Think Locally, Act Locally – A Framework for Adaptive Particle Swarm Optimizers

Maurice Clerc

***Abstract*—In Particle Swarm Optimization, each particle moves in the search space and updates its velocity according to best previous positions already found by its neighbors (and itself), trying to find an even better position. This approach has been proved to be powerful but needs parameters predefined by the user, like swarm size, neighborhood size, and some coefficients, and tuning these parameters for a given problem may be, in fact, quite difficult. This paper presents a framework for designing adaptive particle swarm optimizers without any "supervisor", in order to obtain autonomous methods in which the user has almost nothing to do but describe the problem. To illustrate the adaptation process, a possible implementation for a simple PSO version is given, so that we can examine how different, and globally better, is the behavior of the swarm on some classical test functions.**

***Index Terms*—adaptation, optimization, particle swarm.**

I. Introduction

When using a non-adaptive Particle Swarm Optimizer (PSO), as defined in [1]–[2], you may need to run it several times using different parameter sets before finding a "good" one. To avoid this drawback, some attempts have already been made to define an adaptive PSO [3]–[9], for example by using selection, by modifying a coefficient during the process itself, or by clustering techniques.

Here, we present a more complete framework systematically based on the particle point of view, which has mainly local information. The key question is then "What would I do if I were a particle"? To illustrate this approach by giving a possible implementation, we also build an adaptive PSO version from a very simple non-adaptive one, so that we can examine the global behavior of the swarm and report results on some classical test functions.

II. Principles and Qualitative Reasonings

Let $H=\{x\}$ be the search space, that is to say the set of the possible positions, and let D be its number of dimensions. Let ϕ be the objective function on H, S the target and ε the maximum admissible error (or minimum wanted accuracy). Let $f=|\phi - S|$ be the error function. We do not examine here the important problem of the stopping criterion for the iterations, and we assume we have at least one. When S is known, this stopping criterion is simply $f(x)<\varepsilon$.

A key concept in particle swarm optimization is the "neighborhood" of a particle. No matter how it is defined (using distances or not, in particular), it can always be seen as a list of particles, including the particle whose neighborhood is being defined. The most widely used neighborhood is the "circular" one. For example, if the particles are numbered (0, 1, 2, 3), the neighborhood of size 3 for the particle 3 is {3-1, 3, 3+1} modulo(4) i.e. {2, 3, 0}.

Any PSO algorithm defines, for a given particle P_i, its position $x_i(t)=(x_{i,1}(t),\ldots,x_{i,d}(t),\ldots,x_{i,D}(t))$ and its velocity $v_i(t)=(v_{i,1}(t),\ldots,v_{i,d}(t),\ldots,v_{i,D}(t))$ at time t+1, depending on the ones at time t. To present the adaptation principles, here we choose a simple version of PSO, which is known as PSO Type 1'' [10]. This has just one coefficient, but the principles can easily be extended to more complicated systems. It is defined in detail in [10], and we give here just the main formula:

$$v_{i,d}(t+1)=\chi(v_{i,d}(t)+rand(0\ldots\frac{\phi}{2})(p_{i,d}(t)-x_{i,d}(t))) \quad (1)$$
$$+rand(0\ldots\frac{\phi}{2})(g_{i,d}(t)-x_{i,d}(t))$$

$$x_{i,d}(t+1)=x_{i,d}(t)+v_{i,d}(t+1)$$

with

$$\chi=\frac{2}{\phi-2+\sqrt{\phi^2-4\phi}}$$

where $p_i=(p_{i,1},\ldots,p_{i,d},\ldots,p_{i,D})$ is the best position found so far by the particle P_i, and $g_i=(g_{i,1},\ldots,g_{i,d},\ldots,g_{i,D})$ the best position found so far by the neighbors (remember that one of them is the particle itself), and where $rand(0\ldots u)$ stands for "a random number (uniform distribution) in $[0,u]$." Note that this coefficient is a priori different for each component, so we can not write just a vector equation like

$$v_i(t+1)=\chi(v_i(t)+rand(0\ldots\frac{\phi}{2})(p_i(t)-x_i(t)))$$
$$+rand(0\ldots\frac{\phi}{2})(g_i(t)-x_i(t)) \quad (2)$$

for such a formula will imply exactly the reverse. For example, it will mean that the coefficient $rand(0\ldots\phi/2)$ is computed just once and has the same value for all components of $(p_i(t)-x_i(t))$. However φ is constant and the same for

Manuscript received June 30, 2002.

