Ant-Based Metaheuristics Struggle to Solve the Cartesian Genetic Programming Learning Task

Julian Trautwein, Michael Heider(✉), Henning Cui, and Jörg Hähner

University of Augsburg, 86159 Augsburg, Germany
{julian.trautwein,michael.heider}@uni-a.de

Abstract. Ant-based metaheuristics have successfully been applied to a variety of different graph-based problems. However, for Cartesian Genetic Programming (CGP) only the impact of Max-Min Ant Systems has been tested. In this work, we try to fill this gap by applying four different popular ant-based metaheuristics as the optimizer (and therefore training algorithm) of CGP. The idea of combining CGP with ant-based metaheuristics is not novel but older works' experimental design may not meet today's standard. To compare these metaheuristics to the Evolution Strategies (ESs) commonly used in CGP, we benchmark against a standard CGP variant that uses a simplistic $(1 + 4)$-ES, mutation, and no crossover. Additionally, we include $(\mu + \lambda)$-ES and (μ, λ)-ES in our experiments.

We analyse the performance on datasets from the symbolic regression, regression, and classification domains. By tuning and evaluating various configurations, we can not affirm a significant improvement by using ant-based methods with CGP as we encounter premature convergence—even with those ant-based metaheuristics that were originally proposed to overcome such problems. Despite our results being of negative nature, this work still gives important and interesting insights into the training of CGP models. The key contributions of our work are thus a more thorough benchmarking of these optimizers than has been done before. This should clear up doubts about the capabilities of ant-based metaheuristics in CGP. Furthermore, we include a roadmap on how they can be addressed to solve this complex optimization problem from the model building domain of machine learning.

Keywords: Cartesian Genetic Programming · Evolution Strategies · Evolutionary Algorithm · Ant Colony Optimization · Ant-based Metaheuristic

1 Introduction

Cartesian Genetic Programming (CGP) is a form of *Genetic Programming* (GP) developed by Miller in 1999 [26]. CGP—in contrast to GP—is represented by

a *feed-forward, directed, and acyclic* graph instead of a *tree based representation*. This makes it easy to be applied to graph-based applications like *neural architecture search* [32] or *image processing* [25].

CGP often omits *crossovers*, which are an archetypical operator of genetic algorithms, even though there have been experiments where including crossover shows an increase in fitness [9]. As a result, only selection and mutation operators induce changes to optimise a graph for a given learning task. Metaheuristics or other learning paradigms are typically not considered in the context of CGP. Since CGP is represented by said graph, it can be viewed as a pathfinding problem, for which ant system–related algorithms are a natural solution. In this work, we apply four different variants of ant-based metaheuristics and analyse their impact on CGP.

We start by reintroducing the core principles of CGP in Sect. 2 to serve as an easy entrance to the reader. In Sect. 5 the different ant-based metaheuristics used in this work are presented, which is followed by a summary of previous work on ant-based CGP in Sect. 4. After that, we give a description of the implementation of our ant-based metaheuristics into CGP (Sect. 3). Then, the performance of all used metaheuristics is analysed in Sect. 6. At last, Sect. 7 summarizes our results and shows further research possibilities.

2 Cartesian Genetic Programming

This section reintroduces the core principles of *Cartesian Genetic Programming* (CGP).

2.1 Representation

In CGP, a program is represented as a *feed-forward, directed, and acyclic* graph. Nowadays, it contains *nodes* arranged in a one dimensional grid with $c \in \mathbb{N}^+$ columnsd [28]. CGP takes an arbitrary amount of program inputs and feeds them forward through the graph to get the desired amount of program outputs.

There are three types of nodes present in a CGP graph: input, computational and output nodes. The *input nodes* are the first nodes of the program. They directly relay the program input to the other node types. The *computational nodes* are represented by multiple genes: One *function gene*, that specifies which function the node will apply on the given inputs, and $a \in \mathbb{N}^+$ *connection genes* that define the node's inputs. The value a is set to the highest arity of the defined function set. If a function needs less than a inputs, all unused connection genes will be ignored. The *output nodes* are typically the last nodes of the graph. They only receive the output previous node and redirect it as the output of the program. This node category consist of one connection gene, which refers to the computational or input node they take their output from.

Input and computational nodes can also be divided into active and inactive nodes. *Inactive nodes* are nodes that are not part of a path to any output nodes

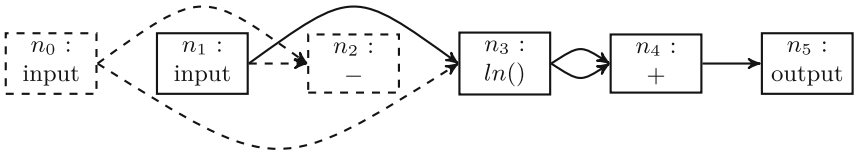

Fig. 1. A graph defined by a CGP genotype. The dashed nodes and connections are inactive.

and therefore do not contribute to the program output. Still, they are beneficial to the optimization process as they lead to genetic drift [33]. *Active nodes* are part of a path to any output nodes by one or more paths—therefore they contribute to the output of the program.

