Randomness matters

Maurice Clerc

Randomness matters

Maurice Clerc

5th May 2012

1 Introduction

Comparing two stochastic algorithms is not easy, even for one single problem. By definition, comparing two algorithms means comparing the estimates of their respective performance measures, and it is difficult to be certain about the reliablities of these estimates. In this study, we will explain why a claim like "On problem P, algorithm A is better than algorithm B" should be carefully examined, and investigate the reasons for such scrutiny. Some reasons are fairly well known, e.g., the number of runs or the position of the solution point, but their importance is sometimes underestimated. Other reasons (e.g., the byte size of the computer on which the algorithm was run, or the kind of randomness that was used) are more subtle and not often taken into account, although their effects may be quite prominent. Both of these, namely, byte size and type of randomness, are in fact quite similar, for both modify the way the algorithm makes use of random numbers. One interesting observation is this : a careful study of these two reveals that it is sometimes possible to get excellent results with a very bad random number generator (RNG).

2 Five reasons to be careful

To compare two optimisation algorithms on a given problem, we run both a large number of times, and we compare the results. However, the conclusions that can apparently be drawn are sometimes questionable, for various reasons. Let us examine some of them. In the following, we use the C version of Standard PSO 2011 as optimiser (4.1), available on the Particle Swarm Central [8] (there is also a Matlab version, slightly different), but the principles are valid for any other stochastic algorithm.

2.1 The position of the solution point

Most iterative algorithms are biased. A simple and effective way to see it is to try solving the following impossible Flat problem,

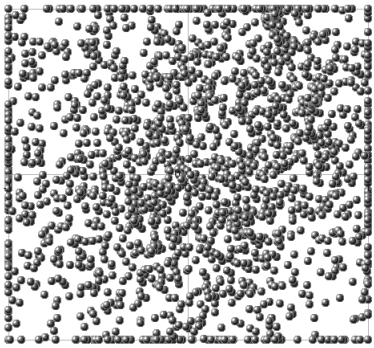


FIGURE 2.1: 2000 successive positions of the particles with SPSO 2011 on the impossible 2D Flat problem. The default confinement method is applied.

$$\begin{cases} f(x) & = & 1 \\ \text{search space} & = & [-1,1]^2 \\ \text{objective} & = & 0 \end{cases}$$

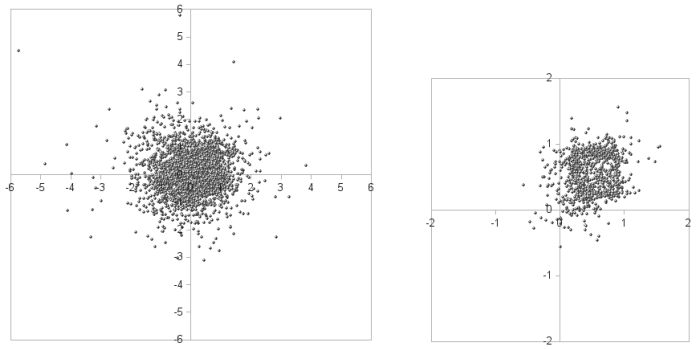
We can plot the successive positions of the particles. By default, in SPSO 2011, a confinement method is applied (the particles are stopped at the boundary of the search space). See figure 2.1. It is not surprising that a lot of positions fall exactly on the boundary. However, what is more unexpected is the higher density near the centre of the search space. This phenomenon is explained in [10] (actually, even for symmetrical functions, there is a higher density along the axes and the diagonals). It is even more obvious if we do not confine the particles (the "let them fly" method). From figure 2.2, we see that the algorithm wrongly "thinks" that the solution is in the middle of the search space. More precisely, with the usual values of the parameter, the swarm simply tends to oscillate around the centre of the initialisation, even if it is not the centre of the search space. For a perfectly unbiased algorithm, on this Flat problem the distribution of the positions should be uniform inside the whole search space.

Actually, whenever an algorithm works "dimension by dimension", this phenomenon may occur. Let us say that a problem is biased when its solution point is on a special position (diagonal, axis, centre). So, for fair comparison of algorithms, it is safer to apply the following rule.

