List Based PSO for Real Problems

Maurice Clerc

List Based PSO for Real Problems

Maurice Clerc

5th May 2012

Note: this is a short version, put online on my site [1], of a more complete paper.

1 Motivation

In the real world, engineers and practitioners often have to solve the same kind of problem, again and again, with just small variations. Also, for some applications, a hardware implementation is needed, which should ideally be small, quick, and deterministic. We show here that is possible to transform a basic Particle Swarm Optimiser into an even simpler one that has these three features.

In order to do that, we start from the concept of *list based optimiser*.

2 List Based Optimisers

In most of stochastic optimisers, we only need random numbers r that are in $[0, 1]$, even if sometimes we have to linearly transform them by a formula like $a + r\,(b - a)$. Now, let us suppose we have a predefined list of numbers in $[0, 1]$, say $L = (r_1, r_2 \cdots, r_n)$. During the optimisation process, wheneveer we need a "random" number, we pick it sequentially and cyclically in L, i.e. we pick r_1, then r_2, ..., then r_n, then again r_1, etc. Obviously the resulting process is completely deterministic. Actually, this is already the case with a random number generator (RNG) like KISS [4] or the ones that are embedded in a language like C, at least if we always keep the same seed. However, the idea here is to reduce as much as possible the length of the list L, and on the same time to improve the performance. So, we will speak of List Based Optimiser (LBO) only when the number of "random" number that is used is relatively small (typically at most a few hundreds for a 30D problem). Experimental results suggest the following

Conjecture 1. *For any problem, for any performance measure, and for any stochastic algorithm that needs only bounded random numbers, there exists a deterministic algorithm that is better*

The length of the list L can be extremely short. For example, for the Tripod problem (4.2.1), which is only two dimensional (but nevertheless not that easy), and for a basic PSO, the minimal size is probably 4. A possible such "magic" list is

$$
\begin{aligned}
L_4 \quad = \quad & (0.66636005245184826151, \\
& 0.48627235377349220524, \\
& 0.30526339730324419941, \\
& 0.00779071032578351665)
\end{aligned}
$$

With this list, the success rate is 100% over 100 runs, as with a better RNG like KISS it is slightly smaller (97%). Some other lists are of course possible (in particular with no so many digits). But what is really interesting is that the *same* list is usable for a lot of problems of the same dimension (see column L_{17} of the table 1. As said, we replace then a stochastic algorithm by a list based one, which is deterministic. Of course, the main difficulty is to build the lists, as few as possible, as short as possible, but that nevertheless give good results.

In the table 1 we use such candidate lists for five well known quasi-real-world problems. It is probably possible to find shorter ones but this is an open question.

3 List Based Classical PSO

For real-world ones, the engineers, and more generally the practitioners, usually want "a reasonably good solution, in a reasonable time, for a reasonable cost". Moreover, they often have to solve the same kind of problems, again and again. A list based optimiser may be then very useful for at least three reasons:

- deterministic
- quick
- can be easily embedded in a hardware optimisation device

The first feature is ensured by transforming a classical PSO into a list based one. For the second and third features, the algorithm has then be simplified as much as possible. The source code, is also available on my PSO site [1]. In short, the main points are:

- no RNG, but a list of numbers. However, for comparison, the RNG KISS has been implemented in the code;
- only the old classical ring topology (not a variable one like in the most recent PSO versions);
- velocity update dimension by dimension (no rotationally invariant as in SPSO 2011 [6], but, as said, it is not really an issue here).

The table 1 gives the results for five classical quasi-real-world problems. On the one hand, one may note that there is no relationship between the dimension and the list size, but, on the other hand, nothing proves that the lists used here are the shortest possible ones. If there exists such a relationship (or with the number of local minima, or the relative sizes or the attraction basins) is another open question.

Table 1: For each quasi-real-world problem, it is possible to define a short list that gives excellent results. Sometimes it is also pretty good for another problem but this is not the general case. The success rates are over 100 runs.