M. Clerc is with the France Télécom Recherche & Développement, 90000, Belfort, France (e-mail: Maurice.Clerc@ WriteMe.com).

all particles.

To transform this model into an adaptive one, we have to modify three parameters during the process: the swarm size (by adding and removing some particles), the neighborhood size of each particle, and a different coefficient ϕ for each particle, called now ϕ_i. Here, we try to do that without any "supervisor", just by local actions, assuming that "local" means "in the neighborhood of a particle". So each particle needs some knowledge, and some abilities.

A. What a particle knows

At each time step *t*, each particle P_i can obtain the following information from particles P_j of its neighborhood:

- the position $x_i(t)$ (written just x_i, to simplify),
- the best position ever found p_i,
- the first error function evaluation (we will need it for improvement evaluation), i.e. $f(x_j(t_{0,j}))$, where $t_{0,j}$ is the time step at which P_j has been generated,
- the velocity v_i,

and also, from itself:

- the coefficient ϕ_i used instead of ϕ in (1),
- the neighbour list h_i,
- the last time instant it has performed any adaptation task,
- the previous position value $f(x_i(t\text{-}1))$.

Although the process is designed to be as local as possible, the particle also needs a few global information:

- an adaptive threshold Δ for "enough improvement", as defined below,
- the time *t*, for, as we will see, adaptation has to be made just from time to time,
- the swarm size $N(t)$ (called simply *N* throughout the rest of the paper).

B. What a particle can do

Using this information, the particle has to "think locally, act locally." We assume it can perform, as in non-adaptive PSO, all computations required by the system (1), and can also compute $f(x)$ for any given position x. Now, for the adaptation process, four actions are possible:

- kill itself,
- generate a new particle,
- modify its coefficient ϕ_i,
- modify its neighbour list.

Killing itself and generating a new particle are both modifying the swarm size. In "classical" PSO, it is constant. Some authors use 20, some others 30, and some studies have been done showing the influence of the swarm size [1], [11]–[12], but we have no rule saying a given size is better than another for a given kind of problem. So it seems simpler to let the algorithm modify this size, from time to time, by removing some particles and adding new ones.

We now give the six rules defining our framework, as qualitative "reasoning" for the adaptive actions. Remember that in these rules, "local" from a particle point of view means "in my neighborhood, amongst my neighbors (including myself)." In the next section, we will give some mathematical formulations for a possible implementation.

1) Suicide

The reasoning proposed is:

> *Rule 1*
> "There has been enough local improvement. However, I can't claim any credit for I am the worst local particle. It's time to kill myself."

Improvement for a particle is evaluated by comparing its position when it has been generated and the best position it has found after that.

2) Generation

Now, the reasoning is:

> *Rule 1'*
> "There has been not enough local improvement, although I am the local best. It's time to generate a new particle."

3) Modifying coefficient

The idea is that if everything goes well, the particle can try to speed up its convergence, taking the risk of decreasing the part of the search space it explores. On the contrary, if its improvement is not that good, it would be better to slow down the rate of convergence, and take time to explore a wider part of the search space.

So, a general pair of rules could be

> *Rule 2*
> "The more I improve myself, the smaller can be the part of the search space I explore."

> *Rule 2'*
> "The less I improve myself, the bigger should be the part of the search space I explore."

In PSO Type 1'', this can be done just by increasing/decreasing the ϕ value. This technical point is not really obvious, but it has been shown in [10] by computing the radius of a circular attractor in the phase space of the particle. However, you can note in (1) that both χ and $\chi\phi$ are indeed decreasing when ϕ increases above 4, and so does velocity.

