Journal of Computational Interdisciplinary Sciences (2008) 1(1): 3-10

ISSN 1983-8409
http://epacis.org

A new multi-particle collision algorithm for optimization in a high performance environment

Eduardo Fávero Pacheco da Luz, José Carlos Becceneri and Haroldo Fraga de Campos Velho

Manuscript received on July 31, 2008 / accepted on October 5, 2008

ABSTRACT

A new meta-heuristics is introduced here: the Multi-Particle Collision Algorithm (M-PCA). The M-PCA is based on the implementation of a function optimization algorithm driven for a collision process of multiple particles. A parallel version for the M-PCA is also described. The complexity for PCA, M-PCA, and a parallel implementation for the MPCA is developed. The efficiency for optimization for PCA and M-PCA is evaluated for some test functions. The performance of the parallel implementation of the M-PCA is also presented. The results with M-PCA produced better optimized solutions for all test functions analyzed.

Keywords: Computational mathematics, computational complexity, high performance computing, meta-heuristics, optimization.

Associated Laboratory for Computing and Applied Mathematics, National Institute for Space Research, Av. Astronautas, 1.756, Jardim Granja, 12227-010 São José dos Campos, SP, Brazil. E-mails: [eduardo.luz, becce, haroldo]@lac.inpe.br

1 INTRODUCTION

Scientists have studied and developed new methods for resolving optimization problems [1]. Some of these methods are nature inspired techniques such as Simulated Annealing (SA) [13, 9], Genetic Algorithm (GA) [6, 5], Particle Swarm Optimization (PSO) [8] and Ant Colony System (ANS) [3] – as systems based on animal behavior, and Invasive Weed Optimization (IWO) [12] – from an emulation for the vegetal growing patterns. More recently, the canonical Particle Collision Algorithm (PCA) [17, 16] has been proposed.

The PCA implements the search for one single particle, responsible for a sequential search in the search space [17]. However, this kind of search can be improved by using many particles, dealing with in a collaborative way. The Multi-Particle Collision Algorithm (M-PCA) can better explore the search space, avoiding premature convergence to a local optimum. The M-PCA is straightforward for implementation in a parallel environment.

In the next section the PCA and M-PCA are described and the complexity for both approaches is discussed, and the algorithms are applied to several test functions. Section 3 presents the parallel implementation of the M-PCA, including the complexity for the parallel implementation, and performance results are shown.

The obtained results for the application of Multi-Particle Collision Algorithm (M-PCA) over some test functions reveals outstanding results, which together with the assurance of less computational efforts ensure the applicability of the proposed method.

2 DESCRIPTION OF THE PCA AND M-PCA SCHEMES

The theory of optimization is a branch of mathematical sciences that studies the methods for finding an optimum set for a given problem. The practical part of the theory is defined by the collection of techniques, methods, procedures and algorithms that can be used to find the optimum [1].

Optimization problems have the goal of finding the best set within a variable set to maximize or minimize a function, defined as an objective function or cost function. Optimization problems can be classified as: continuous optimization (where the variable has real or continuous values); discrete optimization (where the variable has integer or discrete values); and mixed optimization (with integer and continuous values at the same time) [2].

The best method for determining the function optimum strongly depends on the nature of the function in study. Two kinds of algorithms can be used: local optimization algorithms, that given a point in a function sub-domain are able to find the optimum point in this sub-domain (most of this algorithms are deterministic); and global optimization algorithms that seek the optimum point in the totality of the search space (those are stochastic algorithms, mostly metaheuristics).

Deterministic algorithms used in optimization can be divided into three main types:

- Zero order methods: based on the value of the objective function, e.g., Powell's conjugated directions;
- First order methods: based on the value of the objective function and its derivative regarding the project's variables, e.g., Steepest Descent;
- Second order methods: based on the value of the objective function, its derivative and the Hessian matrix, e.g., Quasi-Newton.

