Parametrizing Cartesian Genetic Programming: An Empirical Study

Conference Paper *in* Lecture Notes in Computer Science · September 2017

DOI: 10.1007/978-3-319-67190-1_26

2 authors:

Paul Kaufmann
Johannes Gutenberg-Universität Mainz
57 PUBLICATIONS **401** CITATIONS
SEE PROFILE

Roman Tobias Kalkreuth
Technische Universität Dortmund
21 PUBLICATIONS **39** CITATIONS
SEE PROFILE

Parametrizing Cartesian Genetic Programming: An Empirical Study

Paul Kaufmann[1] and Roman Kalkreuth[2]

[1] Department of Computer Science, Paderborn University, Germany
[2] Department of Computer Science, University of Dortmund, Germany

Abstract. Since its introduction two decades ago, the way researchers parameterized and optimized Cartesian Genetic Programming (CGP) remained almost unchanged. In this work we investigate non-standard parameterizations and optimization algorithms for CGP. We show that the conventional way of using CGP, i.e. configuring it as a single line optimized by an (1+4) Evolutionary Strategies-style search scheme, is a very good choice but that rectangular CGP geometries and more elaborate metaheuristics, such as Simulated Annealing, can lead to faster convergence rates.

1 Introduction

Almost two decades ago Miller, Thompson, Kalganova, and Fogarty presented first publications on CGP—an encoding model inspired by the two-dimensional array of functional nodes connected by feed-forward wires of an Field Programmable Gate Array (FPGA) device [6, 1]. CGP has multiple pivotal advantages:

- CGP comprises an inherent mechanism for the design of simple hierarchical functions. While in many optimization systems such a mechanism has to be implemented explicitly, in CGP multiple feed-forwards wires may originate from the same output of a functional node. This property can be very useful for the evolution of goal functions that may benefit from repetitive inner structures.
- The maximal size of encoded solutions is bound, saving CGP to some extent from "bloat" that is characteristic to Genetic Programming (GP).
- CGP offers an implicit way of propagating redundant information throughout the generations. This mechanism can be used as a source of randomness and a memory for evolutionary artifacts. Propagation and reuse of redundant information has been show beneficial for the convergence of CGP.
- CGP encodes a directed acyclic graph. This allows to evolve topologies. An example is the evolution of Artificial Neural Networks (ANNs) using CGP [7].
- CGP is simple. The implementation complexity in many programming languages is marginal relying on no special programming language properties like the ability to handle tree structures efficiently.

Along with the mentioned advantages, CGP suffers as a combinatorial representation model from the usual sources of epistasis. For instance, rewiring a single input of a functional node can change the overall transfer function dramatically. Additionally, the spatial arrangement of functional nodes on a two-dimensional grid introduces restrictions to the topology of the evolved solutions. Moving a functional node among the grid requires rearranging the genotype, if possible. Additionally, the connection set of an input of a node strongly depends on the location of the node on the grid. These dependencies implicitly impact on the evolvability and make it difficult to realize structural methods for CGP. For instance, a recombination operator needs to restructure large parts of a genotype to be able to swap functionally related substructures among candidate solutions [8, 2]. A trial to free CGP from grid-induced epistasis was made in [3] by assigning each input and output of a node signatures. Best-fitting signatures were then used to clamp wires.

The first systematic investigation on an efficient optimization scheme for CGP was done by Miller 1999 in [5]. Miller employed a regular GA with a uniform recombination operator and a $(1 + \lambda)$ mutation-only search scheme. He configured CGP as a square grid of functional nodes with the maximal length of feed-forward wires of two. In 1999 it was already know that a "neutral selection" scheme that is preferring offspring individuals for propagating into the next generation if they are on par or better than the parent individual is highly beneficial for CGP. In a series of experiments Miller observed that the evolution of digital circuits using CGP can be solved better by local search-like approaches employing "neutral selection" than by GA.

In this work we address the question, whether the popular choice of $(1 + 4)$ search scheme in combination with single-line CGP genotype can be generalized. For this, we rely on an unbiased parameter tuning method to identify (i) well-performing parameterizations of CGP and (ii) efficient optimization schemes.

2 Experimental Setup

The first class of benchmark functions consists of Boolean adder, multiplier, and even parity functions. The set of 2-input Boolean functions that may be used as functional nodes in CGP genotypes is presented in Tab. 1. An experiment is stopped if a perfect solution has been found or the maximal number of fitness evaluations has been exceeded.

