UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo — Trento (Italy), Via Sommarive 14
http://dit.unitn.it/

DO NOT BE AFRAID OF LOCAL MINIMA: AFFINE SHAKER AND PARTICLE SWARM

Roberto Battiti, Mauro Brunato and Srinivas Pasupuleti

May 2005

Technical Report # DIT-05-049

Do not be afraid of local minima: Affine Shaker and Particle Swarm

Roberto Battiti Mauro Brunato
Srinivas Pasupuleti

Department of Computer Science and Telecommunications, Università di Trento
Via Sommarive 14, 38050 Povo, Trento, Italy
{battiti,brunato}@dit.unitn.it

May 2005

Abstract

Stochastic local search techniques are powerful and flexible methods to optimize difficult functions. While each method is characterized by search trajectories produced through a randomized selection of the next step, a notable difference is caused by the interaction of different searchers, as exemplified by the Particle Swarm methods. In this paper we evaluate two extreme approaches, Particle Swarm Optimization, with interaction between the individual "cognitive" component and the "social" knowledge, and Repeated Affine Shaker, without any interaction between searchers but with an aggressive capability of scouting out local minima. The results, unexpected to the authors, show that Affine Shaker provides remarkably efficient and effective results when compared with PSO, while the advantage of Particle Swarm is visible only for functions with a very regular structure of the local minima leading to the global optimum and only for specific experimental conditions.

1 Introduction

The problem being addressed is that of *minimizing* a function mapping a vector $\boldsymbol{x} \in D \subseteq \mathbb{R}^d$ of d real numbers into a real number,

$$\begin{aligned} \min_{\boldsymbol{x}\in D} \quad & f(\boldsymbol{x}) \\ \text{where} \quad & f : D \subseteq \mathbb{R}^d \to \mathbb{R} \end{aligned}$$

where d is the dimension of the domain. Furthermore, the domain of the function, i.e. the search domain, is assumed to be a limited subset of $\mathbb{R}^d$. The

minimum is usually searched for within a d-dimensional interval, where coordinate j is delimited by a lower bound L_j and an upper bound U_j:

$$D = [L_1, U_1] \times [L_2, U_2] \times \cdots \times [L_d, U_d],$$

where we assume that a minimum exists[1].

2 Particle swarm optimization

Particle Swarm Optimization [9], PSO for short, is a population-based stochastic optimization technique for optimizing complex functions through the *interaction of individuals* in a population of particles. Let's define the notation. The parameter d is the dimensionality of the search space, P is the total number of particles, and the position of each individual "particle" $i \in \{1, 2, \ldots, P\}$ is described by a vector $\boldsymbol{x}_i \equiv (x_{i1}, x_{i2}, \ldots, x_{id})$. Each particle follows a trajectory $\boldsymbol{x}_i(t)$ in the search space, $t \geq 0$ being the discrete iteration counter. The best previous position (the position giving the best function value found) of the i-th particle up to time t is referred to as $\boldsymbol{p}_i(t)$ and the globally best position among all particles in the population up to time t is called $\boldsymbol{g}(t)$. More formally, let $\sigma_i(t) \leq t$ be the iteration at which particle i found its personal minimum (if the same value was found at many iterations, then we take the first one):

$$\begin{cases} \sigma_i(t) \leq t & \\ \forall s \leq t & f\Big(\boldsymbol{x}_i\big(\sigma_i(t)\big)\Big) \leq f\big(\boldsymbol{x}_i(s)\big) \\ \forall s & f\Big(\boldsymbol{x}_i\big(\sigma_i(t)\big)\Big) = f\big(\boldsymbol{x}_i(s)\big) \Rightarrow s \geq \sigma_i(t). \end{cases}$$

Then the best previous position at time t for particle i is

$$\boldsymbol{p}_i(t) = \boldsymbol{x}_i\big(\sigma_i(t)\big), \qquad i = 1, \ldots, P. \tag{1}$$

Similarly, let $I(t)$ be the index of the particle which found the global best at time t (the smallest such index, in case of ties):

$$\begin{cases} I(t) \in \{1, \ldots, P\} & \\ \forall i = 1, \ldots, P & f(\boldsymbol{p}_{I(t)}(t)) \leq f(\boldsymbol{p}_i(t)) \\ \forall i = 1, \ldots, P & f(\boldsymbol{p}_{I(t)}(t)) = f(\boldsymbol{p}_i(t)) \Rightarrow i \geq I(t). \end{cases}$$

Then the global best coordinates at time t are

$$\boldsymbol{g}(t) = \boldsymbol{p}_{I(t)}(t). \tag{2}$$

In the following description, the components of the vector are indexed by j. The particle positions are initialized by picking points at random and with

[1]Note that if f is continuous then the assumption is trivial; however, even if a minimum does not exist, the problem of computationally locating a near-infimum value of the function is still well defined.

uniform probability in an initialization range delimited by a lower bound L'_j and an upper bound U'_j: $x_{ij} \in [L'_j, U'_j]$. Such initialization range is always contained in the search range: $L_j \leq L'_j < U'_j \leq U_j$. In the actual usage of the technique, if the function structure and the position of the global optimum are unknown, the initialization range coincides with the entire search range. In the tests on the benchmark functions (where for example the position of the global optimum is known to the researchers) the initialization range will be a proper subset the search range to avoid potential biases caused, for example, by symmetric initialization around a global minimum, as explained in Section 4.

Let $\boldsymbol{\Delta}_i(t) \equiv \big(\delta_{i1}(t), \ldots, \delta_{id}(t)\big)$ be the "velocity" of particle i at iteration t. The initial velocity $\boldsymbol{\Delta}_i(0)$ of particle i is chosen with uniform probability within suitable limits: $\delta_{ij}(0) = \mathrm{rand}(-D_j, D_j)$. The use of the term "velocity" and the choice of the limits will become clear in the following explanation. The discrete dynamical system governing the motion of each particle for $t \geq 1$ is the following [14]:

$$\begin{aligned} \boldsymbol{\Delta}_i(t) &= w\boldsymbol{\Delta}_i(t-1) + c_1\,\mathbf{Rand}_d(0,1)\cdot(\boldsymbol{p}_i(t-1) - \boldsymbol{x}_i(t-1)) \\ &\quad + c_2\,\mathbf{Rand}_d(0,1)\cdot(\boldsymbol{g}(t-1) - \boldsymbol{x}_i(t-1)) \qquad (3) \\ \boldsymbol{x}_i(t) &= \boldsymbol{x}_i(t-1) + \boldsymbol{\Delta}_i(t) \qquad (4) \end{aligned}$$

where $\boldsymbol{\Delta}_i(t)$ is the step in the search space executed at iteration t, w is the *inertia weight* [12, 13], c_1 and c_2 are two positive constants, and $\mathbf{Rand}_d(0,1)$ is a $d \times d$ diagonal matrix of independent random numbers in the range $[0,1]$. Thus, each component of the position vectors is multiplied by a different random number.

An analogy from astronomy for PSO is that of a planet $\boldsymbol{x}_i$ orbiting around two "suns" given by the current best individual position (the "cognitive" component) and the current best global position (the "social" component). The analogy becomes clear if $\boldsymbol{\Delta}_i(t)$ is interpreted as the velocity of a particle and w is equal to 1. In this case the position of the suns determines the velocity variation, i.e., the acceleration, as forces in Physics do. Let's note that the suns are of course changing in time, in some cases abruptly, as soon as a new best position is found either for a specific particle or for the entire swarm. A careful choice of the parameters determines the dynamics of the planets, to avoid that they get burned by the suns too early (*premature convergence*) or that they are spread in distant space (*explosion*). A detailed study of the dynamical system underlying PSO is present in [6]. The authors consider both a simplified deterministic dynamical system and the full stochastic system consisting of a single trajectory, with the assumption that the individual and social influence terms are fixed (i.,e., in the above analogy, that the two suns guiding the motion of the planet have a fixed position). Furthermore, no interaction among different trajectories is considered in the analysis. For the single-particle case, in the above assumptions, the analysis in fact eliminates problem-specific parameters to be fixed by the user.

While the assumptions are quite radical, nonetheless the analysis suggests

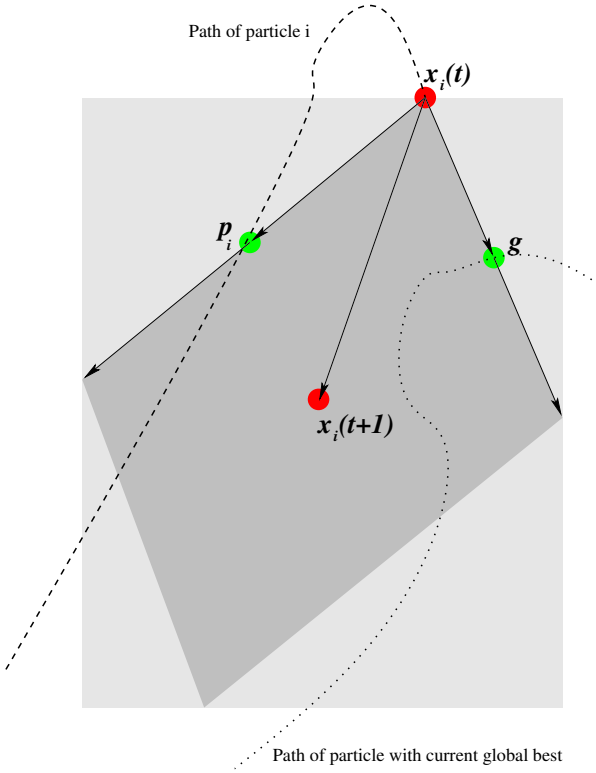

Figure 1: Particle Swarm geometry: randomized selection of the next point, based on "attraction" by the individual best $\mathbf{p_i}$ and global best $\mathbf{p_g}$. Because each component of the two vectors is multiplied by an independent random number, the final vector can also exit the darker gray parallelogram, being only limited by the light gray rectangle. A portion of the previous step $w\mathbf{\Delta}_i(t-1)$ is finally added through the inertia mechanism.

values for the algorithm parameters (like c_1, c_2 and w), whose judicious application can avoid the swarm explosion effect related to the deleterious effects of randomness[2]. Moreover, with suitable parameter values, explosion can be prevented while still allowing for exploration, therefore avoiding a "premature convergence" of the search.

