Semi-continuous challenge

Maurice Clerc Maurice.Clerc@WriteMe.com
Last update: 2004-02-17

1. Why?

The aim is to improve my parameter free (adaptive) particle swarm optimizer Tribes [CLE 03]. In order to do that, I have also written a parametric one, called OEP (means simply PSO in French), with a lot of options. By analyzing what are the best options/parameters for several problems, my hope is I could find better adaptations rules for Tribes.

However, there are now so many PSO variants that I simply can't incorporate all of them in OEP. I have to choose the most promising ones. So the aim of this "challenge" is to ask people if they are aware of a PSO version that gives better results, and, if so, if they can explain how they choose the parameters, if any.

Tribes (and OEP) can easily cope with hybrid problems (with some discrete dimensions), but it is not the case for all PSO versions. So the problems defined below are all continuous (or semi-continuous for the first one).

In such a case, almost any kind of search space can be transformed into a D-cubical one, or, simpler, into a set of such D-cubes (you may see the "For amatheurs only" section). I don't say it is always easy, I just say it is possible. That is why, in order to compare algorithms, you just need to use interval constraints.

1.1. Six continuous or semi-continuous problems

I have carefully chosen these six problems so that it should be quite difficult to solve them all efficiently with a given parametric optimisation method, that is to say by keeping the same parameter set. They are placed in ascending order of theoretical difficulty. For each case, the minimal value is zero, and the objective is to find a function value smaller than 10^{-5}. I also give the C source code in the appendix.

.

Name Difficulty	Formula	Search space	Objective
Tripod 33	$p(x_2)(1+p(x_1)) + \lvert x_1 + 50p(x_2)(1-2p(x_1))\rvert + \lvert x_2 + 50(1-2p(x_2))\rvert$ with $\begin{cases} p(u)=1 \text{ si u} \geq 0 \\ =0 \text{ si } u<0 \end{cases}$	$[-100,100]^2$	0 ± 10^{-5}
Alpine 10D 125	$\sum_{d=1}^{D} \lvert x_d \sin(x_d)+0{,}1x_d\rvert$	$[-10,10]^{10}$	0 ± 10^{-5}
Parabola 30D 273	$\sum_{d=1}^{D} x_d^2$	$[-20,20]^{30}$	0 ± 10^{-5}
Griewank 30D 335	$\frac{\sum_{d=1}^{D}(x_d-100)^2}{4000} - \prod_{d=1}^{D}\cos\left(\frac{x_d-100}{\sqrt{d}}\right)$	$[-300,300]^{30}$	0 ± 10^{-5}
Rosenbrock 30D 370	$f(x)=\sum_{d=1}^{D-1}(1-x_d)^2+100(x_d^2-x_{d+1}$	$[-10,10]^{30}$	0 ± 10^{-5}
Ackley 30D 470	$-20e^{-0{,}2\sqrt{\frac{\sum_{d=1}^{D}x_d^2}{D}}} - e^{\frac{\sum_{d=1}^{D}\cos(2\pi x_d)}{D}} + 20$	$[-30,30]^{30}$	0 ± 10^{-5}

1.2. Representations and comments

Don't forget that these 2D figures give just a faint idea of the problem when the dimension is in fact 10 or 30.

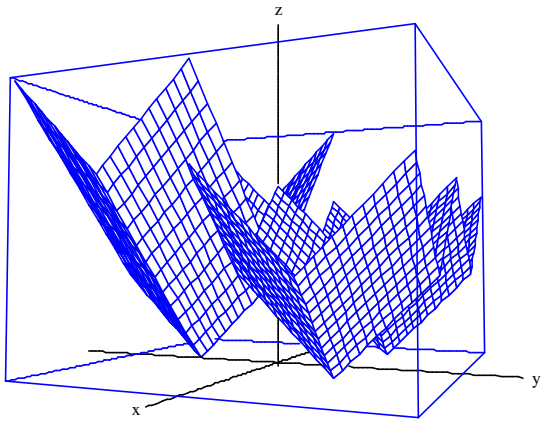

Figure 1. Tripod. The minimum 0 is on (0, -50). Theoretically easy, this problem is in fact difficult for a lot of algorithms that are trapped in the two local minima. Note the function is not a continuous one. It has been published first in [GAC 02]