Figure 1 shows an example graph defined by a CGP genotype. It has $c = 6$ columns, takes two inputs and returns one output. The first two nodes n_0 and n_1 are input nodes and only relay the two program inputs. The nodes n_2, n_3 and n_4 are computational nodes. Node n_3 uses the function $ln()$ on its inputs, which has an arity of one. This leads to n_3 omitting the second input and only calculating with its first input. Therefore, only n_1, n_3 and n_4 are active nodes. At last, n_5 provides the output of the program by relaying the output of the computational node n_4. As a result, this CGP graph describes the following function:

$$\begin{aligned} f : \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ (n_0, n_1) &\to \ln(n_1) + \ln(n_1) \end{aligned}$$

2.2 Common Evolutionary Operators of CGP

Most CGP variants use an elitist $(1+4)$-ES. It is commonly used in combination with neutral search to improve performance and convergence time [27]. In this context, *neutral search* describes the concept that if an offspring of the current parent has the same fitness value as the parent, it will always be chosen as the new parent. This allows for neutral drift to occur and improves the exploration of the search space [28].

As the mutation strategy, either a *probabilistic mutation* [16] or *Single* [14] are used. With the first operator, it simply iterates over all genes and mutates them with a predefined probability. This leads to children possibly not having mutated any active nodes and therefore not altering the program's output. Single, on the other hand, randomly selects genes and mutates them until an active node is mutated. This enforces a change in the phenotype and allows inactive nodes to be mutated. To improve readability, the aforementioned description of CGP will be called Standard in the following sections. There are also many more adaptations to the Evolutionary Algorithm of CGP like [10,13,18], which will not be used in this paper.

3 Ant-Based Metaheuristics

This section will describe the non-ES metaheuristics used in this paper to optimize CGP graphs. The integration and benchmarking of previously unexplored (or unpublished) options of ant-based metaheuristics is the key contribution of this work.

3.1 Ant System

Ant System (AS) [12] was the first proposed ant-based metaheuristic. AS is the simplest ant-based metaheuristic and every other ant-based metaheuristic can be seen as an extension of AS.

Initially, a given number of $m \in \mathbb{N}^+$ ants are randomly distributed among all nodes and each edge gets an initial pheromone level of $\tau_0 \in \mathbb{R}^+$, therefore $\tau_{ij}(0) := \tau_0$ for $i, j = 1, \ldots, \#\text{nodes}$. At each timestep, each ant selects the next node it will move to[1], based on the amount of pheromones present on the given edge and the length of the path to the next node. For the k-th ant, the transition probability to go from node i to node j at time t is defined as:

$$p_{ij}^k(t) := \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha_{AS}} \cdot [\eta_{ij}]^{\beta}}{\sum_{c \in C_k} [\tau_{ic}(t)]^{\alpha_{AS}} \cdot [\eta_{ic}]^{\beta}} & if\ j \in C_k \\ 0 & otherwise \end{cases} \tag{1}$$

where C_k is the set of available nodes that ant k has not yet visited, $\eta_{ij} : -\frac{1}{d_{ij}}$ for d_{ij} being the distance between node i and j, $i, j = 1, ..., \#\text{nodes}$, and $\alpha_{AS}, \beta \in \mathbb{R}^+$ are hyperparameters that scale the importance of the pheromone versus the distance. The intensity of the pheromone trail left on edge (i, j) at timestep $t \in \mathbb{N}$ is represented by $\tau_{ij}(t)$. After all nodes have been visited, each ant lays pheromones on the connections it has used to construct its path to the inputs. The pheromone update is specified by the following two rules:

$$\tau_{ij}(t) := \rho \cdot \tau_{ij}(t - n) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \tag{2}$$

$$\Delta\tau_{ij}^k(t) := \begin{cases} \frac{1}{L_k} & if\ ant\ k\ used\ edge\ (i, j)\ in\ its\ latest\ tour \\ 0 & otherwise \end{cases}$$

where L_k is the fitness (defined by a task-appropriate metric) of ant k and $\rho \in [0, 1[$ is a hyperparameter that simulates the evaporation of pheromones on each edge.

[1] Note that the ants can only select nodes that are "further left" than the current location to avoid the creation of cyclical graphs.

3.2 Ant Colony System

The *Ant Colony System* (ACS) [11] is one of the many improvements of the Ant System. There are two main differences: At first, the authors changed the pheromone update rule and the state transitioning rule of ants as follows:

$$\Delta\tau_{ij}^{k}(t) := \begin{cases} \tau_0 & if\ ant\ k\ used\ edge\ (i,j)\ in\ its\ latest\ tour \\ 0 & otherwise \end{cases}$$

In addition to that, ACS introduced a *global update rule* where only the best ant is allowed to deposit pheromones. This best ant can either be the best ant that has been found so far, or the best ant of the current iteration of the algorithm. The pheromone update rule, with $\tilde{k}$ as the best ant, is described as following for all $i, j = 1, ..., n$:

$$\tau_{ij}(t) := (1 - \alpha_{ACS}) \cdot \tau_{ij}(t - n) + \Delta\tau_{ij}^{\tilde{k}}(t) \tag{3}$$

$$\Delta\tau_{ij}^{\tilde{k}}(t) := \begin{cases} \frac{1}{L_{\tilde{k}}} & if\ (i,j) \in best\ tour \\ 0 & otherwise \end{cases} \tag{4}$$

where $\alpha_{ACS} \in [0, 1[$ is another hyperparameter, that simulates pheromone evaporation on the edges. For the transition rule, they used the same function as the AS, but without the use of the α_{AS} hyperparameter given in AS. In addition to that, ACS introduces the hyperparameter $q_0 \in [0, 1]$, that determines a rate of *exploitation* vs *exploration*.

