Simple Generators for List-Based Optimisers

Maurice Clerc

Simple Generators for List-Based Optimisers

Maurice Clerc

11th June 2014

Abstract

A coded random number generator (RNG) is in fact a generator of a list of numbers (LNG), and any algorithm that uses such a list is deterministic. Herem, we present some experiments that support two claims when the algorithm is an optimiser: namely, a) there is no need of a complicated LNG, and b) the generated list can be very short, so that all possible different runs can be executed, and therefore there is no need of statistical analysis to evaluate and compare the results.

1 Introduction

We consider here the so called "stochastic" optimisers, i.e. optimisers that theoretically make use of a Random Number Generator (RNG). In a previous paper [3], we have explained why, for a given algorithm, using different coded RNGs lead to different performances, making comparisons difficult. In another paper [4], we introduced the concept of *list-based optimisers*, in which the RNG is replaced by a finite list of predefined *l-random* numbers. We have seen that such a list can sometimes be extremely short, with still good performances and that there is in fact no binary distinction between "stochastic" and "deterministic". Rather, it is more a degree of stochasticity. From this point of view, only some hardware systems (like quantic ones) have the maximum degree [2]. That is why, in this paper, we will not talk about RNGs but about LNGs (List Number Generators).

In the following, we present three simple and practical ways to produce interesting lists, illustrated by experimental results. Some mathematical developments are given in the Appendix .

2 Three simple list generators

For the sake of simplicity, we assume here that we want to generate numbers in $]0,1[$. We already have given some methods in [4], in particular one which is based on an irrational number (called *seed* below). We call it the Additive method. Here we will present two other methods, also based on an irrational seed, namely Multiplicative Method, and HP method[1]. The lists generated by Additive and HP are *equidistributed modulo 1* (see the Appendix 6.3) but, as we will see, it does not mean that they are always "better" than Multiplicative. For these three generators, the generated sequences are dense on $]0,1[$, and we never have the same number twice (see the Appendix 6.4).

The first 100 l-numbers generated by these three methods are plotted on circular diagrams in the Figure 1, along with the numbers generated by the Mersenne-Twister algorithm [7], which is commonly used in the "stochastic" optimisers.

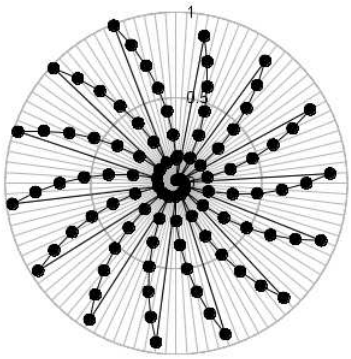
Note that contrary to what is sometimes claimed, all classical LNGs, like KISS [6] or Mersenne-Twister, generate a list that is not equidistributed, and not even dense on $]0,1[$. Also, those are repetitive. There is a simple reason: the lists are finite (for Mersenne-Twister, its size is $2^{19937}-1$). It means that in some intervals there are no generated numbers at all. Of course, for "good" generators all these "empty" intervals are very small, and for most computers is does not make a big difference, if any. With the far simpler methods that we present here there are no such gaps, the sequences are dense, and as we will see, they are perfectly usable in the context of optimisation. On the other hand, such simplistic deterministic methods are obviously not acceptable in some other domains, like, say, cryptography.

3 Experiments with PSO

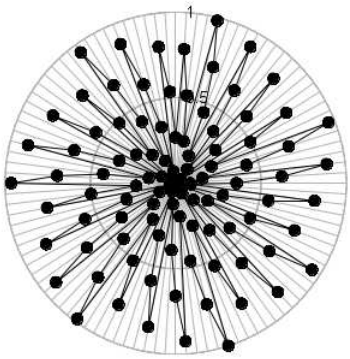
3.1 Infinite lists

The principle of a list-based optimiser is to replace a classical LNG by a *short* list of l-random numbers, which is used cyclically. However, it would be interesting to compare the performances when we let the algorithm generate as many

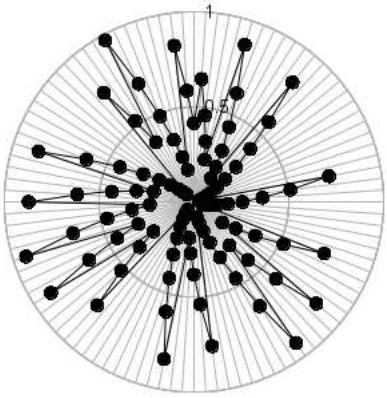
[1] It has been used in some hand-held programmable scientific/engineering calculator made by Hewlett-Packard around 1980, with the seed π.



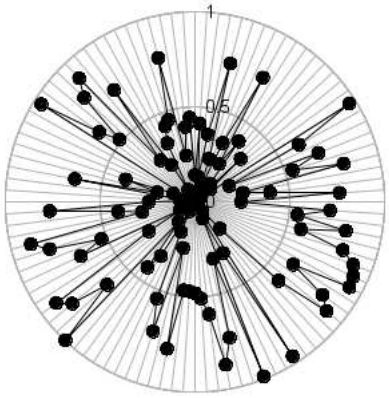
(a) Additive(π)



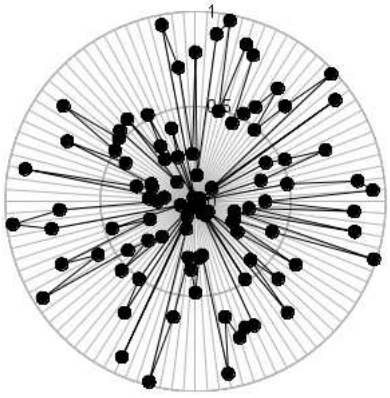
(b) Additive($\sin(1)/\pi$)



(c) Multiplicative($\sqrt{2}$)



(d) Multiplicative(π)



(e) Mersenne-Twister

Figure 1: 100 successive generated numbers plotted on circular diagrams.

Table 1: Three generators. The number α must be irrational. $\{x\}$ denotes the fractional part of x.