Problem	D	$\mathrm{FE}_{\max}$	ε	KISS	L_4	L_9	L_{17a}	L_{17b}
Lennard-Jones	15	30000	10^{-2}	99%	0%	0%	100%	100%
Gear Train	4	20000	10^{-11}	15%	0%	100%	0%	100%
Compression Spring	3	20000	10^{-10}	56%	0%	0%	18%	99%
Pressure Vessel	4	50000	10^{-6}	71%	0%	30%	44%	84%
Frequency Identification	6	50000	10^{-6}	24%	0%	0%	100%	0%

Table 2: A list is really interesting when it is valid for several variants of a given problem, here the Gear Train with different β and γ values. L_9 is perfect, and L_{17b} is still pretty good, even if sometimes the success rate is not 100%.

β	γ	L_9	Solution x^*	$f(x^*)$	L_{17b}	Solution x^*	$f(x^*)$
6.0	2	100%	(12, 12, 36, 24)	2.70e-12	100%	(50, 12, 60, 60)	2.70e-12
	3.5	100%	(16,17,44,59)	3.06e-12	100%	(12, 46, 56, 59)	2.26e-12
6.931	2	100%	(19,16,49,43)	8.57e-16	100%	(19, 16, 49, 43)	8.57e-16
	3	100%	(19,13,30,57)	2.50e-12	100%	(30, 13, 54, 50	1.80e-12
	3.5	100%	(19,13,30,57)	2.69e-12	100%	(133, 12, 53, 52)	32.37e-12
7.2	2	100%	(20,15,48,45)	2.7e-12	51%	(12, 30, 54, 48)	2.70e-12
	2.5	100%	(12,31,47,57)	2.7e-12	51%	(12, 30, 54, 48)	2.70e-12
7.5	2	100%	(12,12,28,60)	2.7e-12	100%	(12, 34, 60, 51)	2.70e-12

The good news, from a practical point of view, is that when a given problem is slightly modified, the same list may be still valid (although not always with the same precision), as seen on the table 2.

One may think that for the algorithm is deterministic, all runs are identical. This is not the case. First, it is of course not the case when the success rate is neither 0% nor 100%. Second, the number of times that a "random" number is needed in a run is very rarely a multiple of the length of the list. Therefore, for the second run the cyclical use of the list does begin at another rank than 1, the third run yet at another rank, etc. So those consecutive runs are not identical, as we can see on the figure 3.1 Of course, some runs *are* indeed identical, particularly when the list if short, and also many runs may finally find the same solution if the number of fitness evaluations is big enough.

For Gear Train with L_9 the convergence is quick (typically after less than 2000 fitness evaluations). However, from an engineer point of view, it may be more interesting to find several quasi-solutions. In order to do that we can allow less fitness evaluations than necessary for complete convergence, say 1000 here. As for L_{17b} the convergence is a bit

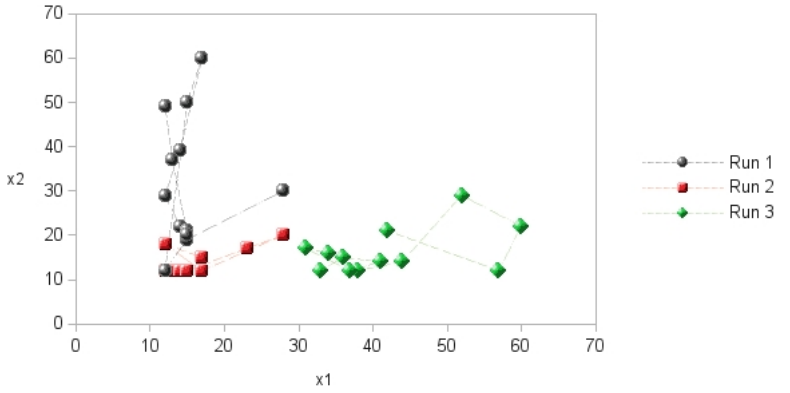

Figure 3.1: Gear Train. Trajectories of the particle 0, for the twelve first moves of three runs with L_{17b}. For clarity, only two dimensions are represented here.

slower (it needs about 2400 evaluations), and as we can see on the table 3, less runs find the same solution.