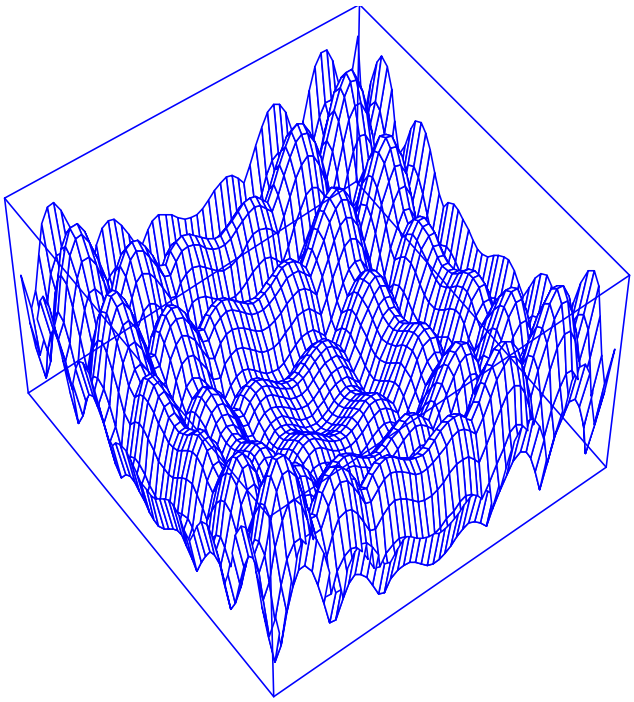

Figure 2. *Alpine. A lot of local and global minima. It is not really symmetrical. This problem is nevertheless quite easy, and can be seen as a kind of pons asinorum for optimisation algorithms. The first version (this one is a bit different, so it should be named Alpine 2), has been published in [CLE 99]*

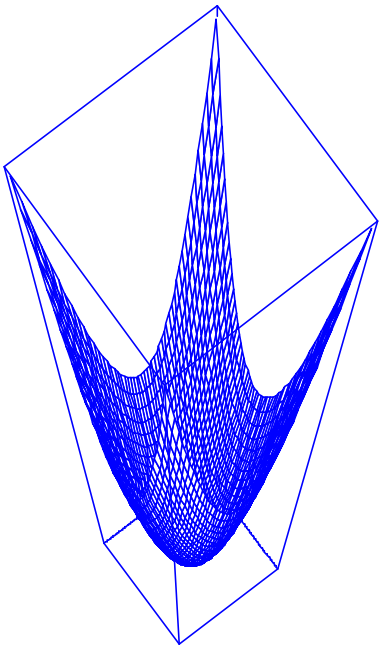

Figure 3. *Parabola. Well known. Just one minimum 0 in (0, 0). Sometimes called Sphere, nobody knows why, maybe because of its equation, but, fortunately, cricket balls don't really have this shape. It is very easy to adjust its difficulty just by modifying the dimension D. With such a function, algorithms like gradient methods usually work very well, so it is by itself a challenge for stochastic methods like PSO*

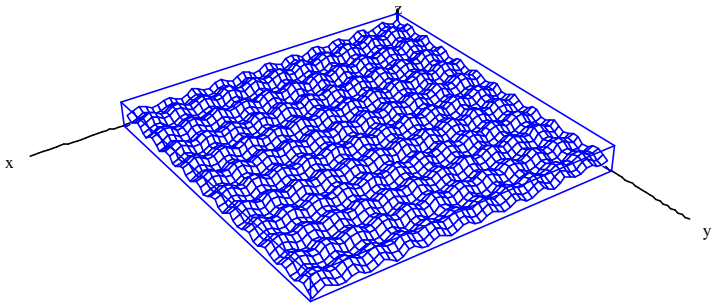

Figure 4. *Griewank. Well known and more difficult. The minimum 0 is on (100, 100), and almost not different from the numerous local minima around. On the one hand, of course, it increases the difficulty, but, on the other hand, as there are so many small local minima, it is still quite easy to escape from them. So, increasing the dimension not necessarily increases the difficulty*

