On the Evolvability of a Hybrid Ant Colony-Cartesian Genetic Programming Methodology

Sweeney Luis and Marcus Vinicius dos Santos

Department of Computer Science, Ryerson University
Toronto, Canada
{sluis,m3santos}@ryerson.ca

Abstract. A method that uses Ant Colonies as a Model-based Search to Cartesian Genetic Programming (CGP) to induce computer programs is presented. Candidate problem solutions are encoded using a CGP representation. Ants generate problem solutions guided by pheromone traces of entities and nodes of the CGP representation. The pheromone values are updated based on the paths followed by the best ants, as suggested in the Rank-Based Ant System (AS_{rank}). To assess the evolvability of the system we applied a modified version of the method introduced in [9] to measure rate of evolution. Our results show that such method effectively reveals how evolution proceeds under different parameter settings. The proposed hybrid architecture shows high evolvability in a dynamic environment by maintaining a pheromone model that elicits high genotype diversity.

Keywords: Ant Colonies, Cartesian Genetic Programming, Rank-Based Ant System, Hybrid Architectures, Evolvability, Dynamic Environments.

1 Introduction

Dynamic problems are those in which the solution changes over time. In such domains, the ability of a population to evolve to a new region in the solution space is key. Tracking a moving optimum or moving from a local to a global optimum is facilitated when the problem representation and optimization methodology interact in ways to provide a high level of evolvability. In natural evolution, evolvability is the capacity for an adaptive response to a dynamic environment (fitness function) [1]. In genetic programming, a machine learning methodology concerned with evolving computer programs, the fitness function is in many cases static, so there is little selection pressure for "evolvability" in the biological sense. In the work presented here we set out to investigate the evolvability of a hybrid methodology that combines the Cartesian Genetic Programming (CGP) [10] representation and the probabilistic techniques for searching the solution space used in Ant Colony Optimization (ACO) [4].

Inspired on the Ants System introduced in [3], Ant Programming (AP) [11] extended ACO to Genetic Programming (GP) tree style representations. The AP system builds and modifies candidate problem solutions in accordance to the pheromone model referred as *pheromone tree*. The pheromone tree is composed of a number of pheromone tables for each tree node containing the pheromone values for the possible functions and terminals for the corresponding node. Using a similar approach, but based on different pheromone model, Dynamic Ant Programming (DAP) [14] introduced a dynamic tabular pheromone model which holds the pheromone values for the possible functions and terminals at each node and uses a *tabu list* restricting the selection of a non-terminal that has already been selected, thereby enabling the system to create diverse individuals. The pheromone table size changes dynamically in each iteration and nodes with low pheromone levels are deleted. This results in the system creating programs of smaller size on average.

In the work presented here, we propose a methodology that extends to CGP the representations introduced in probabilistic model building GP methodologies introduced in [8,5] and combines it with ACO. We begin by presenting in Section 2 the background materials regarding CGP and ACO. Then, in Section 3 we present how we take advantage of the CGP representation to propose a pheromone model that elicits genotype redundancies which have shown to be crucial for the evolvability of problem solutions [7,16]. We also introduce our learning algorithm, which draws on the Rank-Based Ant System (AS_{rank}) [2], where ants are ranked according to the quality of the CGP genomes they generate and the best ranked ants are used to update the pheromone table. The proposed method is similar to Cartesian Ant Programming (CAP) [6], where a pheromone model is sampled to create the genome of a CGP individual and the Max-Min Ant System (MMAS) [15] is used as the learning algorithm to update the pheromone model. To analyze the evolvability of our system, in Section 3 we propose a variant of the nonsynoymous to synonymous substitution ratio k_a/k_s introduced in [9]. In Section 4 we present the experimental design used to benchmark CAP against our approach. Our results show that in a static environment variability helps our system to converge to an optimal solution and neutrality helps preserve the optimal solution met. In a dynamic environment our system is able to maintain genotype diversity throughout the run which makes it highly adaptable to changes in the environment.

2 Background Materials

CGP is an artificial evolution methodology where the individuals are represented as a graph addressed on the Cartesian co-ordinate system and can be executed as a computer program. CGP distinguishes between genotype and phenotype unlike canonical genetic programming. In CGP, the genotype is a string of integers of fixed size which maps to the phenotype which is an executable graph. The genotype represents the graph's input and output connections. Each node consists of inputs and a function, each of which is represented by an integer number.

