List Based Optimisers - Experiments and Open Questions

Maurice Clerc

List Based Optimisers - Experiments and Open Questions

Maurice Clerc

13th September 2013

Abstract

For any iterative stochastic optimisation algorithm, it is possible to replace the Random Number Generator (RNG) that is used by a predefined short list of numbers, used cyclically. We present here some experiments to check this approach. The results show that it may indeed be interesting, for the same list can be used for variants of a given problem, and even for different problems. However, there are still some important open questions, in particular about the possible methods to built the lists, which are, for the moment, quite empirical.

1 Motivation and approach

In the real world, engineers and practitioners have often to solve the same kind of optimisation 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 two simple iterative stochastic methods, namely Particle Swarm Optimisation (PSO), and Adaptive Population-based Simplex (APS), into even simpler ones that has these three features. To do that, we first present the concept of *list based optimiser*, and then some experiments and their results, which are quite good. Building "good" lists is however still challenging, and we suggest some empirical and semi-empirical methods.

2 List Based Optimisers

Most of stochastic optimisers, make use of a coded random number generator (RNG), like KISS [7], or Mersenne-Twister[8], or the ones that are embedded in a language like C. As they are not based on a hardware system, they are in fact deterministic, at least if we always keep the same seed. They generate a long list of pseudo-random numbers (typically in $[0, 1]$), whose length is ideally far bigger than what we need to solve a problem, even if the algorithm is ran several times. In such a case all runs are different, and we do hope that it improves the probability to find a good solution.

So, we can in fact consider we have a predefined list of numbers in $[0,1]$, say $L = (r_1, r_2 \cdots, r_n)$. During the optimisation process, whenever 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.. In order to avoid any confusion with true random numbers, from now on we will call them *l-random numbers*. 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 L is relatively small (typically at most one hundred of l-random numbers for a 10D problem).

The length of the list L can be extremely short. For example, for the Tripod problem (6.3.1), which is only two dimensional (but nevertheless not that easy), and a simple List-based PSO, the minimal size is probably 4. A possible such "magic" list is

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

with a classical RNG like KISS, the success rate is 97% (over 100 runs). With L_4, the run is successful. One could say that the "success rate" is 100%, but, of course, as the process is completely deterministic, if we launch it again we will get exactly the same result, so its "success rate" is either 0% or 100%. However, we could use the same list, but by starting from another element. Then, there are at most $|L|$ different runs, where $|L|$ is the length of the list L. Some of them may be successes, and the others may be failures. This way, we can define a *l-success rate*, whose value is necessarily $100\frac{k}{|K|}$, where k is an integer from $[0, |L|]$. Note that the meaning is not the same that the one of the classical success rate. For the later, no matter how big is it, if you launch the algorithm just once, you can not be completely sure that the run will be successful. On the contrary, as soon as the l-success rate is not null, if you launch the $|L|$ different runs, you are absolutely sure that at least one run will be successful.

In our example, it means we can have four different runs. Here, the l-success rate is 100%. So, we could say that this list is *perfect*: no matter on which element you start, the run is always successful. Having a perfect list may be interesting if we want to obtain several acceptable solutions.

We will now present and comment more experiments. As we will see, what is really interesting is that the same list is sometimes usable for several problems. Building such a list is not always easy, and, for the moment, there are only empirical methods. Also, it seems even more difficult to find a list that is usable for several methods.

3 Experiments with LB-PSO

We start from an already simple PSO ([2]). We can easily transforming it into a list based one. Also, the algorithm has been simplified. The C source code is available on line [3]. Note that the code contains a lot of options (like two different RNGs for comparison,

Table 1: For each quasi-real-world problem, it is possible to define a short list that gives results equivalent to the ones with a classical RNG. Sometimes it is also pretty good for another problem but this is not the general case. When using KISS, the success rate is over 100 runs. When using a list, the table presents the l-success rate.

Problem	D	$\mathrm{FE}_{\max}$	ε	RNG	L_4	L_9	L_{17}
Lennard-Jones	15	30000	10^{-2}	99%	0%	0%	100%
Gear Train	4	20000	10^{-11}	15%	0%	11.111%	5.882%
Compression Spring	3	20000	10^{-10}	56%	0%	0%	35.294%
Pressure Vessel	4	30000	10^{-6}	71%	0%	88.889%	70.588%
Frequency Identification	6	50000	10^{-6}	24%	0%	0%	11.765%

different kinds of initialisation, different topologies, etc.), just for test purpose. So it is longer that it could be. In short, the main points of the basic algorithm are:

- no RNG, but a list of l-random numbers, used cyclically, as said;
- the topology is the old classical bi-directional ring (not a variable one like in more recent PSO versions);
- the swarm size is 40 (not adaptive as in some PSO versions);
- the initial velocity of each particle is set to zero.

