A Highly Efficient Function Optimization with Genetic Programming

Joao C. F. Pujol[1] and Riccardo Poli[2]

[1] Centro de Desenvolvimento da Tecnologia Nuclear,
Rua Prof. Mario Werneck s/n,
30123 970, Belo Horizonte, Brazil
pujol@cdtn.br
[2] University of Essex, Wivenhoe Park, CO4 3SQ, Colchester, UK
rpoli@essex.ac.uk

This paper describes a new approach for function optimization that uses a novel representation for the parameters to be optimized. By using genetic programming using, the new method evolves functions that transform initial random values for the parameters into optimal ones. Moreover, the new approach addresses the scalability problem by using a representation that, in principle, is independent of the size of the problem being addressed. Promising results are reported, comparing the new method with differential evolution and particle swarm optimization on a test suite of benchmark problems.

1 Introduction

Parameter Optimization (*POPT*) problems present themselves in a variety of important domains ranging from engineering to *AI*, from mathematics to biological sciences, in areas such as function optimization, system identification, *AI* search, control, machine learning, design and many others. In order to solve *POPT* problems using computers, one has to specify a *representation* for the parameters of a system and then an *algorithm* that will perform the optimization of these parameters.

A lot of research has gone into search, learning and optimization algorithms, but by no means it can be claimed that a clear understanding of the space of all such algorithms has been achieved. Also, insufficient research has been devoted to the question of choosing a good parameter representation for a system. Indeed, most of the methods of parameter optimization proposed in the literature use simple arrays of numbers as a representation for the system's parameters. Overall performance differences are then the result of algorithms' differences/improvements only. This is not satisfactory because major speedup is often attainable through a change in representation of the solutions rather than in the search algorithm.

In the area of evolutionary computation there has been some interest in different ways of representing parameters. In most cases the research focus has been the so-called genotype/phenotype mapping [1, 2, 9], but rarely has this research considered real-valued mappings.

An issue that also plagues optimization methods is the so called scalability problem. As the number of parameters to be optimized increases, the size of the search space increases exponentially, making the problem intractable. To avoid the exponential growth of the search space, one can impose constraints on the parameters. However this is not always possible nor desirable.

In this work a new approach for parameter optimization based on Genetic Programming (*GP*) [4, 5] is described. The new method introduces a new parameter representation that is independent of the number of parameters to be optimized. In Section 2, the new approach is described. In Section 3, results of experiments are reported. Section 4 concludes with suggestions for further research.

2 Parameter Mapping Approach(*PMA*)

The basic idea behind *PMA* is to use computer programs to represent both the algorithms and the parameters to be optimized at the same time, and to optimize such programs using *GP*. In *PMA* a population of *GP* programs represents a set of trial mapping functions which transform initial sets of parameter values into adapted ones. Under the guidance of an appropriate performance measure, *GP* iteratively improves such functions, until a mapping function is evolved which can satisfactorily transform a given initial set of parameter values into a good set of values. The method is illustrated in Figure 1. It must be pointed out that the initial values assigned to each parameter do not change in the course of the evolutionary process. They are random constants initialized once and for all.

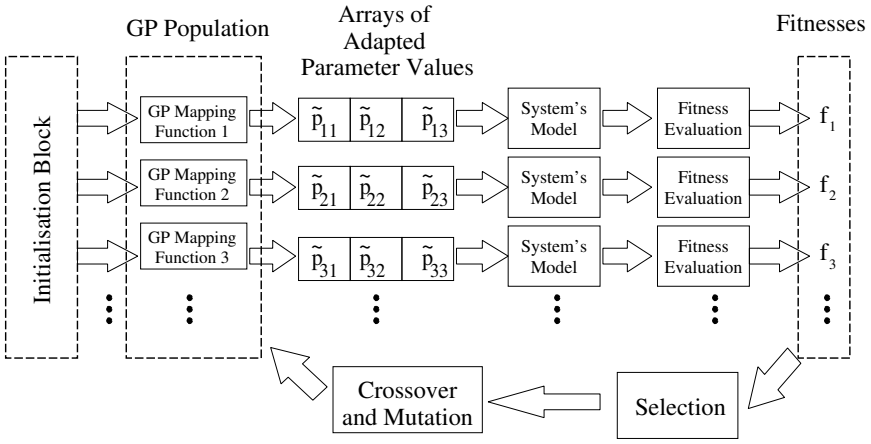

Fig. 1. *PMA* applied to a system with three parameters.

