
3. Investigate the potential performance enhancement of linkage learning in solving MOPPs (e.g., by modifying/combining the multi-objective GOMEA for discrete optimization with single-objective GOMEA for permutation optimization in order to solve MOPPs).

Most of the source code is available in the C programming language.

1. NSGA-II: `https://www.egr.msu.edu/~kdeb/codes.shtml`
2. MOEA/D: `https://dces.essex.ac.uk/staff/zhang/webofmoead.htm`
3. GOMEA: `https://homepages.cwi.nl/~bosman/source_code.php`

## Literature

- P.A.N. Bosman, N.H. Luong, and D. Thierens. Expanding from discrete Cartesian to permutation gene-pool optimal mixing evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2016, pp. 637-644.

- N.H. Luong, H. La Poutré, and P.A.N. Bosman. Multi-objective gene-pool optimal mixing evolutionary algorithms with the interleaved multi-start scheme. Swarm and Evolutionary Computation, vol. 40, pp. 238-254, 2018.
- F. Samanlioglu, W.G. Ferrel Jr., and M.E. Kurz. A memetic random-key genetic algorithm for a symmetric multi-objective traveling salesman problem. Computer & Industrial Engineering, vol. 55, pp. 439-449, 2008.
- H. Ishibuchi and T. Murata. A multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Transaction on Systems, Man, and Cybernetics-Part C: Applications and Reviews, vol. 28, no. 3, pp. 392-403, 1998.
- K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, 2002.
- Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on Evolutionary Computation, vol. 11, no. 6, pp. 712-731, 2007.

# 3 Weight-Tunable Symbolic Regression

Intended supervisor: Marco Virgolin (Marco.Virgolin@cwi.nl)

## Background

Genetic Programming (GP) and Deep Learning (DL) can both be used to learn classification or regression models. The tree-like models evolved by GP are actually not too dissimilar from the network-like models employed by DL: both are directed computation graphs (see Fig. 1). However, while GP's optimization operates by swapping and mutating tree branches, i.e. by changing the architecture of the trees, optimization in DL does not change the architecture of a network, rather, it updates weight components that define the strength of neuron interactions (see Fig. 2). It is interesting to assess wether a mechanism similar to DL's weight updates can be exploited in GP.

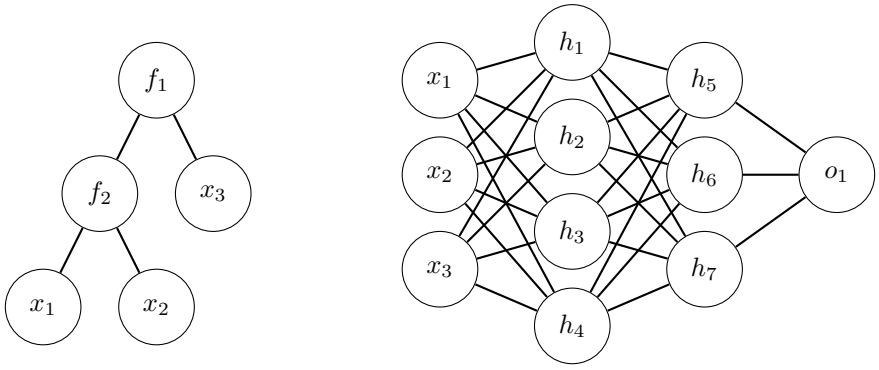


Figure 1: **Left**: Example of GP tree with two binary functions ($f_1$ and $f_2$) and three features ($x_1$, $x_2$, $x_3$). Computation flows bottom-up. **Right**: Example of fully-connected, feed forward neural network, with three-dimensional input layer, 2 hidden layers (first with 4 units and second with 3 units), and unidimensional output layer. Computation flows left-right.

## Assignment

The goal of this assignment is to implement a GP for symbolic regression (or classification) augmented with weight-tuning. For any function node $f$, each input must be transformed by scaling and translation, prior to applying $f$. E.g., for the GP tree of Figure 1, the root will compute $f_1(w_1 f_2 + w_2, w_3 x_3 + w_4)$. In general, a $d$-dimensional function node $f$ with input nodes $i_1, \ldots, i_d$ will compute $f(w_1 i_1 + w_2, \ldots, w_{2d-1} i_d + w_{2d})$.

To tune the weights, you can implement or use an existing real-valued EA, to run within GP. Given a GP tree containing a total of $n$ weights, such EA must build a population where each individual is a $n$-dimensional real-valued vector. These individuals are evaluated by setting the weights of the tree to the values in its genotype, and calling the normal fitness function of GP (e.g., mean-squared-error for regression, accuracy for classification). **Alternative:** If you are familiar with *backpropagation*, you can implement and use that instead of a real-valued EA. Note that this requires the implementation of (1) the derivative of the fitness function; (2) the derivative of each function node considered; (3) a gradient descent algorithm.