A CGP system needs prior definition of the following set of parameters $\{G, n_i, n_o, n_n, F, n_f, n_r, n_c, l\}$, where G is the genotype (fixed set of integers); n_i are the program inputs; n_o is the number of program output connections; n_n is the number of node input connections; F is the set of functions; n_f is the number of functions; n_r is the number of nodes in each row; n_c is number of nodes in each column; l is the level back parameter defining how many previous columns of nodes may have their output connected to the input of a node in the current column. In this paper only feed-forward connectivity is considered. The genotype size is fixed and can be calculated as $n_r * n_c * (n_n + 1) + n_o$.

ACO is a swarm optimization technique inspired by the behavior of some ants species. An ant moves from source to destination guided by the pheromone levels on the available paths the ant can travel. If a path is constantly being used by several ants, then more pheromone gets deposited on that path. Therefore, there is a greater possibility that an ant that comes along this path will choose the path to find its food. Similarly, the lower the pheromone levels on a path the smaller the possibility that an ant will choose that path.

For artificial ants a pheromone model is maintained that holds the pheromone values at each node for the paths possible for the ant to travel from that node. This model is updated after every iteration where the paths traveled by the ant that lead to better solutions have a level of pheromone deposited and the remaining paths have a level of their pheromone value evaporated. Pheromone update is the process where good solutions are rewarded by adding to the level of pheromone on paths chosen to reach that solution and evaporating the pheromone level on paths that did not yield a good solution.

3 Methods

The central hypothesis put forth in this work is that the CGP representation combined with an ACO algorithm that elicits redundancies in genotype to phenotype mapping imparts better evolvability of solutions. In the underlying algorithm, artificial ants iteratively generate quality solutions by updating the pheromone model used to create the programs. The model is updated after each iteration by rewarding the most fit programs of the previous iteration. Such rewarding is achieved by increasing the pheromone level of the inputs and function used in each node, and decreasing the pheromone of unvisited nodes.

Drawing on the work presented in [9] we propose a measurement for rate of evolution. We show that it effectively reflects how evolution is driven by the underlying algorithm, and we perform a study case of the system's evolvability.

Pheromone Model. The pheromone model contains the pheromone values for all available inputs and functions needed in generating a node of the program. All pheromone values are initialized at the start to τ_d for all the available inputs and functions for a node. The inputs that are unavailable to a node have their pheromone value set to zero to refrain them from being selected and generating cyclic graphs, *i.e.,* programs with loops. Table 1 shows an example of a pheromone model for a symbolic regression representation. In the example,

Table 1. Representation of the pheromone model

Node	**Input 1**					**Input 2**					**Function**				**Output Node**					
	1.0	**X**	**0**	**1**	**2**	**1.0**	**X**	**0**	**1**	**2**	$+$	**-**	$\times$	$\div$	**1.0**	**X**	**0**	**1**	**2**	**3**
0	τ_d	τ_d	0	0	0	τ_d	τ_d	0	0	0	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d
1	τ_d	τ_d	τ_d	0	0	τ_d	τ_d	τ_d	0	0	τ_d	τ_d	τ_d	τ_d						
2	τ_d	τ_d	τ_d	τ_d	0	τ_d	τ_d	τ_d	τ_d	0	τ_d	τ_d	τ_d	τ_d						
3	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d						

$n_i = \{1.0, X\}$, $F = \{+, -, \times, \div\}$, $n_n = 2$, $n_r = 1$, $n_c = 4$, $n_o = 1$ and $l = 3$. These inputs and functions are available for all nodes which are initialized to τ_d at the start.

Genome Creation: Each ant creates an individual's genotype by sampling the pheromone model. The ant samples the pheromone table and selects the appropriate input or function available, for each position in the genotype. The probability p_i that the ant selects a particular input or function i is given by $p_i = \frac{\tau_i}{\sum_{j=1}^{n_n} \tau_j}$, where n_n is the number of available inputs and functions.

