Initialisations for Particle Swarm Optimisation

Maurice Clerc

24th December 2008

1 Some distributions[1]

1.1 Uniform random distribution: easy but bad

Let E be a real real search space with D dimensions. We want to define N points in this search space as initial positions of a population based optimisation algorithm. We are supposed to know nothing about the fitness function, so we want an initialisation as "regular" as possible

The most usual way is the following:

- on each dimension, the corresponding coordinate of a given point is chosen randomly, according to an uniform distribution

What is wrong with that method?

First, it is easy only on a D-parallelepiped, say a D-cube, for simplicity. In that case, we have $E = [a, b]^D$, and for each point $x_i = (x_{i,1}, \cdots, x_{i,d}, \cdots, x_{i,D})$, we have $x_{i,d} = (b - a)\, U(0, 1)$. This is for example the method used in Standard PSO 2007 [8]. But it does not work for a more complicated search space (which is a quite common phenomenon when more constraints than just intervals of values are taken into account), particularly when D is high. For example, when E is a D-sphere, it is tempting to apply this method to the tangent hypercube and to keep only the points that are really inside the sphere. In practise it is not possible as soon as the dimension is high (say 6), for the rate $volume(D - sphere)/volume(D - cube)$ rapidly tends to zero when D increases. A completely different method is necessary [2]. However, in this paper, I do not examine this point again, and indeed consider that the search space is a D-cube. Nevertheless, the main reason for which this method is not very good is that the resulting set of points does not "cover" well the search space, as we can see on figure 1. Actually, it is also sometimes pointed out that when D is high, almost all points are near to the boundaries, and that this is another drawback of this method. But this second reason is not really valid.

Let us consider two hypercubes $C_0 := [0, 1]^D$, and $C_\varepsilon := [\varepsilon, 1 - \varepsilon]^D$, and let us call ε-shell the set $C_0 - C_\varepsilon$. A point of C_0, $x = (x_1, \ldots, x_D)$, is in the ε-shell if, and only if it not belongs to C_ε. If x has been generated according to the uniform distribution $U(0, 1)$ independently on each dimension, the probability of this event is $p(x \in \varepsilon - \text{shell}) = 1 - (1 - 2\varepsilon)^D$, i.e. simply $1 - volume(C_\varepsilon)/volume(C_0)$. From an intuitive point of view, this is perfectly acceptable. What is *not* intuitive is the fact that, as we can see on the figure 2, this probability increases quickly with D. For example, for $\varepsilon = 0.01$, and $D = 100$, 87% of the generated positions are in the ε-shell.

[1]The C code of the described distributions, and of the test functions, is included in Balanced PSO, available on my site http://clerc.maurice.free.fr/pso/

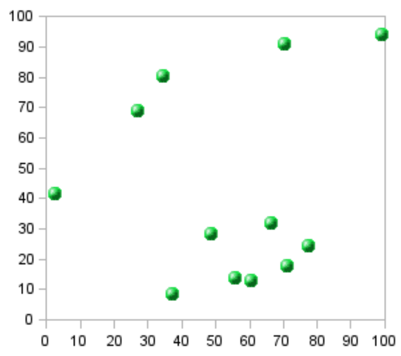
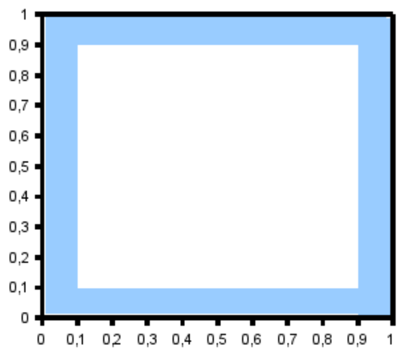
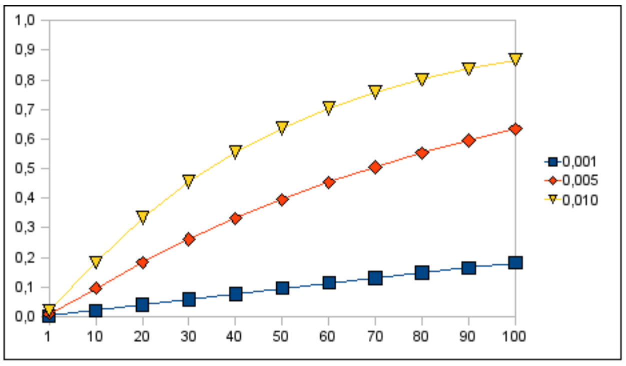


Figure 1: Random initialisation, according to an uniform distribution



(a) An example of "shell"



(b) Membership rates

Figure 2: a) ε-shell for $\varepsilon = 0.1$, and $D = 2$; b) For three values of ε, probability that a point is in the ε-shell, when generated by using $U(0,1)$ on each dimension

In other words, for a constant ε, C_ε becomes small compared to C_0 when D increases. Note it does *not* mean that the diagonal of C_ε becomes also small, compared to the one of C_0, for we have $diagonal(C_\varepsilon)/diagonal(C_0) = 1 - 2\varepsilon$, which is not depending on D. That is why it is so difficult to "see" what happens, and to think there is a bias, as in fact there is none.