Stochastic methods used in optimization are those based on heuristics. Heuristics came from the Greek word *heuriskein*, meaning "to discover", and describing a method "based on the experience or judgment, that leads to a good solution of a problem, but not assuring to produce the optimum solution" [13]. Metaheuristics are those heuristics strongly based on natural processes.

The main metaheuristics are Simulated Annealing (SA), Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), among others [9].

2.1 Particle Collision Algorithm (PCA)

The Particle Collision Algorithm (PCA) was developed by Sacco and co-authors [17, 16, 18, 19], inspired by Simulated Annealing [9].

This algorithm was also greatly inspired by two physical behaviors, namely absorption and scattering that occurs inside a nuclear reactor. The use of the PCA was effective for several test functions and real applications [19].

The PCA starts by selecting an initial solution (Old_Config), that is modified by a stochastic perturbation (Perturbation()), leading to the construction of a new solution (New_Config). The new solution is compared (by the function Fitness()), and the new solution can or cannot be accepted. The main algorithm can be seen in Figure 1.

If the new solution is not accepted, the Metropolis scheme is used (the function Scattering()). The exploration on closer positions is guaranteed by using the functions Perturbation() and Small_Perturbation(). The latter algorithms are shown by Figure 2.

```
Generate an initial solution Old_Config
Best_Fitness = Fitness(Old_Config)
For n = 0 to # of iterations
 Perturbation()
 If Fitness(New_Config) > Fitness(Old_Config)
    If Fitness(New_Config) > Best_Fitness
       Best_Fitness = Fitness(New_Config)
    End-If
    Old_Config = New_Config
    Exploration()
 Else
    Scaterring()
 End-If
End-For

Exploration()
 For n = 0 to # of iterations
 Small_Perturbation()
 If Fitness(New_Config) > Fitness(Old_Config)
    If Fitness(New_Config) > Best_Fitness
       Best_Fitness = Fitness(New_Config)
    End-If
    Old_Config = New_Config
 End-If
Return

Scattering()
 p_scattering = 1 - (Fitness(New_Config) / Best_Fitness)
 If p_scattering > random(0,1)
    Old_Config = Random Solution
 Else
    Exploration()
 End-If
Return
```

Figure 1 – Main PCA pseudo-code.

```
Perturbation()
  For i = 0 to (Dimension-1)
    Upper = Superior_Limit[i]
    Lower = Inferior_Limit[i]
    Rand = Random(0,1)
    New_Config[i] = Old_Config[i] + ((Upper - Old_Config[i]) *
    Rand) - ((Old_Config[i] - Lower)*(1-Rand))
    If (New_Config[i] > Upper)
      New_Config[i] = Superior_Limit[i]
    Else
      If (New_Config[i] < Lower)
        New_Config[i] = Inferior_Limit[i]
      End-If
    End-If
  End-For
Return

Small_Perturbation()
  For i = 0 to (Dimension-1)
    Upper = Random(1.0, 1.2) * Old_Config[i]
    If (Upper > Superior Limit[i])
      Upper = Superior_Limit[i]
    End-If
    Lower = Random(0.8, 1.0) * Old_Config[i]
    If (Lower < Inferior_Limit[i])
      Lower = Inferior_Limit[i]
    End-If
    Rand = Random(0,1)
    New_Config[i] = Old_Config[i] + ((Upper - Old_Config[i]) *
    Rand) - ((Old_Config[i] - Lower)*(1-Rand))
  End-For
Return
```

Figure 2 – Codes for stochastic perturbation in PCA.

If a new solution is better than the previous one, this new solution is absorbed — absorption is one feature involved in the collision process. If a worse solution is found, the particle can be send to a different location of the search space, giving the algorithm the capability of escaping a local minimum, this procedure is inspired by the scattering scheme.

PCA is a robust metaheuristics, only one parameter is required from the user. This parameter is the number of iterations, and it also acts as a stop criteria for the algorithm. According to [18, 19] 10^5 iterations should be a good value for almost every application.