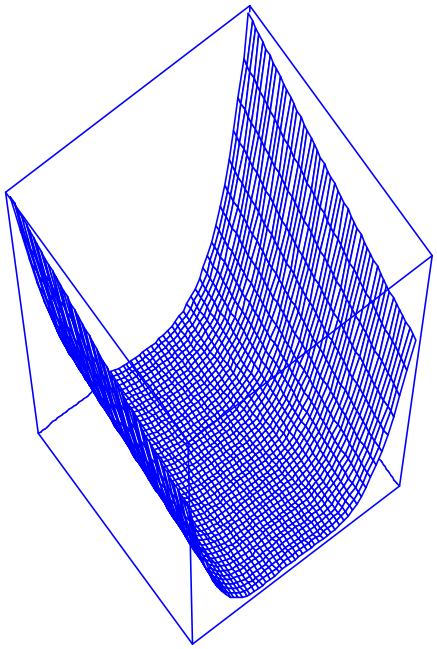

Figure 5. *Rosenbrock. Well known. Deceptively flat, this function is here shown on* $[-10\ 10]^2$. *The global minimum in on (1, 1), and quite difficult to find as soon as the dimension is high.*

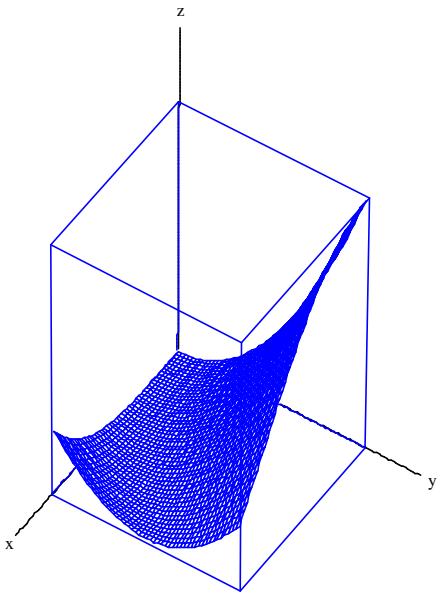

Figure 6. *Rosenbrock again, but on [0 1]x[0 2], so that you can see the minimum.*

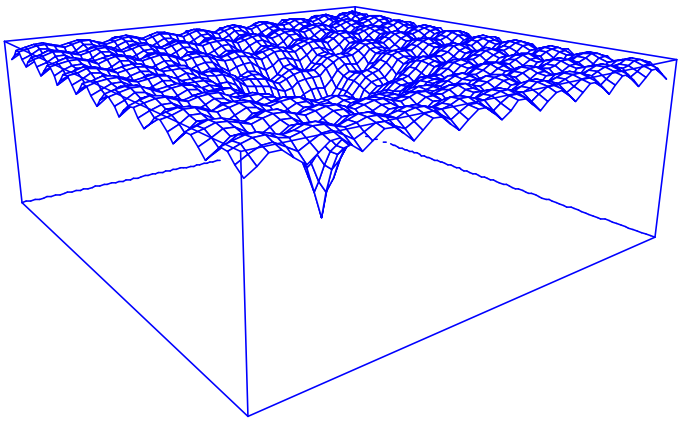

Figure 7. *Ackley. Well known. Looks a bit like Alpine, but in fact more difficult. The "attraction basin" of the global minimum is quite narrow, so random moves can't easily find it*

2. The challenge itself

To initiate this challenge, I give in the table 1 some results obtained by OEP 5. The parameters are the followings:

N, number of explorer particles

M, number of memory particles (for some difficult problems, it is better to have *M*>*N*), or memory swarm size

K, the mean number of information links from a memory to explorers (note that in all cases each explorer has just one link towards the memory swarm)

info_option, which defines the topology of the information graph (fixed or not, random or not, etc.)

mouv_option, which defines the kind of proximity probabilistic distribution (D-parallelepid as in classical PSO, D-sphere, D-gauss, distorted variants, etc.)

For some mouv_option values (typically for the classical PSO), there is an additional parameter φ.

Note that for the moment the program does not incorporate any kind of explicit clustering/niching. It is probably a promising approach, but I am still not sure.

For each function, I ran 100 times the program with a given "search effort" *T*, that is to say a given maximum number of objective function evaluations, and the failure rate is then computed. Of course, it is just an estimation. However, it is easy to prove there is 95 % of chance that it is correct by less than 5 %. It is usually enough to decide whether a result is really better than another one or not.

If the intrinsic difficulty is δ you can then derive that the difficulty for a given *T* is about δ-ln(*T*), as soon as *T* is big enough. So, *T* has been chosen to keep the same difficulty order for the six problems, but also big enough so that it is indeed possible to solve most of them with OEP.

