Modern Heuristics Review for PID Control

P. B. de Moura Oliveira

***Abstract*— A set of modern heuristic techniques is reviewed in the context of PID control structures optimization. The selected techniques are: simulated annealing, genetic algorithm, population based incremental learning algorithm, particle swarm optimization algorithm and the differential evolution algorithm. An introduction to each algorithm is provided followed by an illustrative example based in a simulation assignment of an evolutionary algorithms course. Some conclusions are presented about the effectiveness of the reviewed heuristics based on the simulation results.**

I. Introduction

THE study of optimization techniques within engineering courses is of crucial importance. Indeed, this is well stated in the following quotation from Schwefel [12] extracted from Michalewicz book [10] introductory chapter: "There is scarcely a modern journal, whether of engineering, economics, management, mathematics, physics or social sciences, in which the concept 'optimization' is missing from the subject index". Thus, this is a topic that should be addressed in lecturing modern heuristics to engineering courses as one of their main applications is to solve optimization problems. This is the case of the course entitled *evolutionary algorithms* of the *MSc in Engineering Technologies* at the UTAD University in Portugal. During this course several assignments were given to students consisting either in elaborating a heuristic/evolutionary computation topic survey or to apply an evolutionary inspired algorithm to solve an engineering problem. One of the selected assignments, which motivated this article, concerns the optimization of proportional integral and derivative (PID) controllers in the continuous time domain using different modern heuristics. In this paper a series cascaded control configuration is considered.

The PID controller is the most popular controller used in industrial control applications. The overwhelming dominance of PID controller over other forms of feedback, in the last fifty years, its due to its simple structure and reliability in a wide range of operating conditions. Some estimates state that more than 95% of the controllers used in process control applications are of PID type [1]. Due to the wide acceptance of PID controllers within industry many tuning rules have been proposed for this type of controller, since the original work of Ziegler and Nichols [15]. However, despite the huge amount of existing tuning rules for PI/PID controllers, the test of a large set of industrial plants [4], indicated that 30% of the controllers were operated manually and 65% were poorly tuned. Indeed, plant operators tend to tune PID controllers by trial and error or using very simple tuning rules. A plausible explanation is the lack of appropriate educational background from most of the process control operators [11] concerning the proper use of tuning methodologies. In some cases, the control operator can be responsible for hundreds of process control loops [2], with a wide range of system dynamics, having to design and tune PID controllers to meet performance and robustness specifications. Considering the huge number of tuning rules proposed for different process models and the limited amount of effort that the plant operator can devote to each loop, the existence of an universal tuning method would simplify their role significantly. Because there is not an universal tuning method, the use of an optimization algorithm, particularly using modern heuristics, constitutes a global tool to the design and tuning of PID controllers for a wide range of control engineering applications.

Manuscript received March 20, 2005.

P. B. de Moura Oliveira is with the Universidade de Trás-os-Montes e Alto Douro and CETAV- Centro de Estudos das Tecnologias Ambiente e Vida, 5000-911 Vila Real, Portugal (phone: 351-259-350339; fax: 351-259-350300; e-mail: oliveira@ utad.pt).

II. Review of Some Modern Heuristic Algorithms

This section provides a short review of some modern heuristic algorithms from an optimization perspective. The algorithms description is based on their original forms with minor extensions, in order to maintain the conceptual simplicity that made them easily understandable, motivating their wide implementation and use.

A potential solution for a general combinatorial optimization problem in a *n*-dimensional search space defined by an objective function *f* (1), can be represented by a vector $\vec{x} \in \Omega$ (2), in which *t* represents the current iteration and Ω the feasible set. An important issue within any optimization methodology is how to generate a new

trial solution vector $\vec{x}(t+1) \in \Omega$. This can be generated from the current vector by using equation (3), with $\vec{\delta}$ representing an incremental vector (4).