It is interesting to note that, as the same scalar *GP* function is used to map each parameter, the result of the evolutionary process performed in *PMA* can

be interpreted as a non-iterative *POPT* algorithm that, given the initial set of random parameters, is able to map it into the target set in one pass. A second interpretation is to consider the fact that in one way or another the scalar *GP* function is a representation for a whole set of parameter values since, given as input, each component of the initial set of random values produces as output a corresponding optimum value.

So, in the first interpretation the evolutionary process performed in *PMA* is searching the space of possible non-iterative *POPT* methods for a perfect direct parameter optimization algorithm. In the second interpretation, the evolutionary process performed in *PMA* is searching for a holistic representation for the target set of parameter values. Naturally, these two viewpoints are entirely equivalent.

In previous research [6, 8], an early version of *PMA* was applied to the training of neural networks (*NNs*). This was inspired by the observation that the process of training the weights of *NNs* is simply a (usually iterative) transformation from a set of initial (typically random) raw weights to a set of trained weights, and that that transformation does not have to be necessarily implemented in the iterative gradient-descent way of typical *NNs* learning rules. If the trained weights were known in advance, it would be a simple task to build a training set of pairs (initial-random-weight/trained-weight), and then fit a function to them by error minimization. However, as the target values are usually unknown, Genetic Programming was used to evolve a function by using the output error of the neural network as fitness function. The evolved function was then used to compute the trained weights by using the initial random weights as input. The experiments revealed that *PMA* was more effective than more traditional approaches.[1]

The original version of *PMA* has the following important features that presumably will also be shared by more general forms of *PMA*:

1. Within each individual the parameters to be optimized are not individually encoded: they are represented by a single function. Consequently, the representation does not necessarily grow quickly with the size of the problem of interest (which unavoidably would lead to the exponential growth of the search space and poor scalability).
2. The use of a parameter mapping function does not impose strong constraints on the value of the parameters and, consequently, *GP* can discover and use arbitrary or nearly arbitrary regularities in the parameter values appropriate for a problem.
3. The representation exploits this ability in conjunction with a good set of search operators to beneficially bias the search (with the result of speeding it up and making it scale up well).

[1] The authors were quite aware of the fact that *PMA* was not limited to the training of neural networks, and that it could be applied to other *POPT* problems. However, due to the lack of financial support, further research could not be carried out. As funding has now become available, the exploration of *PMA* was resumed.

3 Experiments

3.1 *GP* implementation

A steady state form of *GP* was implemented, with tournament selection and a *non-destructive headless chicken crossover* [10]. In this form of crossover, a standard subtree crossover is performed between a parent selected from the population and a randomly created individual. The resulting offspring is inserted in the population if it is superior to the parent, otherwise it is discarded.

The *GP* function set included four 4-arity operators: $SUM{=}x \times y{+}u \times v$, $SUB{=}x{\times}y{-}u{\times}v$, $MUL{=}(x{+}y){\times}(u{+}v)$ and *PDV(x+y,u+v)*, where x, y, u and v are the numerical values computed at the root nodes of subtrees. *PDV(num,den)* is the protected division, which returns *num* if *den* is zero. The reason for using 4-arity functions is that this breaks the symmetry of the addition and multipication arithmetic operators, thus reducing the possibility of the so-called *permutation* or *competing convention* problem [16, 15]. The ideia of using these operators was also inspired by neural networks, as for the first two operators, the values of nodes y and u may be seen as *weights* of nodes x and y (or vice-versa).