4) Change its neighborhood

Although different topologies are possible for the neighborhood, no clear and constant effects has been found [13]. So, for the moment, we use the simple circular one we have seen above. However, the *size* of this neighborhood is constantly modified for each particle during the process.

The reasoning is:

> *Rule 3*
> "I am the local best, and I have improved myself enough, so I don't need to ask so many neighbors for more information. I can reduce my neighborhood."

Rule 3'

"I am the local best, but I have not improved myself enough, so I need more information. I have to ask more neighbors."

Note that the above implies a particle that is not the local best does not change its neighborhood size at all. Also, as soon as different particles start having different neighborhood sizes, the relation "to be the neighbor of" ceases to be a symmetric one. In fact, in such a case, it would be better to not use the term "neighborhood" but something like "knowable group" or "informers."

C. Acting, but when?

Checking for adaptations at each time step is not a good idea. Not only because it has a cost, but also because, for example, when you add a new particle, this particle needs some time "to prove itself." Unfortunately, there does not exist a method that is mathematically proven to be optimal. The following rule of thumb seems to work well in practice.

When you use a "circular" neighborhood, an estimation of the number of time steps before the next check is

$$\Delta t = E(N/2) \tag{3}$$

where N is the current swarm size, and $E(u)$ the integer part of u. Intuitively, it is quite logical that it should be an increasing function of N, for if there are a lot of particles, they do not need to spend time to be "intelligent," so they can perform adaptations less often.

Also, it has been found that it is better, in a given neighborhood, to not perform all adaptations for all particles at the same time. In particular, at each time step, just one suicide or one generation per neighborhood is enough. One could say that the "thinking unit" is not the particle but the neighborhood.

III. An implementation

To illustrate the qualitative principles we have seen and to be able to give some numerical results, we give here a *possible* implementation. There is no doubt that better ones can be found.

A. Precise definitions

1) Comparing particles

First, we have to define how to compare the particle P_i and the particle P_j, i.e. to define a relation order $\succ$ (read "better than") on the set of particles. We do not take into account the current positions, but only the best previous ones:

$$f(p_i) \leq f(p_j) \Leftrightarrow P_i \succ P_j \tag{4}$$

Note that it means we have $P_i \succ P_i$. More generally, the order relation makes sense even for particles with exactly the same best error function value (it may occur, particularly in a discrete search space).

2) Improvement

The improvement for a given particle P_i is defined by

$$\delta(P_i) = \frac{f(x_i(t_{0,i})) - f(p_i)}{f(x_i(t_{0,i})) + f(p_i)} . \tag{5}$$

It is a relative improvement, always defined, for if the denominator were equal to zero, it would mean a solution has already been found, and the algorithm already stopped.

3) Improvement threshold

The threshold "enough improvement" Δ is first computed as follows. After the (random) swarm initialization, the worst f value and the best f value for the whole swarm are found. Then the initial threshold is defined by

$$D = 1 - \frac{f_{best}}{f_{worst}} . \tag{6}$$

After that, during the process, it is updated by

$$\Delta := \Delta \left(2 - \frac{1}{e^N} \right) \tag{7}$$

each time a particle is removed, and by

$$\Delta := \frac{\Delta}{\left(2 - \frac{1}{e^N} \right)} \tag{8}$$

each time a particle is generated. The underlying idea is that when there has been enough improvement, this threshold can be increased, and vice versa. Also, the smaller the swarm, the easier it is to increase the improvement threshold, that is to say the less probable it is that some particles will be removed during the next step, and vice versa. Note that, for simplicity, we use here just one global threshold, although we can have one for each neighborhood.