2.2 Complexity of PCA

Analyzing the main algorithm of PCA, presented at Figure 1, and adopting N as the number of iterations defined by the user, we can see that the first loop induces N checking operations over the objective function. At this main loop, there are calls to the Exploration() procedure, that by its turn executes another N operations through a loop inside the procedure. Therefore, $N \times N$ operations are executed by PCA, leading to a $O(N^2)$ complexity.

2.3 Multi-Particle Collision Algorithm (M-PCA)

The new Multi-Particle Algorithm (M-PCA) is based on the canonical PCA, but a new characteristic is introduced: the use of several particles, instead of only one particle to act over the search space.

Coordination between the particles was achieved through a blackboard strategy, where the Best_Fitness information is shared among all the particles in the process.

The pseudo-code for the M-PCA is presented by Figure 3, where the new loop, responsible for the control of the new particles is introduced.

Similar to the PCA, M-PCA also has only one parameter to be determined, the number of iterations. But in this case, the total number or iterations is divided by the number of particles which will be used in the process. The division of the task is the great distinction of M-PCA, which leads to a great reduction of required computing time.

The M-PCA was implemented using MPI libraries in a multiprocessor architecture with distributed memory machine.

```
Generate an initial solution Old_Config
Best_Fitness = Fitness(Old_Config)
Update Blackboard
For n=0 to # of particles
  For n=0 to # of iterations
    Update Blackboard
    Perturbation()
    If Fitness(New_Config) > Fitness(Old_Config)
      If Fitness(New_Config) > Best_Fitness
        Best_Fitness = Fitness(New_Config)
      End-If
      Old_Config = New_Config
      Exploration()
    Else
      Scattering()
    End-If
  End-For
End-For
```

Figure 3 – The new M-PCA and the loop for particle control.

2.4 Complexity of M-PCA

The code of M-PCA, presented in Figure 3, is very similar to PCA, so there are $N \times N$ checking operations in the inner loops, but due to the new loop, introduced by the multiple particle technique, that number of checking operations can be increased to a case where $N \times N \times N$ operations can occur, e.g., the number of particles is equal to the number of iterations.

So, the complexity associated to M-PCA is initially $O(N^3)$, but when we distribute the algorithm through the use of p processors, making sure that $p = n$, n as the number of particles in use, the complexity can return to $O(N^2)$, which is the same complexity of the canonical PCA.

3 HIGH PERFORMANCE ENVIRONMENTS

High performance environments are viable through the use of High Performance Computing (HPC) term that reflects the use of supercomputers or computer clusters to induce the parallel processing.

The process of computing can be defined as the execution of a sequence of instructions in a data set that can be improved by the use of a high performance environment. At this point, the computing sequence in a parallel environment consisted of one or more tasks executed in a concurrent way.

Computational systems that make a high performance environment viable are actually capable of providing teraflops, or 10^{12} floating point operations per second, to the user, enabling the solution of problems in high dimensions through new algorithms adapted to operate in this environment, as proposed in this work.

Flynn's taxonomy [4], is used to classify computers and programs capable of high speed operations in high performance environments, in the following way:

- Single Instruction – Single Data (SISD);
- Single Instruction – Multiple Data (SIMD);
- Multiple Instruction – Single Data (MISD);
- Multiple Instruction – Multiple Data (MIMD).

However, a new classification can be used, based on the analysis of hardware and software solutions adopted [20]. In this simpler scheme, only the main characteristics of each architecture are analyzed.

For the classification based on hardware, the following scheme is possible:

- Vectorial architecture: the same instruction is executed synchronously by every processor, but in distinct data sets;
- Multiprocessor architecture with shared memory: machines with shared memory, usually with a few number of processors that have access to a global memory area, where the communication between the processors occurs;
- Multiprocessor architecture with distributed memory: a set of processors and memory are connected through a communication network. The access to the remote data is done through message passing and the processors execute different sets of instructions in different moments. The synchronism is obtained only through the use of mechanisms provided by the programming language;
- Hybrid system architecture: assumes the existence of several multiprocessed nodes linked by networks similar to a distributed system architecture.