3.3 Max-Min Ant System

The *Max-Min Ant System* (MMAS) [31] is another enhancement of the Ant System. Its characteristic differences to the AS are that only the global best ant or the best ant of the iteration updates the pheromone trail. Furthermore, to avoid stagnation, the pheromone trails are limited to an interval $[\tau_{min}, \tau_{max}]$. Additionally, the pheromone trails are initialized to τ_{max} to get a higher exploration of the search space at the start of the algorithm. The state transition rule used in the MMAS is the same as in AS. MMAS updates its pheromones with the global update rule of ACS, as shown in Equation (3) and Equation (4) and omits the local update of all ants.

3.4 Ant System Local Best Tour

The last ant-based metaheuristic we tested is the *Ant System Local Best Tour* (ASLBT) [35]. The main idea behind ASLBT is to remove a global observer and let every single ant keep track of the best tour it has found so far. The pathfinding algorithm of ants is the same as in AS, which can be seen in Equation (1). The pheromone update rule is also the same rule as Equation (2) with the following addition:

$$\Delta\tau_{ij}^{k}(t) := \begin{cases} \frac{L_{best}^{k}(t)}{L_k(t)} & if\ ant\ k\ used\ edge\ (i,j)\ in\ its\ latest\ tour \\ 0 & otherwise \end{cases}$$

Table 1. Distance functions used by Ant-based metaheuristics.

Distance function	Mathematical definition
Manhattan Distance:	$\eta(i,j) := (i - j_1) + (i - j_2)$
Euclidean Distance:	$\eta(i,j) := \sqrt{(i - j_1)^2 + (i - j_2)^2}$
Logarithmic Distance:	$\eta(i,j) := \sqrt{ln(i - j_1) + ln(i - j_2)}$
Constant:	$\eta(i,j) := 1$

where $L^k_{best}(t)$ is the best fitness value ant k has found until timestep t and $L_k(t)$ is the current fitness value corresponding to ant k.

4 Including Ant-Based Metaheuristics Into CGP

After reintroducing ant-based metaheuristics, we now describe our method of integrating them into the CGP training algorithm. We use two different pheromone matrices for optimising the CGP Graphs. One for optimising the connections of the graph and one for optimising the functions of the individual nodes.[2]

All entries of the connection pheromone matrix are initialised with the respective default $\tau_0, \tau_{max} \in \mathbb{R}^+_0$ values as is typical for the different ant-based metaheuristics. Connections that are not allowed are initialised with 0 to stop ants from generating illegal solutions.

An ant creates a *genotype* by iterating over every node i and choosing the connection according to the probability

$$p_j := \frac{[\tau(i,j)]^{\alpha_{AS}} \cdot [\eta(i,j)]^{\beta}}{\sum_j [\tau(i,j)]^{\alpha_{AS}} \cdot [\eta(i,j)]^{\beta}}$$

where i, j are the two connections defined by the row index of the matrix. We used the parameter $\alpha_{AS} := 1$ for all transition probability calculations as it was proposed that way in [31] and [35] to limit the hyperparameter search space. The parameter α_{ACS} used in ACS still needs to be optimised. We implemented four different distance functions (see Table 1) to improve the optimizers capability. Given their mathematical definition, i is the current node's index, j_1 is the index of the first connection of node i, and j_2 is the respective second connection. To prevent the distance function from differentiating between different inputs, all input nodes got the same index for calculation of the distance (the highest index of all input nodes). Ideally, we would find one distance function that is good or even optimal for all cases. However, we did not find one function to be ideal as our hyperparametertuning selected different functions regularly.

[2] We also looked at having only one pheromone matrix that combines optimizing the graph and the individual nodes. This version showed worse performance on all tested datasets and was therefore discarded.

After the pheromones of all connections have been updated, each ant iterates through the function pheromone matrix and assigns functions according to the probability $p_j := \frac{[\tau(j)]}{\sum_j [\tau(j)]}$, where j is the column index of the matrix. Here, no distance function is used because the probability to use a specific function should not be dependent on a randomly assigned index.

The update of the pheromone level after each iteration is done according to the function defined by each algorithm. The only difference is, that for AS, ACS, and MMAS $\Delta\tau_{ij}^k := \frac{1}{1+f(a_k)}$, where $f(a_k)$ is equal to the fitness of ant k. For the ASLBT pheromone update $\Delta\tau_{ij}^k := \frac{f_{best}(a_k)}{f(a_k)}$ is used, where $f_{best}(a_k)$ is equal to the best fitness ant k has had so far. In all cases, $\Delta\tau_{ij}^k := 0$, if the connection or function was not used by ant k.

The available computation node functions for all problems are: sin (), cos (), tan (), tanh (), ReLu, Sigmoid, exp (), ln (), abs(), $*(-1)$, $+$, $-$, $*$, $/$. Please note that the first ten functions are of arity one while the last four have an arity of two. Therefore, the function matrix consists of 14 different rows.

5 Ant-Based Cartesian Genetic Programming

Our work focuses on analysing the impact on the performance of CGP when the ES is substituted with different ant-based metaheuristics . There is already some existing research in this area, which we will present in the following.

Hara et al. [15] introduced the idea of using an adapted Max-Min Ant System [31] for mutating the connections of CGP nodes and named their system *Cartesian Ant Programming* (CAP). However, contrary to our work, they used fixed alternating function genes that were not mutated during training. Also, in comparison to standard Max-Min Ant Systems and our work, the distance between the nodes was not taken into consideration for the transition rule of ants. This allowed them to use a single ant that walks through the graph moving from one output node to one of the input nodes. After that, the ant backtracks until it reaches a node that has unconnected inputs and repeats the aforementioned step of choosing a single next connection. This was repeated until a complete CGP graph is built.