4) Best neighbor, worst neighbor

The best particle $P_{best,i}$ in the neighborhood of the particle P_i is "better than" (order relation $\succ$) any other in this neighborhood. More precisely, we have

$$P_{best,i} \in \left\{ P_k, P_k \in h_i, \forall P_l \in h_i, P_k \succ P_l \right\} . \tag{9}$$

There may be several particles like that, and in case of a tie, it is chosen at random among them. For the worst particle $P_{worst,i}$, we have a similar definition

$$P_{worst,i} \in \left\{ P_k, P_k \in h_i, \forall P_l \in h_i, P_l \succ P_k \right\} . \tag{10}$$

B. Mathematical formulations for reasoning

1) Swarm size (Rule 1 and Rule 1')

The qualitative reasoning for suicide can now be formulated as follows

$$P_i = P_{worst,i}\ \delta(P_{best,i}) \geq \Delta \Rightarrow \text{remove } P_i \tag{11}$$

Similarly, the formula for generation is

$$P_i = P_{best,i}\ \delta(P_{best,i}) < \Delta \Rightarrow \text{add a particle} \tag{12}$$

Note 1 – There are a lot of ways to add a new particle, randomly or semi randomly (guided generation). As it is not the purpose of this paper to study them, in the examples below we use only pure random generation inside the search space *H*.

Note 2 – When a particle is removed or added, some neighbor lists have to be redefined.

Note 3 – There is no rule for the initial swarm size. You can perfectly begin with just one particle, but, experimentally, three seems to be a better choice.

Note 4 – Theoretically, you do not need to define a minimum swarm size. Nevertheless, in some cases, the swarm disappears completely before finding a solution, so, in practice, a minimum value is indeed given (three, in the examples below). If the swarm size is equal to this value, no suicide is permitted. Also we do not need to define a maximum swarm size but in practice we might have to, considering the limitations of our computer. When this maximum is reached no generation is permitted.

2) Coefficients (Rule 2 and Rule 2')

We have to define how to modify ϕ_i in a given interval $[\phi_{\min}, \phi_{\max}]$, knowing its current value, the improvement of the particle $\delta(P_i)$ and the threshold Δ. As initial value for each particle, we simply choose $(\phi_{\min}+\phi_{\max})/2$, and then, as $\delta(P_i)-\Delta$ is theoretically in $]-\infty, 1]$, we use the following formulas, which are consistent with Rule 2 and Rule 2' and ensure that ϕ_i is always between $\phi_{\min}$ and $\phi_{\max}$:

$$\begin{aligned} &m_i = \delta(P_i) - \Delta \\ &m_i \geq 0 \Rightarrow \delta\phi = (\phi_{\max} - \phi_i)\, m_i \\ &m_i < 0 \Rightarrow \lambda = (\phi_{\max} - \phi_i)/(\phi_i - \phi_{\min}) \\ &\delta\phi = (\phi_i - \phi_{\min})\left((1-m_i)^{-\lambda} - 1\right) \end{aligned} \tag{13}$$

3) Neighborhood sizes (Rule 3 and Rule 3')

Let $\delta|h_i|$ be an incremental modification of the particle P_i's neighborhood size. A possible set of formulas for Rule 3 and Rule 3' is

$$\begin{aligned} &\delta(P_i) \geq \Delta \Rightarrow \delta|hi| := \delta|hi| - \frac{\delta|h_i| - 1}{N-1} \\ &\delta(P_i) < \Delta \Rightarrow \delta|hi| := \delta|h_i| + \frac{N - \delta|h_i|}{N-1} \end{aligned} \tag{14}$$

These formulas are the simplest ones (linear) that assure the neighborhood size is at least equal to 1 (the particle itself) and at most equal to *N* (the whole swarm). Note that the size being an integer, such modifications have a real effect only when their "accumulation" is greater than (or equal to) 1 (in absolute value). Then we have

$$\delta|h_j| \geq 1 \Rightarrow h_j := h_j + 1,\ \delta|h_j| := 0 \quad \text{and}$$
$$\delta|h_j| \leq -1 \Rightarrow h_j := h_j - 1,\ \delta|h_j| := 0\ . \tag{15}$$

IV. Illustration

We now have all we need to really run an adaptive PSO, which we will call APSO 1''. Below we give some examples. The first one is quite detailed, in order to examine the behavior of the swarm; the others to show the global performances. For comparison we use PSO Type 1'', with ϕ value equal to 4.1, a swarm size equal to 20, and a neighborhood size equal to three for all particles. Note that these parameters have been manually chosen after thousands of trials, in order to obtain good results on most of classical test functions.