For the analysis over the software solution, the possibilities includes implementation via:

- High Performance Fortran (HPF): an explicit parallel language. Extension of Fortran 90 with the incorporation of directives for making data parallelism viable;
- OpenMP: a set of programming directives and libraries to parallel application programming, based in a shared memory model using explicit parallelism;
- Parallel Virtual Machine (PVM): a package of integrated libraries and software tools that allows serial, parallel and vectorial computers to be connected into a same network, working as a single computational resource;
- Message Passing Interface (MPI), a message trade library used in a shared memory environment, viable through explicit calls to the communication routines, where every parallelism is explicit.

Besides the classification presented, grid computing techniques, where the computational resources are scattered in a large geographical area, can be classified as a computing system solution, i.e., a software platform that manages the data traffic in a safe way, and administrates the execution of designed tasks.

To ensure the efficiency of a parallel algorithm, we need to perform some verification regarding the efficiency of the proposed algorithm.

Two related analyses can be done, the first one will check the speedup rate of the parallel algorithm in comparison to its serial version. The speedup analysis is defined, according to [20], as the ratio of the execution time of the serial algorithm (T_s) over the execution time of the parallel algorithm (T_p):

$$S_p = \frac{T_s}{T_p} \tag{1}$$

The second analysis, related to the efficiency of the algorithm, is defined as the ratio of speedup (S_p) over the number of used processors (p), trying to check how much the parallelism was explored by the algorithm:

$$E_p = \frac{S_p}{p} \leq 1 \tag{2}$$

Where, to ensure the efficiency of the proposed algorithm, the value of Equation 2 must be less or equal to one, as stated in [20].

3.1 Parallel implementation

A preliminary study of the M-PCA code has led to the chosen parallelization strategy.

Immediately some parallel strategies can be thought: (a) Coarse Grained: where the particles are divided on several clusters of particles, some particles can migrate (this is a new operator) from one cluster to another; (b) Fine Grained: such approach requires a large number of processors because the particles are addressed into a large number of particle clusters, and at each cluster a search strategy is applied, but subject to migration; (c) Global Parallelization: in this approach all search operators and the evaluation of all individuals are explicitly parallelized.

Strategy (a) admits several schemes: insulated clusters (particle migration could be implemented in several ways: particles cross from one cluster to other, or particles from one cluster could be send and/or received between the master cluster and others), cluster ring (particles travel only to the closest cluster, this may also be named as stepping-stone strategy). It is not clear which parallelization scheme is more effective for strategy (a). For a huge amount of particles, the strategy (b) could be applied in massively parallel machines. However, considering few particles (less than 100, for example) this strategy is a good option for many HPC systems, and its implementation is straightforward. For parallelizing the computation of the objective function for many particles to different processors is not a difficult challenge, but the parallelization of other aspects from the PCA is not clear.

Due to the small number of particles adopted for the present implementation of M-PCA, the parallel strategy implemented is the procedure (b).

The related code was parallelized using calls to the message passing communication library MPI [14] and executed on a distributed memory parallel machine. This machine is a HP XC cluster of 112 standard AMD-Opteron 2.2 GHz scalar architecture processors connected by a InfiniBand interconnection.

The prevailing trend in the search for high performance is the use of parallel machines due to their good cost effectiveness. Three parallel architectures are usually considered: vector processors, multiprocessors with shared memory, and distributed memory machines. In the multiprocessors shared class, all processors access a unique memory address space and there are scalability constraints. In the latter class, off-the shelf machines called nodes are interconnected by a network composing a cluster or Massive Parallel Processors (MPP), a parallel machine with hundreds or thousands of nodes using a very fast interconnection scheme. The processors of each node access only their local memories and data dependencies between node memories enforce communication by means of routines of a message passing library, like Message Passing Interface (MPI) or Parallel Virtual Machine (PVM). Data dependencies between processors require calls to the MPI library. On the other hand, a single processor/node may perform tasks like gathering partial results obtained by every node and broadcasting the global result to all other nodes, also by means of MPI calls.