Kushida et al. [23] extended the work of Hara et al. [15] by introducing a function to use the inter-node distance for the transition rule of the ants. They also tried to enhance CAP by dynamically assigning functions to the computational nodes. For this, they used two different pheromone tables, one for the connections and another one for the functions. They updated them with the same rule as Hara et al. [15] and only used functions with arity two. Their approach made the ant choose the two connections of each node independent of each other, which is their main difference compared to our work. This independence might lead to ants not choosing the best order of input nodes for the given function because functions like subtraction or division are not associative.

At last, Luis et al. [24] introduced a rank-based ant algorithm for evolving a CGP graph. For this, they effectively used three different pheromone tables.[3] One is for the first connection of each node, another is for the second connection of each node, and the last is the function of each node. They also omitted the idea of Kushida et al. [23] to use a distance function to make the ants explore shorter connections first. Their method lead to a higher diversity in the population and therefore better adaptation to a dynamic environment. However, it did not lead to a better fitness of the trained model than the approach of Hara et al. [15].

6 Experimental Setup and Evaluation

We logged the mean fitness of the population, the standard deviation of the fitness, the mean absolute error on the regression problems, the best fitness found until the given iteration, the best fitness of the current population, the active nodes, and the number of function evaluations executed so far. To approximate the convergence time of each algorithm, we used the mean number of function evaluations it took to find the best solution during training *mean(F2B)*.

Additionally, in order to compare the different configurations based on solid statistical statements, we ranked the algorithms according to their final fitness values on the test data. Throughout the benchmarks, the fitness is always positive, therefore, a *t-distribution* can not model the data well [22]. Hence, we performed a Bayesian data analysis for the posterior distributions of our results. The model to compare the algorithms is based on the *Plackett-Luce model* described by Calvo et al. [6].[4]

6.1 CGP Variants and Configurations

To allow for a fair comparison a broad set of configurations have to be evaluated in addition to the ant-based metaheuristics. CGP is mostly used in combination with a (1+4)-ES with neutral search (we call it STANDARD; Sect. 2). However, the authors Kaufmann and Kalkreuth [20,21] found that a different parametrization of the $(\mu + \lambda)$-ES helps CGP achieve its full potential. Thus, we include $(\mu + \lambda)$-ES into our experiments to *ensure a fair comparison*. To go one step further, we also examine the impact of a (μ, λ)-ES as this approach is also very close to the $(\mu + \lambda)$-ES. Furthermore, the following three replacement strategies are examined in conjunction with both ES:

- Neutral Search [28]
- Random Selection
- Fitness Uniform Selection Scheme (FUSS) [17]

[3] By contrast, we use two pheromone tables for the functions and the connections, respectively.

[4] We used the Python library *cmpbayes* https://github.com/dpaetzel/cmpbayes.

For STANDARD, $(\mu + \lambda)$-ES, and (μ, λ)-ES, we tested both Single and probabilistic mutation.

To find the best hyperparameters for each metaheuristic, we used a *Tree-structured Parzen Estimator* implemented in the Python library *optuna* [1]. All configurations were tested four times with independent train–test splits and randomly chosen seeds. After finding the best hyperparameters for a benchmark, each CGP variant was run for 10 times with independent seeds and train–test splits for our evaluation. Each algorithm has a different set of hyperparameters that needs to be optimized. These hyperparameters are shown in Table 2.

Table 2. Hyperparameters tuned for each CGP version.

CGP variant	Hyperparameters
STANDARD	#nodes, mutation type $\in \{Single, probability\}$
$(\mu + \lambda)$, (μ, λ)	#nodes, μ, λ, mutation probability p, elitist selection scheme, mutation type $\in \{Single, probability\}$
AS, ASLBT	#nodes, β, m, τ_0, ρ, distance function
ACS	#nodes, β, m, τ_0, ρ, distance function, α, q_0, global/local best
MMAS	#nodes, β, m, τ_{min}, τ_{max}, ρ, distance function, global/local best

Table 3. An overview of the symbolic regression benchmarks used for testing. $U[a, b, c]$ means that c random samples are drawn from a uniform distribution in the range $[a, b]$. $E[a, b, c]$ defines a grid from a to b with spacing c.

Name	Variables	Equation	Data Set
Koza–3	1	$x^6 - 2 \times x^4 + x^2$	$U[-1, 1, 20]$
Pagie–1	2	$\frac{1}{1-x^{-4}} + \frac{1}{1-y^{-4}}$	$E[-5, 5, 0.4]$
Nguyen–7	1	$ln(x + 1) + ln(x^2 + 1)$	$U[0, 2, 20]$

To decrease the search space of ACS we use $q_0 := 0.9$ as it was proposed in [11]. For the Max-Min Ant System, we use the dynamic recalculation of τ_{min} and τ_{max} proposed by Stützle et al. [31] to simplify the search space even further.

6.2 Benchmarks

To evaluate our metaheuristics, 21 different symbolic regression, regression, and classification benchmarks were tested. We used the *symbolic regression* benchmarks Koza–3, Pagie–1 and Nguyen–7 [34] (cf. Table 3). Furthermore, we present results on the *regression* problems Forest Fires [8] and Wine Quality (White) [7]

and the *classification* problems Adult [4] and Chronic Kidney Disease [30].[5] The input data of every regression and classification dataset was normalized by implementing Min-Max Scaling on the range [0, 1]. The outputs of the regression problems were standardized by using the Z-score.