Rule 1 : Never trust biased problems for comparisons

2.2 The number of runs

On a given problem, it is very common to estimate an average performance after several runs. For example, we can save the best fitness values found by the runs, and



(a) Initialisation was in $[-1,1]^2$ (b) Initialisation was in $[0,1]^2$

FIGURE 2.2: Successive positions on the impossible 2D Flat problem, with SPSO 2011, and no confinement.

compute their mean. Or, as for a test problem we do know the solution point x^*, we can define an acceptable error $\varepsilon > 0$, and we can say that a run is successful if it finds a point x so that $|f(x) - f(x^*) < \varepsilon|$. Then we define the success rate. There is nothing wrong with that, except that the number of runs may be too small for a correct estimate. Figure 2.3 shows an example : after 30 runs, the success rate is 23.3%, but after 100 runs, it is more or less stabilised around 17%. Therefore, if you perform only 30 runs, and if you claim that this algorithm is better than another one whose success rate is, say, 18%, you may be completely wrong.

Of course, the number of runs that are needed for such a stabilisation depends on the problem. We may find it "visually" by plotting the curve "Success rate vs Number of runs", or more rigorously by defining a maximum acceptable variance, and by running the algorithm as many times as necessary in order to have a variance below this threshold. Hence, we have

Rule 2 : Be sure that the estimated performance has converged reasonably

However, in some rare cases, the estimated performance may *never* converge. In particular, it may happen if the performance measure is the mean best value and if the algorithm makes use of a probability distribution with an infinite variance, like Lévy. In such a case, this particular performance measure is meaningless, and we have to use another one that does converge, like the success rate (which may be less robust, though, see below).

2.3 The kind of performance measure

As mentioned before, the two most commonly used performance measures (criteria) are the mean best and the success rate. But the important point is the relative "indepen-

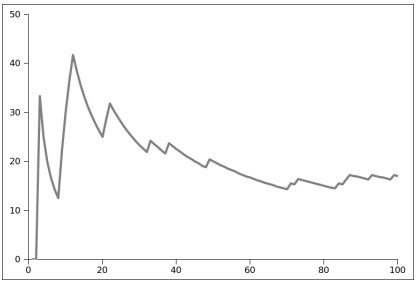


FIGURE 2.3: Success rate vs Number of runs. Shifted Rosenbrock 10D (90000 fitness evaluations/run).

dence" between the two criteria, as noted, for example, in [5]. It may well happen that an algorithm A seems better than an algorithm B with respect to the first criterion, but performs worse than B with respect to the second criterion. See table 1. Of course, this table is just an illustration, with a small number of runs, but the same phenomenon may occur with a bigger number of runs, and with significant difference. Hence we have

Rule 3a : Never trust just one performance criterion

and

Rule 3b : Always perform a statistical test

About the statistical test, a common mistake is to ignore some important assumptions. For example, a t-test can not be applied if the sample size is too small or if the distributions are not more or less normal, and one may have to simply apply a non-parametric test (Mann-Whitney, Friedman, ...), which is more flexible.

TABLE 1: According to the mean best, algorithm A seems better than algorithm B. According to the success rate, it is the opposite. The threshold value for success is 0.01.

	Algorithm A		**Algorithm B**	
Run	Best	Success	Best	Success
1	0.0034	1	0.0069	1
2	0.0098	1	0.0083	1
3	0.0145	0	0.0001	1
4	0.0156	0	0.1292	0
5	0.0182	0	0.0037	1
6	0.0159	0	0.0044	1
7	0.0025	1	0.0025	1
8	0.0132	0	0.1246	0
9	0.0192	0	0.1158	0
10	0.0004	1	0.0178	0
Mean	**0.0113**	**40%**	**0.0413**	**60%**

2.4 The word size

If the numbers are internally coded in the computer as words of n bits, it means that we have access to only 2^n different values. Of course, clever tricks may be used to increase this range, for example using two words to represent one number, but anyway there is a limitation. Such a limitation has an influence on the performance, as we can clearly see from figure 2.4. On the Shifted Sphere 30D problem, we always use the same RNG (Random Number Generator) called KISS, but we simulate a less powerful machine, whose byte size is 5, 7, 9 or 11. The results of the statistical tests are not shown here, but the corresponding success rates are significantly different. For this simple problem, the shorter the byte size, the better the performance, but it need not be the case for all problems. And so, we get the following