An important issue is to maximize the amount of computation done by each processor and to minimize the amount of communication due to the MPI calls in order to achieve good performance. The speedup is defined as the ratio between the sequential and parallel execution times. A linear speedup denotes that processing time was decreased by a factor of p when using p processors and can be thought as a kind of nominal limit. Efficiency is defined as the ratio of the speedup by p and thus it is 1 for a linear speedup. Usually, communication penalties lead to efficiencies lesser than 1. Exceptionally, as data is partitioned among processors, cache memory access can be optimized in such way that superlinear speedup can be attained, i.e. efficiencies greater than 1.

4 TEST RESULTS

The PCA was set to execute 10^5 iterations, and the M-PCA was set to operate with 10 particles, each one executing 10^4 iterations. The test functions used were Easom, Rosenbrock and Griewank

(for minimization) and Shekel (for maximization). The optimum results for these functions are presented in Table 1.

Table 1 – Optimum results for tested functions.

Function	x^*	$f(x^*)$
Easom	(π, π)	–1
Shekel	(–32, –32)	499.002
Rosenbrock	(1, 1)	0
Griewank	(0, 0)	0

The comparative results for the Easom function can be seen in Table 2, where the great advantage of M-PCA over PCA is clear. Even if the results are similar, the time of execution demonstrates a great advantage for M-PCA. The global minimum for this function is located at $x^* = (\pi, \pi)$ with $f(x^*) = -1$.

Table 2 – Easom function results.

	PCA	M-PCA
Time	19.37s	4.29s
x^*	(3.14, 3.14)	(3.14, 3.14)
$f(x^*)$	–1.00	–1.00

Table 3 shows the results for the Shekel function, now a maximization function. Once again, both algorithms reached good results, but the time needed for M-PCA execution is significantly lower. The global minimum for this function is located at $x^* = (–32, –32)$ with $f(x^*) = 499.002$.

Table 3 – Shekel function results.

	PCA	M-PCA
Time	3099.53s	113.56s
x^*	(–32.0, –32.0)	(–32.0, –32.0)
$f(x^*)$	499.002	499.002

For the Rosenbrock function, the results are seen on Table 4. And once more, the execution time for M-PCA reveals its advantages. The global minimum for this function is located at $x^* = (1, 1)$ with $f(x^*) = 0.0$.

Table 4 – Rosenbrock function results.

	PCA	M-PCA
Time	1492.23s	21.21s
x^*	(1.00, 1.00)	(1.00, 1.00)
$f(x^*)$	0.0	0.0

And finally the results for the Griewank function are seen on Table 5, where the M-PCA surpasses its canonical version, both in time and accuracy. The global minimum for this function is located at $x^* = (0, 0)$ with $f(x^*) = 0$, and only M-PCA was able to find it with great results.

Table 5 – Griewank function results.

	PCA	M-PCA
Time	2392.05s	32.03s
x^*	(–3.14, 4.43)	$(-1.82 \times 10^{-8}, -3.25 \times 10^{-8})$
$f(x^*)$	7.39×10^{-3}	3.33×10^{-16}

A speedup and efficiency analysis considering the Easom function for M-PCA results given by Table 2 gives a speedup rate of $S_p = 4.447$ and a efficiency around $E_p = 0.4447$ for the given parameters.

The M-PCA speedup curve for the Easom function solved with 2, 5 and 10 processors is given by Figure 4.