Each dataset was split randomly using a *Monte-Carlo Cross Validation* with an 80%–20% train–test split for ten runs each. For the regression and symbolic regression problems, the *mean squared error* is used as the fitness function. The classification problems are optimised with $1 - |MCC|$ as fitness function, where MCC is the *Matthews Correlation Coefficient*.

With these settings each CGP variant is classified as *solved* once the fitness value reaches less than 0.0001. Furthermore, each ant-based algorithm is seen as *converged* once there has not been an increase in fitness over the last 500 iterations. All algorithms are given a maximum of 100,000 iterations to finish their optimisation process which should ensure that convergence is achieved for even the slowest algorithm. Note that the number of iterations has to be multiplied by the population sizes to arrive at the number of function evaluations if a comparison would be made on budgets.

6.3 Results

We will now discuss our results on all different CGP variations. As our benchmark featured 21 different datasets from different types of learning tasks which generated a lot of data, we made a pre-selection and present the most interesting results in Table 4.[6] This article should be seen as a comprehensive benchmark that focusses on more than one application niche. As all datasets led to the generally same outcomes and findings with regards to the performance of the different optimizers, we are confident that out selection is made on sound assumptions and does not take away from a fair comparison.[7] However, we want to stress that the datasets not featured in the table are still important for our evaluation and discussion points.

[5] Additionally, we tested on the following additional regression and classification datasets (which are all publicly available as part of the UCI repository): Abalone, Air Quality, Appliances Energy Prediction, Bike Sharing Dataset - (Day & Hour), California Housing, Wine Quality (Red); Apnea-ECG, Bach Chorales Harmony, Car Evaluation, Diabetes. Their respective results as well as our *source code* can be found at https://github.com/trautwju/ACM_CGP.

[6] We made our selection for Table 4 based on a number of points: First, we considered datasets/learning tasks frequently used in the evolutionary computation community and featured in GP publications, e.g. Koza–3. Then, we limited the number of classification and regression datasets to give an approximately even representation. Last, we selected datasets that fulfil the criteria above and showed similar outcomes to others of the respective groups.

[7] Again, we refer to our GitHub https://github.com/trautwju/ACM_CGP and the supplementary material for the results on the remaining datasets.

Table 4. Results on a representative subset of our benchmark. We report the mean number of function evaluations until the best solution was found *mean(F2B)*, the mean fitness and its standard deviation *mean* ± *std(fit)*, total number of nodes *#nodes*, and the probability of one configuration being the best p_{best}.

	CGP Variant	mean(F2B)	mean ± std(fit)	#nodes	$\mathbf{p}_{\text{best}}$
Koza–3	Standard	49,003	0.00 ± 0.00	50	0.311
	$(\mu + \lambda)$-ES	67,528	0.00 ± 0.00	300	0.273
	(μ, λ)-ES	1,394,913	0.00 ± 0.00	650	0.270
	AS	212	0.01 ± 0.00	400	0.011
	ACS	126	0.02 ± 0.04	200	0.036
	MMAS	10,486	0.00 ± 0.00	250	0.054
	ASLBT	162	0.00 ± 0.00	500	0.046
Nguyen–7	Standard	152,306	0.00 ± 0.00	300	0.189
	$(\mu + \lambda)$-ES	261,586	0.00 ± 0.00	450	0.356
	(μ, λ)-ES	73,027	0.00 ± 0.00	50	0.396
	AS	771	0.18 ± 0.00	100	0.027
	ACS	1,201	0.59 ± 0.37	400	0.008
	MMAS	5,570	0.27 ± 0.22	400	0.018
	ASLBT	367	0.63 ± 0.47	800	0.005
Pagie–1	Standard	324,146	0.00 ± 0.00	950	0.301
	$(\mu + \lambda)$-ES	2,019,017	0.00 ± 0.00	450	0.417
	(μ, λ)-ES	2,979,659	0.01 ± 0.02	800	0.221
	AS	8,874	0.50 ± 0.17	350	0.006
	ACS	317	0.37 ± 0.21	100	0.016
	MMAS	10,795	0.27 ± 0.06	350	0.025
	ASLBT	410	0.42 ± 0.16	550	0.015
Forest Fires	Standard	253,605	0.41 ± 0.42	50	0.089
	$(\mu + \lambda)$-ES	1,731,673	0.61 ± 0.90	800	0.055
	(μ, λ)-ES	1,512,850	0.06 ± 0.00	550	0.647
	AS	384	0.21 ± 0.00	100	0.14
	ACS	192	2.98 ± 0.04	660	0.075
	MMAS	10,648	0.44 ± 0.86	700	0.114
	ASLBT	451	1.18 ± 1.39	300	0.041
Wine Quality (White)	Standard	389,552	0.74 ± 0.03	800	0.233
	$(\mu + \lambda)$-ES	2,850,189	0.75 ± 0.17	300	0.218
	(μ, λ)-ES	3,419,420	0.70 ± 0.07	550	0.372
	AS	155	0.98 ± 0.00	500	0.089
	ACS	669	0.93 ± 0.06	450	0.052
	MMAS	6,024	1.00 ± 0.03	200	0.015
	ASLBT	1,125	1.00 ± 0.06	200	0.021
Chronic Kidney Dis.	Standard	98,856	0.06 ± 0.06	400	0.255
	$(\mu + \lambda)$-ES	604,656	0.05 ± 0.02	600	0.261
	(μ, λ)-ES	1,050,944	0.02 ± 0.02	200	0.406
	AS	506	0.55 ± 0.31	250	0.008
	ACS	23	1.00 ± 0.00	350	0.003
	MMAS	9,467	0.27 ± 0.05	50	0.044
	ASLBT	590	0.38 ± 0.12	300	0.022
Adult	Standard	342,400	0.22 ± 0.00	350	0.265
	$(\mu + \lambda)$-ES	3,192,479	0.22 ± 0.00	600	0.350
	(μ, λ)-ES	2,358,825	0.22 ± 0.01	300	0.302
	AS	3,250	0.51 ± 0.26	500	0.005
	ACS	1,596	0.25 ± 0.06	550	0.055
	MMAS	12,090	0.31 ± 0.11	100	0.008
	ASLBT	1,520	0.40 ± 0.27	300	0.015