Rule 4 : For reproducible results, do specify the word size

2.5 The kind of randomness

This may be the most important cause of erroneous comparisons, and we will focus on it with the help of figures 2.5 and 2.3. The problems under consideration are the very classical Shifted Sphere (Parabolic (see 4.2.1)) and Shifted Rosenbrock (see 4.2.2). Let us rigorously define the different kinds of "randomness" (or Random Numbers Generator, RNG) that were used to plot these figures :

– "C" means the RNG of this language. Whenever the algorithm needs a random number, it calls the rand() function, and gets an integer between 0 and 32767, in a deterministic cyclic way that depends on the initialisation (the "seed");

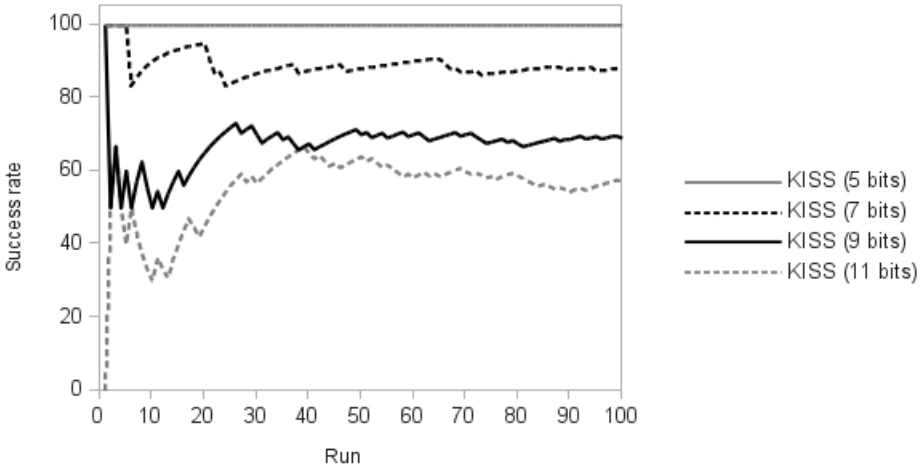


FIGURE 2.4: Shifted Sphere 30D. With the same RNG (KISS here), but different byte sizes, the performances are different. In this particular simple example, the shorter the word size, the better the performance, but it is not necessarily so for all problems.

- "KISS" has been defined in [4]. It is explicitly coded in the algorithm, and its cycle length is far longer (greater than 2^{127}). The deterministic sequence of produced numbers also depends on the "seed". In practice (as in this example) the number of random numbers that are needed is about 1.7e+08, and there is no cycle ;
- "Q" means that each random number is coded by a sequence of bits coming from a quantum system, which is supposed to generate "true" randomness. If we use just 3 bits, there are only 8 different possible values, and therefore there are cycles. With 32 bits, there is probably no cycle for these examples (although it has not been checked).

We can make at least three important remarks :

1. For a given RNG the performance (here the success rate) may be very different for 30 runs and for 100 runs ;
2. For the same number of runs, the performance may also be very different for different RNGs. In this particular case, it is perfect (100%) for the first example (Sphere 30D) with Q (3 bits).
3. There is no relationship between the quality of the randomness and the performance. One may have good performance with a bad RNG like Q (3 bits), but on the other hand the performance may be better with a better random number generator, like Q (32 bits) vs KISS. The figure 2.7 presents some results on four quasi-real-world problems that illustrate this remark.

Therefore, we need to apply

Rule 5 : For reproducible results, do specify the RNG that is used, including its parameters, if any (like the seed)