Additive(α)	$r(1) = \alpha$ $r(t+1) = \{r(t) + \alpha\}$
Multiplicative(α)	$r(1) = \alpha > 1$ $r(t+1) = \{\alpha r(t)\}$
HP(α)	$r(1) = \{\alpha^5\}$ $r(t+1) = \left\{(\alpha + r(t))^5\right\}$

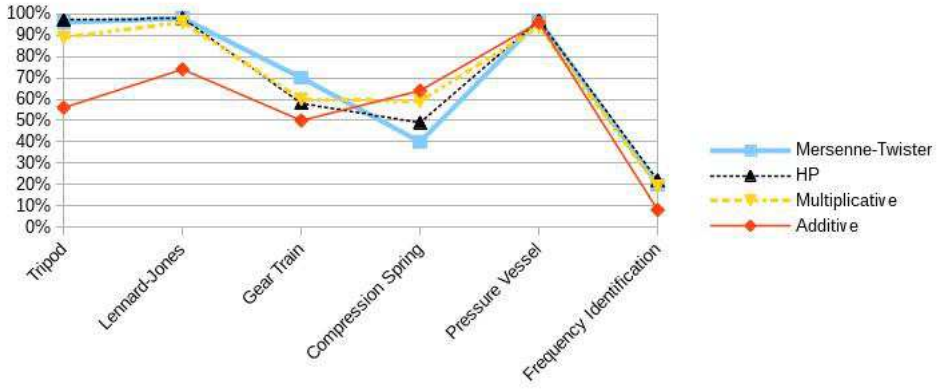
numbers as needed. We use here a List-based Particle Swarm Optimiser, a simplified version of Standard PSO 2007, in which, in particular, the variable topology has been replaced by the fixed Ring one (more details about the differences are given in the Appendix 6.1). We mainly use graphical comparisons, for they are more illustrative than big tables of numbers. The success rate has the usual meaning, and the accuracy levels are given in the Appendix 6.2 (here "optimisation" means "minimisation"). The *normalised mean best* magnifies the differences, and is computed as follows. Suppose we are using n LNGs, and for each function we use K runs, say $K = 100$. Then, for a given function we get n mean bests $m_i, i = 1 \dots n$, one for each LNG. Let m be the smallest one, and M the biggest one. Then each normalised mean best m'_i is given by

$$m'_i = 1 - \frac{m_i - m}{M - m} \tag{1}$$

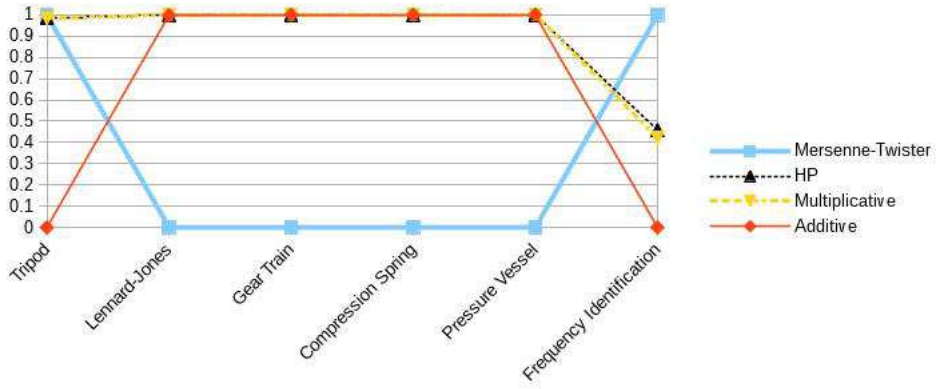
So, the value 1 means "the best", and 0 means "the worst". Let us first consider six low dimensional problems (the detailed description is in [4]). As we can see from Figure 2, Mersenne-Twister is equivalent to Multiplicative and HP for the success rate, but is outperformed with respect to the normalised mean best. Additive is not very bad, but not as good as the others.

Let us now try six more difficult problems (dimension 50, and shifted), taken from the CEC 2008 benchmark [10]. As the figures 3 show, the success rates are not very different, but for the normalised mean best, Multiplicative outperforms the other ones (except HP on just one problem, namely, Griewank). Note that the authors of Mersenne-Twister have coded it in C with about 75 lines. For Multiplicative, we need three lines. Hence our first conclusion:

Claim 1: For optimisation, a very simple LNG can be better than a complicated one.

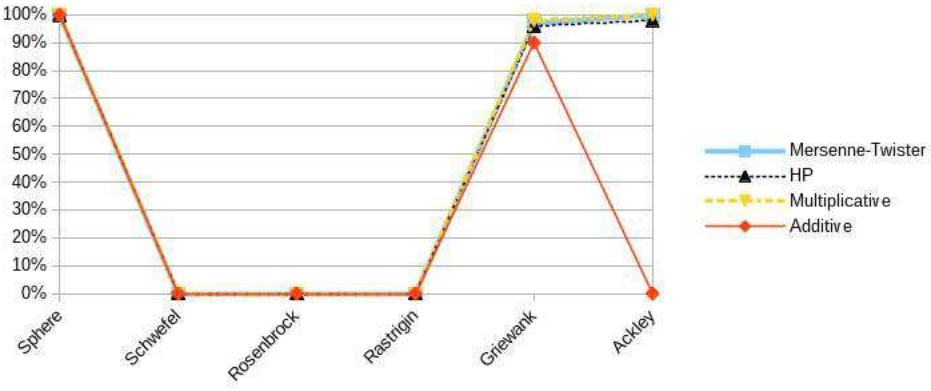


(a) Success rate.

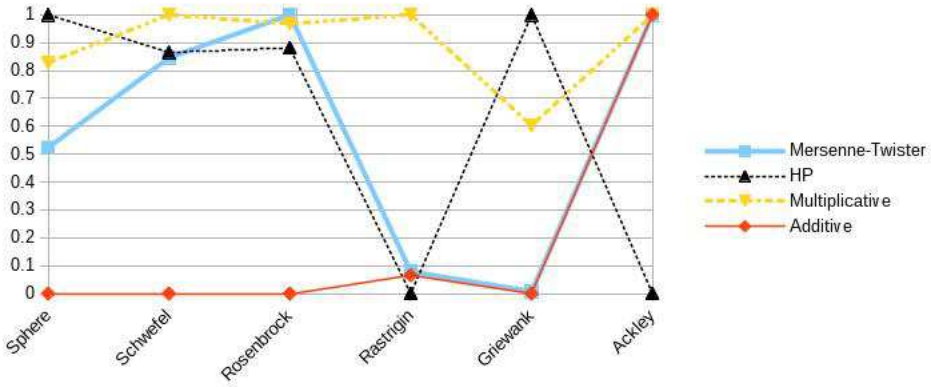


(b) Normalised mean best.