In the current version of *PMA*, a sparse encoding for the initial untrained parameter values was used, where each parameter is actually represented by a specified number of random constants. When executing a *GP* tree to map the untrained parameter into an optimized one, the random numbers representing that parameter are fed into a set of variables used as inputs (terminals) to the tree. The values for these variables are changed when a different parameter is processed. Note that the number of these variables has nothing to do with the number of parameter being optimized. Depending on the problem, in order to improve performance, one can use a single variable, 10, 100 or even more, to represent each parameter. The terminal set also included some random constants that were initialized in generation zero.

3.2 Test functions

To explore the potential of the new method, it was applied to minimize a set of benchmark functions, the well-known De Jong set. This set of functions was chosen because it has been extensively used as a test suite for optimization algorithms. In addition, it includes functions of varying degrees of difficulty, so that one can compare the performance of different optmization methods. Two other functions taken from [14] were also included. The whole test suite is as follows:

1)First De Jong function. $F1(\mathbf{x}) = \sum_{j=1}^{3} x_j^2$, where $x_j \in [-5.12, 5.12]$. Minimum is $F1(0) = 0$.

2) Second De Jong function. $F2(\mathbf{x}) = 100 \times (x_1^2 - x_2)^2 + (1 - x_1)^2$, where $x_j \in [-2.48, 2.48]$. Minimum is $F2(\mathbf{1}) = 0$.

3) Third De Jong function. $F3(\mathbf{x}) = 30 + \sum_{j=1}^{5} \lfloor x_j \rfloor$, where $x_j \in [-5.12, 5.12]$.

Minimum is $F3(-5.12 \leq x_j < -5) = 0$.

4) Fourth De Jong function modified according to [14]. $F4(\mathbf{x}) = \sum_{j=1}^{30}(x_j^4 \times j + \nu)$,where $x_j \in [-1.28, 1.28]$. Minimum is $F4(\mathbf{0}) \leq 30 \times \overline{\nu} = 15$, where $\nu \in [0, 1)$ is a random variable with a uniform distribution, and $\overline{\nu} = 0.5$ is its expected value.

5) Fifth De Jong function.

$$F5(\mathbf{x}) = \frac{1}{0.002 + \sum_{i=1}^{25} \frac{1}{i+(x_1-a_{i1})^6+(x_2-a_{i2})^6}},$$

where $x_j \in [-65.536, 65.536]$
$a_{i1} = [-32, -16, 0, 16, 32, -32, -16, \ldots, 0, 16, 32]$
$a_{i2} = [-32, -32, -32, -32, -32, -16, -16, \ldots,$
$32, 32, 32]$
Minimum is $F5(-32, -32) \approx 1$.

6) Corona's parabola [14]. $F6(\mathbf{x}) = \sum_{j=1}^{4} f_j$,
where $f_j = 0.15 \times (z_j - 0.05 \times sign(z_j))^2 \times d_j \ , \ if \quad \|x_j - z_j\| < 0.05$
$otherwise f_j = d_j \times x_j^2 \, ; \, x_j \in [-1000, 1000]$
with $z_j = \left\lfloor \left\| \frac{x_j}{0.2} \right\| + 0.49999 \right\rfloor \times sign(x_j) \times 0.2$ and $d_j = \{1, 1000, 10, 100\}$
Minimum is $F6(\mathbf{x})$=0, with $\|x_j - z_j\| < 0.05$.

7) Griewangk's function [14]. $F7(\mathbf{x}) = \sum_{j=1}^{10} \frac{x_j^2}{4000} - \prod_{j=1}^{10} \cos(\frac{x_j}{\sqrt{j+1}}) + 1$,

where $x_j \in [-400, 400]$. Minimum is F7($\mathbf{0}$)=0.

The performance of the *PMA* approach was tested on these functions, and the results were compared to those obtained with the two other methods: Differential Evolution (*DE*) and Particle Swarm Optimization (*PSO*), as described in [12, 13]. These two algorithms were chosen because they have been reported to be effectively the state of the art in function optimization.