Note that, though, when the algorithm is Particle Swarm Optimisation , it has been proved[7]that when D is high (typically 100), almost all particles tend to leave the search space after the first move. This is however specific to classical PSO, in which each particle has a velocity, and, moreover, efficient confinement mechanism can be applied (see Standard PSO on [8], and [3]).

1.2 Hammersley

This method is for example described in detail in [9]. I present it here just shortly. Let $P = (p_1, \cdots, p_{D-1})$ be a list of different prime numbers. For any element p of this list, any positive integer i can be expanded by $i = \sum_{k=0}^{r} a_k p^k$. We define then $\Phi_p(i) = \sum_{k=0}^{r} \frac{a^k}{p^{k+1}}$. Finally, the

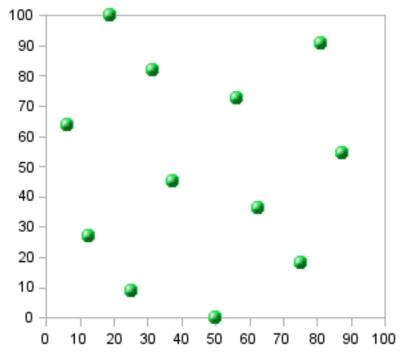


Figure 3: Hammersley distribution

ith point is defined by

$$x_i = \left(\frac{i}{N}, \Phi_{p_1}(i), \cdots \Phi_{p_{D-1}}(i) \right)$$

As we can see, in this pure Hammersley method, the first dimension is particular. So, when the initialisation is for a stochastic algorithm, is it is better to define at random before each run which dimension is “the first”. Figure 3 shows an example which is already far better than the uniform random one.

1.3 Tessellations - Potential method

There are a lot of pure deterministic tessellation methods (Dirichlet, Thiessen, Voronoï, Delaunay, etc.). However, as soon as the dimension is greater than 3, they are very complicated to code, so I do not examine them here. Fortunately, there are some methods that make use of a “potential” that are almost equivalent. The underlying idea is that the frontiers of the search space define a box, and that the points are particles moving inside the box. All elements, the box and the particles are “positively charged”, so there is repulsion between them (note that these particles have nothing to do with the ones of PSO). The equilibrium position is the one we are looking for. The exact potential is quite complicated as soon as the dimension is high, so, in practise, we can approximate the equilibrium position by minimising a simplified potential function, by any method. For example, in the parameter-free optimiser TRIBES [5], it is done by a standard PSO. There are usually several solutions. By starting from random positions for the points we can find different solutions. On the figure 4 we can see a result for $N = 12$ points, and $E = [a,b]^D = [0,1]^2$,when minimising the potential

$$V(E,N) = \frac{2D}{N-1} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{1}{\delta_{i,j}} + \frac{N-1}{4D} \sum_{i=1}^{N} \sum_{d=1}^{D} \left(\frac{1}{|x_{i,d} - (a+b)/2|} - \frac{2}{b-a} \right)$$

where $\delta_{i,j}$ is the Euclidean distance between the point x_i and the point x_j. The second part of the potential is a decreasing attractive force towards the centre of the hypercube, far easier to compute than a repulsive force from the boundaries.

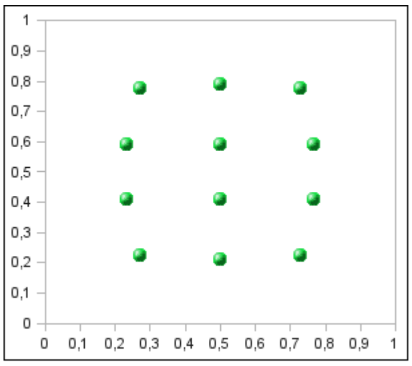


Figure 4: Distribution by minimisation of a potential

1.4 Biased random distribution

What about using a non uniform distribution? At the first glance, it seems a bit strange. We are looking for the point where a given function/landscape is minimum. But, after all, at the very beginning, if we know absolutely nothing about this landscape there is apparently no reason to set a more dense repartition of initial points here or there. But most of the time we do know something. In the worst case, it is almost sure that the function belongs to the Nearer is Better class [4]. It then means that the solution point is more probably "not far" from the centre of the search space. Also, as already noted, in a algorithm like PSO and for high dimension, at the very beginning almost all particles tend to leave the search space. So, for these two reasons, it may be interesting to use a repartition that is more dense around the centre of the search space. Note this is already the case with the potential method defined above. Here I use the following distribution:

$$\begin{cases} \delta & = & U(0,1) - 0.5 \\ x_{i,d} & = & \frac{a+b}{2} + (b-a)\, sign(\delta)\, |\delta|^{1+1/N} \end{cases}$$

2 Application to PSO

For Standard PSO with N particles, we not only need to initialise N positions x, but also N velocities v. So, we have to compare different methods for this initialisation. I consider here the four following ones, for the particle whose initial position is x_i:

Two-rand: $v_{i,d} = (x_{max,d} - x_{min,d})\,(U(0,1) - U(0,1))$.