The second set of benchmark consists of twelve symbolic regression functions (Koza-2, -3, Nguyen-4 . . . -10, Keijzer-4, -6, Pagie-1). A training data set consists of c uniformly sampled from an interval $[a, b]$. The cost function is defined as the sum of absolute differences between functional values of the reference and evolved function at the data points of the according training set. An experiment is terminated if the cost function reaches a value below or equal 0.01 or the maximal number of fitness evaluations has been exceeded.

Table 1: Functional set and arameter space explored by iRace.

Benchmarks	Functional set
$(i,i,1)$-add, (i,i)-mul	$a \wedge b$, $a \wedge \bar{b}$, $\bar{a} \wedge b$, $a \oplus b$, $a \vert b$
even parity	$a \wedge b$, $a \wedge \bar{b}$, $\bar{a} \wedge b$, $a \vert b$, $a \vert \bar{b}$, $\bar{a} \vert b$
Koza	$+$, $-$, $*$, $/$, sin, cos, $\ln(\vert n \vert)$, e^n
Keijzer	$+$, $*$, n^{-1}, $-n$,$\sqrt{n}$

Optimization Algorithms

As the baseline method we select the (1+4) CGP The second and third algorithms are $(1 + \lambda)$ CGP and $(\mu + \lambda)$ CGP, where the number of offspring individuals λ and the number of parents μ are subject to optimization. For all CGPs schemes "neutral selection" has been realized. For optimizing Boolean circuits we have additionally selected SA with the following colling strategy:

$$A \leftarrow \frac{(T_{\text{start}} - T_{\text{end}})(N+1)}{N}; \qquad B \leftarrow T_{\text{start}} - A; \qquad T_t \leftarrow \frac{A}{t+1} + B.$$

Random sampling and random walk have also been investigated in preliminary experiments and sorted out because of inferior results.

Automatic Parameter Tuning: To detect good parameterizations, we are using the Iterated Race for Automatic Algorithm Configuration (iRace) package [4]. iRace was configured to execute 2000 trials for each of the tested algorithm-benchmark pairs.

iRace usually evolves multiple good-performing configurations for an algorithm-benchmark pair. To verify the results of iRace, we have computed for each configuration the median performance in 100 runs. We have then selected for each algorithm-benchmark pair the best performing configuration and report it in this paper.

3 Results

Evolution of Boolean circuits: The first observation that can be made is that the baseline (1+4) CGP on a single-line CGP is never a clear winner regarding the median number of fitness evaluations when evolving functionally correct Boolean circuits (c.f. Tab. 2). Except for the smallest benchmarks, the $(2,2,1)$-adder and the $(2,2)$-multiplier, and for the 8-parity benchmark SA is always a clear winner. For the 8-parity benchmark SA is passed by $(\mu+\lambda)$ CGP only by roughly 2%. For larger benchmarks, like the $(3,3,1)$- and $(4,4,1)$-adder, $(3,3)$-multiplier, and the parity benchmarks, the best performing algorithm is 1.3 to 3 times faster than the baseline (1+4) CGP. When looking at the CE metric, SA is the clear winner for all but the smallest and the $(3,3,1)$-adder benchmarks. Sometimes, the best performing algorithm regarding the median

Table 2: Evaluation of CGP parameters for Boolean functions. Not optimized parameters are marked with an "–". The comparison prefers conventional (1+4) CGP, as iRace budget is set to 2000 for all configurations and challengers have more parameters to optimize. The results are measured in number of fitness evaluations. Best results are printed in *bold*. n_c and n_r - number of CGP columns and rows; m - mutation rate; T_{start} and T_{stop} - starting and stopping temperatures for SA. CE at $z = 99\%$.