Figure 2: **Infinite lists. Low dimension problems. Comparison of the use of four LNGs by List-based PSO.**



(a) Success rate.



(b) Normalised mean best.

Figure 3: **Infinite lists. 50D problems. Comparison of the use of four LNGs by List-based PSO.**

3.2 Short lists

Although the process is completely deterministic when infinite lists are used (or huge ones, for example with Mersenne-Twister), the results can only be statistical estimations[2] because we can not perform all possible runs. If we execute the algorithm 100 times (without reinitialising the LNG, of course), we can not be *sure* that the results would be similar to that of 1000 runs, or even 10,000. That is why the idea of using a short list cyclically is so tempting: if the size of the list is N, only N different runs are possible (as explained in [4]). Therefore, not only the process is of course still deterministic, but moreover no statistical analysis is needed any more. Let us explain what it means, for example, for the success rate, or, more precisely, for the probability of success s, i.e. the probability that a given run is successful.

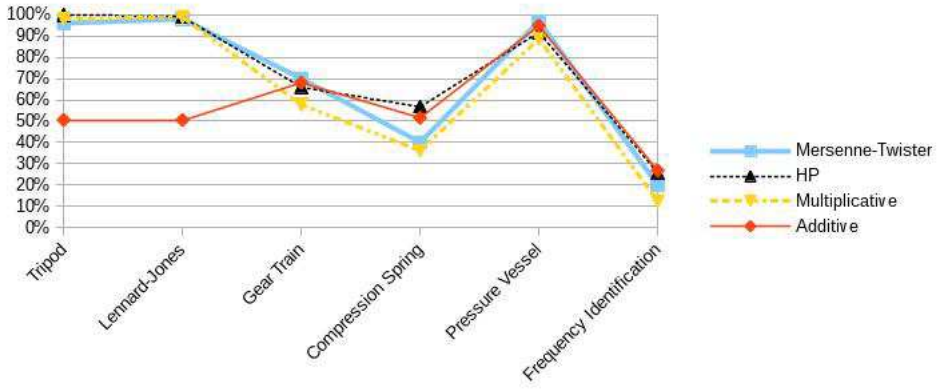
If the list is infinite, after N runs the probability that at least one is successful is $1-(1-s)^N$. If s is small, say 0.02, then even after 100 runs, the probability of success is only 0.87. To increase it, we have to increase the number of runs. However, with a finite list of size N, we do not need to do that. After N runs, we have a definite non-probabilistic measure of the performance, i.e. how many runs are successful, and how many would be for any number of runs greater than N, for anyway any extra run would be identical to at least one of the first N runs.

So, we have to check whether such short lists can induce good performances or not. Let us give here some examples, on the same test functions as above. For Mersenne-Twister, we just re-use the results of Figures 2 and 3, for easier comparison. For the others, the size of the list is set to 97, a prime number close to 100. Why we use a prime number is explained in the Appendix 6.1. It is not absolutely nececessary, but it usually gives better results.

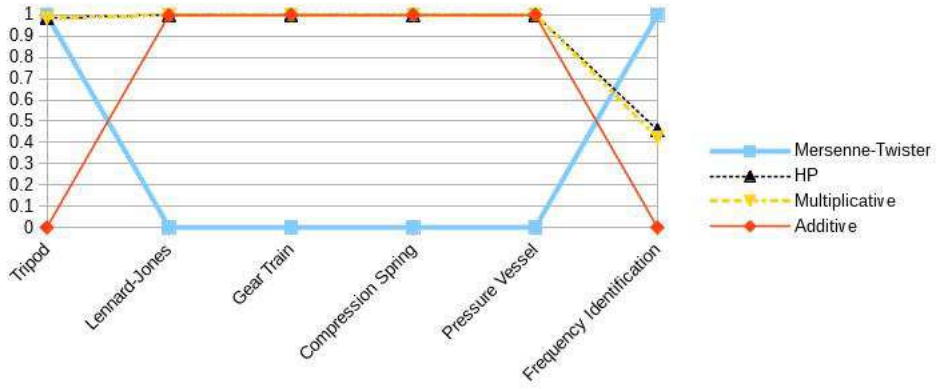
From Figure 4, we can see that the success rates of HP and Multiplicative are still better than or equivalent to Mersenne-Twister. [3]. For the normalised mean best, the figure looks exactly the same (actually there are some differences

[2]From a purely theoretical point of view, it may be worth noting that most statistical analysis methods are not usable, for they do assume that the runs are independent. However, they are not, precisely because the LNG is deterministic.

[3]Although the classical statistical analysis is not really valid here, we do assume that two success rates can be said to be different at the significance level 0.05 only if the difference is greater than 3/number_of_runs [**?**].



(a) Success rate.



(b) Normalised mean best.

Figure 4: **Low dimension problems. List size=97 (except for Mersenne-Twister). List-based PSO.**

but too small to be visible). For the 50D problems, again, HP and Multiplicative are still better. However, Adaptive is now clearly the worst. Nevertheless, we reach our second conclusion:

> *Claim 2*: For optimisation, using short lists cyclically can be equivalent to or even better than using a complicated LNG.

But there is still an issue. What should be the list size?

