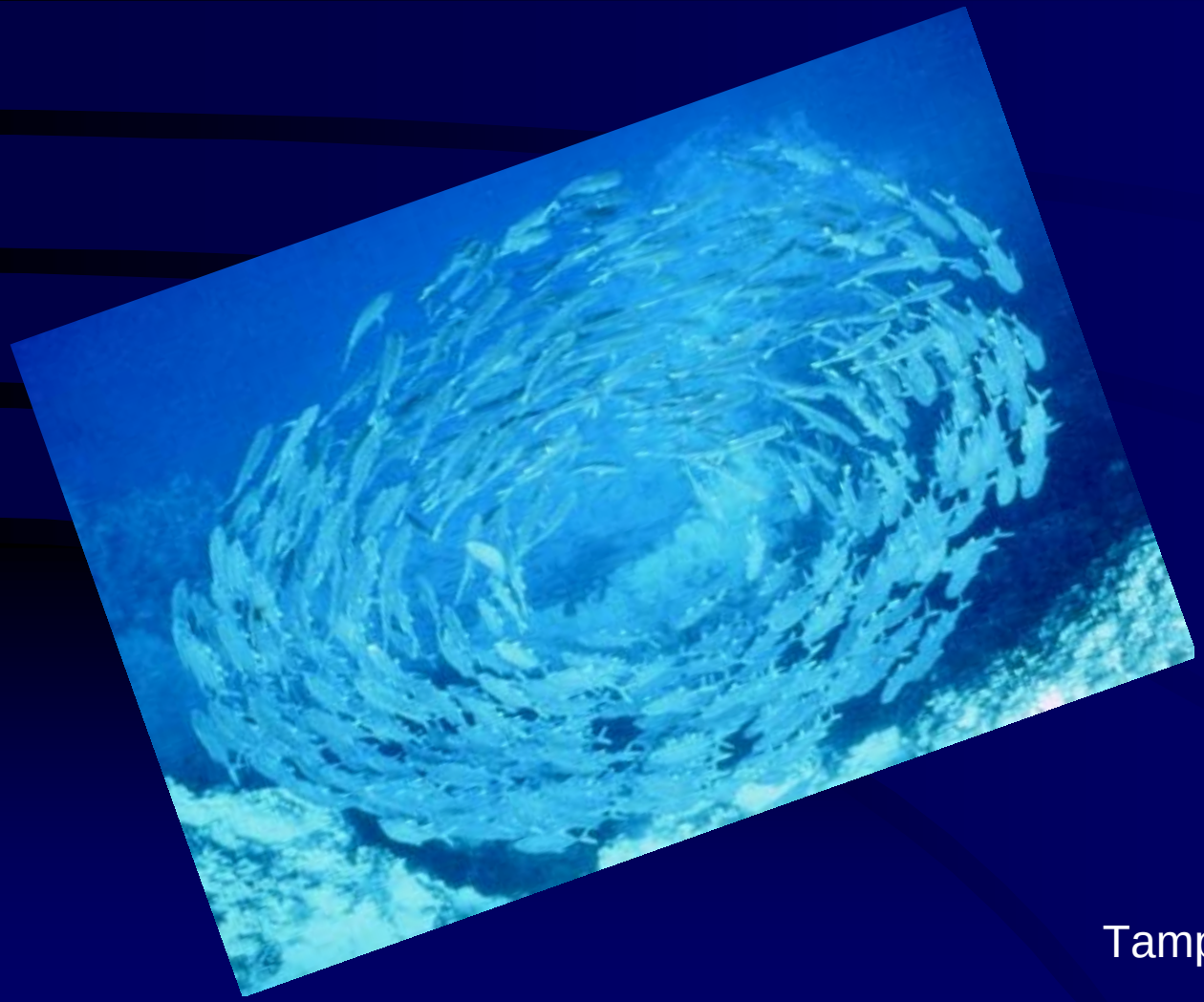
Fractional Particle Swarm Optimization in Multi-Dimensional Search Space