$$f : \Omega \subseteq R^n \rightarrow R, \Omega \neq 0 \quad (1)$$

$$\vec{x}(t) = x_1(t), x_2(t), \ldots, x_n(t) \quad (2)$$

$$\vec{x}(t+1) = \vec{x}(t) + \vec{\delta}(t) \quad (3)$$

$$\vec{\delta}(t) = \delta_1(t), \delta_2(t), \ldots, \delta_n(t) \quad (4)$$

A. Simulated annealing

The basic simulated annealing algorithm proposed by Metropolis et al. [9] uses a temperature variable, T, that is decreased through the optimization process following a specified scheduled or by using a function. Considering that a new solution is generated in the neighborhood of the current solution value, the difference between the current and new solutions is represented by δE (5), known as energy gradient. In the context of minimization problems, the replacement of the current solution by the new solution is performed using a condition defined by (6) based on a Boltzmann probability threshold. Figure 1 illustrates the basic simulated annealing algorithm for minimization problems.

$$\delta E = f(\vec{x}(t+1)) - f(\vec{x}(t)) \quad (5)$$

$$\begin{cases} \vec{x}(t) = \vec{x}(t+1) & \text{if } \delta E < 0 \\ \vec{x}(t) = \vec{x}(t+1) \Leftarrow \dfrac{1}{1+\exp\left(\dfrac{\delta E}{T}\right)} > rand[0,1] & \text{otherwise} \end{cases} \quad (6)$$

```
t=0
T= Tinitial
initialize x(t) randomly
while(!(termination criterion))
  while(!(repetition criterion))
    generate a new trial solution x(t+1)
    δE = f(x(t+1)) − f(x(t))
    if (δE < 0)
      x(t) = x(t+1)
    else if 1/(1+exp(δE/T)) > rand[0,1]
      x(t) = x(t+1)
    end
  end
  t=t+1
  T=annealing(T,t)
end
```

Fig. 1. Basic simulated annealing algorithm.

B. Genetic Algorithm

The genetic algorithm (GA), is a randomly based technique to search for the best problem solutions, and was proposed by John Holland [7]. Over the last 30 years, GAs have been used to solve a wide range of search, optimization, and machine learning problems [5]. The basic GA idea is to maintain a set of data structures, population, that represents candidate solutions for a certain problem. Population elements compete between them to pass their genetic material to future generations by recombination, following the survival of the fittest principle. In this way, the fitter elements of current population take part on the matting pool used to form the new population. Three processes are used to make the transition (reproduction) from one population to the next: selection, crossover and mutation. The basic genetic algorithm is shown in Figure 2, considering that a population *X* of potential solutions with size *m* is represented by (7), with *d* representing the dimension index.

$$X(t) = (x_{i1}(t), x_{i2}(t), \ldots, x_{id}(t)) \quad 1 \le i \le m \quad 1 \le d \le n \quad (7)$$

```
t=0
initialize population X(t)
evaluate X(t)
while(!(termination criterion))
  t=t+1
  select parents to mate
  generate children with crossover
  mutation
  evaluate X(t+1)
  replace old population by the new population elements
end
```

Fig. 2. Basic genetic algorithm.

C. Population based incremental learning

The population based incremental learning algorithm (PBIL) was developed by Baluja [3] and combines features of GA and hill-climbing algorithm. Each element is represented by a binary vector $\vec{b}$, with strings concatenated according with the number of variables involved in the optimization. A probability vector is used to guide the search, $\vec{p}$ (8), where *n* is the binary string length, *d* is the dimension index and *res* is the number of resolution bits used to encode each of the real value elements of the vector $\vec{x}$. Each vector $\vec{p}$ element represent the probability of the string *b* corresponding bits are assigned to *1* as it is shown by (9), and are initialized with a probability of *0.5*. The random function returns a pseudo-random number uniformly distributed in the interval [0,1]. As the search evolves, the probability vector values moves away from the equilibrium point, *0.5*, towards *0* or *1*. The basic PBIL algorithm it is represented by Figure 3.

$$\vec{p}(t) = (p_1(t), p_2(t), \ldots, p_j(t)) \quad 1 \le j \le n = d * res \quad (8)$$

$$\begin{cases} rand[0,1] < p_j(t) \Rightarrow b_{ij} = 1 \\ rand[0,1] \ge p_j(t) \Rightarrow b_{ij} = 0 \end{cases} \quad 1 \le i \le m \quad 1 \le j \le n \quad (9)$$

```
t=0
initialize probability vector p = (0.5,0.5,…,0.5)
while(!(termination criterion))
  t=t+1
  generate new population X(t) using the probability vector
  evaluate X(t)
  update the probability vector
  adjust the probability vector
end
```

Fig. 3: Basic PBIL algorithm.