The table 1 gives the results for five classical quasi-real-world problems (the lists that are used are given in the 6). On the one hand, one may note that there is no clear 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. Whether such a relationship does exist or not (or with the number of local minima, or with the relative sizes of the attraction basins) is an open question. We can see that the same list L_{17} can be used for the five problems. Moreover, if this list is used "as it is", i.e. just once by starting from the beginning, the run is successful for each function. Another good news, from a practical point of view, is that when a given problem is slightly modified, the same list may be still valid as seen on the table 2.

4 Experiments with LB-APS

For PSO, we have specially written a simplified version. Let us try now to start from an existing stochastic method, based on a very different principle. APS (Adaptive Population-based Simplex) is in fact already a simplification of the method described in [6]. The C code used here is exactly the one available on [1], except that the RNG is replaced by a list. In particular it means that the population size is automatically computed, depending on the dimension of the problem. In such a case, it is probably better to use two lists: one for initialisation, and one for the search itself.

Table 2: A list is really interesting when it is valid for several variants of a given problem. Here, for the Gear Train problem, L_9 is usable with different β and γ values. The l-success rate is given over the nine possible runs. In all cases, the first run (i.e. starting from the beginning of the list) is successful.

β	γ	L_9	Best solution x^*	$f(x^*)$
6.0	2	100%	(30, 12, 36, 60)	8.57×10^{-35}
	3.5	100%	(30, 12, 36, 60)	2.41×10^{-60}
6.931	2	11.111%	(17, 21, 55, 45)	2.7×10^{-12}
	3	100%	(15, 21, 56, 39)	1.14×10^{-13}
	3.5	100%	(24, 13, 45, 48)	5.78×10^{-14}
7.2	2	88.889%	(12, 25, 48, 45)	3.80×10^{-35}
	2.5	100%	(23, 16, 50, 53)	9.44×10^{-44}
7.5	2	100%	(16, 29, 60, 58)	3.45×10^{-36}

Table 3: A list that works for LB-PSO does not necessarily work for LB-APS, and vice-versa.

Problem	RNG	L_4	L_9	L_{17}	L_{25}	L_{25} with LB-PSO
Tripod	96%	25%	0%	58.824%	16%	88%
Lennard-Jones	58%	0%	0%	0%	8%	0%
Gear Train	52%	0%	0%	0%	20%	4%
Compression Spring	80%	0%	0%	0%	8%	0%
Pressure Vessel	100%	0%	0%	0%	68%	4%
Frequency Identification	47%	0%	0%	0%	4%	0%

4.1 Generating "good" lists

For the moment, there is no sure way to build a "good" list, i.e. valid for several kinds of problems, at least for a given method. Here are the ones that have been used for this study. For each method, the most tedious point is to find the right size for the list. We have to try a lot of different ones.

4.1.1 Purely empirical methods

Let $|L|$ be the length of the list we are looking for. We divide $]0,1[$ (i.e. without 0, and without 1) into $|L|$ intervals, and in each interval we choose a number at random. Then we randomly permute these $|L|$ numbers to build the list. For these two phases, "random" means "according to any decent RNG" (in this study, Mersenne Twister). 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 a two dimension problems, 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

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

the performance is 100%, as with the L_4 seen in the section 2. We can also define just three intervals, namely "small", "middle", and "high" values. For example, L_{17}, which gives a perfect result with LB-PSO for the five problems used here (and also for Tripod), has been defined by combining three random (uniform) selections: six numbers in $]0,2\varepsilon]$, six numbers in $]0.5-\varepsilon,0.5+\varepsilon]$, and five numbers in $]1-2\varepsilon,1[$, with $\varepsilon = 0.01$. And then, again, all these numbers have been randomly permuted. More generally, it seems that using a non-uniform distribution is more efficient.

4.1.2 Semi-empirical methods

If we have a look at a "good" list obtained by some of the previous methods, we can see that they are oscillating in $]0,1[$ between small and high values, as shown on the figure 4.1

Therefore, it is tempting to apply a mathematical formula that generates similar lists. An easy way is to build an arithmetic progression by starting from a irrational[1] value smaller than 1, say d, which can be also the difference, and then "split" it into $]0,1[$

$$\left\{ \begin{array}{rcl} r_0 & = & d \\ r_{i+1} & = & if\ (r_i+d) > 1\ then\ (r_i+d-1)\ else\ (r_i+d) \end{array} \right. \tag{4.1}$$

[1] With an irrational value we are sure to never generate twice the same number.