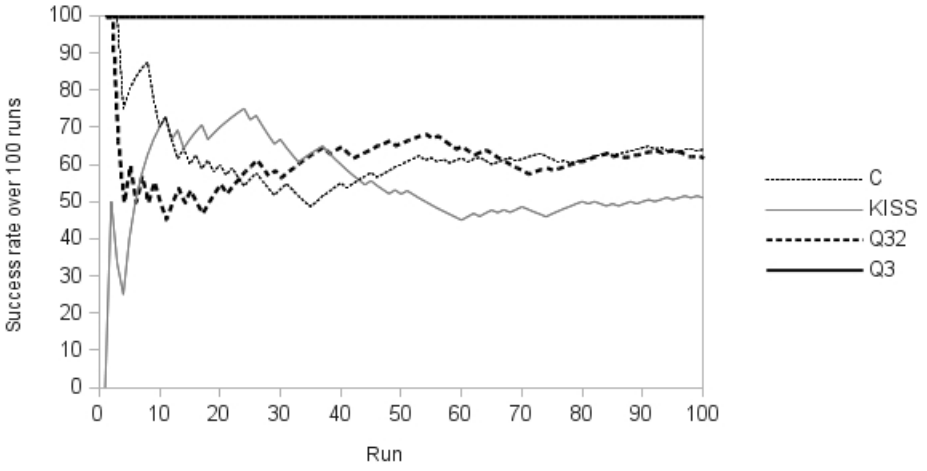


FIGURE 2.5: Shifted Sphere 30D. The performance measure - here the success rate - may be very different when using different RNGs. A bad one, like Q (3 bits) produces a perfect result (100%) here.

Remark 3 above is particularly interesting, for it suggests that it may be possible to replace almost any stochastic algorithm by a deterministic one. The only limitation - and hence the "almost" - is that the random numbers must be bounded. If the stochastic algorithm makes use of unbounded random numbers, sampled, for example from a Gaussian or a Lévy distribution, theoretically, it can not be transformed into a list based one. However, one may note that on a computer, the list of possible numbers is anyway always bounded. So, in practice, such a transformation is always possible. Thus, we replace the concept of stochastic optimisation by the list based one.

3 Future work

The kind of randomness that is used can be seen as a parameter of the optimiser. Therefore, an obvious question is "Can we modify it during the iterative process, in order to improve the performance?" In short, can we define an adaptive randomness? Preliminary results suggest that it is indeed possible. A simple way to do that is, for example, to use the Q randomness (list of bits generated by a true random system),

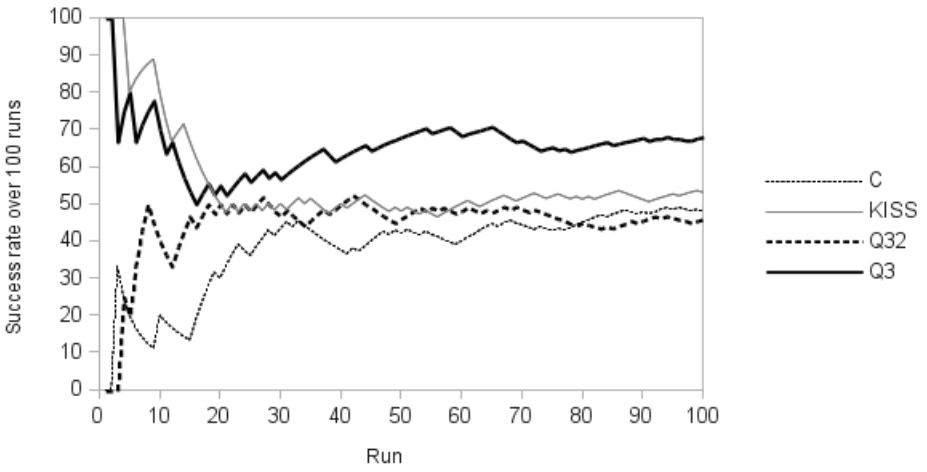


FIGURE 2.6: Shifted Rosenbrock 10D. Here too Q (3 bits) produces a better result.

and to modify the number of bits n that are used to generate real numbers in $[0, 1]$. An adaptation rule could be "if improvement, decrease n, if not, increase it".

4 Appendix

4.1 Standard PSO 2011

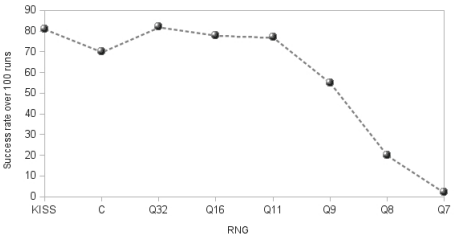
It was well suspected for years that the dimension by dimension method used by most of PSO versions is biased : when the optimum point lies on a axis, or on a diagonal, or worse, on the centre of the system of coordinates, it is easier to find it. The paper [6] was not completely convincing, but a more complete analysis of this phenomenon has been presented in 2010 [10]. That is why in SPSO 2011 the velocity is modified in a "geometrical" way that does not depend on the system of coordinates.