The geometry of the position-updating equations (3) and (4) is illustrated in Fig. 1.

The specific PSO version that we consider is the one proposed in [14], where

[2] "the random weighting of the control parameters [...] results in a kind of explosion or a drunkard's walk as particle velocities and positional coordinates careen towards infinity" [6]

the inertia weight w is linearly varying in time as:

$$w = w_1 + (w_2 - w_1) \cdot \frac{t}{\text{MAXITER}} \tag{5}$$

w_1 and w_2 are the initial and final values of the inertia weight, respectively, t is the current iteration number and MAXITER is the maximum number of allowable iterations Following [11], we shall refer to this PSO variant as PSO-TVIW (time-variant inertia weight). This evolution in time is reminiscent of the use of an annealing schedule in the Simulated Annealing algorithm [10], where the temperature parameter is gradually reduced during the search. The pseudo-code of PSO algorithm is shown in Fig. 2.

The modulus of each component of the velocity vector of a particle ($\delta_{ij}(t)$) is limited by the maximum allowable velocity along that dimension, D_j. In the experiment it is set to half the maximum range: $D_j = (U_j - L_j)/2$, the same choice used in many papers using the same functions for comparing algorithms, in particular [11].

According to [7], by limiting the distance that can be covered in one time step to the largest dimension of the system, one avoids a possible *explosion* of the swarm and makes the system more robust with respect to the choice of the parameters governing its dynamics.

The step coordinates are therefore reset to the maximum allowable modulus when they exceed the limits during the iterations. In the pseudo-code, this check and possible truncation of the step is executed by the function "limit ($\mathbf{\Delta}_i$, (D_j))" at line 18 of Fig. 2 limiting the components of vector $\mathbf{\Delta}_i \equiv (\delta_{i1}, \ldots, \delta_{id})$ as follows:

$$\delta_{ij} \longleftarrow \begin{cases} -D_j & \text{if } \delta_{ij} < -D_j \\ D_j & \text{if } \delta_{ij} > D_j \\ \delta_{ij} & \text{otherwise.} \end{cases}$$

For additional possibilities to control the explosion as well as the convergence of the particles we refer the reader to the detailed analysis of [6].

3 Affine Shaker Algorithm

The Affine Shaker algorithm (or AS for short) originally proposed in [4] is an adaptive random search algorithm based only on the knowledge of function values. The term "shaker" derives from the brisk movements of the search trajectory of the stochastic local minimizer, while the term "affine" derives from the affine transformation executed on the local search region considered for generating the next point along the search trajectory.

The algorithm starts by choosing an initial point $\boldsymbol{x}$ in the configuration space and an initial search region $\mathcal{R}$ surrounding it, see Fig. 4. It proceeds by iterating the following steps:

1. A new tentative point is generated by sampling the local search region $\mathcal{R}$ with a uniform probability distribution.

Variable	Scope	Meaning
w_1, w_2, c_1, c_2	(input)	Parameters defined in the text
P	(input)	Number of particles in the swarm
d	(input)	Dimension of the space
f	(input)	Function to minimize
MAXITER	(input)	Maximum number of iterations
$D_1, \ldots, D_d$	(input)	Speed limit vector
$L'_1, \ldots, L'_d, U'_1, \ldots, U'_d$	(input)	Initialization range
t	(internal)	Iteration counter
$\boldsymbol{x}$, $\boldsymbol{\Delta}$, $\boldsymbol{p}$, $\boldsymbol{g}$	(internal)	status vectors, defined in the text
currentvalue	(internal)	Function value at the current point
bestvalue	(internal)	Per-particle best value array
globalbest	(internal)	Global best value

```
1.  function PSO_TVIW (f, (L'_j), (U'_j), (D_j), P, w_1, w_2, c_1, c_2, MAXITER)
2.      t ← 0;
3.      globalbest ← +∞;
4.      for i ← 1 ... P
5.          x_i ← initial random point in [L'_1, U'_1] × ··· × [L'_d, U'_d];
6.          Δ_i ← initial random velocity;
7.          p_i ← x_i;
8.          currentvalue ← f( x_i);
9.          bestvalue_i ← currentvalue;
10.         if currentvalue < globalbest
11.             globalbest ← currentvalue;
12.             g ← x_i;
13.     while t ≤ MAXITER
14.         t ← t+1;
15.         w ← w_1 + (w_2 − w_1) t / MAXITER;
16.         for i ← 1 ... P
17.             Δ_i ← wΔ_i + c_1 Rand_d ·(p_i − x_i) + c_2 Rand_d ·(g − x_i);
18.             limit ( Δ_i, (D_j));
19.             x_ij ← x_ij + δ_ij;
20.             currentvalue ← f( x_i);
21.             if currentvalue < bestvalue_i
22.                 p_i ← x_i;
23.                 bestvalue_i ← currentvalue;
24.                 if currentvalue < globalbest
25.                     globalbest ← currentvalue;
26.                     g ← x_i;
27.     return g;
```

Figure 2: The Particle Swarm algorithm

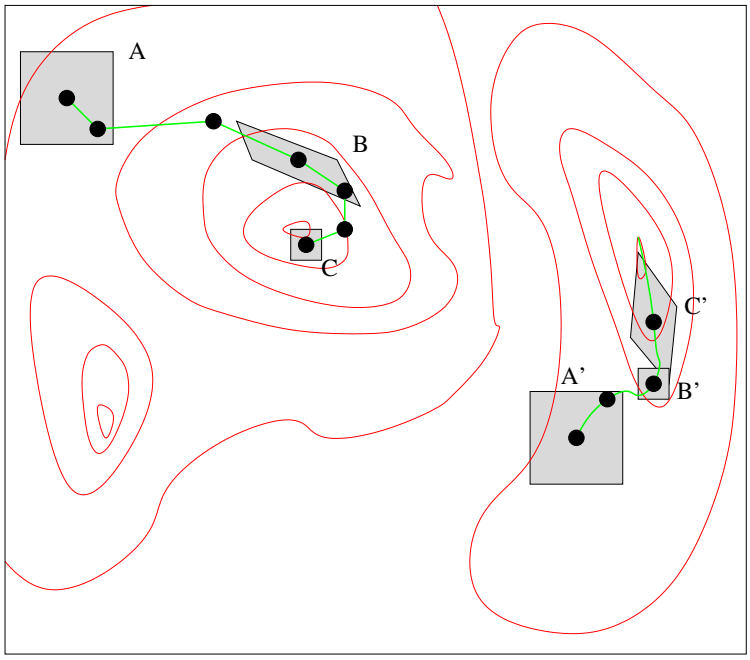

Figure 3: Affine Shaker geometry: two search trajectories leading to two different local minima. The evolution of the search regions is also illustrated.

2. The search region is modified according to the value of the function at the new point. It is compressed if the new function value is greater than the current one (unsuccessful sample) or expanded otherwise (successful sample).

3. If the sample is successful, the new point becomes the current point, and the search region $\mathcal{R}$ is translated so that the current point is at its center for the next iteration.

The geometry of the Affine Shaker algorithm is illustrated in Fig. 3, where the function to be minimized is represented by a contour plot showing *isolines* at fixed values of f, and two trajectories (ABC and A'B'C') are plotted. The pseudo-code of the algorithm is illustrated in Fig. 4. To obtain a fast and effective implementation of the above described framework it is sufficient to use a simple search region around the current point defined by edges given by a set of linearly independent vectors $\mathcal{R} \equiv (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$, and to generate a candidate point through a uniform probability distributions inside the box. In Figure 3 search regions (in grey shade) are shown for some points in the trajectory; being in two dimensions, a couple of independent vectors defines a parallelogram centered on the point. Generating a random displacement is simple: the basis

vectors are multiplied by random numbers in the real range $[-1, 1]$ and added: $\boldsymbol{\Delta} = \sum_j \text{rand}(-1, 1)\boldsymbol{b}_j$.

An improvement of the basic algorithm is obtained by using the *double-shot strategy*: if the first sample $\boldsymbol{x} + \boldsymbol{\Delta}$ is not successful, the specular point $\boldsymbol{x} - \boldsymbol{\Delta}$ is considered. This choice drastically reduces the probability of generating two consecutive unsuccessful samples. The motivation is clear if one considers differentiable functions and small displacements: in this case the directional derivative along the displacement is proportional to the scalar product between displacement and gradient $\boldsymbol{\Delta} \cdot \nabla f$. If the first is positive, a change of sign will trivially cause a negative value, and therefore a decrease in f for a sufficiently small step size. The empirical validity for general functions (not necessarily differentiable) is caused by the correlations and structure contained in most of the functions corresponding to real-world problems.