Pheromone Update: The pheromone table is updated at the end of every iteration. The ants are ranked according to the quality of the solution they generate. Out of the m ants in each iteration only the $(n-1)$ *best-ranked ants* and the *best ant*, *i.e.,* the ant that produced the best solution so far (this ant could be from the current iteration or from a previous iteration) update the pheromone table as given by the following expression:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{r=1}^{n-1} (w - w_r) \Delta\tau_{ij}^{r}(t) + w\Delta\tau_{best} \tag{1}$$

where: $\tau_{ij}(t)$ is the current pheromone level of input(or function) i at node j at iteration t; w is a constant weight assigned at the start of the experiment; r is the rank of the ant; $w_r = w/(n-r)$; $(w - w_r)$ is the weight calculated that rewards higher ranked ants; and $\Delta\tau_{ij}^{r}(t)$ is equal to the fitness of rth-best ant, if ant selects input or function i at node j; otherwise it is zero. Analogously, $\Delta\tau_{best}$ is equal to the fitness of best ant.

Measuring Rate of Evolution: The nonsynonymous to synonymous substitution ratio k_a/k_s is a concept used to measure genetic substitution in molecular biology. This measurement has been used in [9] to quantitatively assess evolvability in Linear Genetic Programming (LGP). A change in the genome of an individual that brings about a change in its fitness is known as a *nonsynonymous* change, where as if a genome change does not cause a change in fitness, then it is called a *synonymous* change. We use the same terminology and a similar approach in the work presented here. One key difference, is that in the context of Ants (and Estimation of Distribution Algorithms) changes are brought about by probabilistic sampling of the pheromone table rather than by the application of genetic operators, which is the case in [9].

In our simulated studies we measure the k_a/k_s ratio by observing the changes brought about in the *best-ranked* ants. To determine the value of nonsynonymous (and synonymous) change we compare each individual I and individual J that were brought about by the N best-ranked ants of iteration t and $t-1$, respectively. The value of nonsynonymous change $m^I_{ak}(t)$ on each gene k of each individual I from iteration t is calculated as follows: if gene k did not change, i.e., $I_k = J_k$, then $m^I_{ak}(t) = m^I_{sk}(t) = 0$. Otherwise, if change was *silent*, i.e., individuals I and J have same fitness, then $m^I_{ak}(t) = 0, m^I_{sk}(t) = 1$. If change was not silent, then $m^I_{ak}(t) = 1, m^I_{sk}(t) = 0$.

We compute the number of nonsynonymous substitutions $M_a(t)$ and the number of synonymous substitutions $M_s(t)$ as follows: $M_a(t) = \sum_{i=1}^{N} m^i_{ak}(t),\ M_s(t) = \sum_{i=1}^{N} m^i_{sk}(t)$

Like in [9], we keep a record of all changes to each gene during the iterations of the algorithm. We compute such accumulated numbers of nonsynchronous $c_{ak}(t)$ and synchronous $c_{sk}(t)$ changes in gene k up to iteration t, as follows: initially $c_{ak}(0) = c_{sk}(0) = 0$, for all genes k in the genome. We update these values for each gene k and individual I brought about by the N best-ranked ants of iteration t, as follows: $c_{ak}(t) = c_{ak}(t-1) + m^I_{ak}(t),\ c_{sk}(t) = c_{sk}(t-1) + m^I_{sk}(t)$.And we compute the *potential* of a gene k being changed nonsynonymously or synonymously (also called the *sensitivity* of a gene) as follows: $n_{ak}(t) = \frac{c_{ak}(t)}{c_{ak}(t)+c_{sk}(t)},\ n_{sk}(t) = \frac{c_{sk}(t)}{c_{ak}(t)+c_{sk}(t)}$

We add up the sensitivities of all genes in the representation to obtain the total nonsynonymous and synonymous sensitivities $N_a(t)$ and $N_s(t)$: $N_a(t) = \sum_{k=1}^{N} n_{ak}(t),\ N_s(t) = \sum_{k=1}^{N} n_{sk}(t)$

Finally, we compute the nonsynonymous and the synonymous substitution rates k_a and k_s of iteration t as $k_a(t) = M_a(t)/N_a(t)$, $k_s(t) = M_s(t)/N_s(t)$, which enables us to obtain the rate of evolution R_e in iteration t:

$$R_e(t) = k_a(t)/k_s(t) \tag{2}$$

4 Experimental Design and Results

We begin by testing how effectively the rate of evolution reflects the dynamics of the underlying algorithm. We then compare the evolvability of our method with that of CAP in two different environmental conditions: a fixed target and a moving target environment.