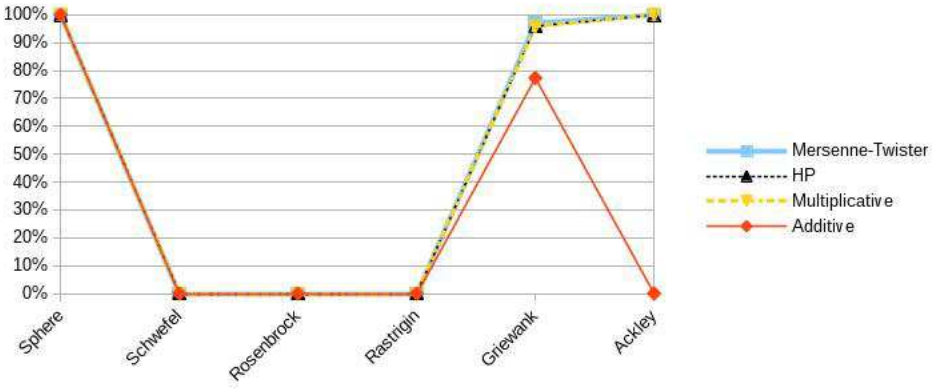
3.3 About the list size

We can try different list sizes, and compare the results. We omit the details here and instead give only a global indicator: the mean success rate over our six low dimension problems. It is clear from Figure 6 that the list size should not be "too small", and that there are a few annoying discontinuities. Such discontinuities are due to the fact that the list size L and the swarm size S may be commensurable (i.e. S/L is a rational number). For this reason, to avoid such cases, our List-based PSO gives an option that automatically defines the list size depending on the swarm size, (see the Appendix 6.1). If we apply it, the performance for the best simple generator, Multiplicative, is slightly improved, as we can see for example from Figure 7.

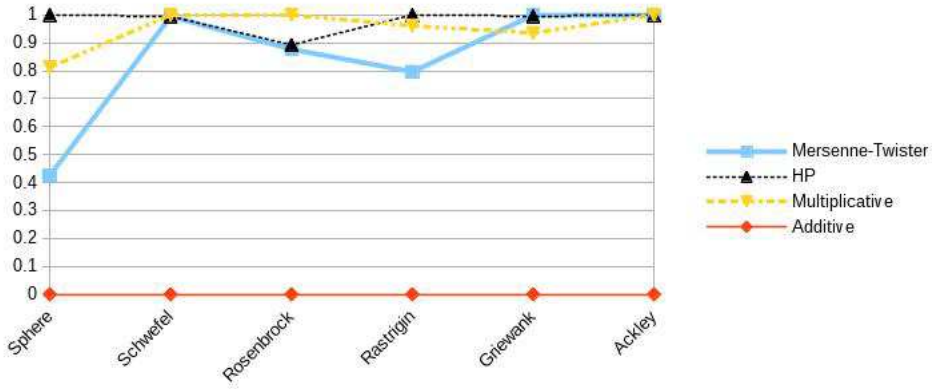
4 Experiments with APS

The good point with our List-based PSO is that the l-numbers are used only to generate the displacements of the particles. For this reason, relatively small lists are enough for good performances. However, the Adaptive List-based Simplex method [1] theoretically makes use of randomness for three very different tasks:

- to select a simplex amongst the whole population;



(a) Success rate.



(b) Normalised mean best.

Figure 5: **50D CEC 2008 problems. List size=97 (except for Mersenne-Twister). List-based PSO.**

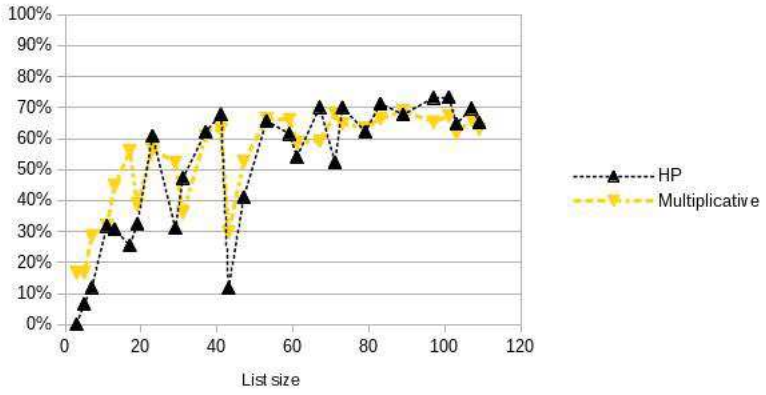


Figure 6: **Mean success rate vs list size. Low dimension problems. List-based PSO.**

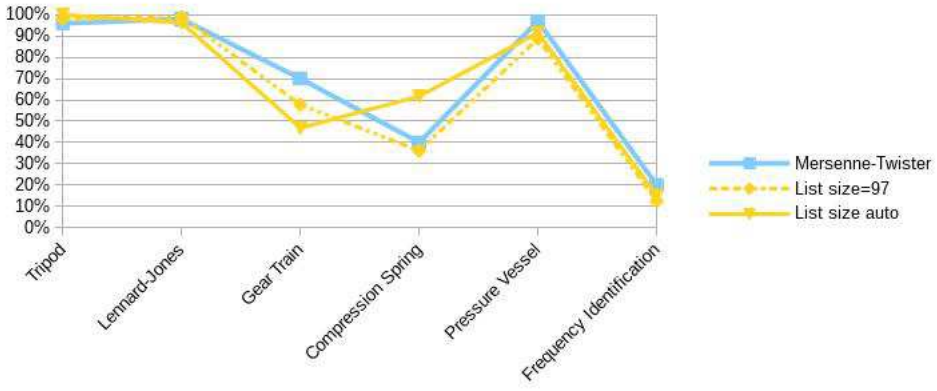


Figure 7: **Low dimension problems. Success rate with automatically defined list size. Multiplicative generator. List-based PSO.**

Table 2: **Success rate vs very small list size (in bold for the smallest possible one, i.e. D+1). Low dimension problems. List-based APS.**

	3	5	7	17
Tripod	100 %	0 %	28.57 %	5.88 %
Lennard-Jones				41.18 %
Gear Train		0 %	0 %	23.53 %
Compression Spring		20 %	0 %	23.53 %
Pressure Vessel		20 %	0 %	41.18 %
Frequency Identification		0 %	0 %	11.76 %

- to define an adaptive probability threshold;
- to perform a local search inside a hypersphere (and for this task Gaussian "random" numbers are required .

With infinite lists, in terms of performance, Multiplicative and HP are still equivalent to or even better than Mersenne-Twister, as shown in the Figures 8 and 9. However, because of the above three roles, a correct size for a finite list has to be greater than the one used with our List-based PSO. Even for low dimension problems, it has to be about 180 (more precisely 179, as we use prime numbers), as we can see from Figure 10. Note though that for some problems, it may happen that a very short list is enough (see Tripod in the Table 2), but there is apparently no way to predict such perfect performances.