For APSO 1'', we define a minimum swarm size equal to three, a minimum neighborhood size equal to three, no maximum for the swarm size, and, for all ϕ_i, the interval $[4, 4.2]$. Note that, from a theoretical point of view, this last choice is quite arbitrary and, thus, quite frustrating, for it is just a rule of thumb. In all the examples where we run PSO Type 1'', we can see (no rigorous proof yet) that there is indeed an optimal value near to 4.1.

A. A detailed example

Let us study first in detail a well-known small example: the Rosenbrock's /Banana function on $[-10,10]^2$.

To see the difference between the two approaches, we can look at the evolution of some global descriptors: the swarm energies.

The (relative) kinetic energy of the particle P_i is defined by

$$e_{k,i}(t) = \frac{1}{2}\sum_{d=1}^{D}\left(\frac{v_{i,d}(t)}{x_{\max}(d) - x_{\min}(d)}\right)^2, \tag{16}$$

where $x_{\max}(d)$ (resp. $x_{\min}(d)$) is the maximum (resp.

minimum) for the d[th] component of any position inside the search space.

The kinetic energy of the whole swarm is then

$$E_k(t)=\sum_{i=1}^{N} e_{k,i}(t) . \quad (17)$$

The potential energy of the particle is defined by

$$e_{p,i}(t)=\frac{f(x_i(t))}{\varepsilon} . \quad (18)$$

It represents the number of energy steps the particle has to go down to reach a solution point. The potential energy of the whole swarm is then

$$E_p(t)=G\sum_{i=1}^{N} e_{p,i}(t) . \quad (19)$$

Here, G is simply an arbitrary constant, chosen so that the evolution of E_p, E_k, and the swarm size can be easily shown on the same figure. In this example, G=5.10^{-14}.

As the swarm moves, energies tend to go to zero for both methods, but not so regularly for the adaptive one, for, from time to time, a new particle is added. The swarm has then still the possibility to search in a larger domain, and to find a solution more quickly. For two typical runs, Fig. 1 and Fig. 2 show energies versus the number of evaluations, instead of the number of time steps. For a constant swarm size, they are the same, but not for a variable swarm size, so we use this representation for easier comparisons. Of course, as there is some randomness, the exact number of evaluations may be different for another run, but the global "patterns" are the same. It is not globally true that an adaptive algorithm is *always* better than a non-adaptive one. For instance, in this particular case, with the non-adaptive PSO Type 1'', a constant swarm size equal to 11, a constant neighborhood size equal to 4, the result is equivalent. However, the point is that you may have to try a lot of different sets of parameters to find such a nice one, and unless you are lucky, the sum of all your trials is far worse than any single run of the adaptive method.

In Fig. 3, we can see how the neighborhood size is modified for some particles during a run. According to Rule 3, for a "good" particle the neighborhood size is decreasing. So, we can guess that particle 5 (or 17, generated later) will find a solution (it is particle 17, in fact). Note that when a new particle is generated it is numbered (labeled) by adding 1 to the largest label ever used. So, if some particles have been removed, a particle can perfectly have a label greater than the current swarm size.

In Fig. 4, we see how particle 17 globally increases its ϕ parameter during the process, even if from time to time it has to decrease it. It means it reduces its search domain and, finally, it will indeed find a solution point.

Let us now see some results with more difficult test functions. For fair comparisons, we define a "unit adaptation cost" u, proportional to the processor time related to one error function evaluation, during the iterative process. If we take $u=1$ for PSO Type 1'' and for a given test function, we always have $u>1$ for APSO 1'' (adaptation does have a cost). This rate may be decreased with better coding, but, anyway, we have to take it into account.

B. Some global results

For each example the following method has been used. First, PSO Type 1'' is launched 100 times, with 40000 function evaluations (that is to say 2000 iterations, for the swarm size is 20). Results are noted, and the mean computer time T for a run is evaluated. Second, APSO 1'' is launched 100 times, and for every launch it was permitted to run for the time T. Of course, because of the adaptation process, it means less function evaluations can be done, but thanks to the adaptation process results should be at least equivalent if not better.