First, I tried to carefully tune the parameters differently for each function, in order to know what is possible to do with the program. It appears that just one problem (Rosenbrock) can absolutely not be solved (in less than 40000 evaluations). After that, I choose a given parameter set, and used it for all functions. For the time being, one of the best one I have found is the following:

N=*M*=45, *K*=3

info_option: random choice of *K* explorers for each memory, if there has been no global improvement after the iteration

mouv_option: if no global improvement after the previous iteration, use distorted positive spherical D-sectors, else use the pivot method (adapted from [SER 97]).

φ=2.17

As expected, I don' t have been able to find a parameter set that give good result for all the six functions, so I choose to accept worse results for easy problems, in order to still have quite good ones for difficult problems. The opposite is possible, but, interestingly, the mean failure rate seems to be then a bit higher.

Problem	Search effort (max. number of evaluations) and difficulty	Failure rate when parameters are tuned for each problem	Failure rate with the same parameter set for all problems
Tripod 2D	40000 (22)	0 %	19 %
Alpine 10D	15000 (122)	0 %	25 %
Parabola 30D	15000 (264)	0 %	0 %
Griewank 30D	40000 (325)	0 %	5 %
Rosenbrock 30D	40000 (360)	100 % (8,22)	100 % (29,20)
Ackley 30D	40000 (460)	0 %	0 %
		Mean failure rate	**25 %**

Table 1. Failure rates in different cases, for 100 runs, when using OEP5. For Rosenbrock, as the failure rate is 100%, I have added in parenthesis the mean of the 100 best values found. When using just one parameter set, in order to obtain the best mean failure rate, I have to accept a quite big one for the easiest problems

3. For amatheurs only

3.3. Anything is a cube, or Handling constraints by homeomorphism

An optimisation problem is usually given as follows, when the search space is a continuous one:

minimise $f(x)$, with $x_d \in [x_{d,\min}, x_{d,\max}], \forall d \in \{1,...,D\}$

After that, it is often said something like "Now, let' s add some constraints$g_i(x) \leq 0$ and $h_j(x) = 0$". A lot of remarks can be said about such a representation. The most important ones are:

- the initial problem is already a constrained one, for we have $x_d \in [x_{d,\min}, x_{d,\max}] \Leftrightarrow \begin{cases} x_{d,\min} - x_d \leq 0 \\ x_d - x_{d,\max} \leq 0 \end{cases}$

- all inequality constraints can be transformed into equalities, for we have $g_i(x) \leq 0 \Leftrightarrow g_i(x) + |g_i(x)| = 0$. It is particularly interesting when they are just indicative constraints, to satisfy "as well as possible", and not imperative ones. The problem can then be seen as a multiobjective one.

- the initial problem can be rewritten: minimise $\phi(y)$, with $y \in [0,1]^D$

just by the bijective continuous transformation (homeomorphism) $y_d = \frac{x_d - x_{d,\min}}{x_{d,\max} - x_{d,\min}}$, and by defining ϕ by $\phi(y) = f(x)$. So the initial search space, which was a D-parallelepid, is now the unit D-cube $C(D)$.

The last point can be generalised. First, note that the constraints define a subspace of the initial search space *H*. Let us call it "restricted search space" H_r. Second, there is a strange theorem saying that there are the same "number" of points in $[0,1]$ than in any other real interval (of course, it is not true for discrete finite search spaces). More generally, there is a theorem saying that it is possible to map H_r to $C(D)$ by using a homeomorphism, if the topological genus of H_r is zero. In practice, this last condition is not really important, for, as long as H_r is not too mathematical monstrous, you can always cut it into a finite number of parts whose genus is indeed 0, and then map each of them to $C(D)$.

I give here just two examples, so that you get the idea.

Example 1. (disc/4)=square

minimise $f(x_1, x_2)$, search space $H = [0,1]^2$, constraint $\sqrt{x_1^2 + x_2^2} \leq 1$.