Let $G_i(t)$ be the centre of gravity of three points : the current position, a point a bit "beyond" the best previous position (relative to $x_i(t)$), and a point a bit "beyond" the best previous position in the neighbourhood. Formally, it is defined by the following formula, in which t has been removed for simplicity

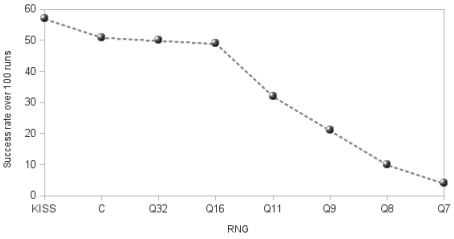
$$G_i = \frac{x_i+(x_i+c(p_i-x_i))+(x_i+c(l_i-x_i))}{3} = x_i + c\frac{p_i+l_i-2x_i}{3} \quad (4.1)$$

We define a random point x'_i (uniform distribution) in the hypersphere

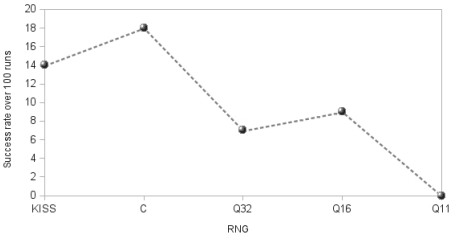
$$\mathcal{H}_i(G_i, \|G_i - x_i\|) \quad (4.2)$$



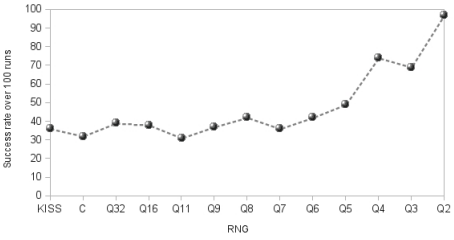
(a) Compression Spring



(b) Gear Train



(c) Pressure Vessel



(d) Lennard-Jones

FIGURE 2.7: Usually, when the number of possible random values decreases, the performance also tends to decrease (a,b,c). However it is not always the case (d).

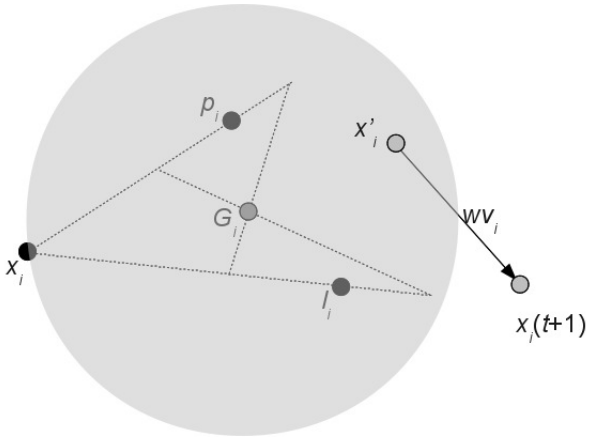


FIGURE 4.1: SPSO 2011. Construction of the next position. The point x'_i is chosen at random inside the hypersphere $\mathcal{H}_i\left(G_i, \|G_i - x_i\|\right)$

with centre G_i and of radius $\|G_i - x_i\|$. Then the velocity update equation is

$$v_i\left(t+1\right) \quad = \quad wv_i\left(t\right) + x'_i\left(t\right) - x_i\left(t\right) \tag{4.3}$$

It means that the new position is simply

$$x_i\left(t+1\right) \quad = \quad wv_i\left(t\right) + x'_i\left(t\right) \tag{4.4}$$

Note that with this method it is easy to rigorously define "exploitation" and "exploration". There is *exploitation* when $x_i\left(t+1\right)$ is inside at least one hypersphere $\mathcal{H}_j$, and *exploration* otherwise.

The source code contains some options (like hyperspherical Gaussian distribution instead of the uniform one) that are not described here.