Table 3: Gear Train. Results of seven consecutive runs with L_9 and L_{17b} after 1000 evaluations. One finds more different solutions with a longer list.

	L_9		L_{17b}	
Run	Solution x^*	$f(x^*)$	Solution x^*	$f(x^*)$
1	(12,35,52,56)*	2.35e-9	(14,30,52,56)*	2.35e-9
2	(12,35,52,56)	2.35e-9	(30,13,54,50)*	2.73e-8
3	(12,35,52,56)	2.35e-9	(13,12,40,27)*	2.73e-8
4	(12,35,52,56)	2.35e-9	(30,13,54,50)	2.73e-8
5	(21,17,45,55)*	1.36e-9	(36,13,60,54)*	2.73e-8
6	(12,33,49,56)*	1.26e-9	(16,19,39,54)*	4.92e-9
7	(12,35,52,56)	2.35e-9	(30,13,54,50)	2.73e-8

4 Appendix

4.1 Generating "magic" lists

We want to transform a stochastic algorithm in a list based one. We are looking for a list L_D for a given dimension D, and valid if possible for many ariants of a given problem. We define a benchmark with such variant. We run the algorithm on this benchmark with a classical RNG, say KISS, and we save the results. Now, we are looking for a list that replaces the RNG so that the resulting list based algorithm gives never worse results, and at least sometimes better ones. The idea is to start from a list length equal to $2D$, and to increase then this length if we are not enable to find a valid list. To build a list, we have at least two ways: semi-manual or entirely automatic, by meta-optimisation.

4.1.1 Semi-manual methods

Let $|L_D|$ be the length of the list we are looking for. We divide $]0,1[$ (i.e. without 0, and without 1) into $|L_D|$ intervals, and in each interval we choose a number at random. Then we randomly permute these $|L_D|$ numbers to build the list. For these two phases, "random" means "according to any decent RNG" (in this study, KISS). The intervals may be of the same length. However, experimental results suggests that the first one and the last one should be smaller than the other ones. For example, for $D = 2$, we can define the four intervals $\{]0, 0.2[, [0.2, 0.5[, [0.5, 0.8[, [0.8, 1[\}$.

For a given small problem, this method may be enough. For example, for the Tripod problem, you can easily find that with the following list

$$\begin{array}{rcl} L_{4b} & = & (0.915702, \\ & & 0.394833, \\ & & 0.514620, \\ & & 0.013374) \end{array}$$

the performance is 100%, as with the L_4 seen in the section 2. A variant of this method is to not divide $]0,1[$ at all. We just choose the points at random in the whole interval. However, it seems that using a non-uniform distribution may be interesting. For example L_{17b}, that gives a perfect result for the 5 atoms Lennard-Jones problem (and also for Tripod), has been defined by using a linearly decreasing one, thanks to the following formula

$$|rand\,(0,1) + rand\,(0,1) - 1| \tag{4.1}$$

However, the result was not perfect, so three values have been then manually divided by 100.

4.1.2 Meta-optimisation

We consider the search space $]0,1[^{|L_D|}$. Each point of this search space is a possible list, which defines a list based optimiser when replacing the RNG of our stochastic optimiser.

We apply it many times (say 100) to all the problems of the benchmark, in order to compute an averaged performance, which can be "mean success rate AND inverse of variance of the success rates". The aim of this meta-optimisation is to find the point of the search space that maximises this performance. Of course, this process is *very* time consuming, but this is computer time, not human one! And you have to do it just once for each kind of problem.

This method has been used for Tripod, and found L_4, which is then probably one of the shortest possible list.

4.1.3 Three lists

They have been first generated at random in [0,1]. A few values have been then manually modified.