Two-rand-Half-diff: $v_{i,d} = 0.5\,(x_{max,d} - x_{min,d})\,(U(0,1) - U(0,1))$. This is the method used in Standard PSO 2007

One-rand: $v_{i,d} = (x_{max,d} - x_{min,d})\,U(0,1) - x_{i,d}$

Two-particles-rand: $v_{i,d} = x_{j,d} - x_{i,d}$, where j is chosen at random (uniformly) in $\{1,\ldots,N\}$

Zero: $v_{i,d} = 0$

It means we have finally to compare 4×5 initialisation methods. After a lot of tests, I found that, apparently, the best combination is (Hammersley, One-rand). In the table 1a comparison with Standard PSO, which uses the (Random, Two-rand-Half-diff) pair. The three first functions are coming from the CEC 2005 benchamrk[1] (in particular, they are "shifted"). The Tripod

Table 1: A short comparison between classical initialisations and a different pair. Mean of the best final value over 100 runs

	Search space	Number of evaluations	(Random, Two-rand-Half-diff)	(Hammersley, One-rand)	Relative improvement
Parabola/Sphere	$[-100, 100]^{30}$	10000	0.0026	0.0022	14 %
Rosenbrock	$[-100, 100]^{10}$	5000	68.7	12.9	81 %
Rastrigin	$[-5, 5]^{10}$	10000	7.36	6.5	12 %
Tripod	$[-100, 100]^{2}$	10000	0.50	0.44	13 %

Table 2: Results with some other initialisations

	(Random, One-rand)	(Random, Zero)	(Random, Two-rand)	(Potential, One-rand)	(Biased, Two-rand)	(Biased, One-rand)
Parabola/Sphere	0.0048	0.0052	0.0042	0.0030	21.7	0.0027
Rosenbrock	57.8	56.2	57.7	45.0	65.4	39.5
Rastrigin	6.8	7.9	7.3	7.5	7.3	6.71
Tripod	0.63	0.61	0.52	0.54	0.59	0.53

function is defined in [6]. Although quite simple, it is very deceptive for most of algorithms. The relative improvement is given by the formula $100\frac{value_1 - value_2}{value_1}$. In the table 2, we can see some other results. It is interesting to note that no initialisation of the positions is the best one by itself (i.e. no matter what the initialisation of velocities is), and vice-versa. Only some *pairs*, like (Hammersley, One-rand) seem to be better than all the others.

3 Conclusion

For a population based algorithm like PSO, it is possible to significantly modify the results just by using different kind of initialisations. Therefore, when comparing two PSO variants that make use of the same fixed number of particles, it is necessary to use the same initialisation method for the positions, and for the velocities, if they exist. If we do not do that, it is impossible to know whether a claimed improvement is really due to the modification of the strategy used by the particles, or just due to a different kind of initialisation.

References

[1] CEC. Congress on evolutionary computation benchmarks,http://www3.ntu.edu.sg/home/epnsugan/, 2005.

[2] Maurice Clerc. Math stuff about pso, http://clerc.maurice.free.fr/pso/.

[3] Maurice Clerc. Confinements and biases in particle swarm optimisation. Technical report, 2006.

[4] Maurice Clerc. When nearer is better, https://hal.archives-ouvertes.fr/hal-00137320. Technical report, 2007.

[5] Yann Cooren, Maurice Clerc, and Patrick Siarry. *Initialization and Displacement of the Particles in TRIBES, a Parameter-Free Particle Swarm Optimization Algorithm*, volume 136 of *Adaptive and Multilevel Metaheuristics*, pages 199–219. 2008.

[6] Louis Gacôgne. Steady state evolutionary algorithm with an operator family. In *EISCI*, pages 373–379, Kosice, Slovaquie, 2002. http://www.iie.cnam.fr/ gacogne première présentation de la fonction Tripod.

[7] Sabine Helwig and Rolf Wanka. Theoretical analysis of initial particle swarm behavior. In *10th International Conference on Parallel Problem Solving from Nature (PPSN 2008).*, pages 889–898, Dortmund, Germany, September 13-17 2008. PPSN X, Springer.

[8] PSC. Particle Swarm Central, http://www.particleswarm.info.

[9] Tien-Tsin Wong, Wai-Shing Luk, and Pheng-Ann Heng. Sampling with Hammersley and Halton points. *Journal of Graphics Tools*, 2 (2):9–24, 1997.