We may sometimes have $l_i\left(t\right) = p_i\left(t\right)$ when the particle i is precisely the one that has the best previous best in its neighbourhood. In such a case $l_i\left(t\right)$ is simply ignored, and the equation 4.1 that defines the gravity centre G_i becomes

$$G_i = \frac{x_i + \left(x_i + c\left(p_i - x_i\right)\right)}{2} = x_i + c\frac{p_i - x_i}{2} \tag{4.5}$$

SPSO 2011 keeps the same variable neighbourhood topology as the previous standard versions (2006, 2007), i.e. the topology is modified partly at random after each unsuccessful iteration. For more details, see the original source code on the Particle Swarm Central [8]. For this study, some options have been added, in particular more possible RNGs. The modified code (a C version) is available on my personal site [1]. The parameter values are the suggested ones for the original version, i.e. $S = 40$ (swarm size), $K = 3$ (number of particles informed at random by a given one), $w = 1/\left(2ln\left(2\right)\right)$ (inertia weight),

$c = 0.5 + ln(2)$ (common value for cognitive and social coefficient). All runs have been performed on a 64 bit machine.

4.2 Test problems

4.2.1 Shifted Sphere (Parabola)

The function to minimise is

$$f(x) = -450 + \sum_{d=1}^{30} (x_d - o_d)^2$$

The random offset vector $O = (o_1, \cdots, o_{30})$ is defined below. The search space is $[-100, 100]^{30}$. The function is unimodal and O is the solution point, on which $f = -450$.

Offset O for Sphere/Parabola (C source code)

static double O[30] = { -3.9311900e+001, 5.8899900e+001, -4.6322400e+001,
-7.4651500e+001, -1.6799700e+001, -8.0544100e+001, -1.0593500e+001,
2.4969400e+001, 8.9838400e+001, 9.1119000e+000, -1.0744300e+001,
-2.7855800e+001, -1.2580600e+001, 7.5930000e+000, 7.4812700e+001,
6.8495900e+001, -5.3429300e+001, 7.8854400e+001, -6.8595700e+001,
6.3743200e+001, 3.1347000e+001, -3.7501600e+001, 3.3892900e+001,
-8.8804500e+001, -7.8771900e+001, -6.6494400e+001, 4.4197200e+001,
1.8383600e+001, 2.6521200e+001, 8.4472300e+001 };

4.2.2 Shifted Rosenbrock

The function to minimise is

$$f(x) = 390 + \sum_{d=2}^{10} \left(100 \left(z_{d-1}^2 - z_d\right)^2 + \left(z_{d-1} - 1\right)^2\right)$$

with

$$z_d = x_d - o_d + 1$$

The search space is $[-100, 100]^{10}$. The random offset vector $O = (o_1, \cdots, o_{10})$ is defined below. This is the solution point, on which $f = 390$. There is also a local minimum $(o_1 - 2, \cdots, o_{30})$, on which the fitness value is 394.

Offset O for Rosenbrock (C source code)

static double O[10] = { 8.1023200e+001, -4.8395000e+001, 1.9231600e+001,
-2.5231000e+000, 7.0433800e+001, 4.7177400e+001, -7.8358000e+000,
-8.6669300e+001, 5.7853200e+001};

4.2.3 Compression Spring

For more details, see[9, 2, 7]. There are three variables

$$\begin{array}{rcll} x_1 & \in & \{1,\ldots,70\} & \text{granularity} \quad 1 \\ x_2 & \in & [0.6,3] & \\ x_3 & \in & [0.207,0.5] & \text{granularity} \quad 0.001 \end{array}$$

and five constraints

$$\begin{array}{rcl} g_1 & := & \frac{8C_f F_{max} x_2}{\pi x_3^3} - S \leq 0 \\ g_2 & := & l_f - l_{max} \leq 0 \\ g_3 & := & \sigma_p - \sigma_{pm} \leq 0 \\ g_4 & := & \sigma_p - \frac{F_p}{K} \leq 0 \\ g_5 & := & \sigma_w - \frac{F_{max} - F_p}{K} \leq 0 \end{array}$$

with

$$\begin{array}{rcl} C_f & = & 1 + 0.75\frac{x_3}{x_2 - x_3} + 0.615\frac{x_3}{x_2} \\ F_{max} & = & 1000 \\ S & = & 189000 \\ l_f & = & \frac{F_{max}}{K} + 1.05\,(x_1 + 2)\,x_3 \\ l_{max} & = & 14 \\ \sigma_p & = & \frac{F_p}{K} \\ \sigma_{pm} & = & 6 \\ F_p & = & 300 \\ K & = & 11.5 \times 10^6 \frac{x_3^4}{8 x_1 x_2^3} \\ \sigma_w & = & 1.25 \end{array}$$

and the function to minimise is

$$f(x) = \pi^2 \frac{x_2 x_3^2\,(x_1 + 1)}{4}$$

The best known solution is $(7, 1.386599591, 0.292)$ which gives the fitness value $2.62542145 78$. To take the constraints into account, a penalty method is used. In this study, the maximum number of evaluations is 20,000.