Also, the automatic list size mechanism is not as efficient as with List-based PSO (see the Figures 12). However, for both low dimension and 50D problems, with a list size of 179, HP and Multiplicative are stil more or less equivalent to Mersenne-Twister (more precisely, both slightly worse, and HP slightly better than Multiplicative). Note that Additive is now not that bad. The reason for that is not clear for the moment.

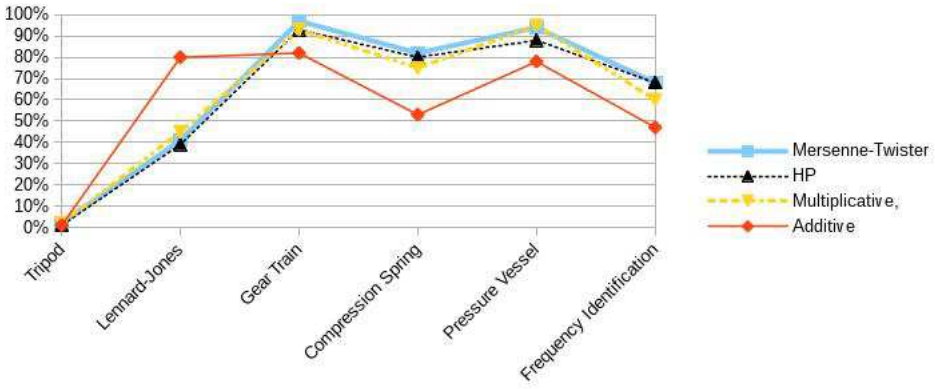
5 Using another seed

An obvious point of interest is whether using different seeds leads to different results. Experimentally, as illustrated by the Figures 14 and 15, transcendental numbers seem to be slightly better than just irrational ones, and amongst them, π seems to be a good choice. But of course, this needs a theoretical analysis or, at least, more experiments with more different seeds.

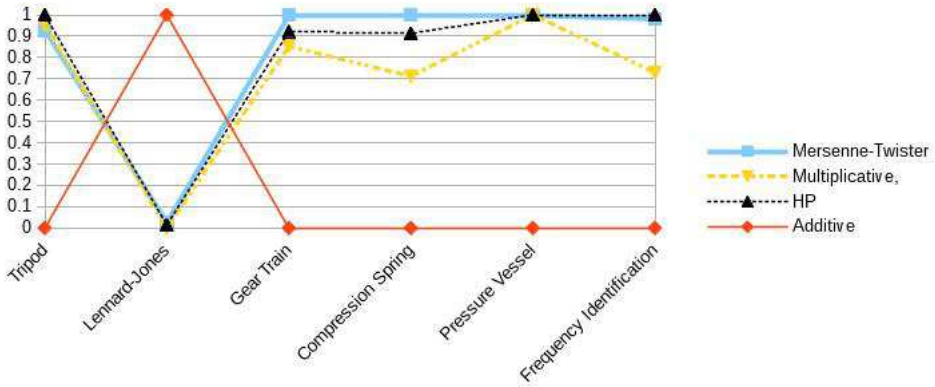
6 Appendix

6.1 From SPSO 2007 to List-based PSO

As said before, to define the List-based PSO used here we started from SPSO 2007; in particular from the C version, available on the Particle Swarm Central [8]. There are a few differences, though:



(a) Success rate.



(b) Normalised mean best.

Figure 8: **Infinite lists. Low dimension problems. Comparison of the use of four LNGs by List-based APS.**

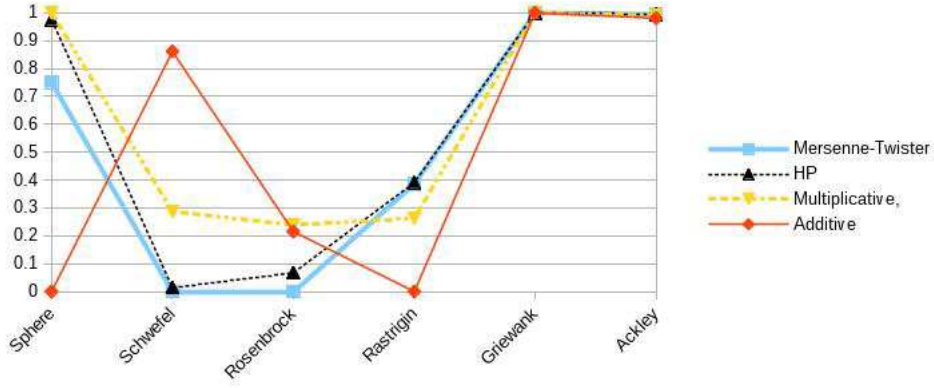


Figure 9: **Infinite lists. 50D problems. Normalised mean best. Comparison of the use of four LNGs by List-based APS.**