goal function	algo-rithm	evolved parameters							termination	no. evaluations		Comp. Effort	restart at eval.
		n_c	n_r	μ	λ	m[%]	T_{start}	T_{stop}	1Q	median	3Q		
(2,2,1) add	1+4 CGP	200	–	–	–	2.1112	–	–	14916	26532	49840	160753	91840
	1+λ CGP	100	200	–	3	0.3215	–	–	11316	18933	28797	89280	34350
	μ+λ CGP	200	50	1	1	0.3803	–	–	8114	**13129**	21723	**67860**	19849
	SA	200	2	–	–	1.8976	1299	0.0348	12242	20052	35411	109530	42284
(3,3,1) add	1+4 CGP	200	–	–	–	2.1512	–	–	113168	194120	326156	**689115**	689112
	1+λ CGP	150	1	–	3	1.9464	–	–	105789	178344	302211	929794	581961
	μ+λ CGP	100	4	1	3	0.8396	–	–	122460	190539	330936	1018919	451407
	SA	70	4	–	–	1.3706	4671	0.4366	88335	**149817**	246126	750368	621896
(4,4,1) add	1+4 CGP	200	–	–	–	1.2341	–	–	424924	697152	1182452	2830424	2404400
	1+λ CGP	300	2	–	2	0.6852	–	–	303080	501550	698950	2206982	1680482
	μ+λ CGP	100	4	1	1	1.1503	–	–	364545	545438	936699	2469195	2097544
	SA	150	3	–	–	0.6693	3610	0.6437	283038	**400832**	723341	**2034761**	1422236
(2,2) mul	1+4 CGP	100	–	–	–	2.9542	–	–	3452	5564	9136	28434	14864
	1+λ CGP	100	100	–	3	0.8680	–	–	2121	3417	5474	**16512**	9009
	μ+λ CGP	100	30	1	1	1.4332	–	–	2079	**3322**	5465	17349	7279
	SA	30	14	–	–	2.4941	58	0.0889	2661	4183	6801	21275	9959
(3,3) mul	1+4 CGP	2000	–	–	–	0.5008	–	–	274228	447220	722280	2103815	1787156
	1+λ CGP	200	20	–	2	0.2988	–	–	149824	288368	459822	1203021	1203020
	μ+λ CGP	150	30	1	2	0.2971	–	–	130250	224178	498888	1382722	361496
	SA	200	100	–	–	0.1622	3336	0.0870	84844	**148145**	356305	**949607**	169289
7-parity	1+4 CGP	300	–	–	–	1.2582	–	–	175628	271048	427788	1347746	645976
	1+λ CGP	300	8	–	2	0.7142	–	–	100408	186250	262668	762572	381284
	μ+λ CGP	300	2	1	2	0.9089	–	–	118996	186674	291118	696589	696588
	SA	150	8	–	–	0.7584	1528	0.2000	87773	**140463**	238599	**539214**	458054
8-parity	1+4 CGP	2000	–	–	–	0.9057	–	–	336420	461948	739504	2113156	1374636
	1+λ CGP	200	6	–	3	1.0381	–	–	310524	486894	798396	2408346	932859
	μ+λ CGP	300	6	1	1	0.5578	–	–	192417	**323192**	455204	1404562	702280
	SA	300	4	–	–	0.6733	417	0.3479	213877	329472	479532	**1196482**	1196482
9-parity	1+4 CGP	2000	–	–	–	0.8718	–	–	628536	1011220	1718660	5380705	1487336
	1+λ CGP	150	3	–	2	0.7050	–	–	617418	959194	1570728	2859287	2859286
	μ+λ CGP	300	3	1	1	0.8519	–	–	512420	755543	1239866	3073095	1774561
	SA	300	10	–	–	0.3784	2209	0.2907	392406	**579111**	910828	**2209561**	1876989

number of fitness evaluations is not the winner regarding the CE. However, the differences in medians and CE values between the winner algorithm regarding the median and the winner algorithm regarding the CE are small to marginal.

Although we have showed for Boolean benchmarks that the conventional way of parameterizing CGP can always be outperformed, we would like to emphasize the following fact: Neither the best performing algorithm regarding the median nor the best algorithm regarding the CE metric can be in general considered dominant when it comes to the computational complexity of optimization and with it, time. The reason for this is the inaccurate assumption that the computational complexity of a fitness evaluation is constant among all CGP parameterizations. For example, $(\mu + \lambda)$ CGP is the best-performing algorithm regarding the median and CE metrics for the $(2, 2, 1)$-adder. However, despite worse median and CE values, (1+4) CGP operating on a single-line CGP and SA evolve functionally correct adders in much shorter time. This is because the genotype sizes found by iRace are much smaller for the two algorithms than for the $(\mu + \lambda)$ CGP. But even having identical CGP geometries the functional

Table 3: Evaluation of CGP parameters for symbolic regression functions. Not optimized parameters are marked with an "–". The comparison prefers conventional 1+4 CGP, as iRace budget is set to 2000 for all configurations and challengers have more parameters to optimize.