4.2.4 Gear Train

For more details, see[9, 7]. The function to minimise is

$$f(x) = \left(\frac{1}{\beta} - \frac{x_1 x_2}{x_3 x_4}\right)^{\gamma}$$

The search space is $\{12, 13, \ldots, 60\}^4$. In the original problem, $\beta = 6.931$, and $\gamma = 2$. There are several solutions, depending on the required precision. For example $f(19, 16, 43, 49) =$

2.7×10^{-12}. So, if we set the objective value to zero and the acceptable error to 10^{-11}, any run that finds this solution is successful.

4.2.5 Pressure Vessel

Just in short. For more details, see[9, 2, 7]. There are four variables

$$\begin{array}{lll} x_1 \in [1.125, 12.5] & \text{granularity} & 0.0625 \\ x_2 \in [0.625, 12.5] & \text{granularity} & 0.0625 \\ x_3 \in]0, 240] & & \\ x_4 \in]0, 240] & & \end{array}$$

and three constraints

$$\begin{array}{lll} g_1 & := & 0.0193x_3 - x_1 \leq 0 \\ g_2 & := & 0.00954x_3 - x_2 \leq 0 \\ g_3 & := & 750 \times 1728 - \pi x_3^2 \left(x_4 + \frac{4}{3}x_3\right) \leq 0 \end{array}$$

The function to minimise is

$$f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + x_1^2 (3.1611x_4 + 19.84x_3)$$

The analytical solution is $(1.125, 0.625, 58.2901554, 43.6926562)$which gives the fitness value 7,197.72893. To take the constraints into account, a penalty method is used.

4.2.6 Lennard-Jones

For more details, see for example [3]. The function to minimise is a kind of potential energy of a set of N atoms. The position X_i of atom i has three co-ordinates, and therefore the dimension of the search space is $3N$. In practice, the coordinates of a point x are found by simply writing the coordinates of all the atoms $X_i$in order. In short, we can write $x = (X_1, X_2, \ldots, X_N)$, and we then have

$$f(x) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(\frac{1}{\|X_i - X_j\|^{2\alpha}} - \frac{1}{\|X_i - X_j\|^{\alpha}} \right)$$

In this study $N = 5$, $\alpha = 6$, and the search space is $[-2, 2]^{15}$.

Références

[1] Maurice Clerc. Math Stuff about PSO, http ://clerc.maurice.free.fr/pso/.

[2] Maurice Clerc. *Particle Swarm Optimization.* ISTE (International Scientific and Technical Encyclopedia), 2006.

[3] S. Das and P. N. Suganthan. Problem Definitions and Evaluation Criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems. Technical report, Jadavpur University, Nanyang Technological University, 2010.

[4] G. Marsaglia and A. Zaman. The KISS generator. Technical report, Dept. of Statistics, U. of Florida, 1993.

[5] Rui Mendes. *Population Topologies and Their Influence in Particle Swarm Performance*. PhD thesis, Universidade do Minho, 2004.

[6] Christopher K. Monson and Kevin D. Seppi. Exposing Origin-Seeking Bias in PSO. In *GECCO'05*, pages 241–248, Washington, DC, USA, 2005.

[7] Godfrey C. Onwubolu and B. V. Babu. *New Optimization Techniques in Engineering*. Springer, Berlin, Germany, 2004.

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

[9] E. Sandgren. Non linear integer and discrete programming in mechanical design optimization, 1990. ISSN 0305-2154.

[10] William M. Spears, Derek T. Green, and Diana F. Spears. Biases in particle swarm optimization. *International Journal of Swarm Intelligence Research*, 1(2) :34–57, 2010.