The generation of each population member is done randomly for each bit by sampling the corresponding element of the probability vector using expression (9). In each PBIL cycle the solutions are decoded and evaluated, allowing finding the best member in terms of performance (best). The probability vector is thus updated towards the direction of the best solution found, using (10) in which LR represents the Learning Rate.

$$p_j(t+1) = (1-LR)* p_j(t) + LR * b_{best\ j} \quad 1 \le j \le n \quad (10)$$

The probability vector moves away from the worst population element in each generation, only in the bits that are different from the ones in the best element:

$$p_j(t+1) = (1-NLR)* p_j(t) + NLR * b_{best\ j} \quad \Leftarrow b_{best\ j} \neq b_{worst\ j} \quad 1 \le j \le n \quad (11)$$

in which *NLR* represents the *Negative Learning Rate*. In the final block of the basic PBIL cycle (Figure 3) an adjustment of the probability vector is done to avoid premature convergence in a local maximum. This adjustment can be implemented by using the original mutation operator:

$$p_j(t+1) = p_j(t) - FF\left(p_j(t) - 0.5\right) \quad 1 \le j \le n \quad (12)$$

where: ρ represents the mutation probability and ρ_d represents the quantity that the corresponding bit is mutated. It can also be done by using a mutation variation proposed by [6], that uses a *Forgetting Factor (FF)*:

$$p_j(t+1) = p_j(t) - FF\left(p_j(t) - 0.5\right) \quad 1 \le j \le n \quad (13)$$

which updates the probability vector towards the neutral value, *0.5*, in a small quantity.

D. Differential evolution

Considering a set X of potential solutions represented by (14) with size m, equation (3) can be rewritten as (15) with *d* representing the dimension index.

$$X_i(t) = \left(x_{i1}(t), x_{i2}(t), \ldots, x_{in}(t)\right) \quad 1 \le i \le m \quad (14)$$

$$x_{id}(t+1) = x_{id}(t) + \delta_{id}(t+1) \quad 1 \le i \le m \quad 1 \le d \le n \quad (15)$$

In the differential evolution algorithm (DE) proposed by Storn and Price [14], a trial vector $\vec{x}_v \in R^n$ is generated in each iteration for each population member using (16) and the increment δ is evaluated using (17). In these expressions $\vec{x}_{r_1}$, $\vec{x}_{r_2}$, $\vec{x}_{r_3}$ are vectors that may be selected randomly from the population set X in each iteration, and $F \in R^+$ is a constant defined prior to the search procedure. In this case $\vec{x}_{r_1}$ is the best solution vector found in the previous iteration.

$$x_{vid}(t+1) = x_{r_1 d}(t) + \delta_{id}(t+1) \quad 1 \le i \le m \quad 1 \le d \le n \quad r_1 \in [1,m] \neq i \quad (16)$$

$$\delta_{id}(t+1) = F\left(x_{r_2 d}(t) - x_{r_3 d}(t)\right) \quad 1 \le i \le m \quad 1 \le d \le n \quad r_2 \in [1,m] \neq i \neq r_1 \neq r_2 \quad r_3 \in [1,m] \neq i \neq r_1 \neq r_2 \quad (17)$$

$$\vec{x}_{ci}(t+1) = crossover\left(\vec{x}_i(t), \vec{x}_{vi}(t+1)\right) \quad 1 \le i \le m \quad (18)$$

$$\vec{x}_i(t+1) = \vec{x}_{Ci}(t+1) \quad if \quad f\left(\vec{x}_{Ci}(t+1)\right) \le f\left(\vec{x}_i(t+1)\right) \quad 1 \le i \le m \quad (19)$$