The size of the population for the *PMA* runs was 100 individuals, except for the third test function, where the population was 50 individuals. The tournament size to select the parent was equal to the size of the population. The terminal set included 100 variables to encode the initial untrained values of the parameter being optimized (see Section 3.1). The variables and the random constants of the terminal set were randomly initialized in the range (-1,+1). The *GP* trees were initialized by using a *grow* strategy [4], where each node had a 50% chance of becoming an internal or a leaf node. In each experiment, the value of the parameters controlling the performance of each method was selected to obtain the best performance.

The results of these experiments are discussed in the next section.

3.3 Experimental results

To evaluate the performance of the three methods, 100 independent runs were carried out. The 100 runs were divided into 10 batches of 10 runs each. For each batch, average values for the number of successful runs (HITS) and for the number of fitness evaluations (NFE) to achieve that result were obtained. The results for the 10 batches were then used to compute standard deviations.

First set of experiments In the first set of experiments, when the value of a parameter generated exceeded the boundaries defined for the function being optimized, the closest boundary value was assigned to the parameter. The results of this set of experiments are summarized in Table 1, 2 and 3.

In terms of number of fitness evaluations to find a solution, *PMA* outperformed *DE* in 5 out of 7 of the test set functions, and was superior to *PSO* in 6 of them. The three methods converged to a solution in almost all runs. However, for functions *F6* and *F7*, *PSO* did not converge to any solution at all, and for function *F4*, *DE* had a poor convergence rate.

Although outstanding, these results should not lead one to jump to conclusions. For most of the functions optimized, the minimum values are either *0* or *1*, and these values are extremely easy to generate by using the *PMA* representation. For example, a *GP* tree encoding the algebraic expression *x-x*, will always produce *0* as a result. Similarly, the algebraic expression *x/x* will always produce *1*, provided that x is greater than zero. Although this is a characteristic of the parameter mapping approach, it might be considered an unfair advantage (for other problems this might be a hindrance). Therefore, to eliminate this advantage, modifications were introduced in the original benchmark functions, and a second set of experiments was carried out.

Second set of experiments In the second set of experiments, the degree of difficulty was increased by introducing a random shift in each dimension of the function $F(x)$ being optimized. A different random shift was applied to each dimension. So that the functions are now $F(x_1 - r, \ldots, x_i - r, \ldots, x_N - r)$, where r is a random constant in the range (-1,1), generated by using a uniform distribution. As a consequence, the optimal values are not *0s* and *1s* anymore, but are shifted by the amount provided by the random constant. The boundaries constraining the search were also shifted by -1 or 1, depending on whether the random shift was negative or positive. This way, the boundaries were shifted to guarantee that a solution could be found, without actually giving information about the value of the random shifts. The results of this set of experiments are summarized in Table 4, 5 and 6.

As expected, things were harder for *PMA* this time. In terms of fitness evaluations it was outperformed by *DE* in the optimization of 4 out of 7 of the test set functions. Moreover, it was also outperformed by *PSO* in two cases.

Although slower in some cases, *PMA* seems to be more robust, since convergence was still achieved in 100% of the cases, irrespective of the problem, while

for *DE* and *PSO*, there were problems where the performance was poor or they simply failed to converge to any solution at all.

Third set of experiments For problems where the parameters to be optimized have completely different ranges, it might be useful to use a scaling procedure. To investigate whether this might bring any benefit to the optimization of the current test set, the hyperbolic tangent was used to squash the output of the *GP* tree to the range (-1,+1). The results were then rescaled to the appropriate parameter range by using a linear transformation. This transformation was not beneficial to neither *DE* nor *PSO*, and so these algorithms were not modified. The benchmark functions were shifted as in the second set of experiments. The results for *PMA* are summarized in Table 7.

By comparing Tables 6 and 7, one can see that the results for function *F3* and *F5* improved, whereas things got worse for function *F7*. For the other functions the results were basically unchanged. So, by considering the *PMA* results only, it is up to the user to apply a scaling procedure to the output of the *GP* tree or not. However, *PMA* now beat *DE* in 4 out of 7 cases and was superior to *PSO* in all test functions in terms of number of fitness evaluations.

