Optimization via Parameter Mapping with Genetic Programming

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

[1] CDTN, 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

Abstract. This paper describes a new approach for parameter optimization that uses a novel representation for the parameters to be optimized. By using genetic programming, the new method evolves functions that transform initial random values for the parameters into optimal ones. This new representation allows the incorporation of knowledge about the problem being solved to improve the search. 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. Moreover, the new algorithm allows the incorporation of information about the problem being dealt with, in order to improve the search. 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.

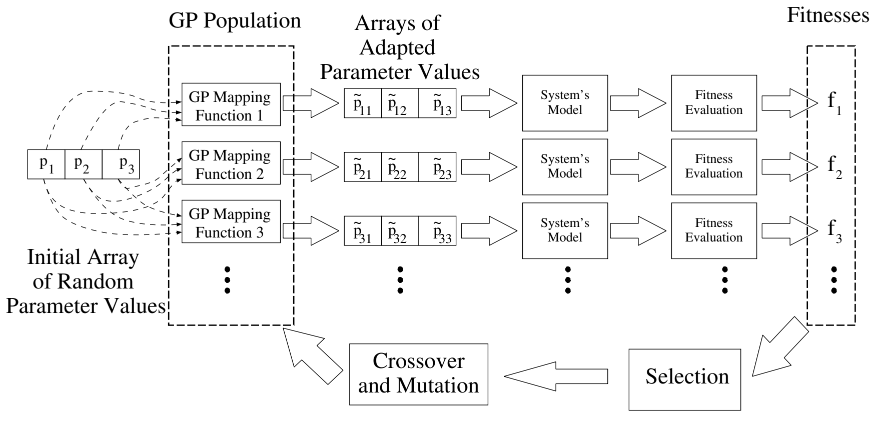
GP Population
Arrays of Adapted Parameter Values
Fitnesses
GP Mapping Function 1
GP Mapping Function 2
GP Mapping Function 3
p_1 p_2 p_3
$\tilde{p}_{11}$ $\tilde{p}_{12}$ $\tilde{p}_{13}$
$\tilde{p}_{21}$ $\tilde{p}_{22}$ $\tilde{p}_{23}$
$\tilde{p}_{31}$ $\tilde{p}_{32}$ $\tilde{p}_{33}$
System's Model
Fitness Evaluation
f_1
f_2
f_3
Initial Array of Random Parameter Values
Selection
Crossover and Mutation

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 pairSs (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 system.
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.

4. As will be shown in the experiments, *PMA* allows the incorporation of information about the problem being addressed in order to improve the search.

3 Experiments

3.1 *GP* implementation

To carry out the experiments a steady state form of *GP* was used, with standard subtree crossover and tournament selection (tournament size=2). Mutation was implemented as a replacement of nodes by random nodes of the same arity. An elitist strategy was also implemented, by performing crossover with the best individual of the population. In all experiments, 50% of the offspring were created by crossover. The other 50% were generated by mutation. The offspring is inserted in the population, if it is superior to at least one of the parents. Replacement is carried out by negative tournament.

The *GP* function set included the arithmetic operators of *addition*, *subtraction*, *multiplication* and *protected division* (which returns the numerator if the denominator is zero).

The terminal set included a variable representing the parameter to be optimized. In some of the experiments (identified in the text) the terminal set was extended to include other variables, to convey information about the problem being tackled, and also random constants.

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 [13] 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 [13]. $F4(\mathbf{x}) = \sum_{j=1}^{30} (x_j^4 \times j + \nu)$,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 $x_j \in [-65.536, 65.536]$
$a_{i1} = [-32, -16, 0, 16, 32, -32, -16, \dots, 0, 16, 32]$
$a_{i2} = [-32, -32, -32, -32, -32, -16, -16, \dots, 32, 32, 32]$

Minimum is $F5(-32, -32) \approx 1$.

6) Corona's parabola [13]. $F6(\mathbf{x}) = \sum_{j=1}^{4} f_j$,

where $f_j = 0.15 \times (z_j - 0.05 \times sign(z_j))^2 \times d_j\ ,\ if\ \ \|x_j - z_j\| < 0.05$
$otherwise f_j = d_j \times x_j^2$; $x_j \in [-1000, 1000]$
with $z_j = \lfloor \| \frac{x_j}{0.2} \| + 0.49999 \rfloor \times sign(x_j) \times 0.2$ and $d_j = \{1, 1000, 10, 100\}$

Minimum is *F6*($\mathbf{x}$)=0, with $\|x_j - z_j\| < 0.05$.