For some classical functions Table I shows that it is indeed the case. Also, as we can see, the mean neighborhood size is never very different from three but the mean swarm size is really interesting. We could think that by using this mean value (more precisely the nearest integer value) as a constant swarm size with PSO Type 1'', we should find a better result. In fact, it is not necessarily the case. For example, for the Rosenbrock function, when we use a constant swarm size of 22, we find a mean value of 58.66 (instead of 48 with a swarm size of 20). As the result with APSO 1'' is far better (21.84), it means the fluctuations of the swarm size has indeed been cleverer than the use of a constant size. Finally, when we compare the number of function evaluations in PSO Type 1'' (that is, 40000) to the mean number of evaluations, we have an estimate of the "unit adaptation cost." As we can see, even though this rate is a bit high for the Ackley example (1.82), the result justifies this extra cost (mean result of 0.03x10^{-8} instead of 1.75x10^{-8} for the same processor time).

An interesting question is to know whether the three kind of adaptations (swarm size, neighborhood sizes, coefficients ϕ_i) are all useful. A lot of tests have been done, keeping one or two of them as constant, with different values and it appears that the conjunction of the three adaptations is indeed most of the time better than any other combination. An example is shown in Table II when one parameter is kept constant. There are only two cases where it would have better to keep either the φ_i parameters or the neighborhood sizes constant. Remember, though, that the constant values used here have been specifically tuned to give good results with these kind of functions.

V. Conclusion and Future Work

In the above, we showed that an adaptive method for PSO might be a very relevant and strong idea. In the non-adaptive method some parameters may need to be estimated after a lot of trial runs, which can be difficult. On

the other hand, the adaptive method does not constrain the user in any way; the user only has to describe the problem; and the results are equivalent or often better in the adaptive method.

However, the implementation example is not completely satisfying, for at least three reasons: starting point is just a particular non-adaptive PSO, the adaptation formulas may be too simple, and, importantly, there is still a parameter (ϕ) whose values need to be arbitrarily confined within a small interval, without any mathematical justification. The first two problems are mainly technical ones, and thanks to the general framework defined here, that we also proved to be efficient, it would not be that difficult to study them. In particular it would be interesting to see if an adaptive deterministic model could be at least as good as the classical non-adaptive ones, which use randomness. The third problem is more difficult and clearly needs a theoretical breakthrough.

Also, studying the adaptive process, it appears that the best description level to understand the behavior of the swarm is probably not the particle but what we have called the "knowable group." It would certainly be interesting to examine how knowable groups exchange information as soon as they have some particles in common, and how they decide to increase or decrease, speed up or slow down.

REFERENCES

[1] R. C. Eberhart and J. Kennedy, "A New Optimizer Using Particle Swarm Theory," presented at Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995.

[2] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," presented at IEEE International Conference on Neural Networks, Perth, Australia, 1995.

[3] P. J. Angeline, "Using Selection to Improve Particle Swarm Optimization," presented at IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4-9, 1998.

[4] A. Carlisle and G. Dozier, "Adapting Particle Swarm Optimization to Dynamics Environments," presented at International Conference on Artificial Intelligence, Monte Carlo Resort, Las Vegas, Nevada, USA, 1998.

[5] M. Clerc, "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization," presented at Congress on Evolutionary Computation, Washington DC, 1999.

[6] J. Kennedy, "Stereotyping: Improving Particle Swarm Performance With Cluster Analysis," presented at Congress on Evolutionary Computation, 2000.

[7] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," presented at International Congress on Evolutionary Computation, San Diego, California, 2000.

[8] R. C. Eberhart and Y. Shi, "Evolving artificial neural networks," presented at International Conference on Neural Networks and Brain, Beijing, 1998.

[9] S. Naka and Y. Fukuyama, "Practical Distribution State Estimation Using Hybrid Particle Swarm Optimization," presented at IEEE Power Engineering Society Winter Meeting, Columbus, Ohio, USA, 2001.

[10] M. Clerc and J. Kennedy, "The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional Complex Space," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 58-73, 2002.