4 Conclusion and further research

In this paper, a new approach for parameter optimization has been presented. The results of the experiments have shown that the new paradigm can be more robust and more efficient than the best function optimization methods. It should be pointed out that the ideas presented here should be further explored. For example:

1. The *GP* function set may include functions that affect particular parameters but not others. This may help the evolutionary process to reduce or increase the interaction between parameters, as necessary.
2. Context information can be provided to the mapping function so that the value of the updated parameter can be made sensitive to context information, such as the value of neighboring or otherwise related parameters. In conventional learning methods a similar strategy is beneficially used. For example, for training neural networks, where the updating of a particular weight depends on the value of the others.
3. Although for the problems addressed in this paper the *GP* function set used has produced good results, for other classes of problems it might be interesting to include functions of different arities. The evolutionary process will then evolve the appropriate combination of them.

In any case, the parameter mapping approach need to be better understood and positioned in terms of other *POPT* algorithms. This can be achieved through comparative experimentation involving a range of optimization algorithms applied to a variety of interesting test problems, exhibiting various levels of complexity.

Table 1. Results of the application of differential evolution to the optimization of the benchmarck functions. *SF* is the scaling factor, and *CP* is the crossover probability.

F_i	DE			
i	SF	CP	NFE(σ)	HITS(σ)
1	0.5	0.3	937(23)	10(0)
2	0.9	0.5	961(64)	10(0)
3	0.8	0.3	157(13)	10(0)
4	0.2	0.2	892,622(93,531)	2.2(1.8)
5	0.9	0.3	2,097(121)	10(0)
6	0.4	0.2	2,413(39)	10(0)
7	0.2	0.2	62,875(2,451)	10(0)

Table 2. Results of the application of particle swarm optimization to the benchmarck functions. *MS* is the maximum speed. σ is the standard deviation.

F_i	PSO		
i	MS	NFE(σ)	HITS(σ)
1	0.01	4,378(251)	10(0)
2	0.01	6,193(604)	10(0)
3	10	49(2.9)	10(0)
4	0.01	99,291 (2,128)	0.1(0.3)
5	1.0	2,638(224)	10(0)
6	0.01-5	2,000,000(0)	no hit
7	0.01-5	2,000,000(0)	no hit

Table 3. Results of the application of the parameter mapping approach to the optimization of the benchmarck functions. *FC* is the fraction of terminals initialized as random constants. σ is the standard deviation.

F_i	PMA		
i	FC	NFE(σ)	HITS(σ)
1	0.5	360(88)	10(0)
2	0.9	2,070(729)	10(0)
3	0.9	29(9)	10(0)
4	0.5	17,775(5,847)	10(0)
5	0.9	20,763(5,530)	10(0)
6	0.5	972(254)	10(0)
7	0.5	541(159)	10(0)

Table 4. Results of the application of differential evolution to the optimization of the benchmarck functions randomly shifted. *SF* is the scaling factor, and *CP* is the crossover probability.

F_i	DE			
i	SF	CP	NFE(σ)	HITS(σ)
1	0.5	0.3	500(29)	10(0)
2	0.9	0.5	902(44)	10(0)
3	0.8	0.3	142(11)	10(0)
4	0.2	0.2	902,137(69,548)	1.9(0.8)
5	0.9	0.3	1,309(61)	10(0)
6	0.4	0.2	3,613(42)	10(0)
7	0.2	0.2	63,153(1,930)	10(0)

Table 5. Results of the application of particle swarm optimization to the benchmarck functions randomly shifted. *MS* is the maximum speed. σ is the standard deviation.

F_i	PSO		
i	MS	NFE(σ)	HITS(σ)
1	0.01	4,083(339)	10(0)
2	0.01	5,828(556)	10(0)
3	10	49(3.4)	10(0)
4	0.01	95,186(3,349)	1.1(0.5)
5	1.0	2,7333(230)	10(0)
6	0.01-5	2,000,000(0)	no hit
7	0.01-5	2,000,000(0)	no hit

Table 6. Results of the application of the parameter mapping approach to the optimization of the benchmarck functions randomly shifted. *FC* is the fraction of terminals initialized as random constants. σ is the standard deviation.