The trial vector obtained with (16) and (17) is then crossed with the current population element $\vec{x}_i$ accordingly to a crossover scheme, binomial or exponential [14], with a pre-defined probability defined by $p_c \in [0,1]$ resulting in a new vector $\vec{x}_{ci}$ (18). If the value returned by the objective function for the crossed trial vector is better or equal than the value obtained for the current vector, the latest is replaced by the former. This is represented by (19) for a minimization problem. The basic DE algorithm is illustrated by Figure 4.

```
t=0
initialize population X(t)
while(!(termination criterion))
  evaluate X(t)
  t=t+1
  while(i<=population size)
    generate the incremental vector δ_i(t+1)
    generate new solution vector x_vi(t+1)
    x_ci(t+1) = crossover(x_i(t), x_vi(t+1))   1 ≤ i ≤ m
    replace current solution by new solution if
    f(x_Ci(t+1)) ≤ f(x_i(t+1))
    i=i+1
  end
end
```

Fig. 4: Basic differential evolution algorithm.

E. Particle swarm optimization

In the particle swarm optimization algorithm (PSO) proposed by Kennedy and Eberhart [8] each particle of a

swarm with size *m*, moves through a *n*-dimensional search space, with positions and velocities defined for the current evolutionary iteration *t* by (20) and (21), respectively. The velocity vector set, *V* (21), corresponds to the set of incremental vectors as it is indicated by (22).

$$X_i(t) = (x_{i1}(t), x_{i2}(t), \ldots, x_{in}(t)) \quad 1 \le i \le m \quad (20)$$

$$V_i(t) = (v_{i1}(t), v_{i2}(t), \ldots, v_{in}(t)) \quad 1 \le i \le m \quad (21)$$

$$\Delta_i(t) = (\delta_{i1}(t), \delta_{i2}(t), \ldots, \delta_{in}(t)) = V_i(t) = (v_{i1}(t), v_{i2}(t), \ldots, v_{in}(t)) \quad 1 \le i \le m \quad (22)$$

The increment (or velocity) is evaluated using (23) and it is used to determine particles new position with (24), in which $p_{id}(t)$ and $p_{gd}(t)$ represent the best previous position of particle *i* and the global best, respectively, in the current iteration *t*, for a pre-defined neighborhood type. Parameter φ_c is known as the *Cognitive Constant* and φ_s as the *Social Constant*, representing uniformly distributed random numbers generated in a pre-defined interval.

$$\delta_{id}(t+1) = v_{id}(t+1) = v_{id}(t) + \varphi_c.(p_{id}(t) - x_{id}(t)) + \varphi_s.(p_{gd}(t) - x_{id}(t)) \quad 1 \le i \le m \quad 1 \le d \le n \quad (23)$$

$$x_{id}(t+1) = x_{id}(t) + \delta_{id}(t+1) \quad 1 \le i \le m \quad 1 \le d \le n \quad (24)$$

An additional parameter was incorporated into equation (24) by [13] resulting in (25), in which ω was termed *Inertia Weight*.

$$\delta_{id}(t+1) = v_{id}(t+1) = \omega v_{id}(t) + \varphi_c.(p_{id}(t) - x_{id}(t)) + \varphi_s.(p_{gd}(t) - x_{id}(t)) \quad 1 \le i \le m \quad 1 \le d \le n \quad (25)$$

```
t=0
Initialize population X(t)
while(!(termination criterion))
  Evaluate X(t)
  t=t+1
  while(i<=population size)
    generate the incremental vector  δ_i(t+1)
    generate new solution vector  x_i(t+1)
    i=i+1
  end
  replace X(t) by X(t+1)
end
```

Fig. 5: Basic particle swarm optimization algorithm.

The value given to the inertia weight will affect the type of search in the following way: a large inertia weight will direct the PSO for a global search while a small inertia weight will direct the PSO for a local search. The PSO algorithm is illustrated by Figure 5.

III. Evolutionary Algorithms Assignment: Optimization of Cascaded PID Control Structures

Cascade control is one of the most successful methods for enhancing single input single loop systems control performance. A series cascade control strategy combines two feedback controllers, with the *primary* controller output serving as the *secondary* controller set-point as it is shown in Figure 6.