[11] Y. Shi and R. C. Eberhart, "Parameter Selection in Particle Swarm Optimization," presented at Evolutionary Programming VII, 1998.

[12] F. Van den Bergh and A. P. Engelbrecht, "Effects of Swarm Size on Cooperative Particle Swarm Optimisers," presented at GECCO 2001, San Francisco, USA, 2001.

[13] J. Kennedy, "Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance," presented at Congress on Evolutionary Computation, Washington D.C., 1999.

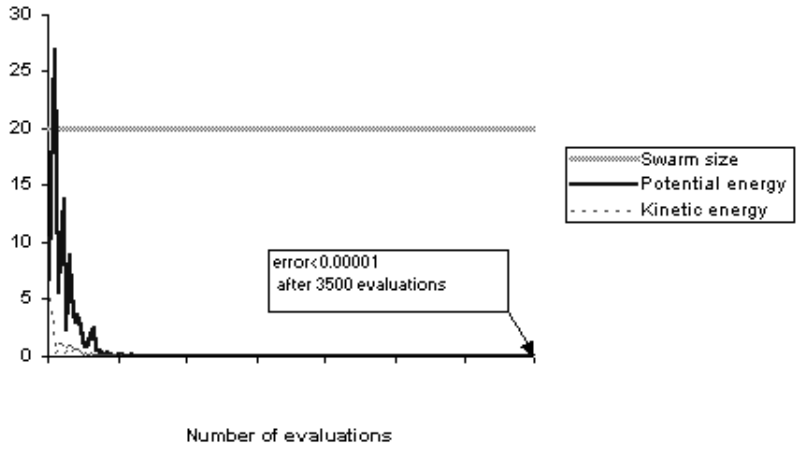


Fig. 1. ROSENBROCK 2D WITH PSO TYPE 1''. Energy evolution for a typical run. Near the solution, as kinetic energy can not appreciably re-increase, improvement is difficult.

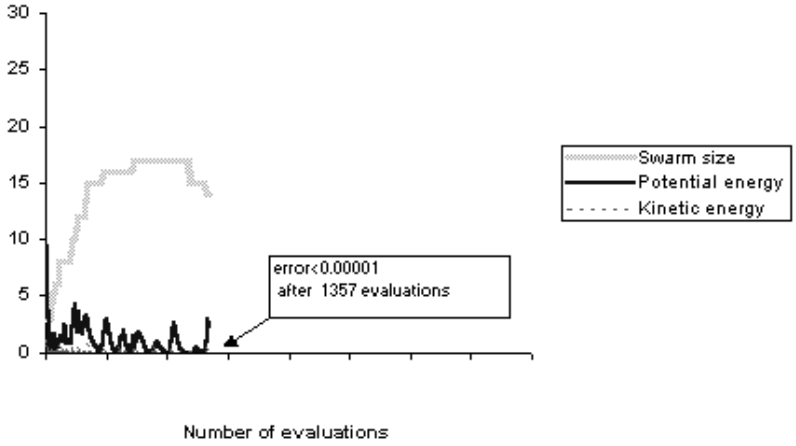


Fig. 2. ROSENBROCK 2D WITH APSO 1''. Energy evolution for a typical run. When the swarm does not find a solution, it globally re-increases its energy by adding new particles, that is to say its ability to explore the search space. And when the process seems to converge nicely, it decreases it by removing "bad" particles.