So H_r is the positive quarter of the unit disc centred in $(0,0)$.

$$\text{mapping } \begin{cases} (x_1, x_2) \in H_r \xrightarrow{\mu} (y_1, y_2) \in D(2) \\ y_1 = \sqrt{x_1^2 + x_2^2} \\ y_2 = \dfrac{2}{\pi} \operatorname{atan}\left(\dfrac{x_2}{x_1}\right) \end{cases}$$

The function ϕ is then defined by $\forall (y_1, y_2) \in C(2), \phi(y_1, y_2) = f\left(y_1 \cos\left(\dfrac{\pi}{2} y_2\right), y_1 \sin\left(\dfrac{\pi}{2} y_2\right)\right)$

and the equivalent problem is: minimise $\phi(y_1, y_2)$, search space $C(2)$.

Example 2. triangle = square

minimise $f(x_1, x_2)$, restricted search space defined by $\begin{cases} x_1 \geq 0 \\ x_2 \geq 0 \\ x_1 + x_2 \leq 1 \end{cases}$.

H_r is then the triangle $\{(0,0), (1,0), (0,1)\}$.

$$\text{mapping } \begin{cases} (x_1, x_2) \in H_r \xrightarrow{\mu} (y_1, y_2) \in D(2) \\ y_1 = x_1 + x_2 \\ y_2 = \dfrac{x_2}{x_1 + x_2} \end{cases}$$

The function ϕ is then defined by $\forall (y_1, y_2) \in C(2), \phi(y_1, y_2) = f(y_1(1 - y_2), y_1 y_2)$. Again, the equivalent problem is: minimise $\phi(y_1, y_2)$, search space $C(2)$.

Remarks

As you have certainly noted, Example 1 and Example 2 are in fact the same (use the intermediate variables $z_1 = x_1^2$ and $z_2 = x_2^2$ to transform the first one into the second one). For most real problems, constraints are (or can be transformed into) linear ones. The restricted search space is then a polyhedron. Any polyhedron can be cut into D-triangles (real triangles if D=2, tetrahedrons if D=3), and any D-triangle can be mapped to the unit D-cube. Theoretically, you could then unify these unit cubes by mapping them to a single one, but in practice, for optimisation, it is not necessary for you can look for the optimum successively inside each D-cube.

3.1. Some difficulty level estimations

The theoretical difficulty is given by $-\ln(\sigma)$, where σ is the probability to find a solution by choosing a point at random (uniform distribution) in the search space.

3.1.1. Tripod

Let ε be the required accuracy. It is supposed smaller than 1, so that there is no need to take local minima into account. The solution space it then a square whose surface is $2\varepsilon^2$. As the whole search space is $[-100\ 100]^2$, the difficulty level is given by

$$difficulty = -\ln\left(\frac{2\varepsilon^2}{200^2}\right) = 2\ln(200) - 2\ln(\varepsilon) - \ln(2)$$

for $\varepsilon = 10^{-5}$, the value is then about 33.

3.1.2. Rosenbrock

Here, it has just been statistically estimated. Note that the difficulty is almost an increasing linear function of the dimension *D*, as shown in the table below. Don't forget, though, it is a logarithm, so the real difficulty is exponentially increasing.

Dimension	Difficulty
2	20
5	60
10	120
20	245
30	370

Just for fun, it is also possible to perform an analytical estimation, by using the Taylor's formula. On the position $\vec{1} = (1,...,1)$, where the minimum is, first order and second order partial derivatives are equal to 0. So, an estimation of the function around this point is given by $f(\vec{1} + h) = h^2 (1 + (D-1)101)$. As we want that to find a value smaller than ε, it give us the edge of the D-cube where are all the solution points, $h = 2\sqrt{\varepsilon/(1 + (D-1)101)}$.

So, in our example, with $\varepsilon = 10^{-5}$, $D = 30$, and the search space $[-10\ 10]^{30}$ whose volume is 20^{30}, the theoretical difficulty is given by

$$difficulty = -\ln\left(\left(\frac{2\sqrt{10^{-5}/2930}}{20}\right)^{30}\right) \cong 362$$

Using this method, the value is necessarily smaller than the true one. So, we see that the statistical estimation 370 is quite acceptable.