4.1 Rate of Evolution under Different Parameter Settings

In this section we describe the experimental setting we have designed to study the influence of different parameter settings on factors related to the Rate of Evolution in a symbolic regression (SR) problem

In these experiments the target expression is the polynomial $x^4 + x^3 + x^2 + x$. The training set, *i.e.,* the fitness cases, consists of 40 equidistant example points in the interval $[-2.0, +2.0]$. Fitness of an individual is computed as the inverse of the accumulated error between the actual example points and the values output by the individual. Formally, let the output of the ith training example be o_i. Let the output of individual g on the ith example from the training set be g_i. Then, for a training set of $n = 40$ examples the fitness f_g of g is calculated as follows: $f_g = \frac{1}{1+\sum_{i=1}^{n} |o_i - g_i|}$

The parameters chosen for this experiment are shown in Table 2. The exponentially weighted moving average method, with a smoothing factor 0.1, is used to smoothen the curves.

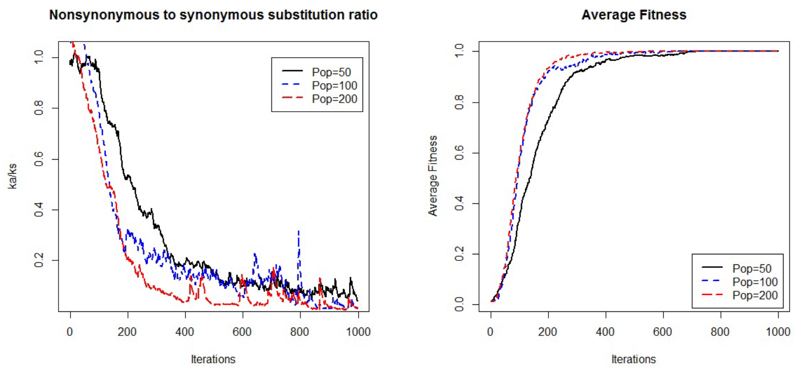
Table 2. Parameter values for experiments SR, FTE and MTE

Parameter	**Experiment**		
	SR	**FTE**	**MTE**
n_i	$\{x, 1.0\}$	ditto	ditto
F	$\{+, -, \times, \div\}$	ditto	ditto
Number of Runs	100	ditto	ditto
Number of Iterations	1000	ditto	ditto
Number of Individuals	50	100	100
τ_d	1.0	ditto	ditto
Number of best-ranked ants	10	ditto	ditto
# of equidistant fitness cases	40	ditto	ditto
Interval	$[-2.0, 2.0]$	ditto	ditto

Population Size. In this experiment we alter the population sizes to 50, 100 and 200. From Figure 1(b) we see having a system with a larger population converges to the solution faster than a system with a smaller population.

The rate of evolution (R_e) in Figure 1(a) is synchronous with natural evolutions as it continues to be at the maximum value slightly above 1.0. As the run progresses the value of R_e continues to descend till it reaches *zero* evolution. The system with a larger population size reached *zero* evolution quicker than a system with a smaller population.

Initial Pheromone Level. In this experiment we study the influence of the initial pheromone value on R_e. The initial pheromone level is set to 0.1, 0.5 and 1.0 and analyze the effect on the rate of evolution and average fitness. From Figure 2(b) we see, a setup with a lower initial pheromone level converges to a solution quicker which reflects the findings in [14]. Observing R_e in Figure 2(a) in all conditions the maximum value is around 1.0 and proceeds towards *zero* evolution. With a lower initial pheromone level *zero* evolution reaches quicker than with a setup where the initial pheromone level is higher.



(a) $R_e = k_a/k_s$ (b) Average fitness

Fig. 1. Varying population sizes

4.2 CGP-ACO vs. CAP

Our approach differs from Cartesian Ant Programming (CAP) [6] in two key aspects: the learning algorithm and the method of updating the pheromone model.

CAP uses Max-Min Ant System (MMAS) [15] as the learning algorithm, where the *best ant* updates the pheromone model at the end of the iteration and the unused pheromone trails are subjected to evaporation. In MMAS, the pheromone model is updated according to the equation below:

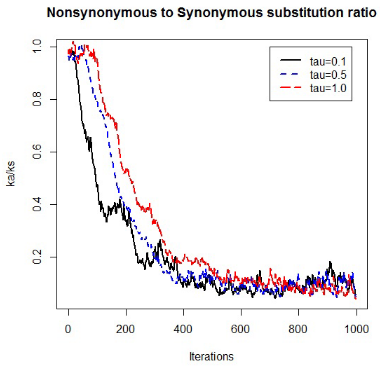
$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{best}(t) \tag{3}$$

where: $\tau_{ij}(t)$ is the current pheromone level of input(or function) i at node j at iteration t; ρ is the evaporation rate; $\Delta\tau_{ij}^{best}$ is equal to the fitness of the best ant, if best ant selects input or function i at node j; otherwise it is zero.