goal function	optimization algorithm	optimized parameters					best fitness quartiles			Success Rate
		n_c	n_r	μ	λ	m[%]	1Q	2Q	3Q	
Koza-2	$1+4$ CGP	150	–	–	–	5	0.0095	0.0099	0.0325	0.65
	$1+\lambda$ CGP	150	3	–	128	2	0.0091	0.0098	0.0364	0.68
	$\mu+\lambda$ **CGP**	**150**	**3**	**18**	**2048**	**10**	**0.0085**	**0.0099**	**0.0140**	**0.65**
Koza-3	$1+4$ CGP	150	–	–	–	7	0.0104	0.0325	0.0328	0.21
	$1+\lambda$ CGP	120	10	–	16	2	0.0087	0.0099	0.0325	0.49
	$\mu+\lambda$ **CGP**	**80**	**20**	**14**	**4096**	**5**	**0.0091**	**0.0100**	**0.0327**	**0.53**
Nguyen-4	$1+4$ CGP	120	–	–	–	10	0.0120	0.0324	0.0487	0.21
	$1+\lambda$ CGP	40	8	–	64	15	0.0129	0.022	0.0395	0.06
	$\mu+\lambda$ CGP	**60**	**6**	**18**	**2048**	**10**	**0.0101**	**0.0283**	**0.0498**	**0.24**
Nguyen-5	$1+4$ CGP	60	–	–	–	7	0.0090	0.0100	0.0240	0.50
	$1+\lambda$ CGP	150	10	–	16	2	0.0099	0.0099	0.0229	0.50
	$\mu+\lambda$ **CGP**	**150**	**20**	**22**	**4096**	**1**	**0.0085**	**0.0096**	**0.0100**	**0.77**
Nguyen-6	$1+4$ CGP	100	–	–	–	2	0.0270	0.0382	0.0392	0.17
	$1+\lambda$ **CGP**	**60**	**20**	–	**8**	**1**	**0.0091**	**0.0191**	**0.0381**	**0.44**
	$\mu+\lambda$ CGP	80	14	–	4096	5	0.0100	0.0381	0.0407	0.25
Nguyen-7	$1+4$ CGP	200	–	–	–	7	0.0157	0.0262	0.0534	0.18
	$1+\lambda$ **CGP**	**120**	**8**	–	**4096**	**7**	**0.0099**	**0.01866**	**0.0382**	**0.25**
	$\mu+\lambda$ CGP	150	6	2	32	2	0.0116	0.0216	0.0288	0.20
Nguyen-8	$1+4$ CGP	150	–	–	–	15	0.0084	0.0111	0.0415	0.53
	$1+\lambda$ CGP	80	10	–	16	2	0.0072	0.0084	0.0098	0.85
	$\mu+\lambda$ **CGP**	**150**	**6**	**2**	**32**	**2**	**0.0072**	**0.0088**	**0.0095**	**0.98**
Nguyen-9	$1+4$ **CGP**	**150**	–	–	–	**15**	**0.2475**	**0.4184**	**1.2077**	**0.00**
	$1+\lambda$ CGP	200	4	–	16	7	0.2707	0.6189	1.0801	0.01
	$\mu+\lambda$ CGP	120	20	22	4096	15	0.5325	0.7245	1.0079	0.00
Nguyen-10	$1+4$ CGP	60	–	–	–	20	0.5728	0.9185	1.1150	0.01
	$1+\lambda$ CGP	120	10	–	4096	20	0.3718	0.5727	0.7346	0.01
	$\mu+\lambda$ **CGP**	**150**	**20**	**8**	**4096**	**15**	**0.2975**	**0.4020**	**0.5921**	**0.00**
Keijzer-4	$1+4$ CGP	22	–	–	–	5	3.6828	3.6828	3.6828	0.00
	$1+\lambda$ CGP	200	20	–	16	7	2.1038	2.3413	2.4953	0.00
	$\mu+\lambda$ **CGP**	**120**	**20**	**22**	**1024**	**10**	**2.0837**	**2.2254**	**2.3484**	**0.00**
Keijzer-6	$1+4$ CGP	100	–	–	–	2	0.3229	0.4883	0.6438	0.00
	$1+\lambda$ CGP	60	20	–	64	10	0.1538	0.2184	0.3445	0.00
	$\mu+\lambda$ **CGP**	**200**	**20**	**6**	**256**		**0.0516**	**0.1008**	**0.2390**	**0.07**
Pagie-1	$1+4$ CGP	150	–	–	–	20	31.5965	34.0846	35.2309	0.00
	$1+\lambda$ CGP	200	20	–	512	10	14.9535	21.4781	30.7461	0.00
	$\mu+\lambda$ **CGP**	**200**	**20**	**14**	**256**	**15**	**14.7931**	**21.3225**	**30.1226**	**0.00**

evaluation complexity can vary greatly, as the number of active genes that are processed by the fitness evaluation procedure can be different.

The second observation is that when tuning for λ or for λ and μ, small values are identified by iRace as beneficial. With this, HC and its close derivatives seem to work better for CGP when optimizing Boolean circuits.