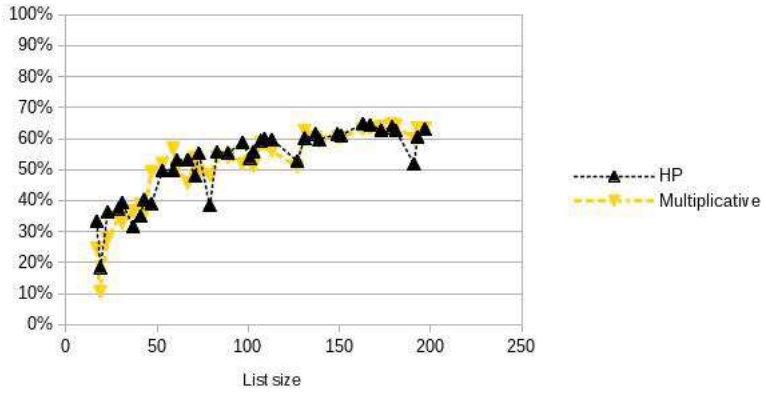
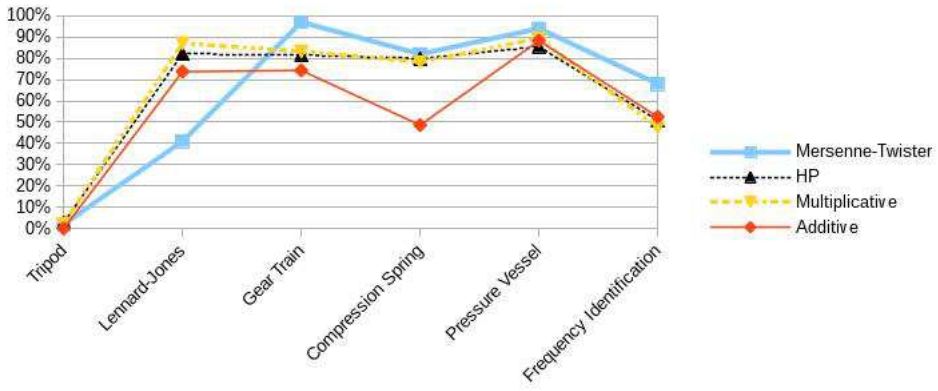
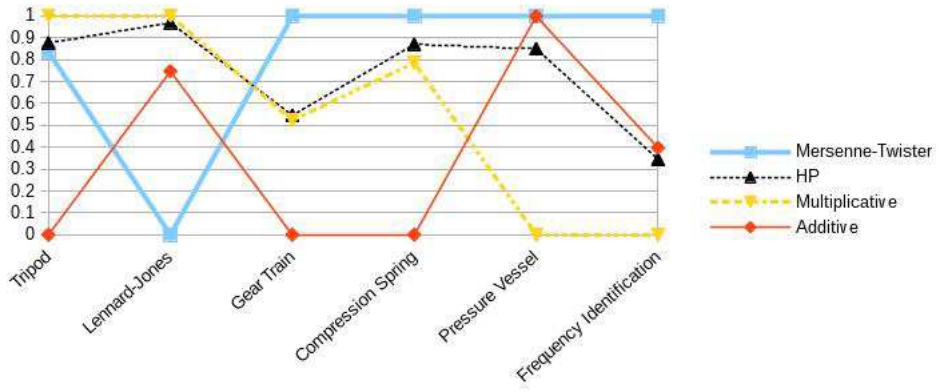


Figure 10: **Mean success rate vs list size. Low dimension problems. List-based APS.**



(a) Success rate.



(b) Normalised mean best.

Figure 11: **Low dimension problems. List size=179 (except for Mersenne-Twister). List-based APS.**

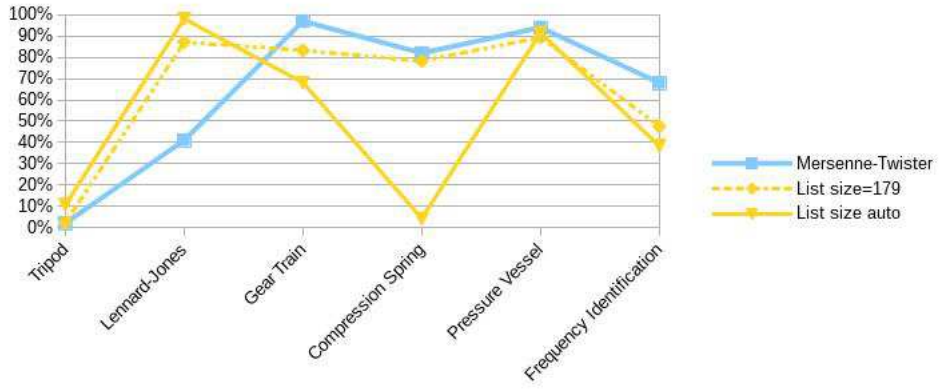


Figure 12: **Low dimension problems. Success rate with automatically defined list size. Multiplicative generator. List-based APS.**

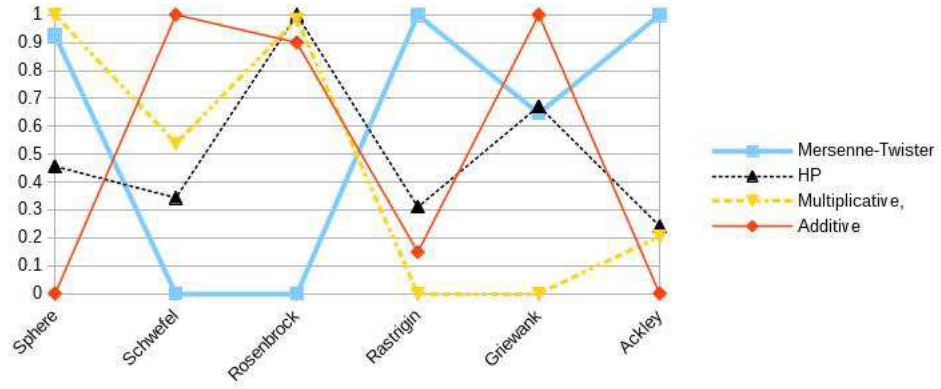


Figure 13: **50D CEC 2008 problems. List size=179 (except for Mersenne-Twister). List-based APS.**