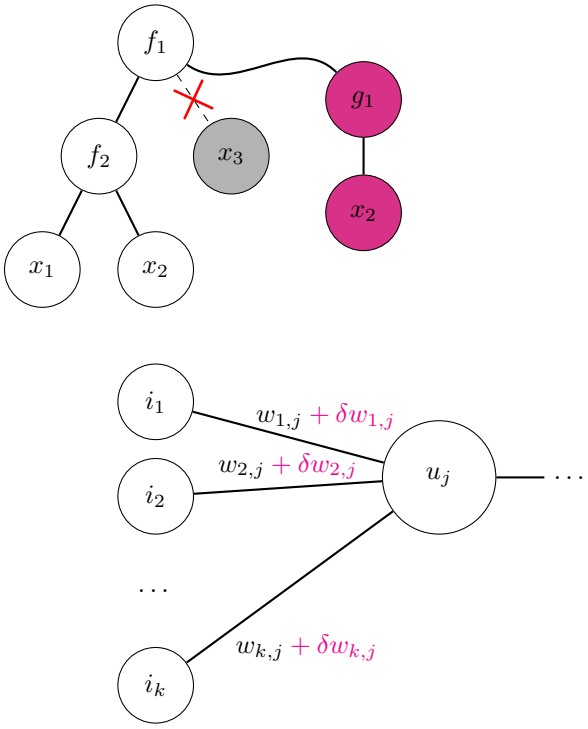


Figure 2: **Top**: Example of subtree mutation in GP: a random subtree is chosen (in gray), and replaced by a newly sampled subtree (in magenta). **Bottom**: Example of weight update in neural networks. A sub-part of the neural network is shown: a unit $u_j$, its input units $i_1, \ldots, i_k$, and the relative connections. For each and every connection, the respective weight $w$ is modified by adding a $\delta w$ term (these terms are computed with the *backpropagation* algorithm, not discussed here).

Code of a typical GP will be provided that performs symbolic regression. Together with the implementation of weight tuning, you will need to experiment how to best use it within GP. E.g.: Shall it be used every generation? With what rate? On the whole population or on some trees?

You can choose to work on classification or on regression, on a dataset of your choice, as long as it is approved by the supervisor. Do not forget to cross-validate!

In summary:

1. Implement the use of weights in GP trees, to scale and translate the inputs of function nodes;
2. Implement or use an existing real-valued EA, to be run within GP to tune the weights of trees. Alternatively, implement backpropagation;
3. Experiment how to best use your weight-tuning algorithm within GP, for a symbolic regression or classification task;
4. Write a report that describes your implementation and experiment choices, presents the results, and draws the conclusions.

## Literature


- Alexander Topchy and William F Punch. Faster genetic programming basedon local gradient search of numeric leaf values. In Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, pages 155162. Morgan Kaufmann Publishers Inc., 2001.
- Qiongyun Zhang, Chi Zhou, Weimin Xiao, Peter C Nelson, and Xin Li. Using differential evolution for GEP constant creation. In Proc. Late breakingpaper at Genetic and Evolutionary Computation Conf., 2006.

# 4 Gray-Box Cooperative Coevolution

Intended supervisor: Anton Bouter (Anton.Bouter@cwi.nl)

## Background

Cooperative coevolution (CC) is a well-known concept that is frequently used for large-scale optimization, both in discrete and real-valued optimization. In CC, as illustrated in Figure 3, the optimization problem is decomposed into smaller subproblems, and a separate (sub)population is used for each of these subproblems. Each subpopulation is optimized with an arbitrary EA, e.g., AMaLGaM for real-valued optimization, and each individual in a subpopulation is evaluated by combining its variables with the elitist solutions of the remaining subpopulations.

Generally, CC is used in a Black-Box Optimization (BBO) setting, where the optimization is preceded by an algorithm to determine a problem decomposition. In the Gray-Box Optimization (GBO) setting that we consider, such a problem decomposition is known prior to optimization. Furthermore, the evaluation of individuals in a subpopulation can be performed in a very efficient way by only evaluating the problem variables that have been modified, instead of evaluating all variables.

Previously, a GBO setting was shown to greatly improve the performance and scalability of the real-valued GOMEA (RV-GOMEA), allowing for the optimization of problem with millions of variables. Theoretically, a GBO setting can also be exploited by CC, but this has never before been tested.

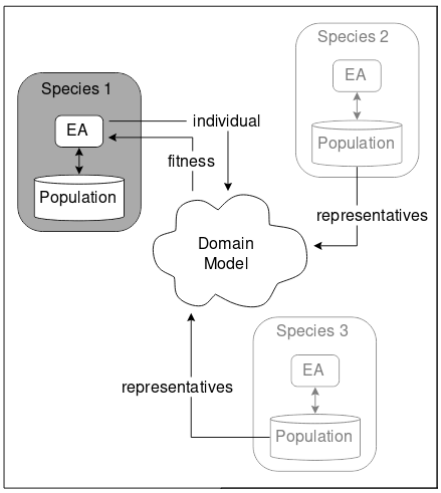


Figure 3: Illustrating cooperative coevolution from the perspective of subpopulation 1 (Potter and De Jong; 2000).

## Assignment

The goal of this assignment is to test the performance and scalability of CC for GBO on various decomposable and non-decomposable benchmark functions. These results should then be compared to RV-GOMEA, which is known to perform well in a GBO setting. For this purpose, an implementation should be made of CC using AMaLGaM as the optimizer for each subpopulation. Code of AMaLGaM will be provided in C, which can be extended in C or C++. Code of RV-GOMEA will be provided to compare experimental results to.

## Literature


- M.A. Potter and K.A. De Jong. "Cooperative coevolution: An architecture for evolving coadapted subcomponents." Evolutionary computation 8.1 (2000): 1-29.

- P.A.N. Bosman, J. Grahl, and D. Thierens. "Benchmarking Parameter-free AMaLGaM on Functions With and Without Noise." Evolutionary computation 21.3 (2013): 445-469.
- A. Bouter, T. Alderliesten, C. Witteveen, and P.A.N. Bosman. "Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm." Proceedings of the Genetic and Evolutionary Computation Conference. ACM, 2017.