It is of interest to compare the method used for generating a new point along the trajectory with the one used in Particle Swarm. In both cases the new point is generated with uniform probability in a search frame, but in AS the frame is centered on the current point, which coincides with the best point $\boldsymbol{g}$ encountered during the search. In fact, the current point moves only if a lower function value if found. When a new best point is found the search frame is centered on the new point. There is no "social" component because a single searcher is active. In addition, the volume of the search frame is dynamically varied depending on the success of the last tentative move (i.e., volume becomes bigger if the last tentative move lead to a new best value, smaller otherwise), and the preferred direction of search is determined depending on the direction of the steps executed.

The design criteria are given by an *aggressive search for local minima*: the search speed is increased when steps are successful (points A and A' in Figure 3), reduced only if no better point is found after the double shot. When a point is close to a local minimum, the repeated halving of the search frame produces a very fast convergence of the search (point C in Figure 3). Note that another cause of reduction for the search region can be a narrow descent path (a "canyon", such as in point B' of Figure 3, where only a small subset of all possible directions improves the function value); however, once an improvement is found, the search region grows in the promising direction, causing a faster movement along that direction. A single run of AS run is terminated if for a predefined number S of consecutive steps one has $\| \boldsymbol{\Delta} \| < \epsilon$. A sequence of S consecutive steps is considered before termination to reduce the probability that, by chance, a small step is executed because of the randomized selection and not because the point is close to a local minimum. The number of steps is set to $S = 8$ in the tests. The ϵ value is the precision with which the user wants to determine the position of a local minimum. The default value $\epsilon = 10^{-6}$ has been used in the tests.

By design, AS searches for local minimizers and is stopped as soon as one is found. A simple way to continue the search after a minimizer is found is to restart from a different initial random point, leading to the Repeated Affine Shaker algorithm described in Fig. 5. This leads to a "population" of AS

Variable	Scope	Meaning
f	(input)	Function to minimize
$\boldsymbol{x}$	(input)	Initial point
$\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$	(input)	Vectors defining search region $\mathcal{R}$ around $\boldsymbol{x}$
ρ_e, ρ_r	(input)	Box expansion and reduction factors
d	(input)	Dimension of the space
t	(internal)	Iteration counter
$\mathbf{P}$	(internal)	Transormation matrix
$\boldsymbol{x}$, $\boldsymbol{\Delta}$	(internal)	Current position, current displacement

1. **function AffineShaker** (f, $\boldsymbol{x}$, ($\boldsymbol{b}_j$), ρ_e, ρ_r)
2. $t \leftarrow 0$;
3. **repeat**
4. $\boldsymbol{\Delta} \leftarrow \sum_j \mathrm{rand}(-1, 1)\boldsymbol{b}_j$;
5. **if** $f(\ \boldsymbol{x}+\ \boldsymbol{\Delta}) < f(\ \boldsymbol{x})$
6. $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{\Delta}$;
7. $\mathbf{P} \leftarrow \mathbf{I} + (\rho_e - 1)\dfrac{\boldsymbol{\Delta}\boldsymbol{\Delta}^T}{\| \boldsymbol{\Delta} \|^2}$;
8. **else if** $f(\ \boldsymbol{x}\text{- }\boldsymbol{\Delta}) < f(\ \boldsymbol{x})$
9. $\boldsymbol{x} \leftarrow \boldsymbol{x}$ - $\boldsymbol{\Delta}$;
10. $\mathbf{P} \leftarrow \mathbf{I} + (\rho_e - 1)\dfrac{\boldsymbol{\Delta}\boldsymbol{\Delta}^T}{\| \boldsymbol{\Delta} \|^2}$;
11. **else**
12. $\mathbf{P} \leftarrow \mathbf{I} + (\rho_c - 1)\dfrac{\boldsymbol{\Delta}\boldsymbol{\Delta}^T}{\| \boldsymbol{\Delta} \|^2}$;
13. $\forall j\ \boldsymbol{b}_j \leftarrow \mathbf{P}\ \boldsymbol{b}_j$;
14. $t \leftarrow t{+}1$
15. **until** convergence criterion;
16. **return** $\boldsymbol{x}$;

Figure 4: The Affine Shaker algorithm

Variable	Scope	Meaning
f	(input)	Function to minimize
ρ_e, ρ_r	(input)	Box expansion and reduction factors
$L_1, \ldots, L_d, U_1, \ldots, U_d$	(input)	Search range
$L'_1, \ldots, L'_d, U'_1, \ldots, U'_d$	(input)	Initialization range
d	(input)	Dimension of the space
$\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$	(internal)	Vectors defining search region $\mathcal{R}$ around $\boldsymbol{x}$
t	(internal)	Iteration counter
$\boldsymbol{x}$, $\boldsymbol{x}'$	(internal)	Current position, final position of run

1. **function** RepeatedAffineShaker (f, ρ_e, ρ_r, (L'_j), (U'_j), (L_j), (U_j))
2. $\quad \forall j\ \boldsymbol{b}_j \leftarrow \dfrac{U_j - L_j}{4} \cdot \boldsymbol{e}_j$;
3. $\quad$ **repeat**
4. $\quad\quad \boldsymbol{x} \leftarrow$ random point $\in [L'_1, U'_1] \times \cdots \times [L'_d, U'_d]$;
5. $\quad\quad \boldsymbol{x}$' $\leftarrow$ AffineShaker(f, $\boldsymbol{x}$, ($\boldsymbol{b}_j$), ρ_e, ρ_r);
6. $\quad$ **until** termination criterion
7. $\quad$ **return** best position found;

Figure 5: The Repeated Affine Shaker algorithm

searchers, but in this case each member of the population is *independent*, completely unaware of what other members are doing. The range of initialization $[L'_1, U'_1] \times \cdots \times [L'_d, U'_d]$ used to generate initial random points in the tests is the same as that used for the PSO algorithm.

While the algorithm runs, the number of function evaluations is registered, as well as the best value found. To avoid cluttering the algorithm description, these standard bookkeeping operations are not shown in the figures.

4 Benchmark optimization problems

The benchmarks used for simulation are given in Table 1. We keep the same function names used in other recent PSO papers. All benchmark functions except the Schaffer's f_6 function, which is two-dimensional, are tested with 30 dimensions. The first two functions are unimodal functions whereas the next functions are multimodal. The global minimum value for the f_1, f_2, f_3, f_4 functions is 0.00.

The Schaffer's f_6 function is designed to have a global maximum at the origin, surrounded by circular "valleys" designed to trap methods based on local search, see Fig. 6. To maintain the original purpose of this test function, we maximize it. The same function is considered in [5] as a challenging problem for discrete optimization based on the hill-climbing local search and the Reactive Search method. As it is clear from the figure, while the local structure of f_6 is designed to trap local minima searchers, the *global structure of the local minima*

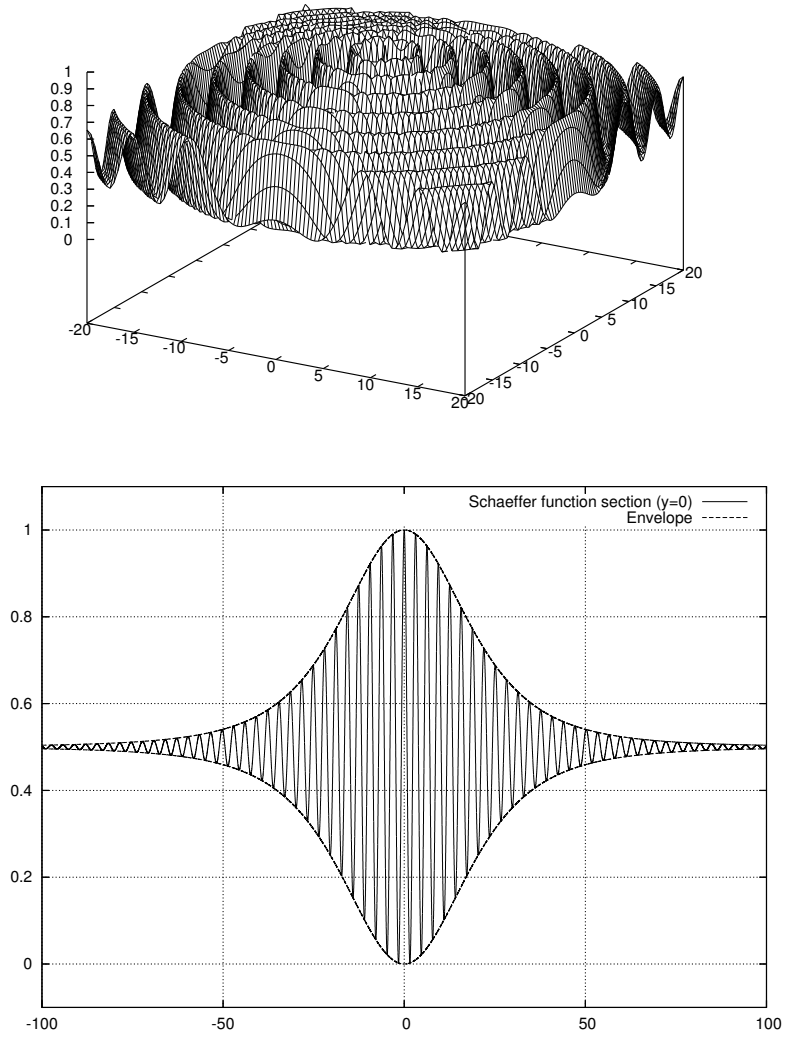

Figure 6: Schaffer f_6 function (top) and a cross-section at $y = 0$ (bottom). The needle-like maximum is at $(0, 0)$.