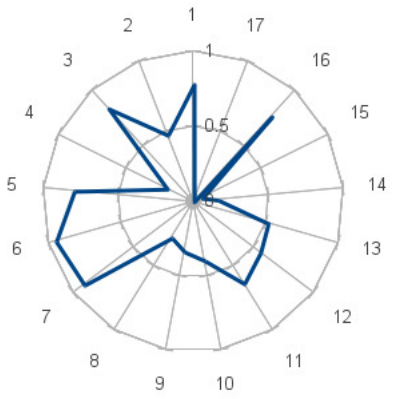

Figure 4.1: An empirical "good" list (L_{17}).

For example L_{25} has been built with $d = \frac{\sqrt{2}}{2}$. Note that if you built this way L_{24} and L_{26}, they are both not as good (null l-success rate for some of our five problems). For Tripod and LB-APS, the same method finds a list of size three. With

$$L_3 = \{0.12132, 0.70710, 0.41421\}$$

the l-success rate is 66.67% (i.e. 2/3, the third run, starting from 0.041421, fails). The first run finds an acceptable solution (error 2.08×10^{-5}) after 2565 fitness evaluations. Actually, as the only number we need to define is d, what could say that to solve the problem just one number is enough (see the Annexe for more examples).

4.1.3 Meta-optimisation

We consider the search space $]0, 1[^{|L|}$. 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 to all the problems of the benchmark, in order to compute an averaged performance, which can be "mean l-success rate AND inverse of variance of the l-success rates". The aim of this meta-optimisation is to find the point of the search space (i.e. the list) that maximises this performance. Of course, this process is *very* computer time consuming, but we have to do it just once. At least, we can more easily apply this method to just one problem. For example, for Tripod, it finds L_4, which is then probably one of the shortest possible perfect lists for this problem and LB-PSO.

5 Open questions

The above experiments (and more not presented here) raise several questions, theoretical and practical. We assume that we have a set of methods (stochastic algorithms) and a

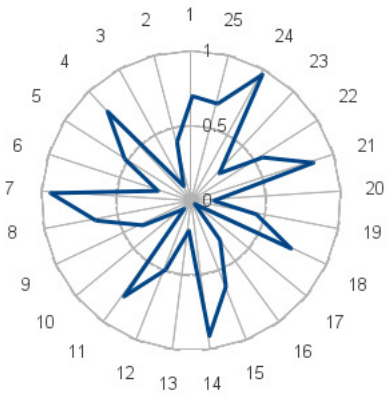

Figure 4.2: Semi-empirical list (L_{25}), generated thanks to an arithmetical progression whose difference is $\frac{\sqrt{2}}{2}$.

set of problems (benchmark).

- if the original method is successful at least once on a problem, it means that the set of lists that can successfully replace the RNG for this problem is not empty. But what is the size of the shortest list(s) of this set? And how to build such a list?
- for a given method, is there a list that can successfully replace the RNG on the whole benchmark? If so, how to build such a list? If not, how to build at least a "good" list (successful on as many problems as possible)?
- it is not rare that even if a list L is not very good, a sub-list (of consecutive numbers) is better. But to find such a sub-list is there a clever way than exhaustive search?

6 Appendix

6.1 When one number (and a formula) is enough

We have seen that thanks to an arithmetical progression defined by just one number $d = \frac{\sqrt{2}}{2}$, and the formula 4.1, we can generate a very short list ($|L| = 3$) which is enough for LB-PSO to solve the Tripod problem. Here are a few more examples. Some problems are coming from the CEC 2005 benchmark [11]. They all are shifted. Note that in their definitions the maximum number of fitness evaluations is linearly increasing with the dimension. As we can see on the table 4, even for relatively difficult problems short lists are usable. For example exactly the same list of length six can be used for the four first

problems. It is not shown here, but similar results can be obtained with some other "seeds", for example $d = \frac{e}{10}$.

Table 4: Some results with LB-PSO and finite lists generated by an arithmetical progression based on $d = \frac{\sqrt{2}}{2}$, and used cyclically. The table gives the length $|L|$ of the smallest list for which the l-success rate is not null, and this l-success rate itself. Remember that it may nevertheless be null for a *longer* list. D is the dimension of the search space for the CEC 2005 problems.

(a) CEC 2005 problems.

Problem	$D = 5$	$D = 10$	$D = 30$
Sphere (F1)	6 (100%)	11 (100%)	31 (100%)
Schwefel (F2)	6 (100%)	11 (27.27%)	37 (83.78%)
Schwefel +noise (F4)	6 (83.33)	19 (47.37%)	>100
Rosenbrock (F6)	6 (33.33%)	19 (5.26%)	37 (2.70%)
Rastrigin (F9)	11 (18.18%)	>100	>100

(b) Other problems used in this study.