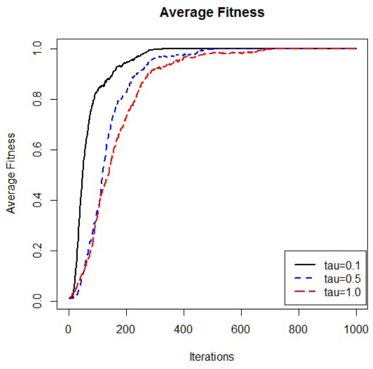
In regards to the method of updating the pheromone model, after evaluating the individual, CAP traverses the model beginning from the output nodes and proceeds backwards towards the input nodes, only updating the model for the nodes that appear in the individuals phenotype. As such, only the pheromone values of the used nodes are updated. In our method, however, we update the pheromone model in a forward manner beginning from the input nodes and moving towards the output nodes, updating the pheromone values for all nodes that appear in the individual's genotype.

Fixed Target Evolution (FTE). In this experiment we evolve the expression x^5-2x^3+x. Figure 3(d) shows that our approach attains maximum fitness faster with a steady convergence rate throughout the run than the approach used in CAP.

Figure 3(a) shows that our method engenders a high number of nonsynonymous substitutions from the start of the run, and that number increases after the



(a) $R_e = k_a/k_s$



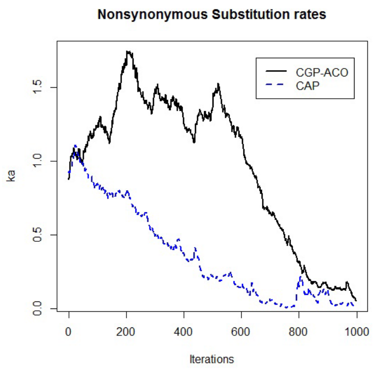
(b) Average fitness

Fig. 2. Varying initial pheromone level

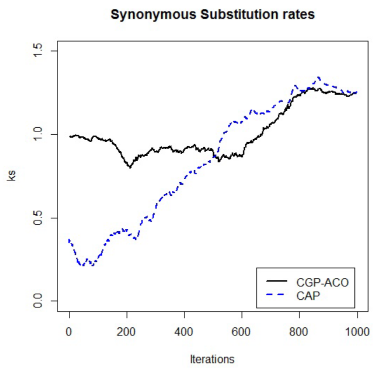
system converges to a solution. Using the method of updating the pheromone model as described in CAP, the system fixates on making a maximum number of nonsynonymous substitutions at the start of the run which results in the synonymous substitutions to be low at the start of the run and increase as the system converges to a solution of maximum fitness. In Figure 3(b) we plot the synonymous substitutions which shows us that using the CGP-ACO approach results in making a larger number of substitutions throughout the run which increases after the system converges to a solution. The method used in CAP involves making a fewer number of substitutions compared to the CGP-ACO approach, the system makes most of the substitutions after the system has converged to a solution. The result of the substitutions is seen in the rate of evolvability R_e in Figure 3(c), the CGP-ACO approach has a higher rate of R_e throughout the run where the system converges to *zero* evolvability towards the end of the run. Using the pheromone update method as used in CAP the model the system has a high value of R_e at the start of the run and reaches *zero* evolvability sooner than the CGP-ACO approach.

We conduct the Multiple Hypothesis Testing [13] to compare the performance of the two approaches. We test for the average fitness at different points of the run using a significance level $\alpha = 0.05$. We find CAP has a better performance in the initial stages of the run (at iteration 250, the p-value equals 0.012). However in the middle (at iteration 500, p-value = 0.58) and final stages (at iteration 750, p-value = 0.33) of the run there is no evidence of statistical difference between the two approaches.