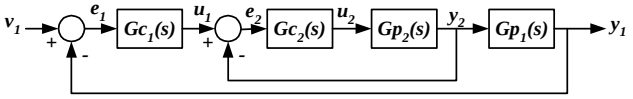

Fig. 6: Series Cascade Control System.

The cascaded PID controllers are governed by:

$$u_{1,2}(t) = K_{p1,2}\left[e_{1,2}(t) + \frac{1}{T_{i1,2}}\int_o^t e_{1,2}(t)dt - T_{d1,2}\frac{d\,y_{1,2f}(t)}{dt}\right] \quad (26)$$

in which: $e_1(t) = v(t) - y_1(t)$, $e_2(t) = u_1(t) - y_2(t)$ are the primary and secondary loop error signals, $y_1(t)$, $y_{1f}(t)$, $y_2(t)$ and $y_{2f}(t)$ are the primary and secondary loop non-filtered and filtered outputs, respectively, K_p represents the proportional gain, T_i and T_d the integral and derivative time constants. The derivative action is applied to the system output in order to avoid derivative kicking and filtered using (27), with N_f representing the filter constant.

$$\frac{T_d}{N_f}\frac{d\,y_{f1,2}(t)}{dt} + y_{f1,2}(t) = y_{1,2}(t) \quad (27)$$

In order to use the reviewed algorithms to optimize the PID gains it is necessary to encode the tuning parameters. Thus each solution vector represents the proportional, integral and derivative parameters for both loops, by using either a binary or real-based coding scheme. The controller design is accomplished by optimizing the system response to a unit set-point change minimizing a time-domain performance criterion. In this study the well known *Integral of the Time-Weighted Absolute Error (ITAE)* is adopted.

IV. Simulation Results

The models selected for the primary and secondary loop processes dynamics are represented by (28), were deliberately selected equal in order to make the design procedure more complicated. Thus, the simultaneous design of the primary and secondary cascaded controllers is recommended, and an optimization algorithm can be used. The search space for the controller parameters is defined by the set [0 0 0.1,5 5 5] for the primary controller and [0 0

0,5 5 5] for the secondary controller. The time period considered for each step response is 100 seconds and $N_f=10$.

$$G_{p1,2}(s)=\frac{1}{(s+3)}e^{-s} \quad (28)$$

The number of objective function evaluations is set equal to 1500 in order to perform a fair comparison between all the optimization techniques. The initial set is generated randomly, and thus may include non-viable solutions corresponding to gains that make the system unstable.

In the *SA* algorithm the number of iterations used is N=15 and the repetition cycle is executed 100 times per iteration. The initial temperature is set to *Tinitial=100* and the annealing function is represented by $T=0.99*T$. The GA was implemented with: a population of size *m=30*, a number of generations *N=50*, a binary encoding with 10 bits per dimensional parameter, a stochastic sampling selection scheme, a single point crossover scheme with probability *pc=0.6* and a mutation probability of *pm=0.033*. The new population replaces the old population, except for the best member which is duplicated in the new population replacing the worst element. In the PBIL algorithm the population size used is *m=30*, the number of iterations is *N=50*, a binary encoding is used with 10 bits per dimensional parameter, a learning rate *LR=0.1* and a forgetting rate of *FF=0.5%*. The update and adjustment of the probability vector is done using expressions (10) and (13). The population size used in the PSO algorithm is *n=30* and the number of iterations is *N=50*, The inertia weight ω is set to change linearly in the interval [0.7,0.4] during the evolution. Parameters φ_1 and φ_2 are uniformly distributed random numbers generated in the interval [0,1]. The population size used in the DE algorithm is *n=30* and the number of iterations is *N=50*. A binomial crossover operator is adopted with a probability $p_c=0.8$, and the amplification parameter is set to *F=0.8*. The results obtained with all the reviewed algorithms for the cascaded PID control design are presented in Tables 1 and 2, for a number of 20 runs. The results show that PSO algorithm achieved the best ITAE trial result, while in terms of mean best result and standard deviation the DE algorithm yields better results.