Table 1: Benchmarks for simulations

	Function Name	Mathematical Representation
a)	Sphere function	$f_1(x) = \sum_{i=1}^{n} x_i^2$
b)	Rosenbrock function	$f_2(x) = \sum_{i=1}^{n} \big(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\big)$
c)	Rastrigrin function	$f_3(x) = \sum_{i=1}^{n} \big(x_i^2 - 10\cos 2\pi x_i + 10\big)$
d)	Griewank function	$f_4(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\frac{x_i}{\sqrt{i}} + 1$
e)	Schaffer's f_6 function	$f_6(x) = 0.5 - \frac{(\sin\sqrt{x^2+y^2})^2 - 0.5}{(1.0 + 0.001(x^2+y^2))^2}$

(or of the local maxima) shows a very regular structure, with a "gradient" of values clearly pointing the way towards the central optimum. From the cross section (and from simple analysis of the function) one appreciates how the knowledge of more local minima creates compelling evidence about the structure and should therefore be used by methods employing a population of interacting searchers, like PSO.

The Rastrigrin (f_3) function is designed to pose similar problems to a localized minimum search algorithm. As shown in Fig. 7, it exposes a plethora of proper local minima surrounding the actual global minimum in $(0, 0)$.

For the purpose of comparison, the asymmetric initialization method used in[1, 2, 8] is adopted for initializing the population in PSO and the independent searchers in the Repeated Affine Shaker, see Table 2. The choice is motivated by [8], which shows that the typical initialization used to compare evolutionary computations can give false impressions of relative performance. In many comparative experiments, the initial population is uniformly distributed about the entire search space which is usually defined to be symmetric about the origin. In addition, many of the test functions are crafted in such a way as to have optima at or near the origin, including the test functions for this study. This method of initialization has two potential biases when considered alone. First, if an operator is an averaging operator involving multiple parents, such as intermediate crossover often used in evolution strategies, recombining parents from opposite sides of the origin will naturally place the offspring close to the center of the initialization region. Second, given that the location of the optima is generally not known, there is no guarantee that any prescribed initialization method will

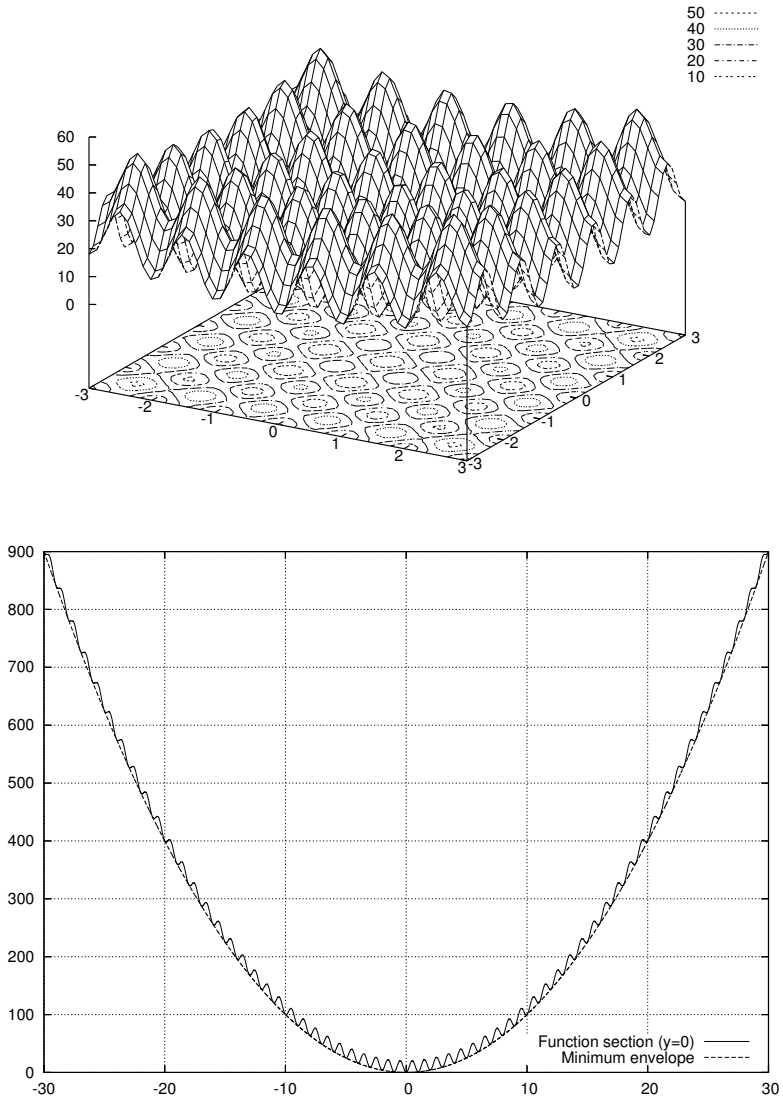

Figure 7: Rastrigrin f_3 function (top) and a cross-section at $y = 0$ (bottom). The global minimum is at $(0, 0)$.

Table 2: Search range and initialization range for the benchmark functions

Function	Search range $[L_1, U_1] \times \cdots \times [L_d, U_d]$	Initialization range $[L'_1, U'_1] \times \cdots \times [L'_d, U'_d]$
f_1	$[-100, 100]^d$	$[50, 100]^d$
f_2	$[-100, 100]^d$	$[15, 30]^d$
f_3	$[-10, 10]^d$	$[2.56, 5.12]^d$
f_4	$[-600, 600]^d$	$[300, 600]^d$
f_6	$[-100, 100]^2$	$[15, 30]^2$

include the optima. Consequently, [8] suggests initializing in regions that deliberately do not include the optima during testing to verify results obtained for symmetric initialization schemes.

When comparing the effectiveness of different optimization methods we are interested in the trade-off between computational complexity and values delivered by the technique. For the values delivered at a given iteration we consider the best value found during the previous phase of the search (the value $\boldsymbol{g}$ in Fig. 2). To measure the computational effort we make the assumption that the evaluation of the function values at the candidate points (calculation of $f(\boldsymbol{x}_i)$) is the dominating factor. This assumption is usually valid for significant optimization tasks of interest for real-world problems.

The experiments are designed to model a situation where a user needs to optimize a function, has a maximum budget of computational resources to devote to the task or, equivalently, a maximum time budget within which the best solution found has to be delivered.

The general assumption is that no clear-cut termination criterion is available. For a generic function there is no way to determine whether the global optimum has been found by the algorithm and therefore to determine when the search can be stopped. Termination is therefore dictated by exhaustion of the time/resource budget, or by obtaining a sufficiently good solution (of course a problem-specific criterion).

After evaluating the time required for a single function evaluation the maximum number of function evaluations is derived. In selecting a technique we assume the user will prefer the one that, on average, delivers the lowest value for the given number of function evaluations. The standard deviation of the results can influence the decision if multiple runs are considered, or if the risk associated to obtaining a value that is far from the average is significant.

Because the algorithms contain a stochastic component, the value $f(\boldsymbol{g})$ will evolve in different ways for different runs (of course the random number generator seed is different for each run) and both the average and error on the average (the standard deviation of the results divided by the square root of the number of tests) are reported in the tests: these data permit conclusions about the expected performance and variability of performance of the various techniques.

5 Comparison Between Particle Swarm and Affine Shaker

In the experiments, for the PSO algorithm of Fig. 2, the constants c_1 and c_2 are fixed at value 2.0 and the value of *inertia weight* w is varied from $w_1 = 0.9$ at the beginning of the search to $w_2 = 0.4$ at the end of the search.

The experiments are conducted with different population sizes $P = 10$, $P = 20$ and $P = 40$. The PSO-TVIW and Repeated Affine Shaker algorithms are run for 50 trials and the average optimum value $f(\boldsymbol{g})$ with standard deviation are measured as a function of the number of function evaluations. The error on the average (obtained by dividing the standard deviation of results by the square root of the number of tests plus one) has been used to assure the statistical significance of the results. To avoid cluttering the plots with too much information, the plots with error on the average are presented in the Appendix.

A subtle but important issue to be decided for the tests is how to fix the value of the maximum number of iterations MAXITER to derive the evolution in time of the inertia weight following equation (5). If one considers the situation encountered in the applications of a heuristic for combinatorial optimization, the user is of course not aware of the value and position of the global optimum (otherwise he would not need to search for it!). For most problems, even determining that a configuration is indeed the global optimum is not possible in polynomial time. These results have a strong theoretical foundation in the theory of computational complexity: if some problems could be solved in polynomial time, a large class of related "difficult" problems in the NP-hard class could also be solved in polynomial time, a result that is currently believed to be extremely implausible by the community of researchers in Computer Science.

The user therefore must make a reasonable allocation of computing time and be satisfied with the best solution found by the heuristic when the allotted time elapses, or repeat the search with a longer computational effort if the results are not satisfactory. In the assumption that computational effort is dominated by the number of function evaluations, for the considered PSO version this means that the user has to fix a reasonably large value of function evaluations at the beginning. The number of function evaluations in PSO is given by MAXITER $\cdot$ P. Therefore the MAXITER parameter is varied to get the performance of PSO for a fixed population size with respect to different number of function evaluations.