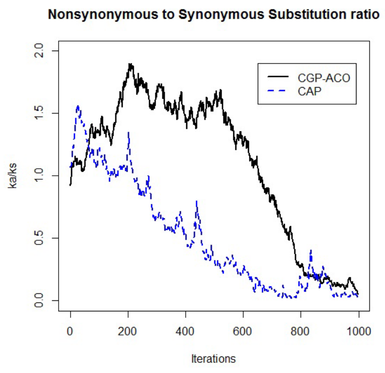
Moving Target Evolution (MTE). In this experiment we create a moving target by increasing the degree n of the polynomial $\sum_{i=1}^{n} x^i$ at regular intervals. We start our target for the first 200 iterations at $i = 3$ where the target expression is $x^3 + x^2 + x$. From iteration 200 to 500 $i = 4$ and iteration 500 onwards $i = 5$.



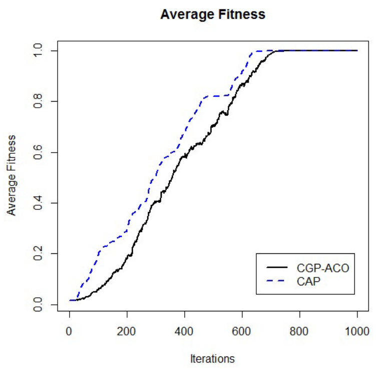
(a) k_a



(b) k_s



(c) $R_e = k_a/k_s$



(d) Average fitness

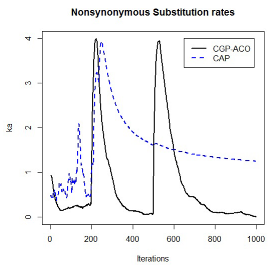
Fig. 3. FTE: CGP-ACO vs CAP

We use an enhanced version of the AS_{rank} model update method that includes evaporation. We incorporate evaporation to balance the effects of depositing a large amount of pheromone when the target is small which hinders exploration of alternative paths when the target is changed. For this experiment we set the evaporation rate to $\rho = 0.2$.

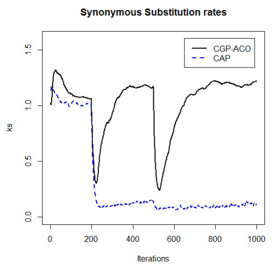
We emulate CAP's methodology using MMAS as the learning algorithm and with an evaporation rate of $\rho = 0.1$.

In Figure 4(d) we plot the average fitness for both systems. We take note that the convergence of the solution is almost identical for both systems. Analyzing the substitution rates in Figure 4(a) and 4(b) we see that both systems follow a similar trend in evolvability from iteration 0 to 200 where the target is an expression of a lower degree. From iteration 200 to the end of the run, as the target is changed and the degree of the polynomial keeps increasing, we notice that the rate of nonsynonymous substitutions in CAP is greater than in CGP-ACO. Also the number of synonymous substitutions made during this period in CAP is low compared to CGP-ACO. Moreover, Figure 4(d) shows that immediately after the target change CGP-ACO is able to produce solutions with higher fitness than CAP, which makes one wander if the CGP-ACO pheromone model elicits a more diverse genotype than the CAP pheromone model. Figure 4(e) shows that there is strong evidence pointing in that direction. Genotype diversity, in this case, was calculated using the diversity measure introduced by Shapiro in [12] (section 3 of that paper).

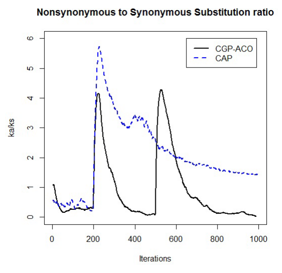
To assess the statistical significance we conduct the same Multiple Hypothesis Test as in the previous section (with $\alpha = 0.05$). Testing for the average fitness at different points of the run we find no evidence of statistical difference between the two approaches (at iteration 100, p-value = 0.71; at iteration 350, p-value = 0.3 and at iteration 750, p-value = 0.76). Testing for diversity at different points of the run we find very strong evidence of a statistical difference between the two approaches, showing CGP-ACO elicits higher diversity as the p-value at different points of the run are extremely small.