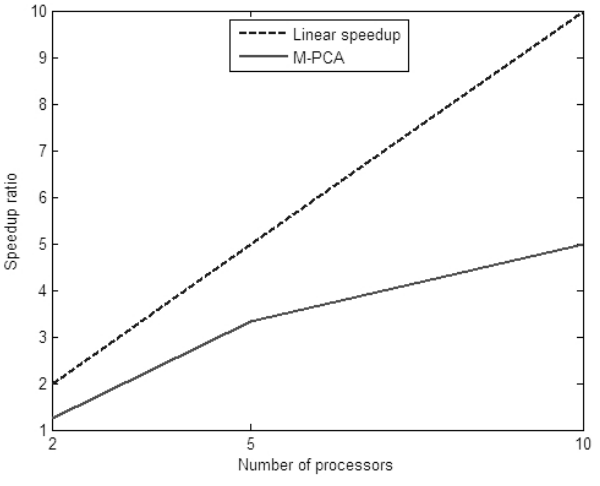

Figure 4 – Speedup curve for M-PCA with 2, 5 and 10 processors.

The speedup curve of M-PCA is bellow the desired curve (for a linear speedup) due to the high amount of communication between the processors, triggered by the update of Best_Fitness information in the blackboard.

A way to improve the performance of M-PCA is to update the Best_Fitness after a determined number of cycles of iterations. Such a feature will decrease the communication messages among processors, resulting in a better performance for the parallel M-PCA.

In preliminary tests, the application of a 1000 iteration communication cycle, i.e. the execution of 1000 iterations until the implementation of communication amongst processors, leads

to an speedup and efficiency improvement, $S_p = 6.933$ and $E_p = 0.6933$, as we can see in Figure 5.

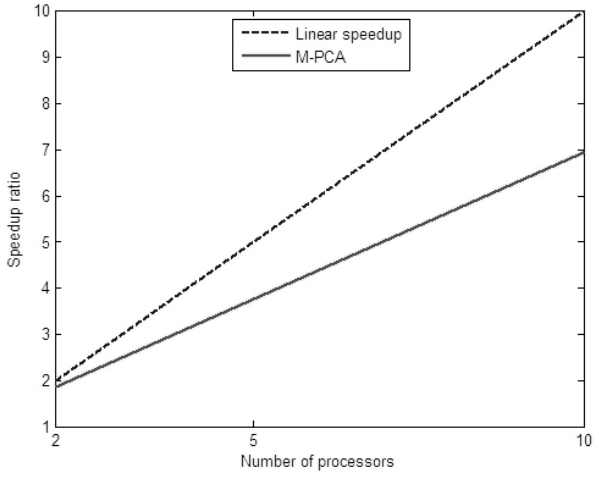

Figure 5 – Speedup curve for M-PCA with 2, 5 and 10 processors and 1000 iterations for communication cycles.

So, the M-PCA proves its advantages over the canonical version, particularly: under the Griewank test function, when analyzing its accuracy; and over all the test functions, when analyzing its execution time.

5 CONCLUSIONS

A multi-particle collision algorithm was developed, and its complexity analysis was carried out ($O(N^2)$ order complexity in a parallel implementation). Results applying the PCA and M-PCA considering several test functions were presented. The M-PCA produced better results (computational time) in all cases analyzed. Parallel versions for the M-PCA were discussed, and a global parallel approach was implemented using MPI instructions. Performance for the parallel M-PCA was also presented.

Optimization algorithms are used for a large range of applications. A very important application for the optimization process is to solve inverse problems. Bio-geo-chemical cycles for greenhouse effect gases (such as methane and carbon dioxide) can be identified using regularized inverse solutions [7], employing an implicit approach. There is work on techniques to improve inverse solutions obtained using quasi-Newton deterministic optimization [15] and PSO stochastic optimization [10, 11], for overcoming local minima and also to compute better inverse solutions. In the employed implicit methodology, the inverse problem is formulated as an optimization problem which minimizes an objective function. For each candidate solution, the value of this function is given by the square difference between experimental values and the data given by the direct model [15]. A typical estimation may demand thousands of iterations and, therefore, optimization performance is an important issue. Therefore, parallel implementation for solving implicit inverse problems is a very relevant issue to improve the performance of the inversion process.