This problem is not present in the Repeated Affine Shaker: because no time-dependent parameter are present in the algorithm, the user can start the search and terminate as soon as the best value found is acceptable, or as soon as the maximum allotted time elapses.

The first series of tests, described in Section 5.1 are simulating this situation. In particular, the maximum number of function evaluations considered is fixed to 100000 for all runs.

In a second series of tests we make the assumption that an *oracle* is telling the correct number of function evaluations to be used by PSO on a problem.

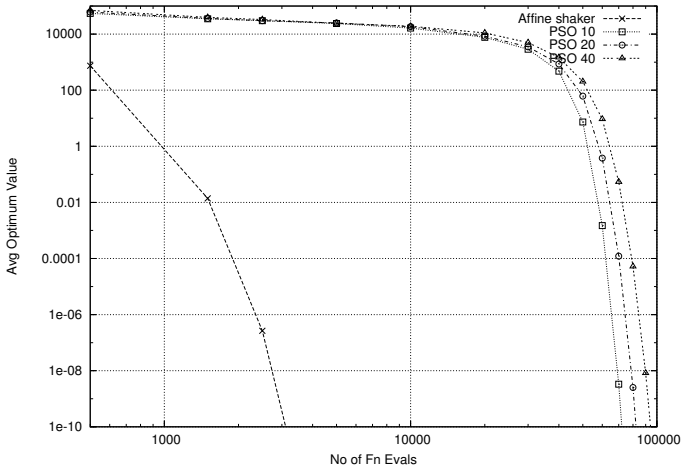

Figure 8: Sphere Function f_1 (fixed MAXITER)

The search is then run up to that number of function evaluations and the best final results are reported. While not realistic, this second series of tests permits to evaluate the effectiveness of PSO as a function of the choice of MAXITER, therefore providing a much larger set of experimental data. Of course, the second choice is not suggested in the applications because the computational effort is much larger: depending on the ΔM interval considered for the MAXITER value the total computational effort is the sum of the individual effort of the run up to ΔM, the independent run up to $2\Delta M$, ..., the run up to MAXITER. This second series of tests is presented in Sec. 5.2.

5.1 PSO versus AS, fixed maximum number of iterations

Figs. 8–13 show the comparison graphs between the Affine Shaker and PSO algorithm for the different benchmark functions considered. The plots are on a log-log scale to show the evolution over a very wide range of values and for up to a large number of function evaluations. In the Appendix Fig. 20–25 show the same comparison graphs but with error of the average included.

For all functions apart from the Schaffer f_6, the Repeated Affine Shaker finds very rapidly low values of f and it tends to maintain the advantage even for large number of iterations. Better values are found by PSO only in the second part of the search (after about 50000 function evaluations).

For the f_6 function, if one consider minimization (and therefore is looking for a point at the bottom of the first circular ring surrounding the needle at the origin) Repeated Affine Shaker lags behind PSO in the initial phase but rapidly and consistently catches up after about 20000 function evaluations, see Fig. 12. If one considers maximization, see Fig. 13 where the difference from

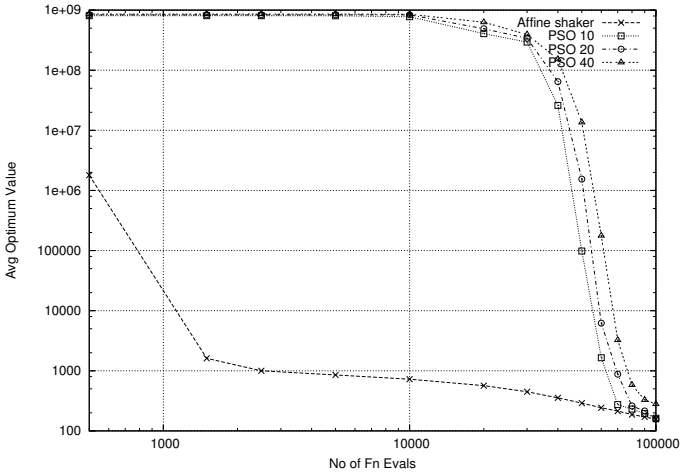

Figure 9: Rosenbrock Function f_2 (fixed MAXITER)

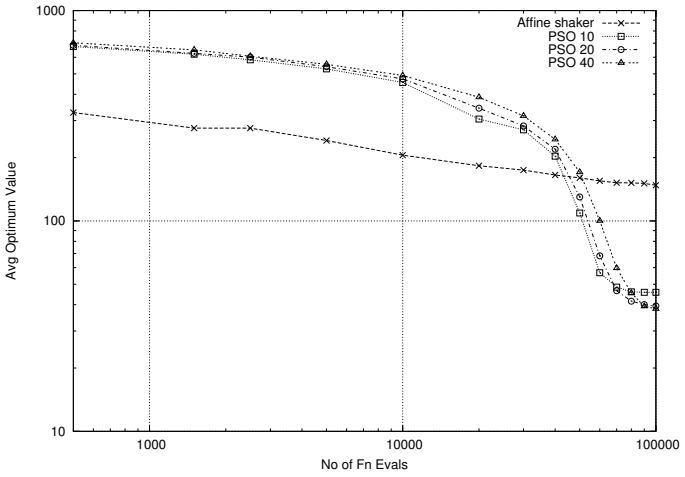

Figure 10: Rastrigrin Function f_3 (fixed MAXITER)

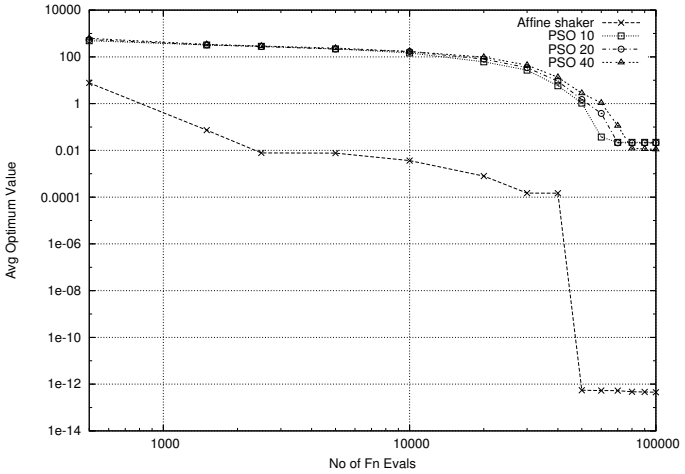

Figure 11: Griewank Function f_4 (fixed MAXITER)

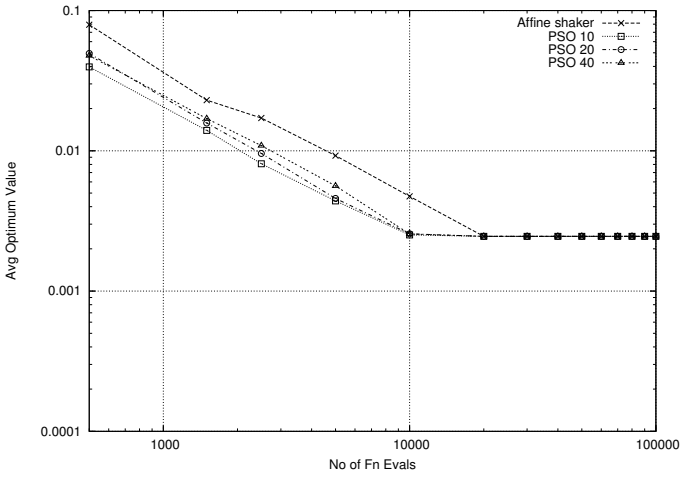

Figure 12: Schaffer Function Minimizingf_6 (fixed MAXITER)

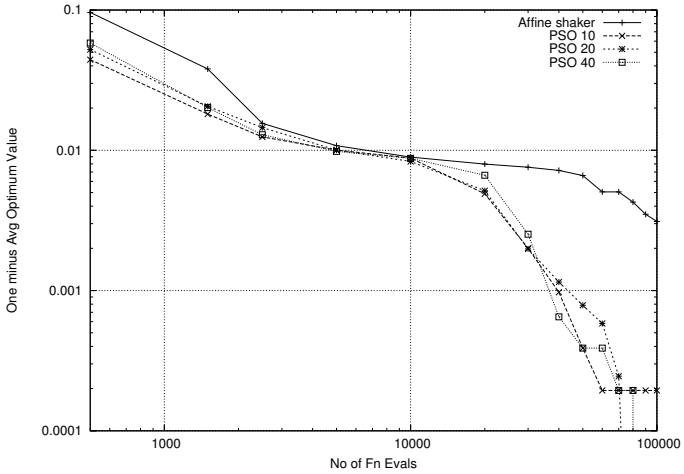

Figure 13: Schaffer Function Maximizingf_6 (fixed MAXITER)

the optimum $1 - f(\boldsymbol{g})$ is shown, Affine Shaker is lagging at the beginning, providing similar good quality values (within 0.01 of the maximum) at about 10000 function evaluations, and then showing a slower improvement during the final part of the search. Considering the structure of the f_6 function, maliciously designed to trap methods based on hill-climbing, with a local optimum at the center surrounded by a very small attraction basin (the "needle" in Fig. 6), these results are hardly surprising. In fact, PSO is influenced by the envelope of the function, see again Fig. 6, that is clearly showing the way towards the global optimum, while Affine Shaker without any interaction starts each search by hoping that random extraction of trajectory points will by chance fall inside the "needle" at the center.