L_9

```
0.0078309 0.4773970 0.8401877 0.1975513 0.7984400 0.9522297 0.0628870
0.0076822 0.0036478
```

L_{17a}

```
0.78309922375860585575  0.47739705186216024879  0.84018771715470952355
0.19755136929338396046  0.79844003347607328536  0.95222972517471282661
0.91164735793678430831  0.27777471080318777430  0.33522275571488902024
0.39438292681909303816  0.63571172795990094073  0.55396995579543051313
0.51340091019561551189  0.1619506800370065225  0.062887092476192441026
0.76822959481190400410  0.003647844727918433940
```

L_{17b}

```
0.007830992237586058557 0.47739705186216024879 0.84018771715470952355
0.19755136929338396046 0.79844003347607328536 0.95222972517471282661
0.91164735793678430831 0.27777471080318777430 0.33522275571488902024
0.39438292681909303816 0.63571172795990094073 0.55396995579543051313
0.51340091019561551189 0.16195068003700652250 0.062887092476192441026
0.76822959481190400410 0.003647844727918433940
```

4.2 Test problems

4.2.1 Tripod (2D)

The function to minimise is

$$
\begin{aligned}
f(x) &= \tfrac{1-sign(x_2)}{2}\left(|x_1| + |x_2 + 50|\right) \\
&+ \tfrac{1+sign(x_2)}{2}\tfrac{1-sign(x_1)}{2}\left(1 + |x_1 + 50| + |x_2 - 50|\right) \\
&+ \tfrac{1+sign(x_1)}{2}\left(2 + |x_1 - 50| + |x_2 - 50|\right)
\end{aligned}
$$

with

$$
\begin{cases}
sign(x) = -1 & \text{if} \quad x \leq 0 \\
\qquad\quad = 1 & \text{else}
\end{cases}
$$

The search space is $[-100, 100]^2$,and the solution point is $(0, -50)$, on which the function value is 0. This function has also two local minima.

4.2.2 Compression Spring

For more details, see[7, 2, 5]. There are three variables

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

and five constraints

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

with

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

and the function to minimise is

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

The best known solution is $(7, 1.386599591, 0.292)$ which gives the fitness value 2.6254214578. To take the constraints into account, a penalty method is used. In this study, the maximum number of evaluations is 20,000.

4.2.3 Gear Train

For more details, see[7, 5]. 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 $f(19, 16, 43, 49) = 2.7 \times 10^{-12}$. So, if we set the objective value to zero and the acceptable error to 10^{-11}, any run that finds this fitness value (or a smaller one) is successful.

4.2.4 Pressure Vessel

Just in short. For more details, see[7, 2, 5]. 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

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

The function to minimise is

$$f(x) = 0.6224 x_1 x_3 x_4 + 1.7781 x_2 x_3^2 + x_1^2 (3.1611 x_4 + 19.84 x_3)$$

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.

4.2.5 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 the atom i has tree coordinates, and therefore the dimension of the search space is $3N$. In practice, the coordinates of a point x are the concatenation of the ones of the X_i. In short, we can write $x = (X_1, X_2, \ldots, X_N)$, and we have then

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

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

4.2.6 Frequency modulation sound parameter identification

For more details, see for example [3]. The function to minimise is

$$f(x) = \sum_{t=0}^{100} \left(y(t) - y_0(t) \right)^2$$

with $\theta = \pi/50$, and

$$\begin{cases} y(t) &= x_1 \sin\left(x_2 t\theta + x_3 \sin\left(x_4 t\theta + x_5 \sin\left(x_6 t\theta\right)\right)\right) \\ y_0(t) &= \sin\left(5t\theta + 1.5 \sin\left(4.8t\theta + 2\sin\left(4.9t\theta\right)\right)\right) \end{cases}$$

The search space is $[-6.4, 6.35]^6$. Obviously, a solution point is $x^* = (1, 5, 1.5, 4.8, 2, 4.9)$, with $f(x^*) = 0$, but there are in fact several ones, for example $x^* = (-1, -5, 1.5, -4.8, -2, 4.9)$. They all are quite difficult to find.

References

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

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

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