Symbolic Regression. On the symbolic regression benchmarks, the ant-based metaheuristics perform worse on every problem other than Koza–3. On Koza–3, all CGP variations apart from AS and ACS are able to solve the problem, but STANDARD, $(\mu + \lambda)$-ES, and (μ, λ)-ES still outperform all other metaheuristics.

On the other symbolic regression problems all ant-based algorithms clearly show a worse performance than STANDARD, $(\mu + \lambda)$-ES, and (μ, λ)-ES. ACS shows a relatively high standard deviation of the fitness value and a very low F2B as a locally optimal ant is found early on but never really improved upon. This is due to using the locally or globally best ant to update the pheromone table which apparently restricts the search space too much.

Still, the high standard deviation shows that it is possible for the ants to find better solutions. A possible improvement here is to implement some sort of restart or pheromone smoothing algorithm to reset the search and help ACS escape a local fitness minimum. MMAS shows a high standard deviation on Nguyen–7 which is unexpected because setting τ_{min} and τ_{max} should help with exploring the search space and our tuning process did not find values for these constants that achieved better results.

Regression. On the forestfire dataset, ASLBT did converge fast, but mostly ended up with a bad fitness score, although some runs did find good solutions quickly. Another interesting result is that AS outperforms STANDARD on Forest Fires. However, the wine quality white dataset shows that the ant-based metaheuristics often times cannot compete with the already established algorithms.

Classification. The classification problems show a similar result. Here, the ant-based metaheuristics can compete with STANDARD, $(\mu + \lambda)$ and (μ, λ)-ES in some runs but mostly reach a worse fitness score. However, the ant-based metaheuristics did converge considerably faster even when finding good results.

General Discussion. The additional implementations of the modified replacement schemes showed worse or at best only similar result to the established neutral search. Therefore, it seems to not have any positive impact to include further options which confirms the long-standing practice of using neutral search in CGP. The (μ, λ)-ES showed a significantly larger *mean(F2B)* than the STANDARD or $(\mu + \lambda)$-ES on all problems apart from Nguyen–7, but showed no significant improvement in fitness other than on Forest Fires. The $(\mu + \lambda)$-ES also had a larger *mean(F2B)* than STANDARD without an improvement in fitness. Probabilistic mutation outperformed Single most of the times with a well chosen mutation probability.

The optimal hyperparameters for each ant-based metaheuristic seem to be very dependent on the given problem, therefore making it impossible to decide on one generalist hyperparameter configuration. On most problems, the use of the globally best ant to update the pheromone tables for ACS and MMAS showed significantly better performance than using the iteration's best. Therefore, it

seems feasible to remove this hyperparameter from the search space for future tuning. Looking at the different distance functions, most commonly the constant distance lead to the best result, closely followed by the logarithmic distance. When the number of nodes are examined, we can not see any trends between the different CGP configurations. Similarly to the other hyperparameters, the required number of nodes are completely dependent on the given problem statement in combination with its respective CGP variant.

Despite the high variations of achieved fitnesses during testing, the ant-based metaheuristics showed small standard deviations during the training process. This comes from the fact that they assume that the optimal solution of a problem is very close to the locally best found solution so far. Furthermore, the biased exploration, which all ant-based algorithms use, increases the possibility of generating the same solution in different iterations because there is no guarantee an active node has been changed. Therefore, it could be argued that the optimal graph in CGP requires a lot more exploration of the search space.

Early during training, MMAS does seem to suffer less from a lack of exploration which is in line with the original proposal of the system being better at exploring the search space. This is the case because τ_{min} makes the ants choose non-optimal connections and functions more often. The lack of exploration in later iterations could be caused by our use of the pheromone update function, which might put too much emphasis on good solutions early on and therefore leading to a loss of exploration of other solutions.

Another interesting discovery is that the *intensive* tuning of the pheromone evaporation rate ρ did not always positively influence the exploration and exploitation balance of the search space. Instead, exploitation seems to be too high as all ant-based metaheuristics except MMAS converge within the first few hundred iterations. This hypothesis is supported by the high standard deviation of their fitness values: Given better exploration, ant-based CGP *should* be able to converge towards a much better mean fitness value and do so more consistently. However, we can assume that—judging from the very high *mean(F2B)*'s of the ES variants—they might very well still be faster than their competition.