7) Griewangk's function [13]. $F7(\mathbf{x}) = \sum_{j=1}^{10} \frac{x_j^2}{4000} - \prod_{j=1}^{10} \cos(\frac{x_j}{\sqrt{j+1}}) + 1$,

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

The performance of the *PMA* approach was tested on these function, and the results were compared to two other methods: Differential Evolution (*DE*) and Particle Swarm Optimization (*PSO*), as described in [11, 12].

In all experiments, when the value of a parameter generated by *GP* or the other methods, exceeded the boundaries defined for the function being optimized, the closest boundary value was assigned to the parameter.

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 optimizations and for the number of fitness evaluations to achieve that result were obtained. The results for the 10 batches then were used to compute standard deviations.

First set of experiments In the first set of experiments, only a single variable representing the parameter to be optimized was used in the *GP* terminal set. The results of this set of experiments are summarized in Table 1a.

In terms of number of fitness evaluations (*NFE*) to find a solution, , with the exception of functions *F3* and *F5*, *PMA* considerably outperformed both *DE* and *PSO*. 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. This effect becomes clear on the optimization of functions *F3* and *F5*, where the minimum is neither *0* nor *1*, *PMA* was outperformed by both *DE* and *PSO*. Actually, *PMA* behaved surprisingly well for these two functions, since no constants (random or otherwise) were included in the *GP* terminal set to help build the solutions.

The three methods converged to a solution in almost all runs. However, their performance varied considerably for some of the functions. As expected, for function *F5*, *PMA* performed poorly as compared to the other two methods. On the other hand, for function *F4*, *PMA* was successful in 63% of the cases, as compared to 22% and 1% yielded by *DE* and *PSO*, respectively. Moreover, for functions *F6* and *F7*, *PSO* did not converge to any solution at all. These results seem to indicate that *PMA* is more robust than *DE* and *PSO*.

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. The same 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 the same amount.

For this set of experiments the *GP* terminal set was expanded to include three other variables: one to represent the random shift r, and two others to represent the boundaries. By using these additional variables, one is actually taking advantage of available information about the problem being addressed. Table 1b shows the results of this set of experiments.

In terms of number of fitness evaluations to find a solution, the situation was even better than in the first set of experiments, as *PMA* was only outperformed by the other two methods to find a solution for function *F5*. In terms of number of successful runs for this function, *PMA* was still outperformed by *DE* and *PSO*, but its rate of success increased to 73%. This gain in performance is a consequence of the use of available information about the problem to improve the search.

Third set of experiments In a third set of experiments, the situation was made even harder by introducing a different random shift in each dimension of the function $F(x)$ being optimized. Except for the variable representing the parameter to be optimized, no additional variables to convey information about the problem, were included in the terminal set this time. That means, the functions to be optimized were supposed to be black boxes. In this case, no information neither about the random shifts nor the boundaries is available and, consequently, was not given as input. Instead, random constants in the (-1,+1) were added to the *GP* terminal set to allow the evolutionary process to build appropriate numerical constants. Table 1c summarizes the results for this set of experiments.

As expected, things were much harder for *PMA* this time. In terms of fitness evaluations it was outperformed by *DE*, except on functions *F3* and *F4*. As compared to *PSO*, *PMA* was superior on functions *F4*, *F6* and *F7*. As for the number of successful runs, although *PMA* has not converged to a solution all the time, it has clearly outperformed *DE* and *PSO* on function *F4*. Once again, *PSO* failed to produce any solution to functions *F6* and *F7*.

Although slower, *PMA* proved to be quite robust, achieving the optimum in at least 40% of the runs, irrespective of the problem, while for *DE* and *PSO*, there were problems where the performance dropped very significatly. This is particularly true if one uses a constant setup for *DE* and *PSO*. Actually, for many settings *DE* and *PSO* simply failed to converge.