3.2. Difficulty depending on the search effort

The theoretical difficulty is of course decreasing if you accept several random choices for the position. Let *T* be the number of these choices. As the success probability for *T*=1 is σ, the failure probability is $(1-\sigma)$, and the probability to don't having found a solution after *T* draws is $(1-\sigma)^T$. So, the probability to *have* found a solution is its complement to 1. Finally, by taking the logarithm, we obtain the theoretical difficulty depending on the search effort

$$difficulty(T) = -\ln\left(1 - (1-\sigma)^T\right) \cong -\ln(T\sigma) = -\ln(\sigma) - \ln(T)$$

4. Appendix

4.1. Some C source code

```
// Tripod
x1=x.x[0]; x2=x.x[1];

if(x2<0) {f=fabs(x1)+fabs(x2+50); }
else { if(x1<0) f=1+fabs(x1+50)+fabs(x2-50);
          else f=2+fabs(x1-50)+fabs(x2-50);}

// Parabola
f=0; for(d=0;d<D;d++) f=f+x.x[d]*x.x[d];

// Alpine
f=0;  for( d=0;d<D;d++) { zz=x.x[d]; f=f+fabs(zz*sin(zz)+0.1*zz);}
```

```
// Griewank
f=0; p=1;
for (d=0;d<D;d++) {xd=x.x[d]-100; f=f+xd*xd; p=p*cos(xd/sqrt(d+1));}
f=f/4000 -p +1;

// Rosenbrock
f=0;
for (d=0;d<D-1;d++) {xd=1-x.x[d]; f=f+xd*xd; xd= x.x[d]*x.x[d]-x.x[d+1]; f=f+100*xd*xd;}

// Ackley
E=exp(1); two_pi=2*acos(-1);
sum1=0;sum2=0;
for (d=0;d<D;d++) {xd=x.x[d]; sum1=sum1+xd*xd; sum2=sum2+cos(two_pi*xd);}
f=(-20*exp(-0.2*sqrt(sum1/(double)D))-exp(sum2/(double)D)+20+E);
```

4.2. Good parameter sets for each function

The following parameter sets are reffering to OEP v. 5. Except for Rosenbrock, they give a failure rate equal to zero.

Tripod
N=95, *M*=30, *K*=3, φ=2.3
info_option=random, modified if no improvement
mouv_option=classical constricted PSO

or
N=40, *M*=40, *K*=3, φ=2.08
info_option= circular, fixed
mouv_option=classical constricted PSO

Alpine
N=18, *M*=18, *K*=3, φ=2.11
info_option=random, modified if no improvement
mouv_option=classical constricted PSO

or
N=18, *M*=18, *K*=3, φ=2.09
info_option=circular, fixed
reorg= 1. Each memory checks its two neighbours. If they are both better, it choose one at random and take its position.
mouv_option=classical constricted PSO

Parabola
N=18, *M*=18, *K*=3, φ=2.11
info_option=random, modified if no improvement
mouv_option=classical constricted PSO

or
N=18, *M*=18, *K*=3, φ=2.09
info_option=circular, fixed
reorg= 1. Each memory checks its two neighbours. If they are both better, it choose one at random and take its position.
mouv_option=classical constricted PSO

or
N=45, *M*=45, *K*=3, φ=2.17
info_option=random, modified if no improvement
mouv_option= if no improvement, use distorted positive spherical D-sectors, else use the pivot method

Griewank
N=95, *M*=95, *K*=3, φ=2.17
info_option=random, modified if no improvement
mouv_option= if no improvement, use distorted positive spherical D-sectors, else use the pivot method

Rosenbrock
N=45, *M*=45, *K*=15, φ=2.07
info_option=random
mouv_option=spherical distributions (equivalent volume method). Instead of D-parallelepids as in classical constricted PSO, it uses spheres that has the same volume.

Ackley
N=45, *M*=45, *K*=3, φ=2.17
info_option=random, modified if no improvement
mouv_option= if no improvement, use distorted positive spherical D-sectors, else use the pivot method

or
N=45, *M*=45, *K*=3, φ=2.08
info_option=random, modified if no improvement
mouv_option=classical constricted PSO

5. References

[CLE 99] Clerc M., "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization", Congress on Evolutionary Computation, Washington DC, 1999.

[CLE 03] Clerc M., "TRIBES - Un exemple d'optimisation par essaim particulaire sans paramètres de contrôle", OEP'03 (Optimisation par Essaim Particulaire), Paris, 2003.

[GAC 02] Gacôgne L., "Steady state evolutionary algorithm with an operator family", EISCI, Kosice, Slovaquie, 2002.

[SER 97] Serra P., Stanton A. F., Kais S., "Pivot method for global optimization", Physical Review, vol. 55, 1997, p. 1162-1165.