As stated at the beginning of this section, we also performed extensive statistical testing to confirm the results. Even though the raw numbers were quite expressive we added the tests to adhere to good scientific practices and encourage future researchers to do the same, especially when the results are not this clear. What can be seen from the tests (p_{best} in Table 4) is that the probability of an ant-based CGP being the best according to fitness is almost zero. In fact, we reach typically suggested thresholds for automated decision making (e.g. at least 80% according to Benavoli et al. [5]) to eliminate ant-based metaheuristics from consideration as the best optimizer of a CGP graph. The ES-based CGP approaches show similar probabilities to each other and overall, there is no clear picture, but we can discern a small tendency towards using the (μ, λ)-ES with elitism. However, we want to stress that this is also by far the slowest approach we tested.

7 Conclusion

In this work, we investigated the effect different ant-based metaheuristics have on CGP performance. We compared these algorithms with a standard CGP variant and the commonly used $(\mu + \lambda)$-ES and (μ, λ)-ES.

In our testing, we found that the ant-based metaheuristics show no significant benefit with regards to achieved fitness. While previous work [15,23,24] shows similar results of the Max-Min ant system on chosen symbolic regression problems, we came to the conclusion that using ant-based metaheuristics most of the time leads to an overall worse result. Even though the introduction of different distance functions lead to significantly more active nodes over the CGP graph, this increase of active nodes does not have any positive impact on the fitness and is barely visible in the number of nodes needed for the optimal solution.

One redeeming quality of ant-based metaheuristics is their fast convergence by a factor of 100 or more compared to ES-based CGP approaches (which could also lead to premature convergence). Thus, given the same training budget for all configurations, more CGP graphs optimized by ant-based metaheuristics can be generated. Paired with the high standard deviation of their fitness values, ant-based CGP has a high possibility to generate a good solution with only a fraction of the computational power needed. We would recommend that the ant-based CGPs will be run multiple times with far fewer iterations. Besides multiple independent runs, a possible strategy is to implement a random restart scheme where the pheromone matrices and the random seed are reset after some set budget was used. If the current best solution is now placed into an archive, it is still available after training. We assume that some of the solutions in the archive will be far off the optimal one, similarly to what we did experience in some of our runs. Nonetheless, there should be solutions that do fit the learning task well and that all of this can be achieved with a number of function evaluations equal to the ESs'.

Interestingly, a premature convergence contradicts the claims of some ant-based metaheuristics. For example, the *Max-Min Ant System* introduces hyper-parameters that should improve exploration. This, however, seems to not be enough to improve graphs defined by CGP. Similar notions of ants often stuck in local optima were put forth by Prakasam and Savarimuthu [29]. The problem of locality of the search performed could also be investigated more in-depth. This could provide more insights into the behaviour of ant-based CGP, and why it performs badly.

Therefore, for additional future works, some improvements for ant-based metaheuristics that were proposed for optimization tasks could be tested. Scaling the value of β down over the course of iterations was proposed by Kushida et al. [23]. This leads to ants exploring shorter connections early and being able to choose longer connections later in the training, which might improve search space exploration. Furthermore, algorithms that help ACS escape local minima—like *pheromone trail smoothing* [31] or *2 Phase reinitialisation* [2]—should be tested to analyse if ant-based metaheuristics can improve the performance of CGP. At last, a different pheromone update function could be used to stop ants from

landing in a local fitness minimum early on. Another possibility is to introduce ideas from *Artificial Bee Colony* (ABC) [19]—or even completely substitute the ant-based metaheuristics. Because the population of ABC features specialised individuals with different tasks (individuals for exploration, exploitation, etc.), ABC may be better suited for optimizing graphs defined by CGP [3].

Overall, we find that our rigorous benchmarking rejects the use of ant-based metaheuristics in the same way we would use ESs in CGP. Still, there are clear paths to exploit the orders of magnitude faster convergence speeds towards achieving better results than previously possible. This becomes especially pronounced when computation budgets are limited.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019)
2. Altiparmak, F., Karaoglan, I.: A genetic ant colony optimization approach for concave cost transportation problems. In: 2007 IEEE Congress on Evolutionary Computation, pp. 1685–1692 (2007). https://doi.org/10.1109/CEC.2007.4424676
3. Baykasoğlu, A., Özbakir, L., Tapkan, P.: Artificial bee colony algorithm and its application to generalized assignment problem. In: Chan, F.T., Tiwari, M.K. (eds.) Swarm Intelligence, chap. 8. IntechOpen, Rijeka (2007). https://doi.org/10.5772/5101
4. Becker, B., Kohavi, R.: Adult. UCI Machine Learning Repository (1996). https://doi.org/10.24432/C5XW20
5. Benavoli, A., Corani, G., Demšar, J., Zaffalon, M.: Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. J. Mach. Learn. Res. **18**(1), 2653–2688 (2017)
6. Calvo, B., Ceberio, J., Lozano, J.A.: Bayesian inference for algorithm ranking analysis. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO '18, pp. 324–325. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3205651.3205658
7. Cortez, P., Cerdeira, A., Almeida, F., Matos, T., Reis, J.: Wine Quality. UCI Machine Learning Repository (2009). https://doi.org/10.24432/C56S3T
8. Cortez, P., Morais, A.: Forest Fires. UCI Machine Learning Repository (2007). https://doi.org/10.24432/C5D88D
9. Cui, H., Heider, M., Hähner, J.: Positional bias does not influence cartesian genetic programming with crossover. In: Affenzeller, M., et al. (eds.) Parallel Problem Solving from Nature – PPSN XVIII, pp. 151–167. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-70055-2_10
10. Cui, H., Pätzel, D., Margraf, A., Hähner, J.: Weighted mutation of connections to mitigate search space limitations in cartesian genetic programming. In: Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms. FOGA '23, pp. 50–60. Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3594805.3607130
11. Dorigo, M., Gambardella, L.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997). https://doi.org/10.1109/4235.585892

12. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **26**(1), 29–41 (1996). https://doi.org/10.1109/3477.484436
13. Fang, W., Gu, M.: FMCGP: frameshift mutation cartesian genetic programming. Complex Intell. Syst. **7**(3), 1195–1206 (2021). https://doi.org/10.1007/s40747-020-00241-5
14. Goldman, B.W., Punch, W.F.: Reducing wasted evaluations in cartesian genetic programming. In: Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A.Ş, Hu, B. (eds.) Genetic Programming, pp. 61–72. Springer, Heidelberg (2013)
15. Hara, A., Watanabe, M., Takahama, T.: Cartesian ant programming. In: 2011 IEEE International Conference on Systems, Man, and Cybernetics, pp. 3161–3166 (2011). https://doi.org/10.1109/ICSMC.2011.6084146
16. Harding, S., Graziano, V., Leitner, J., Schmidhuber, J.: MT-CGP: mixed type cartesian genetic programming. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation. GECCO '12, pp. 751–758. Association for Computing Machinery, New York (2012). https://doi.org/10.1145/2330163.2330268
17. Hutter, M., Legg, S.: Fitness uniform optimization. IEEE Trans. Evol. Comput. **10**(5), 568–589 (2006). https://doi.org/10.1109/TEVC.2005.863127
18. Kalkreuth, R.: Two new mutation techniques for cartesian genetic programming. In: Proceedings of the 11th International Joint Conference on Computational Intelligence, IJCCI 2019, pp. 82–92. SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT (2019). https://doi.org/10.5220/0008070100820092
19. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Global Optim. **39**(3), 459–471 (2007). https://doi.org/10.1007/s10898-007-9149-x
20. Kaufmann, P., Kalkreuth, R.: An empirical study on the parametrization of cartesian genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO '17, pp. 231–232. Association for Computing Machinery, New York (2017). https://doi.org/10.1145/3067695.3075980
21. Kaufmann, P., Kalkreuth, R.: On the parameterization of cartesian genetic programming. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2020). https://doi.org/10.1109/CEC48606.2020.9185492
22. Kruschke, J.K.: Bayesian estimation supersedes the t test. J. Exp. Psychol. Gen. **142**(2), 573–603 (2013). https://doi.org/10.1037/a0029146
23. Kushida, J.I., Hara, A., Takahama, T., Nagura, S.: Cartesian ant programming with transition rule considering internode distance. In: 2016 IEEE 9th International Workshop on Computational Intelligence and Applications (IWCIA), pp. 101–105 (2016). https://doi.org/10.1109/IWCIA.2016.7805756
24. Luis, S., dos Santos, M.V.: On the evolvability of a hybrid ant colony-cartesian genetic programming methodology. In: Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A.Ş, Hu, B. (eds.) EuroGP 2013. LNCS, vol. 7831, pp. 109–120. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37207-0_10
25. Margraf, A., Stein, A., Engstler, L., Geinitz, S., Hahner, J.: An evolutionary learning approach to self-configuring image pipelines in the context of carbon fiber fault detection. In: 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 147–154 (2017). https://doi.org/10.1109/ICMLA.2017.0-165
26. Miller, J.F.: An empirical study of the efficiency of learning Boolean functions using a cartesian genetic programming approach. In: Proceedings of the 1st Annual

Conference on Genetic and Evolutionary Computation. GECCO'99, vol. 2, pp. 1135–1142. Morgan Kaufmann Publishers Inc., San Francisco (1999)
27. Miller, J.F.: Cartesian genetic programming. In: Miller, J.F. (ed.) Cartesian Genetic Programming. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17310-3_2
28. Miller, J.F.: Cartesian genetic programming: its status and future. Genet. Program Evol. Mach. **21**(1), 129–168 (2020)
29. Prakasam, A., Savarimuthu, N.: Metaheuristic algorithms and polynomial turing reductions: a case study based on ant colony optimization. In: Proceedings of the International Conference on Information and Communication Technologies, ICICT, vol. 46, pp. 388–395. Procedia Computer Science (2015). https://doi.org/10.1016/j.procs.2015.02.035
30. Rubini, L., Soundarapandian, P., Eswaran, P.: Chronic Kidney Disease. UCI Machine Learning Repository (2015). https://doi.org/10.24432/C5G020
31. Stützle, T., Hoos, H.H.: Max-min ant system. Future Gener. Comput. Syst. **16**(8), 889–914 (2000). https://doi.org/10.1016/S0167-739X(00)00043-1
32. Suganuma, M., Kobayashi, M., Shirakawa, S., Nagao, T.: Evolution of deep convolutional neural networks using cartesian genetic programming. Evol. Comput. **28**(1), 141–163 (2020). https://doi.org/10.1162/evco_a_00253
33. Turner, A.J., Miller, J.F.: Neutral genetic drift: an investigation using Cartesian Genetic Programming. Genet. Program Evol. Mach. **16**(4), 531–558 (2015). https://doi.org/10.1007/s10710-015-9244-6
34. White, D.R., et al.: Better GP benchmarks: community survey results and proposals. Genet. Program Evol. Mach. **14**(1), 3–29 (2013). https://doi.org/10.1007/s10710-012-9177-2
35. White, T., Kaegi, S., Oda, T.: Revisiting elitism in ant colony optimization. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 122–133. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_11