TABLE 1: BEST PID GAINS.

Algorithm	Best PID Gains					
	Kp1	Ti1	Td1	Kp2	Ti2	Td2
SA	2,517	0,092	2,522	1,315	0,309	2,570
GA	0,753	0,312	0,368	4,615	0,285	0,318
PBIL	1,984	0,142	1,310	2,185	0,196	1,144
PSO	1,175	0,301	0,197	5,000	0,000	0,115
DE	1,122	0,317	0,100	4,307	0,000	0,000

TABLE 2: STATISTICAL TEST DATA.

Algorithm	Best ITAE	Worst ITAE	Mean Best ITAE	Standard Deviation
SA	1166,51	1753521	180926,3	524221,7
GA	1010,84	3406,80	1991,30	831,836
PBIL	977,68	1958,07	1271,80	449,273
PSO	637,53	2910,67	1063,08	735,260
DE	710,88	1818,74	828,547	330,344

The cascaded system simulated responses to a reference step input are shown in Figures 7 and 8, for the system output and control system, respectively. The PID controller parameters used correspond to the best results obtained with the respective optimization technique. Figure 9 illustrates the convergence rate over 20 runs corresponding to the mean ITAE performance index for each DE and PSO algorithms. It shows that the DE algorithm has a faster convergence rate in the beginning and end of the runs.

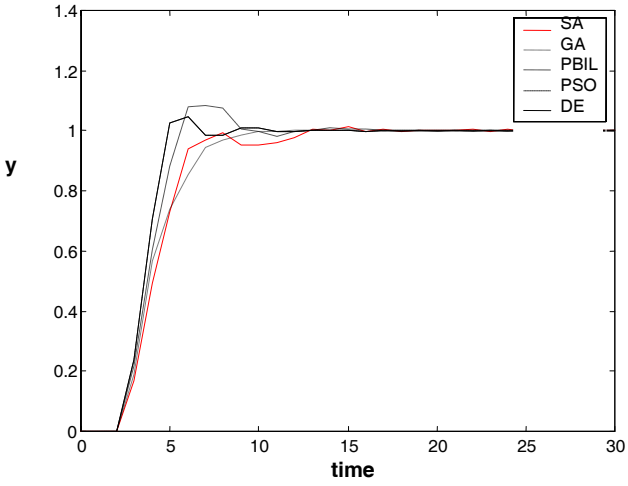

Fig. 7. Set-point tracking responses for the cascaded system

V. CONCLUSION

Some of the most popular modern heuristics were reviewed in the context of an evolutionary algorithms option assignment, requiring the optimization of PID controllers for cascaded feedback loops. The selected techniques are: simulated annealing, genetic algorithm, population based incremental learning algorithm, particle swarm optimization algorithm and the differential evolution algorithm. The best test result was achieved with the particle swarm optimization algorithm, while for the mean of the best result and standard deviation, the differential evolution algorithm delivered the best results. It is important to note that the result analysis is conditioned by the assignment specification in terms of the small number of function evaluations and population sizes used in each optimization trial. This requirement is important in practice

to make the PID controller optimization as fast as possible. Further work should be carried out by considering use a higher number of function evaluations and population sizes for each trial as well as different settings for each optimization technique.

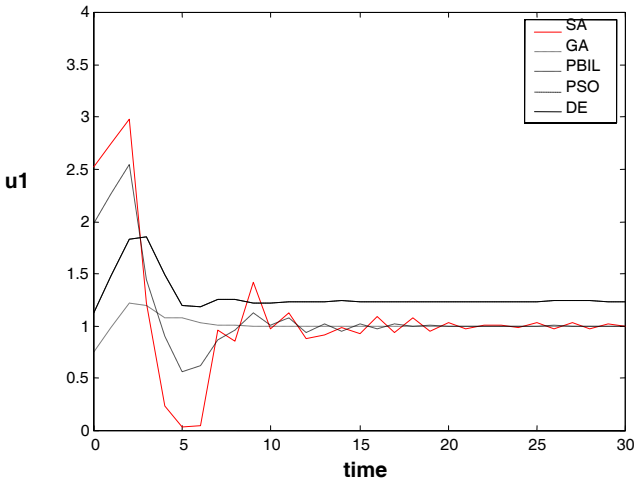

Fig. 8. Control signals for the primary controller of the cascaded system.