F_i	PMA		
i	FC	NFE(σ)	HITS(σ)
1	0.5	1,685(167)	10(0)
2	0.9	5,798(1,552)	10(0)
3	0.9	85(44)	10(0)
4	0.5	18,068(3,874)	10(0)
5	0.9	53,078(14,929)	10(0)
6	0.5	10,528(4,406)	10(0)
7	0.5	11,896(1596)	10(0)

Table 7. Results of the application of the parameter mapping approach to the optimization of the benchmarck functions randomly shifted, by using a scaling procedure. *FC* is the fraction of terminals initialized as random constants. σ is the standard deviation.

F_i	PMA		
i	FC	NFE(σ)	HITS(σ)
1	0.5	1,656(245)	10(0)
2	0.9	5,319(1,418)	10(0)
3	0.9	28(7)	10(0)
4	0.5	20,012(4,507)	10(0)
5	0.9	1,092(535)	10(0)
6	0.5	12,778(2,071)	10(0)
7	0.5	20,887(7,858)	10(0)

References

1. L. Altenberg. *Genome growth and the evolution of the genotype-phenotype map.* In W. Banzhaf and F. H. Eeckman, editors, *Evolution as a Computational Process*, pages 205–259. Springer-Verlag, Berlin, Germany,1995.
2. W. Banzhaf. *Genotype-phenotype-mapping and neutral variation – A case study in genetic programming.* In Y. Davidor *et al.*, editors, *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 322–332, Jerusalem, 9-14 Oct. 1994. Springer-Verlag.
3. D. Fogel and A. Ghozeil. *A note on representations and variation operators. IEEE Trans. on Evolutionary Computation*, 1(2):159–161, 1997.
4. J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.
5. W. B. Langdon and R. Poli. *Foundations of Genetic Programming.* Springer, 2002.
6. J. Pujol. *Evolution of Artificial Neural Networks Using a Two-dimensional Representation. PhD thesis, School of Computer Science, University of Birmingham, UK, Apr. 1999.* (Available from `http://www.cdtn.br/~pujol/tese19.ps`)
7. J. Pujol and R. Poli. *Evolution of neural networks using a two-dimensional aproach.* In L. C. Jain, editor, *Evolution of Engineering and Information Systems and Their Applications*, CSC Press international series on computational intelligence. CRC Press, Boca Raton, Florida, USA, 1999.
8. J. Pujol and R. Poli. *Evolution of neural networks using weight mapping.* In W. Banzhaf *et al.*, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1170–1177, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann. (Available from `http://cswww.essex.ac.uk/staff/poli/papers/Pujol-GECCO1999.pdf`)
9. M. Shackleton, R. Shipman, and M. Ebner. *An investigation of redundant genotype-phenotype mappings and their role in evolutionary search.* In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 493–500, La Jolla, California, USA, 6-9 July 2000. IEEE Press.

10. P. J. Angeline. *Subtree crossover: building block engine or macromutation?* In *Genetic Programming 1997: Proceedings of the Second Annual Conference*,pages 9-17, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

11. D. Wolpert and W. Macready. *No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.

12. J. Kennedy and R. Eberhart. *The Particle Swarm: Social Adaptation in Information-Processing Systems.* In D. Corne *et. al*, editors. *New Ideas in Optimization*, pages 379-387, McGraw-Hill Publishing Company,1999.

13. K. Price. *An Introduction to Differential Evolution.* In D. Corne *et. al*, editors. *New Ideas in Optimization*, pages 379-387, McGraw-Hill Publishing Company,1999.

14. R. Storn and K. Price. *Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces.* TR-95-012, International Computer Science Institute, Berkeley, USA,1995.

15. P. Hancock. *Genetic Algorithms and Permutation Problems: a Comparison of Recombination Operators for Neural Structure Specification. Proceedings of COGANN Workshop, IJCNN*, 1992. IEEE Computer Society Press.

16. N. Radcliffe. *Genetic set recombination and its application to neural networks topology optimization. Technical Report EPCC-TR-91-21*, University of Edinburgh, Scotland,1991.