While the plots show the entire story, to simplify the comparison and to consider another measure based on average time required by a run to reach a threshold value, it is useful to select some fixed threshold and to measure what is the average number of function evaluations used by the different algorithms to reach the thresholds. Table 3 list the number of function evaluations while Table 4 show the relative numbers, as speedup of AS with respect to PSO.

It can be noted that, for the chosen thresholds, Affine Shaker consistently provides a substantial speedup with respect to PSO.

Finally, a third measure of performance is obtained as the average value found by the two methods at the end of the 100000 function evaluations. These results are collected in Table 5. Again, Affine Shaker is better than or comparable to PSO in all cases apart from the Rastrigrin function which shows a clear advantage for PSO.

Table 3: Function Evaluations

Algorithm (goal)	Sphere (< 0.1)	Rosenbrock (< 10000)	Rastrigrin (< 200)	Griewank (< 0.2)	Schaffer (Max, > 0.99)
AShaker	1500	1040	15410	1500	2140
PSO-10	55370	53580	40750	55850	3830
PSO-20	61370	58880	42460	61200	4160
PSO-40	68530	66410	46790	68410	3940

Table 4: Speed-Up Table

Algorithm	Sphere	Rosenbrock	Rastrigrin	Griewank	Schaffer (Max)
PSO-10	37	51	2.6	37	1.79
PSO-20	41	57	2.7	41	1.94
PSO-40	46	64	3	45	1.84
AShaker	1	1	1	1	1

Table 5: Average Optimum Value at the end of 100000 Function Evaluations

Algorithm	Sphere	Rosenbrock	Rastrigrin	Griewank	Schaffer (Max)	Schaffer (Min)
PSO-10	8.29e-30	162.27	45.81	0.022	0.999	0.0024
PSO-20	5.55e-22	156.94	39.42	0.021	1	0.0024
PSO-40	3.25e-13	279.61	38.26	0.011	1	0.0024
AShaker	1.69e-18	155.91	147.91	4.52e-13	0.996	0.0024

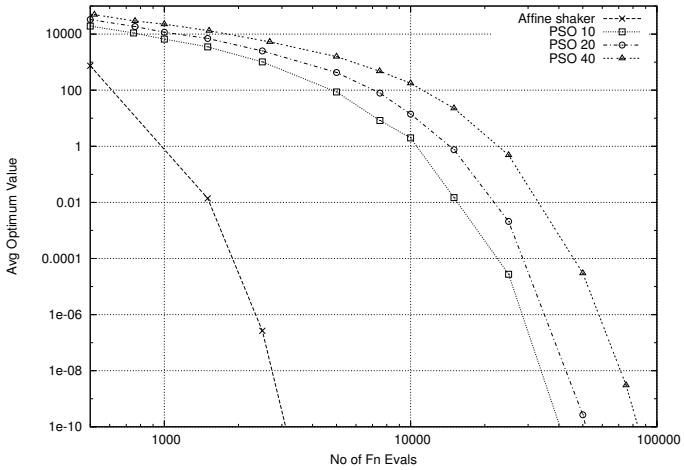

Figure 14: Sphere Function f_1

5.2 PSO versus AS, variable maximum number of iterations

As mentioned, this second series of tests presents the same data for the Affine Shaker, while it reports end values obtained by multiple runs of PSO up to different values of maximum iterations (or, equivalently, maximum number of function evaluations). For each data point at a given number of function evaluations 50 independent runs of PSO are executed. The points on the plots are connected for better visibility but refer to different experimental runs, where the oracle gives a different MAXITER value.

Even in the case that help by the oracle is used by PSO, it can be observed that the Repeated Affine Shaker algorithm takes considerably less number of function evaluations to find comparable values when compared to PSO with different population sizes for the functions f_1, f_2, and f_4. This result is consistent for different thresholds put on the value to be reached.

Given the simplicity of the Repeated Affine Shaker, without any interaction among the multiple runs, these results were unexpected, in particular for the more complex f_2 and f_4 functions.

For the Rastrigrin f_3 and Schaffer f_6 the situation is more varied. For f_3 AS finds very rapidly good local minima, while PSO finds better values for a fixed number of function evaluations if one searches for lower values, see Fig. 16. When the Schaffer function is considered, both minimization and maximization are more rapid for PSO.