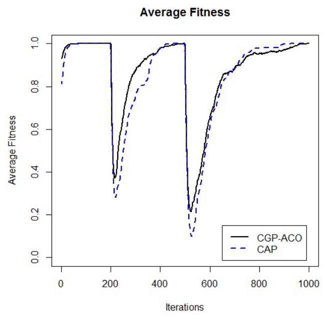
(a) k_a



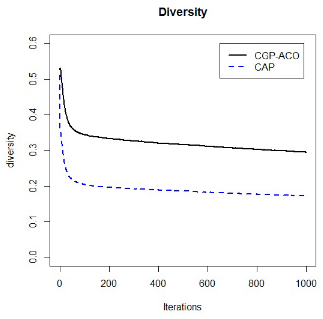
(b) k_s



(c) $R_e = k_a/k_s$



(d) Average Fitness



(e) Diversity

Fig. 4. MTE: CGP-ACO vs CAP

5 Conclusion

In this work we introduced a hybrid optimization algorithm that combines Cartesian genetic programming with ant colonies. Using the rate of evolution we tested the evolvability of our system under different parameter settings and compared our results with CAP in two different environmental conditions

Analyzing the results we observed that a lower level of initial pheromone and a bigger population size helps in faster convergence. CGP-ACO has shown to be highly adaptable to the aforementioned environmental conditions. Our results also showed that CGP-ACO is on par with CAP with regards to average fitness of the evolved population. CGP-ACO, however, showed better adaptiveness when faced with a dynamic environment by maintaining a highly diverse genotype population.

The system imposes variability as a driving force when it needs to attain an optimal solution, where as neutrality helps the system preserve an optimal solution. Adaptiveness is a major characteristic of our system as changes in the environment causes the system to reflect those changes in the pheromone model, thus resulting in the creation of individuals that are highly fit for the environmental conditions.

References

1. Altenberg, L.: The evolution of evolvability in genetic programming. In: Advances in Genetic Programming, pp. 47–74. MIT Press, Cambridge (1994)
2. Bullnheimer, B., Hartl, R.F., Strauß, C.: A new rank based version of the ant system - a computational study. Central European Journal for Operations Research and Economics 7, 25–38 (1997)
3. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed Optimization by Ant Colonies. In: European Conference on Artificial Life, pp. 134–142 (1991)
4. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization – artificial ants as a computational intelligence technique. IEEE Comput. Intell. Mag. 1, 28–39 (2006)
5. Ghoulbeigi, E., dos Santos, M.V.: Probabilistic developmental program evolution. In: Proceedings of the 2010 ACM Symposium on Applied Computing, SAC 2010, pp. 1138–1142. ACM, New York (2010)
6. Hara, A., Watanabe, M., Takahama, T.: Cartesian ant programming. In: SMC, pp. 3161–3166 (2011)
7. Harding, S., Miller, J.F., Banzhaf, W.: Smcgp2: self modifying cartesian genetic programming in two dimensions. In: GECCO, pp. 1491–1498 (2011)
8. Holker, G., dos Santos, M.V.: Toward an estimation of distribution algorithm for the evolution of artificial neural networks. In: Proceedings of the Third C* Conference on Computer Science and Software Engineering, C3S2E 2010, pp. 17–22. ACM, New York (2010)
9. Hu, T., Banzhaf, W.: Neutrality and variability: two sides of evolvability in linear genetic programming. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO 2009, pp. 963–970. ACM, New York (2009)
10. Miller, J.F., Thomson, P.: Cartesian genetic programming (2000)
11. Roux, O., Fonlupt, C.: Ant programming: Or how to use ants for automatic programming. In: From Ant Colonies to Artificial Ants 2nd International Workshop on Ant Colony Optimization (2000)
12. Shapiro, J.L.: Diversity Loss in General Estimation of Distribution Algorithms. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 92–101. Springer, Heidelberg (2006)

13. Shilane, D., Martikainen, J., Dudoit, S., Ovaska, S.J.: A general framework for statistical performance comparison of evolutionary computation algorithms. Inf. Sci. 178(14), 2870–2879 (2008)
14. Shirakawa, S., Ogino, S., Nagao, T.: Dynamic ant programming for automatic construction of programs. IEEJ Transactions on Electrical and Electronic Engineering TEEE 3, 540–548 (2008)
15. Stützle, T., Hoos, H.H.: MAX-MIN Ant System (November 1999)
16. Woodward, J.R.: Complexity and cartesian genetic programming. In: Mirkin, B., Magoulas, G. (eds.) The 5th Annual UK Workshop on Computational Intelligence, London, September 5-7, pp. 273–280 (2005)