In related work it was shown that the efficiency of (1+4) CGP on single-line CGP increases with rising n_c. This can be observed also in Tab. 2. However, the efficiency of CGP can be improved using rectangular grids and slightly different (μ+λ) CGP schemes as well as SA. This is our third observation for the evolution of Boolean functions.

Evolution of Symbolic Regression Functions: The first observation of Tab. 3 is that the regular (1+4) CGP can be outperformed always regarding approximation accuracy except for the Nguyen-8 benchmark. The second observation is that $(\mu + \lambda)$ CGP is very successful. Except for three benchmarks it is constantly better than all the other algorithms. For the symbolic regression we cannot observe increased efficiency for single-line CGP when increasing n_c. However, and this is our next observation, the number of offspring individuals is usually very large. This is similar to regular GP, where often large populations

are used. Unlike to GP, the mutation operator in CGP is working on single individuals. CGP mutation and GP recombination are operators with very similar mechanisms and effects. It is an open question we want to investigate in future work: Assuming the intuition of the inner principle of GP is correct, i.e. parts of the goal solutions are randomly sampled initially and distributed among individuals of a large population, then the goal of GP is put this puzzle together correctly; Could GP also be solved effectively by a single-individual recombination (similar to CGP's mutation) and with smaller population sizes?

The last two findings in in Tab. 3 are: Similar to Boolean functions, rectangular CGP geometries are more efficient than single-line CGP and successful mutation rates are rather high, which is in contrast to prior findings suggesting to set the mutation rate as low as possible.

4 Conclusion and Future work

In this paper, we proposed an empirical study investigating if the regular way CGP is parameterized and optimized in related work is good. The results are that, indeed, the single-line CGP with an (1+4) CGP scheme is good for Boolean benchmarks but that much better results can be achieved for Boolean and symbolic regression functions when using rectangular CGP grids and differently parameterized $(\mu + \lambda)$ CGP schemes as well as SA. Furthermore, we could observe that similar to GP, CGP greatly benefits from large exploration rates, i.e. large offspring populations and high mutation rates, when evolving symbolic regression functions. This behavior is surprising and requires further investigation. It is especially interesting, if the former results on inner CGP mechanisms, like "neutrality", are still valid.

Following recommendations can be drawn from our experiments.

- For simple Boolean functions (1+1) HC applied on CGP with 30 to 50 rows and 100 to 200 columns performs best.
- For complex Boolean functions SA applied on CGP with 3 to 10 rows and 30 to 300 columns performs best. Increasing the number of rows to 100 might help in case of heavy functions, such as the multiplication.
- For Boolean functions the best observed mutation rate interval is $[0.1, 1.6]\%$.
- For continuous functions CGP with 3 to 20 rows and 80 to 200 columns performs best.
- For continuous functions CGP with $\mu = 2 \dots 22$ and $\lambda = 2048 \dots 4096$ performs best. It is worth investigating $\lambda = 8 \dots 32$ in cases where large λ values do not result in fast convergence.
- For continuous functions the mutation rate may vary from 1% to 15% with higher mutation rates being more successful for larger genotypes.

We will extend the benchmark set in our future work to more popular functions, like classification and image-processing tasks, and approach the questions regarding similarity of inner mechanisms to GP. Additionally we will try understand properly the ambivalent nature of CGP making it successful for combinatorial and continuous benchmarks.

References

1. T. Kalganova and J. F. Miller. Evolutionary Approach to Design Multiple-valued Combinational Circuits. In *Proc. Intl. Conf. Applications of Computer Systems (ACS)*, 1997.
2. P. Kaufmann and M. Platzner. Advanced techniques for the creation and propagation of modules in cartesian genetic programming. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1219–1226, Atlanta, GA, USA, 12-16 July 2008. ACM.
3. M. Lones. *Enzyme Genetic Programming*. PhD thesis, University of York, 2003.
4. M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43 – 58, 2016.
5. J. F. Miller. An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach. volume 2, pages 1135–1142. Morgan Kaufmann, 1999.
6. J. F. Miller, P. Thomson, and T. Fogarty. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study, 1997.
7. A. J. Turner and J. F. Miller. The Importance of Topology Evolution in NeuroEvolution: A Case Study Using Cartesian Genetic Programming of Artificial Neural Networks. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI)*, pages 213–226. Springer International Publishing, Cham, 2013.
8. J. A. Walker and J. F. Miller. Evolution and acquisition of modules in cartesian genetic programming. In *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 187–197, Coimbra, Portugal, 5-7 Apr. 2004. Springer-Verlag.