ACKNOWLEDGMENTS

The authors wish to thank Dr. W.F. Sacco (UERJ) for the discussion about the PCA. Special thanks go to the C-PAD team at INPE/SJC. This work was partially supported by CAPES.

REFERENCES

[1] ANTONIOU A & LU W-S. 2007. Practical optimization: algorithms and engineering applications. Springer, New York.

[2] BECCENERI JC. 2008. Chapter Meta-Heurísticas e Otimização Combinatória: Aplicações em Problemas Ambientais. INPE, São José dos Campos.

[3] DORIGO M & STUTZLE T. 2004. Ant Colony Optimization. The MIT Press, Cambridge.

[4] FLYNN MJ. 1966. High-speed computing systems. Proceedings of the IEEE, 54(12): 1901–1909.

[5] GOLDBERG DE. 1989. Genetic Algorithms in search optimization and Machine Learning. Addison-Wesley, Boston, MA, USA.

[6] HOLLAND JH. 1992. Adaptation in natural and artificial systems. MIT Press, Cambridge, MA, USA.

[7] KASIBHATLA P, HEIMANN M, RAYNER P, MAHOWALD N, PRINN RG & HARTLEY DE (Eds.). 2000. Inverse Methods in Global Biogeochemical Cycles. American Geophysical Union, Washington, USA.

[8] KENNEDY J & EBERHART EC. 1995. Particle swarm optimization. IEEE Int. Conf. Neural Networks, 4: 1942–1948.

[9] KIRKPATRIK S, GELATT CD & Vecchi MP. 1983. Optimization by simulated annealing. Science, 220: 671–680.

[10] LUZ EFP. 2007. Estimação de fonte de poluição atmosférica usando otimização por enxame de partículas. Master's thesis, Computação Aplicada, INPE, São José dos Campos.

[11] LUZ EFP, VELHO HFC, BECCENERI JC & ROBERTI DR. 2007. Estimating atmospheric area source strength through particle swarm optimization. Florida: Proceedings of IPDO.

[12] MEHRABIAN AR & LUCAS C. 2006. A novel numerical optimization algorithm inspired from weed colonization. Ecological Informatics 1: 355–366.

[13] METROPOLIS N, ROSENBLUTH AW, ROSENBLUTH MN, TELLER AH & TELLER E. 1953. Equation of state calculations by fast computing machines. Journal of Chemical Physics, 21: 1087–1092.

[14] PACHECO P. 1996. Parallel programming with MPI. Morgan Kaufmann Publishers, San Francisco, USA.

[15] ROBERTI DR, ANFOSSI A, VELHO HFC & DEGRAZIA GA. 2005. Estimation of emission rate of pollutant armospheric source. Proceeding of ICIPE 2005, 3: R03-1–R03-8.

[16] SACCO WF & DE OLIVEIRA CRE. 2005. A new stochastic optimization algorithm based on a particle collision metaheuristic. Proceedings of 6th WCSMO.

[17] SACCO WF, FILHO HA & PEREIRA CMNA. 2007. Cost-based optimization of a nuclear reactor core design: a preliminary model. Proceedings of INAC.

[18] SACCO WF, LAPA CMF, PEREIRA CMNA & FILHO HAA. 2006. Two stochastic optimization algorithms applied to nuclear reactor core design. Progress in Nuclear Energy, 525–539.

[19] SACCO WF, LAPA CMF, PEREIRA CMNA & FILHO HAA. 2008. A metropolis algorithm applied to a nuclear power plant auxiliary feedwater system surveillance tests policy optimization. Progress in Nuclear Energy, 15–21.

[20] SAMBATTI SBM. 2004. Diferentes estratégias de paralelização de um algoritmo genético epidêmico aplicadas na solução de problemas inversos. Master's thesis. Computação Aplicada, INPE, São José dos Campos.