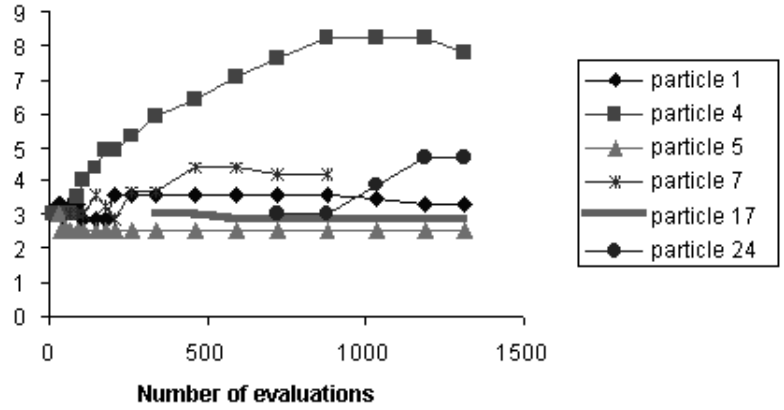
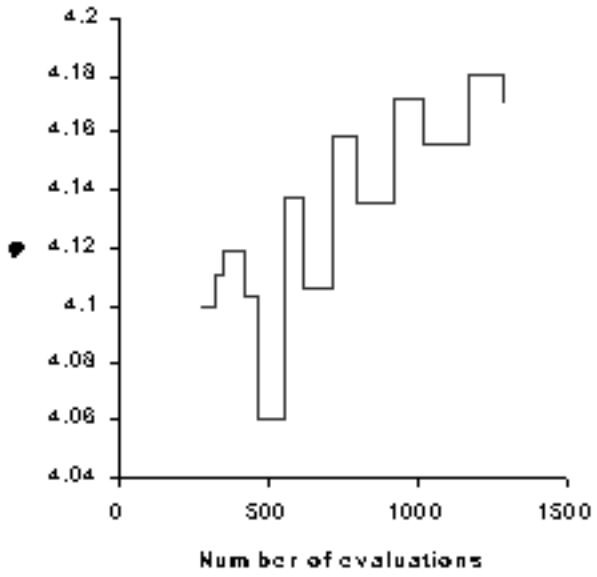


Fig. 3. ROSENBROCK 2D WITH APSO 1''. Evolution of the neighborhood size of some particles during a typical run. A particle which does not need to increase its neighborhood, like 5 or 17 (which has been generated after about 400 evaluations), is necessarily a "good" one. A solution will be found by particle 17.



.
Fig. 4. φ VALUE EVOLUTION. Particle 17 reaches a solution point first for it succeeds to globally increase its φ coefficient, i.e. decrease its search domain.

Table I. NON-ADAPTIVE VS ADAPTIVE. This table shows the results of PSO Type 1'' after 40000 evaluations, and comparison with APSO 1'' running for the same processor time. For each function the global minimum is zero. The unit cost is equal to 40000/(number of evaluations for APSO 1''). The adaptive version is significantly better than the non-adaptive one

		PSO Type 1'' 100 runs of 40000 evaluations		APSO 1'' 100 runs of time T				
Function	Search space	Result (standard deviation)	Processor time units T	Result (standard deviation)	Mean swarm size	Mean neighb. size	Mean number of evaluations	Adaptation cost
Ackley	$[-32,32]^{30}$	$1.75\text{x}10^{-8}$ ($1.86\text{x}10^{-8}$)	1156	$0.03\text{x}10^{-8}$ ($0.05\text{x}10^{-8}$)	12.83	3.35	21921	1.82
Griewank	$[-300,300]^{30}$	0.008 (0.016)	1278	0.006 (0.010)	21.96	3.32	30054	1.33
Rastrigin	$[-5.12,5.12]^{30}$	79.31 (19.01)	1203	51.35 (16.80)	30.98	3.18	32992	1.21
Rosenbrock	$[-10,10]^{30}$	48.00 (32.02)	1147	23.38 (2.74)	21.84	3.32	31669	1.26

Table II. ALL THREE ADAPTATIONS ARE USEFUL. When one parameter is kept constant, on the same examples as in Table 1, the results are globally not so good (mean after 100 runs). We have a better result just for the Griewank function with a constant neighborhood size equal to 3, and the Rastrigin function with a constant φ equal to 4.1.

Function	$N=20$	$\phi=4.1$	$\lvert h_i\rvert=3$
Ackley	$386.9\text{x}10^{-8}$	$56511.7\text{x}10^{-8}$	$2789.8\text{x}10^{-8}$
Griewank	0.026	0.006	**0.0048**
Rastrigin	84.51	**40.03**	51.62
Rosenbrock	48.36	37.411	54.31