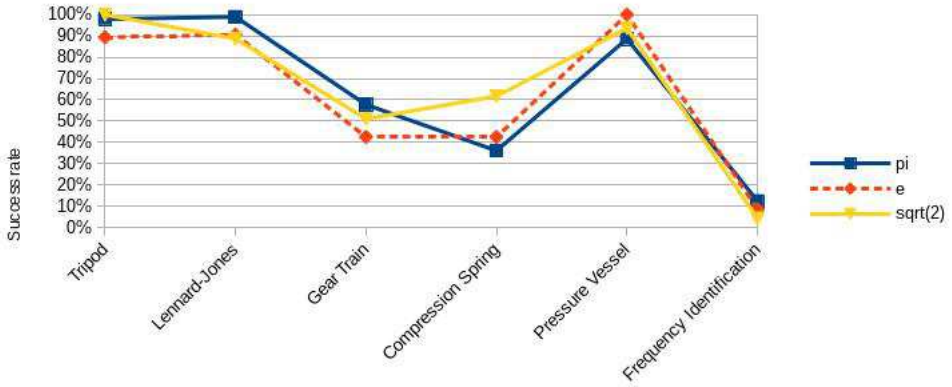


Figure 14: **Low D. List size=97. Multiplicative. List-based PSO.**

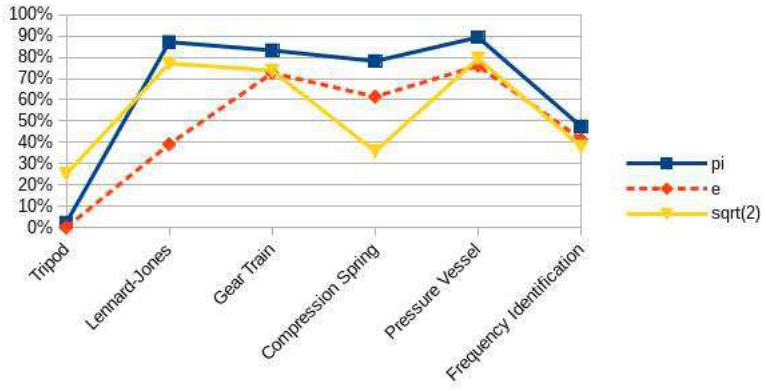


Figure 15: **Low D. List size=179. Multiplicative. List-based APS.**

Table 3: Test problems.

Problem	D	FE_{max}	*Accuracy*
Tripod	2	5000	1.00E-004
Lennard-Jones	15	30000	1.00E-002
Gear Train	4	20000	1.00E-010
Compression Spring	3	20000	1.00E-010
Pressure Vessel	4	30000	1.00E-006
Frequency Identification	6	50000	1.00E-006
All CEC 2008 problems	50	250000	1.00E-010

- for simplicity, the topology is the classical Ring one, instead of the adaptive one used in SPSO 2007;
- the initial velocity of each particle is set to zero;
- the formula that estimates the swarm size S as a function of the dimension D is modified, and is now $S = \max\left(20 + 2\sqrt{D}, \sqrt{40^2 + (D+2)^2}\right)$. Also the result S can optionally be forced to be "the prime number that is the nearest one to S" (and if there are two possible numbers, like 17 or 19 for $S = 18$, the smallest one is chosen);
- there is an option that automatically computes the list size as the smallest prime number greater than S;
- and, of course, different LNGs can be used, either of "infinite" size or finite. In that case, the number of runs is automatically set to list size, and each run starts from a different number.

For finite lists, it experimentally appears that using prime numbers both for the list size and the swarm size usually improves the performance slightly (not always, though). This is because in our List-based PSO each iteration (including initialisation) needs $2D$ "random" numbers, where D is the dimension of the search space. Therefore, if S is a multiple of $2D$, the velocity update formula of each particle uses exactly the same numbers at each iteration. Said differently, each particle has always the same behaviour, and it implies a lack of diversity. Using prime numbers reduces this risk.

6.2 Problems

The detailed description of the problems used here can be found in [4] for the low dimension ones, and in [10] for the 50D ones. Here, we just give the dimension, the maximum number of fitness evaluations, and the wanted accuracy level for each one of those.

6.3 Equidistribution modulo 1

As said, the list generated by Additive is equidistributed modulo 1.

Note that there is nothing new here. In particular, an important equidistribution theorem has been independently proved by Bohl, Sierpinski and Weyl in 1909-1910. It is nevertheless usually called Weyl's criterion, according to the more

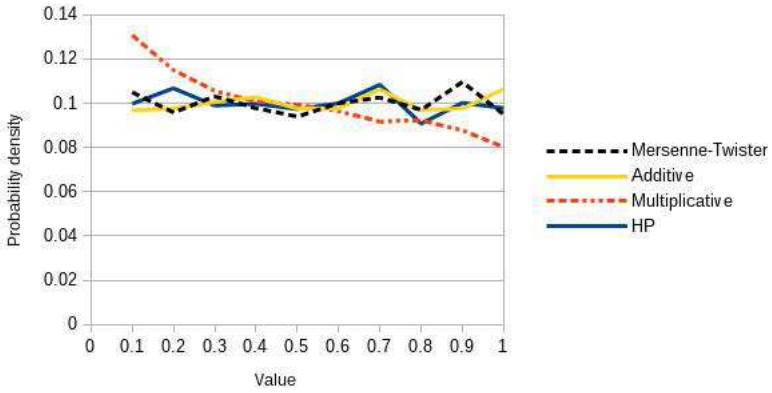


Figure 16: **Estimated distributions over 5000 generated numbers, for the seed π (except for Mersenne-Twister). HP seems to be equidistributed, whereas Multiplicative is clearly not. Additive is theoretically equidistributed. Mersenne-Twister too seems to be so in practice, on the 32-bit computer that has been used.**

Table 4: Geometrical and Multiplicative sequences are not equivalent, here for the seed π.

t	Geometrical	Multiplicative
1	0.1416	0.1416
2	0.8696	0.4448
3	0.0063	0.3975
4	0.4091	0.2487
5	0.0197	0.7812

complete 1916 paper by this author [11]. Let us explain the equidistribution of the Additive generator with a seed α, whose algorithm is given in the Table 1.

In short, the statement is that the sequence α, 2α, 3α,... mod 1 is uniformly distributed on the unit interval if and only if α is irrational. What does it mean? Let $\{\alpha\}$ be the fractional part of the number α. For example, if $\alpha = \pi$, then $\{\alpha\} = 0.1415926535\ldots$. The list that we consider is in fact $\{\alpha\}$, $\{2\alpha\}$, $\{3\alpha\}$, ... Let us suppose we have generated n numbers. Roughly speaking, the equidistribution property is that for any interval $[a, b]$ of $[0, 1]$, the "proportion" of the numbers generated in this interval tends to $b - a$ when n increases. If we consider these numbers as occurrences from an underlying distribution, it immediately implies that this distribution has the same moments to that of the random uniform distribution, in particular the mean and the variance**.**

One may think that our Multiplicative method generates the geometrical sequence $g(t) = \{\alpha^t\}$, and should therefore be equidistributed [5, 9]. However, it is not the case with the seed π (see the slightly decreasing distribution curve on the Figure 16). This is because the fractional operator is not applied the same way. Indeed, for Multiplicative we have $r(1) = \{\alpha\} = g(1))$, but then $g(2) = \{\alpha^2\}$, whereas $r(2) = \{\alpha\{\alpha\}\}$. Clearly, if $\alpha > 1$, the result is not the same as $\{\alpha^2\}$, as we can see from Table 4. Note though, that for some big seed values, say 100π, the distribution (not represented here) is very similar to a uniform one.