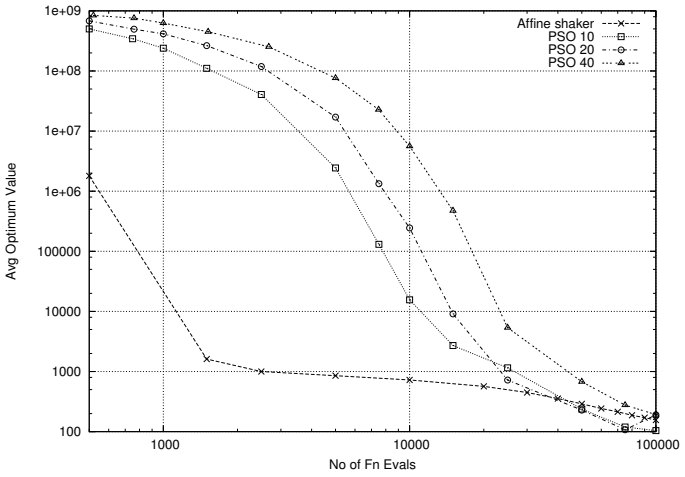

Figure 15: Rosenbrock Function f_2

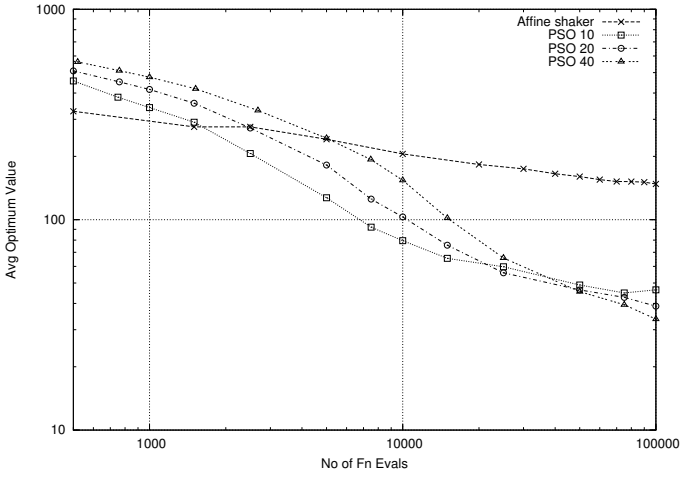

Figure 16: Rastrigrin Function f_3

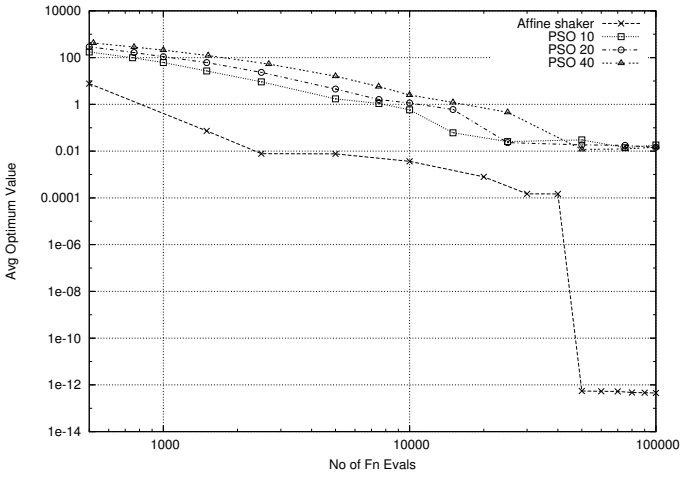

Figure 17: Griewank Function f_4

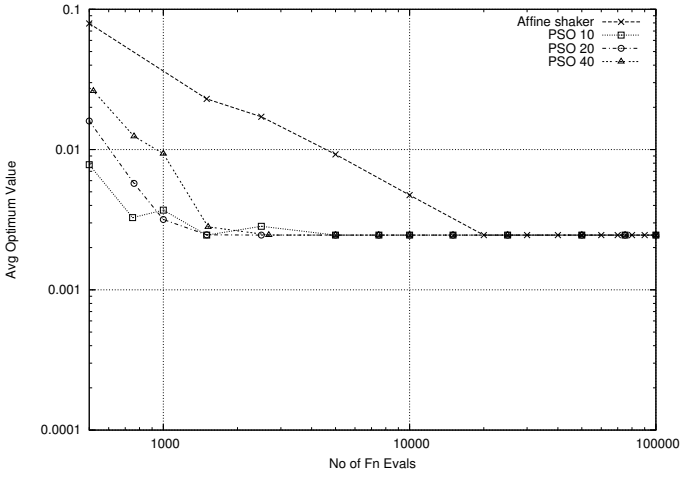

Figure 18: Schaffer Function Minimizingf_6

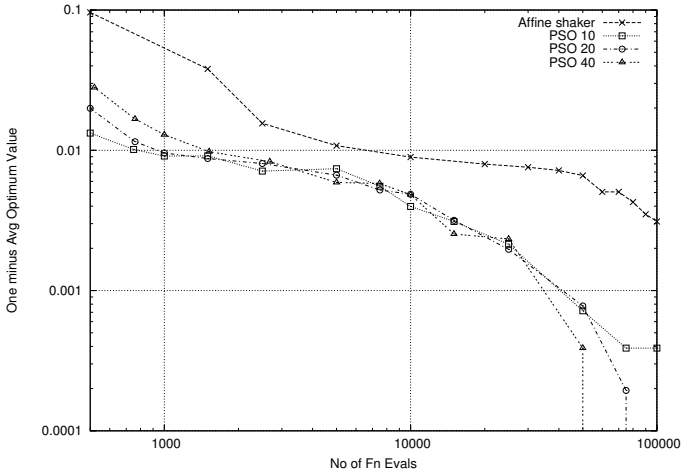

Figure 19: Schaffer Function Maximizing f_6

6 Conclusions

This paper compares two stochastic local search techniques for the minimization of continuous functions. Both techniques generate a search trajectory by producing the next point through a randomized extraction in a localized region, and both consider multiple trajectories, but they differ in a radical manner in the amount of interaction among the different searchers. In one case (Repeated Affine Shaker), the *different searchers are independent*, no information is transferred from one trajectory to another one. Running searches in parallel or sequentially produces the same results. In the other case (Particle Swarm), each trajectory is influenced also by the collective information derived by the other trajectories. Because of the larger amount of information available in the second case and the possibility to "intensify" the search in promising regions, one expects a much larger efficiency. The experiments are designed to simulate a situation where a user selects a large number of iterations before starting the search (not knowing *a priori* the appropriate number of function evaluations needed to reach acceptable results) and a different situation in which an *oracle* tells the user the appropriate MAXITER value. Let's note that intermediate cases can also be considered, so that the user is ignorant at the beginning but progressively tunes parameters depending on preliminary results, the detailed examination of this context is considered in an extension of this research.

The experimental results are unexpected: in the first experimental setting Affine Shaker tends to provide significantly better results in a much smaller number of function evaluations in many conditions, while PSO demonstrates an advantage only for Rastrigrin f_3 at the end of the search and for the "malicious" Schaffer f_6 function. In the second experimental setting, when an oracle sug-

gests the appropriate value for the maximum number of iterations to run PSO with, for some functions, AS gives better results, or reaches a given result with a smaller number of function evaluations, in other cases PSO has a better performance. When PSO has a significantly better performance, a closer analysis reveals that the function structure is "designed" so that the values of the local minima in space clearly point the way towards the global optimum.

Interestingly enough, similar results related to the need to specify the evolution in time of a parameter guiding the search have been found in a very different context related to comparing Simulated Annealing and Tabu Search on the Quadratic Assignment Problem [3].

A tentative conclusion of this preliminary work is that the effort to avoid a premature convergence of the swarm causes a performance penalty in particular in the initial part of the search, when good local minima can be lost because the particles are moving with high velocity over a portion of the search region (the "fear of local minima" effect if we want an analogy). The second technique (AS) is very aggressive in searching for local minima: rapid convergence is desired because diversification can be obtained with the simple multi-start technique.

Given the remarkable performance of the simple AS strategy it is of course of interest to consider integration of swarm intelligence methods with AS, to see whether adding interactions between the individual searchers by Particle Swarm methodologies will provide even better results. A second issue of relevance is to extend the results obtained on a selected number of carefully selected functions to a wider selection of functions with a more complex and "unpredictable" structure. The two issues are currently investigated in our group.

Acknowledgement

We acknowledge support by the WILMA project funded by the Autonomous Province of Trento, and by the QUASAR project funded by MIUR.

References

[1] P. J. Angeline. Evolutionary Optimization versus Particle swarm optimization:philosophy and performance difference. *Annual Conference on Evolutionary Programming*, 1998.

[2] P. J. Angeline. Using Selection to Improve Particle Swarm Optimizaation. *IEEE International Conference on Evolutionary Computation*, May 1998.

[3] R. Battit and G. Tecchiolli. Simulated annealing and tabu search in the long run: a comparison on QAP tasks. *Computer and Mathematics with Applications*, 28(6):1–8, 1994.

[4] R. Battiti and G. Tecchiolli. Learning with first, second and no derivatives: A case study in high energy physiscs. *Neurocomp*, 6:181–206, 1994.

[5] Roberto Battiti and Giampietro Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[6] M. Clerc and J. Kennedy. The particle swarm – explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[7] R. C. Eberhart and Y. H. hi. Comparing inertia weights and constriction factors in particle swarm optimization. *Proc. IEEE Int. Congr. Evolutionary Computation*, 1:84–88, 2000.

[8] D. Fogel and H. G. Beyer. A Note on the Empirical Evaluation of Intermediate Recombination. *Evolutionary Computation*, 3(4), 1995.

[9] Kennedy J. and Eberhart R.C. Particle Swarm Optimization. *IEEE Int. Conf. Neural Networks*, pages 1942–1948, 1995.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[11] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240–255, 2004.

[12] Y. H. Shi and R. C. Eberhart. A Modified Particle Swarm Optimizer. *IEEE International Conference on Evolutionary Computation*, May 1998.

[13] Y. H. Shi and R. C. Eberhart. Parameter Selection in Particle Swarm Optimization. *Annual Conference on Evolutionary Programming*, March 1998.

[14] Y. H. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. *Proc. IEEE Int. Congr. Evolutionary Computation*, 3:101–106, 1999.

7 Appendix

Results for a fixed number of 100000 function evaluations with error bars (error on the average) are illustrated in Fig. 20–25, while results related to an oracle specifying the number of function evaluations to use in PSO are reported in Fig. 26–31.