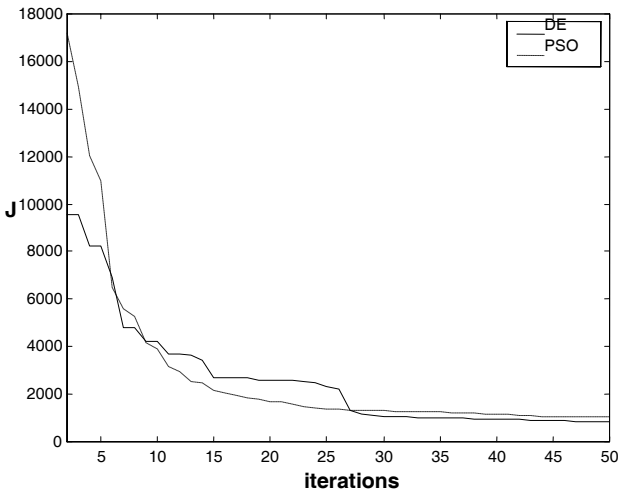

Fig. 9. Convergence plot for the mean best fitness over the 20 runs for the DE and PSO algorithms.

REFERENCES.

[1] Åström, K. J. and Hägglund, T. (1995). PID Controllers: Theory, Design and Tuning, Instrument Society of America, Research Triangle Park, 2nd edition.

[2] Åström, K. J. and Hägglund, T. (2000a). The Future of PID Control, IFAC Work. on Digital Control: Past, present and future, Spain, Terrassa, pp. 19-30.

[3] Baluja, S., (1994). Population Based Incremental Learning: A Method for Integrating Genetic search Based Function Optimization and Competitive Learning, Technical report CMU-CS-95-163, School of Computer Science, Carnegie Melon University, USA.

[4] Ender D. B. (1993). Process Control Performance: not as Good as you Think, Control Engineering, September, pp. 180-190.

[5] Goldberg E D, (1989). Genetic Algorithms in Search, Optimization and Machine Learning, Adison Wesley P.C.

[6] Greene J. R., (1997). A Role for Simple, Robust 'Black-Box' Optimisers in the Evolution of Engineering Systems and Artefacts., Second IEE Conference on GAs in Eng. Systems: Innovations and Applications (GALESIA'97), No. 446, pp. 427-432 , September, Sheffield, UK.

[7] Holland J. H., (1975). Adaptation in Natural and Artificial Systems, 1st MIT Press ed.

[8] Kennedy J. and Eberhart R.C. (1995). Particle swarm optimization. Proc. IEEE Int. Conf. on Neural Networks, Perth, Australia, pp. 1942-1948.

[9] Metropolis, N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H. and Teller, E., (1953), Equation of state calculation by fast computing machines,. J. of Chem. Phys., 21, pp. 1087-1091.

[10] Michalewicz, Z. (1992). Genetic Algorithms+Data Structures=Evolution Programs, Second edition, Springer Verlag, pp.16.

[11] Pomerleau, A. and Poulin, É. (2002). A New Approach for Teaching Process Control, Proc. of IASTED MIC'2002, February, Innsbruck, Austria, pp. 275-279.

[12] Schwefel, H. P. (1981). Numerical Optimization for Computer Models, John Wiley, Chicester, UK.

[13] Shi Y. and Eberhart R. C. (1999). Empirical Study of Particle Swarm Optimization, *Proceedings of the 1999 IEEE Congress of Evolutionary Computation*, 3, pp. 1945-1950.

[14] Storn R. and Price K. (1997). Differential Evolution: a Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces. Journal of Global Optimization Vol. 11, Nº 4, pp. 341-349.

[15] Ziegler, J. G. and Nichol,s N. B., (1942). Optimum Settings for Automatic Controllers, Transactions of the ASME, **64**, pp. 759-768.