Problem	$\|L\|$ (l-success rate)
Tripod	3 (33.33%)
Lennard-Jones	11 (100%)
Gear Train	13 (7.69%)
Compression Spring	6 (16.66%)
Pressure Vessel	7 (100%)
Frequency Identification	18 (5.56%)

Actually, using one number as "seed", and then a formula to generate pseudo-random numbers is exactly what are doing all coded RNGs. However, they use very complicated formulae so that the generated numbers seem to be as random as possible. But it may be not necessary in the context of stochastic optimisation. This is a bit out of the scope of this paper, so we just present a few results in the table 5. For some problems the performance is significantly better than with a classical RNG (see table 1). But also sometimes significantly worse. Nevertheless it suggests it may be worth investigating this approach.

6.2 Some lists

L_9 (empirical)

```
0.9046044347 0.4113702427 0.5567391497 0.8074334206 0.2958179712
0.7310219268 0.377415836 0.0002342685 0.5491584423
```

Table 5: Some results with LB-PSO and infinite lists generated by an arithmetical progression based on $d = \frac{\sqrt{2}}{2}$. Here the success rate is the classical one, estimated over 100 runs.

(a) CEC 2005 problems.

Problem	$D = 5$	$D = 10$	$D = 30$
Sphere (F1)	100%	100%	100%
Schwefel (F2)	100%	100%	99%
Schwefel +noise (F4)	100%	0%	0%
Rosenbrock (F6)	11%	4%	0%
Rastrigin (F9)	39%	0%	0%

(b) Other problems used in this study.

Problem	Success rate
Tripod	100%
Lennard-Jones	98%
Gear Train	7%
Compression Spring	11%
Pressure Vessel	3%
Frequency Identification	14%

L_{17} (empirical)

```
0.015736 0.496893 0.500413 0.990438 0.495484 0.490454 0.982801
0.018933 0.000385 0.495026 0.997142 0.005010 0.498098 0.014213
0.990877 0.1989995 0.991221
```

L_{25} (generated by an arithmetic progression)

```
0.7071067812 0.4142135624 0.1213203436 0.8284271247 0.5355339059
0.2426406871 0.9497474683 0.6568542495 0.3639610307 0.0710678119
0.7781745931 0.4852813742 0.1923881554 0.8994949366 0.6066017178
0.3137084990 0.0208152802 0.7279220614 0.4350288425 0.1421356237
0.8492424049 0.5563491861 0.2634559673 0.9705627485 0.6776695297
```

6.3 Test problems

6.3.1 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. In this study, the maximum number of fitness evaluations (when using a RNG) is 10,000, and the acceptable error is 10^{-4}. Any run that finds this error value (or a smaller one) is said to be successful.

6.3.2 Lennard-Jones

For more details, see for example [5]. 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 $[-2, 2]^{15}$. The objective value is -6, and the acceptable error 10^{-2}.

6.3.3 Gear Train

For more details, see[10, 9]. 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 $\gamma = 2$. The objective value is 0, although it can not be reached, and the acceptable error is 10^{-11}.

6.3.4 Compression Spring

For more details, see[10, 4, 9]. 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

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

with

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

and the function to minimise is

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

The best known solution is $(7, 1.386599591, 0.292)$ which gives the fitness value 2.6254214578. This is the objective here, and the acceptable error is 10^{-10}. To take the constraints into account, a penalty method is used.

6.3.5 Pressure Vessel

Just in short. For more details, see[10, 4, 9]. There are four variables

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

and three constraints

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

The function to minimise is

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

The analytical solution is $(1.125, 0.625, 58.2901554, 43.6926562)$ which gives the fitness value 7,197.72893, which is therefore the objective. The acceptable error is 10^{-6}. To take the constraints into account, a penalty method is used.

6.3.6 Frequency modulation sound parameter identification

For more details, see for example [5]. The function to minimise is

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

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

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

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. The acceptable error is 10^{-6}.

References

[1] APS. Adaptive Population-based Simplex, http://aps-optim.info.

[2] D. Bratton and J. Kennedy. Defining a standard for particle swarm optimization. In *IEEE Swarm Intelligence Symposium*, pages 120–127, June 2007.

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

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

[5] 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.

[6] Changtong Luo and Bo Yu. Low dimensional simplex evolution: a new heuristic for global optimization. *Journal of Global Optimization*, 52(1):45–55, January 2012.

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

[8] 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.

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

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

[11] PN Suganthan, N Hansen, JJ Liang, K Deb, YP Chen, A Auger, and S Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real parameter optimization. Technical report, 2005.