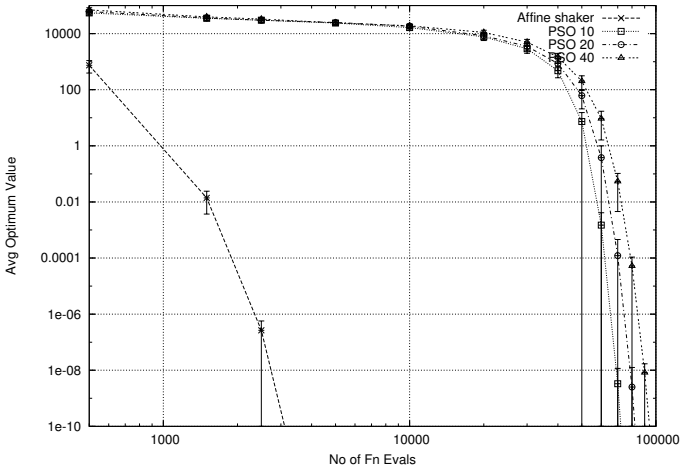

Figure 20: Sphere Function f_1

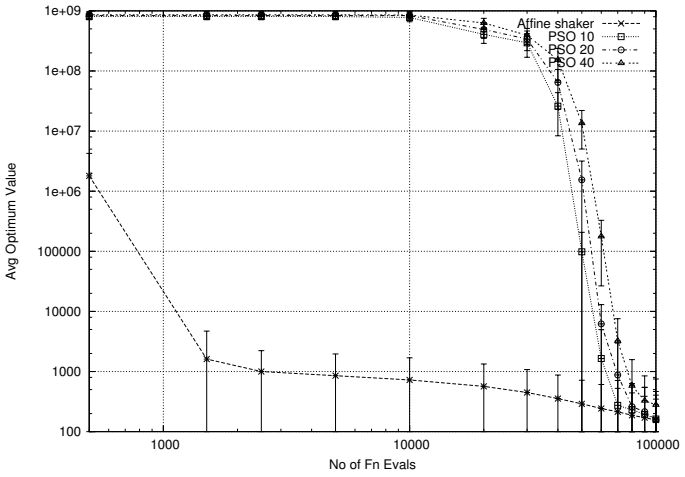

Figure 21: Rosenbrock Function f_2

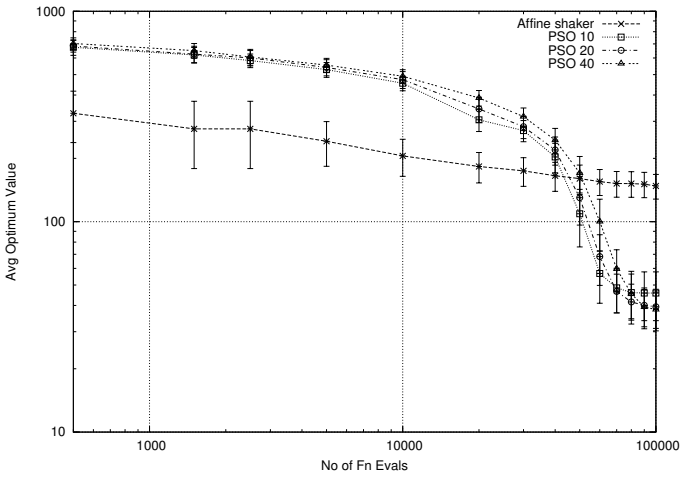

Figure 22: Rastrigrin Function f_3

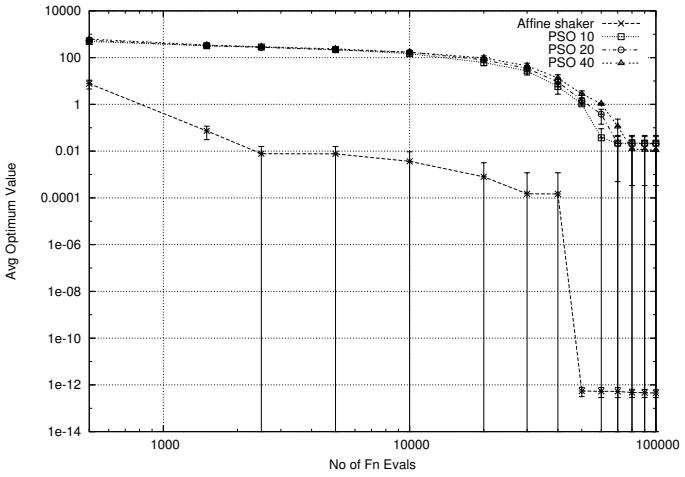

Figure 23: Griewank Function f_4

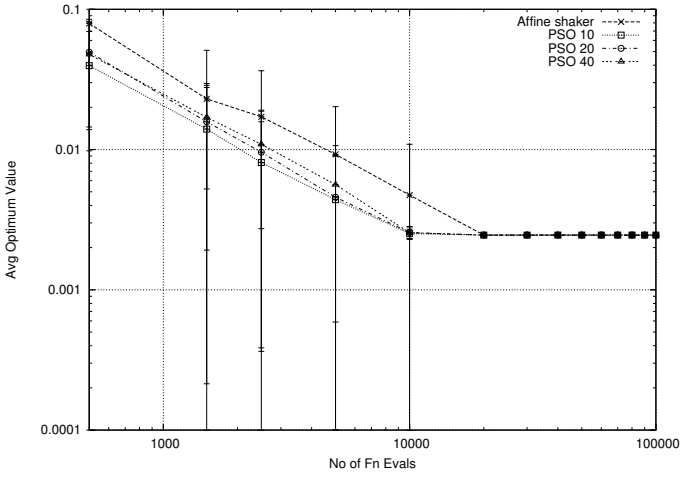

Figure 24: Schaffer Function Minimizingf_6

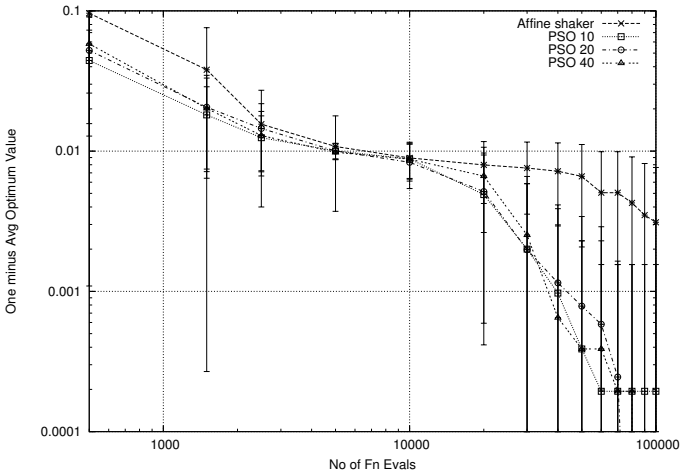

Figure 25: Schaffer Function Maximizingf_6

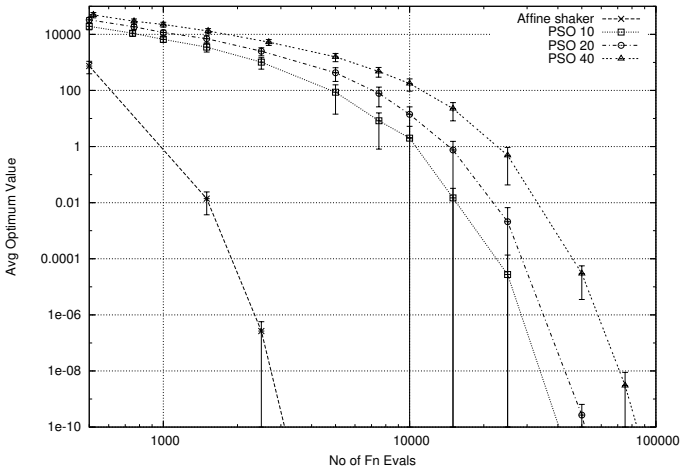

Figure 26: Sphere Function f_1

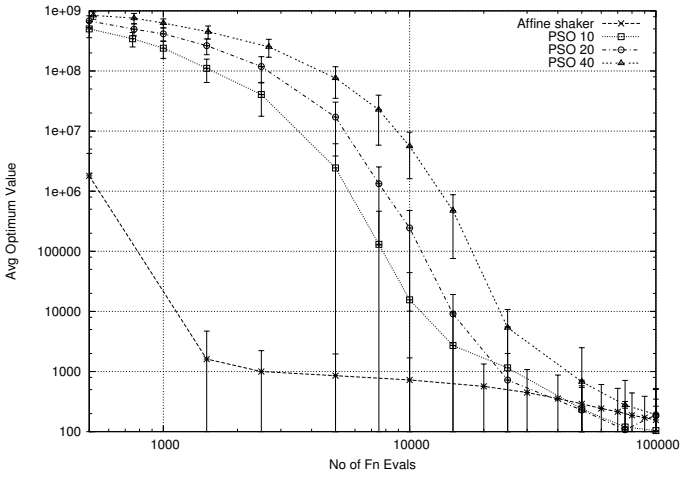

Figure 27: Rosenbrock Function f_2

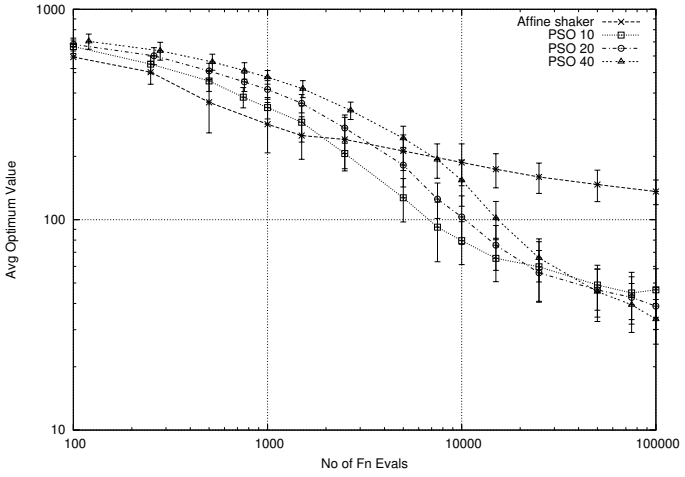

Figure 28: Rastrigrin Function f_3

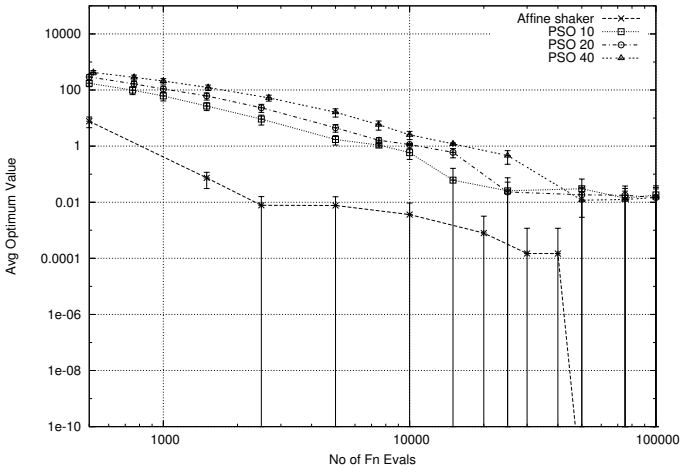

Figure 29: Griewank Function f_4

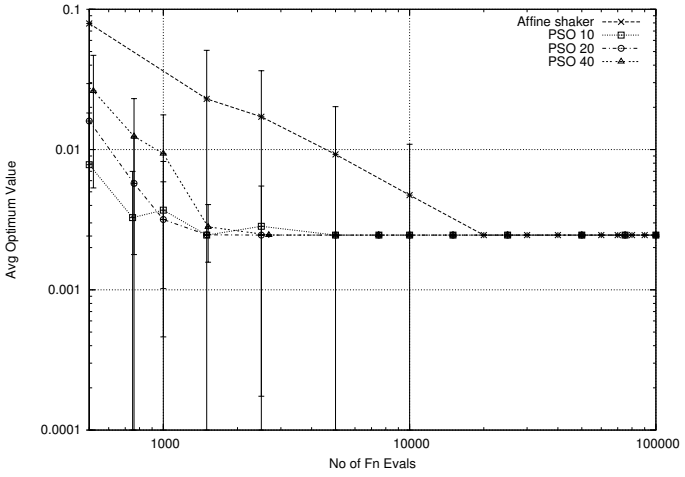

Figure 30: Schaffer Function Minimizingf_6

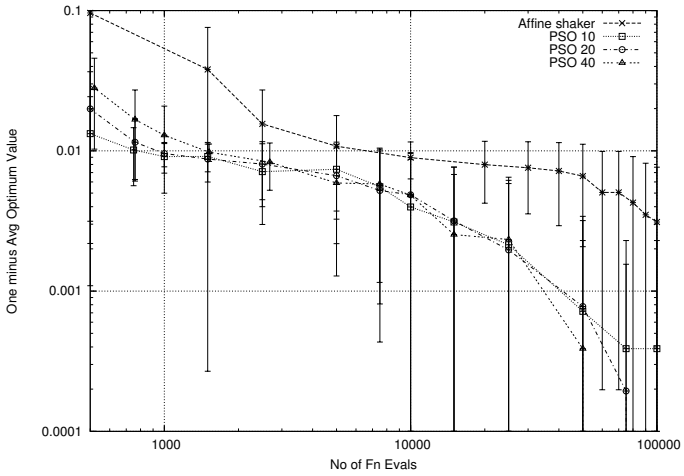

Figure 31: Schaffer Function Maximizing f_6