Table 1. (a) Results for the first set of experiments. (b) Results for the second set of experiments. (c) Results for the third set of experiments. *NFE* represents the average number of fitness evaluations. *HITS* is equal to the average number of successful runs. *SF* is the scaling factor, and *CP* is the crossover probability. *MS* is the maximum speed. σ is the standard deviation.

F_i	DE				PSO			PMA	
i	SF	CP	NFE(σ)	HITS(σ)	MS	NFE(σ)	HITS(σ)	NFE(σ)	HITS(σ)
1	0.5	0.3	937(23)	10(0)	0.01	4,378(251)	10(0)	10(3)	10(0)
2	0.9	0.5	961(64)	10(0)	0.01	6,193(604)	10(0)	24(5.7)	10(0)
3	0.8	0.3	157(13)	10(0)	10	49(2.9)	10(0)	238(66)	10(0)
4	0.2	0.2	892,622(93,531)	2.2(1.8)	0.01	99,291 (2,128)	0.1(0.3)	94,514(17,980)	6.3(1.2)
5	0.9	0.3	2,097(121)	10(0)	1.0	2,638(224)	10(0)	32,438(6,577)	1.4(0.8)
6	0.4	0.2	2,413(39)	10(0)	0.01-5	2,000,000(0)	no hit	12(4.2)	10(0)
7	0.2	0.2	62,875(2,451)	10(0)	0.01-5	2,000,000(0)	no hit	9(3.4)	10(0)

(a)

F_i	DE				PSO			PMA	
i	SF	CP	NFE(σ)	HITS(σ)	MS	NFE(σ)	HITS(σ)	NFE(σ)	HITS(σ)
1	0.5	0.3	481(10)	10(0)	0.01	4,335(217)	10(0)	44(8)	10(0)
2	0.9	0.5	1,021(64)	10(0)	0.01	6,175(586)	10(0)	469(158)	10(0)
3	0.8	0.3	160(12)	10(0)	10	50(3)	10(0)	20(7)	10(0)
4	0.2	0.2	473,027(20,807)	1.3(0.8)	0.01	97,153 (3,095)	0.8(0.8)	52,010(18,752)	8.1(1.0)
5	0.9	0.3	2,112(125)	10(0)	1.0	2,741(306)	10(0)	9,970(1,970)	7.3(1.3)
6	0.4	0.2	2,442(38)	10(0)	0.01-5	2,000,000(0)	no hit	42(9)	10(0)
7	0.2	0.2	32,947(1,366)	10(0)	0.01-5	2,000,000(0)	no hit	44(6)	10(0)

(b)

F_i	DE				PSO			PMA	
i	SF	CP	NFE(σ)	HITS(σ)	MS	NFE(σ)	HITS(σ)	NFE(σ)	HITS(σ)
1	0.5	0.3	500(29)	10(0)	0.01	4,083(339)	10(0)	60,757(10,306)	9.7(0.5)
2	0.9	0.5	902(44)	10(0)	0.01	5,828(556)	10(0)	50,794(7,110)	9.2(1)
3	0.8	0.3	142(11)	10(0)	10	49(3.4)	10(0)	110(34)	10(0)
4	0.2	0.2	902,137(69,548)	1.9(0.8)	0.01	95,186 (3,349)	1.1(0.5)	134,062(52,689)	8.6(1.2)
5	0.9	0.3	1,309(61)	10(0)	1.0	2,733(230)	10(0)	59,473(21,340)	8.6(1.2)
6	0.4	0.2	3,613(42)	10(0)	0.01-5	2,000,000(0)	no hit	281,183(41,306)	6(1.3)
7	0.2	0.2	63,153(1,930)	10(0)	0.01-5	2,000,000(0)	no hit	1,332,880(106,524)	4(1.9)

(c)

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 than other methods. It has been also shown how the new approach can take advantage of information about the problem being addressed, in order to improve performance. It should be pointed out, that the ideas presented here should be further explored. For example:

1. Instead of using random-valued variables as input to the GP trees, one can use integers that identify the parameters to be optimized. So that the GP tree would actually operate as an addressed-by-integers memory that returns the target values of the parameters.
2. 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.
3. 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.

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.

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 Marriott Hotel La Jolla, California, USA, 6-9 July 2000. IEEE Press.

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

11. 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.

12. 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.

13. 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.