For Additive the sequence is the same as $\{t\alpha\}$ (easy to prove by induction), and that is why Weyl's theorem is valid in this case.

6.4 Non repetitivity and denseness

For these three methods, two interesting properties are true or seem to be, assuming the seed is correctly chosen:

1. all generated numbers are different, and therefore the generated sequence is infinite (proved);
2. the generated sequence is dense on$[0, 1]$ (proved only for Additive).

6.4.1 Additive

As it is equidistributed, it is necessarily dense. Now, let us suppose that it is repetitive. Then for two integers k and k', with $k' > k$, $\{k'\alpha\} = \{k\alpha\}$. It implies $\{(k' - k)\alpha\} = 0$, which is impossible for α is irrational. Actually, another way is

to simply note that if it were repetitive, the number of different generated values would be finite, which is not compatible with denseness.

6.4.2 Multiplicative

Let us prove by induction that $r(t)$ is a polynomial in α. It is true for $t=1$, for $r(1)=\{\alpha\}=\alpha-k(1)$, where $k(1)$ is a positive integer (for α is greater than 1). Now, let us suppose that $r(t)=P_t(\alpha)$, where P_t is a polynomial of degree t. Let $\lfloor u \rfloor$ be the integer part of u. We have then $r(t+1)=\{\alpha P_t(\alpha)\}=\alpha P_t(\alpha)-\lfloor \alpha P_t(\alpha) \rfloor = \alpha P_t(\alpha)-k(t+1)=P_{t+1}(\alpha)$, where P_{t+1} is a polynomial of degree $t+1$, which proves the claim. Actually, it is easy to see that the exact formula is

$$r(t)=P_t(\alpha)=\alpha^t-\sum_{i=1}^{t} k(i)\,\alpha^{t-i} \tag{2}$$

in which all $k(i)$ are non negative integers. More precisely, we have

$$\left\{ \begin{array}{rcl} k(1) & = & \lfloor \alpha \rfloor \\ k(i+1) & = & \lfloor \alpha r(i-1) \rfloor \end{array} \right. \tag{3}$$

Note that it implies we always have $k(i) \leq \lfloor \alpha \rfloor$ (one could even prove that we will indeed achieve the strict inequality $k(i)<\alpha$ sometimes; for, if not, $r(t)$ would be negative as soon as t is "big enough"). What is important here is that if the sequence were repetitive, for two integers t and t', with $t'>t$, we would have $P_{t'}(\alpha)-P_t(\alpha)=Q_{t',t}(\alpha)=0$, where $Q_{t',t}$ is a polynom of degree t'. If α is transcendental (like π, e, or $\sin(1)$), it is impossible, and therefore there is no repetition. Note that with just an irrational seed α, the sequence can easily be repetitive. For example, if α is the positive root of x^3-x^2-1, i.e.

$$\frac{1}{3}\left(1+\sqrt[3]{\frac{29-3\sqrt{93}}{2}}+\sqrt[3]{\frac{23+3\sqrt{93}}{2}}\right) \simeq 1.4655712319 \tag{4}$$

then for $t \geq 3$ all $r(t)$ are null. This of course happens because we precisely have $P_3(\alpha)=\alpha^3-\alpha^2-1$. Similarly, if α is the positive root of x^4-3x^3-x-1, i.e.

$$\frac{1}{4}\left(3+\sqrt{5}+2\sqrt{\frac{11+7\sqrt{5}}{2}}\right) \simeq 3.1342729984 \tag{5}$$

then for $t \geq 4$ all $r(t)$ are null.

Let us now prove that the sequence is dense on $]0,1[$. On the other hand, if we call *M-number* any α for which the sequence is finite, experiments suggest some conjectures:

- The number of M-numbers is infinite.
- However the measure of their set is null. In practice, it means that you can use almost any non-integer α greater than 1.
- If α is not a M-number, then the sequence is dense on $[0,1]$.
- When α increases, the distribution of the sequence tends to the uniform one. When α is small, say 1.1, the distribution is quickly decreasing on $[\{\alpha\},1]$, a property which may be sometimes useful.

6.4.3 HP

The general form of this generator is $r(t+1)=\left\{(\alpha+r(t))^{\beta}\right\}$, where β is an integer. As for Multiplicative, a repetition would imply $r(k')-r(k)=P(\alpha)=0$, where P is a polynomial, which is impossible if α is transcendental. The proof of denseness is also similar to the one for Multiplicative. ***** TO COMPLETE *****

Again, if α is just irrational, the sequence may easily be repetitive. A trivial example is $\alpha=n^{1/\beta}$, where n is an integer. In such a case, all $r(t)$ are equal to zero. But as for Multiplicative, experiments suggest that almost any non-integer α generates a sequence that *seems* to be non repetitive and dense on $[0,1]$.

References

[1] APS. Adaptive population-based simplex (http://aps-optim.info/mediawiki/index.php?title=welcome).

[2] Cristian S. Calude, Michael J. Dinneen, Monica Dumitrescu, and Karl Svozil. How random is quantum randomness? an experimental approach. *arXiv:0912.4379 [quant-ph]*, December 2009.

[3] Maurice Clerc. Randomness matters (http://hal.archives-ouvertes.fr/hal-00764990). Technical report, May 2012. 14 pages.

[4] Maurice Clerc. List based optimisers - experiments and open questions. *International Journal of Swarm Intelligence Research*, 4(4), 2014.

[5] Jurjen Ferdinand Koksma. Ein Mengentheoretischer Satz Über die Gleichverteilung modulo Eins. *Compos. Math.*, 2:250–258, 1935.

[6] George Marsaglia. KISS PRNG. 2011.

[7] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8 (1):3–30, 1998.

[8] PSC. Particle swarm central (http://particleswarm.info).

[9] Georges Rhin. Deux théorèmes sur l'équirépartition modulo 1 de suites fn (θ). volume 9 (2), 1967.

[10] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang. Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. Technical report, 2008.

[11] Hermann Weyl. Über die Gleichverteilung von Zahlen mod. Eins. *Math. Ann. 77 (3)*, pages 313–352, 1916.