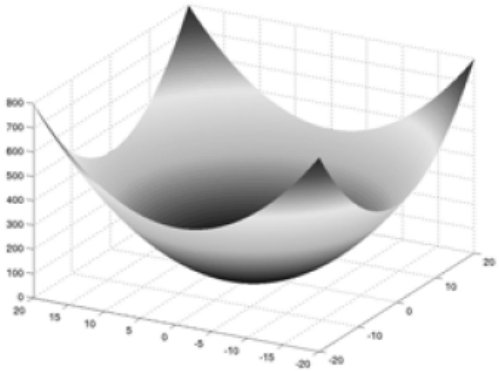
Serkan KIRANYAZ

Tampere Univ. of Technology

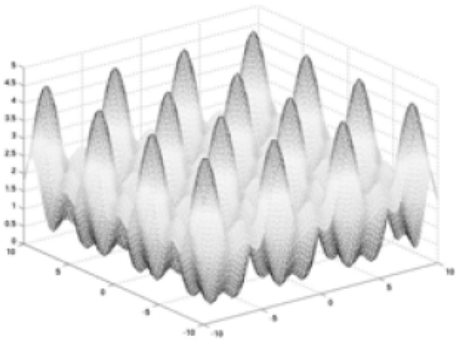
Pattern Recognition & Optimisation

- These two things tend to come up a lot when we think of what we would *like* to be able to do with software, but usually *can't* do.
- *But these are things that seems to be done very well indeed in Biology.*
- *So it seems like a good idea to study how these things are done in biology – i.e. (usually) how computation is done by biological machines*

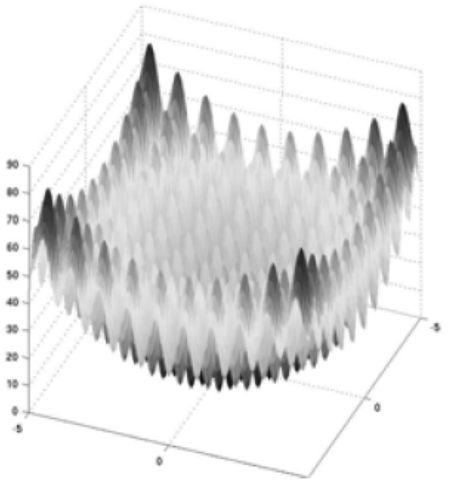
Optimization, but How?



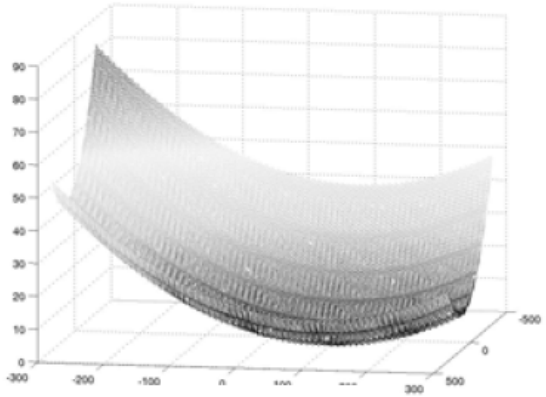
a) Sphere



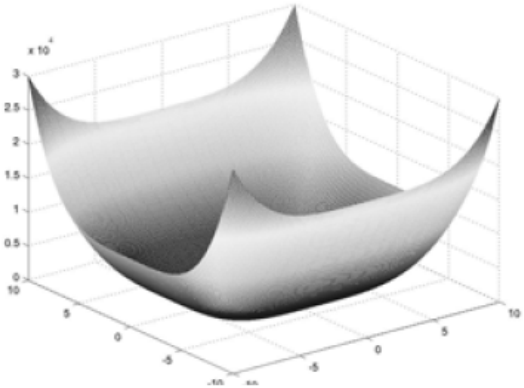
b) Giunta



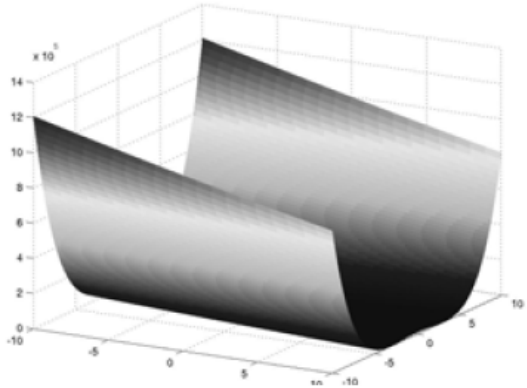
c) Rastrigin



d) Griewank



e) DeJong



f) Rosenbrock

How Biology Optimizes?

- We wish to design something – we want the best possible (or, at least a very good) design.
- The set *S* is the set of all possible designs. It is always much too large to search through this set one by one, however we want to find good examples in *S*.
- In nature, this problem seems to be solved wonderfully well, again and again and again, by ***evolution***
- Nature has designed millions of extremely complex machines, each almost ideal for their tasks (assuming an environment that doesn't go haywire), using evolution as the only mechanism.

Swarm Intelligence

Swarm Intelligence

- How do swarms of birds, fish, etc … manage to move so well as a unit? How do ants manage to find the best sources of food in their environment? Answers to these questions have led to some very powerful new optimisation methods, that are different to EAs. These include *Ant Colony Optimisation*, and *Particle Swarm Optimisation*.
- Also, only by studying how real swarms work are we able to simulate realistic swarming behaviour (e.g. as done in **Jurassic Park, Finding Nemo**, etc …)

Evolutionary Computation Algorithms

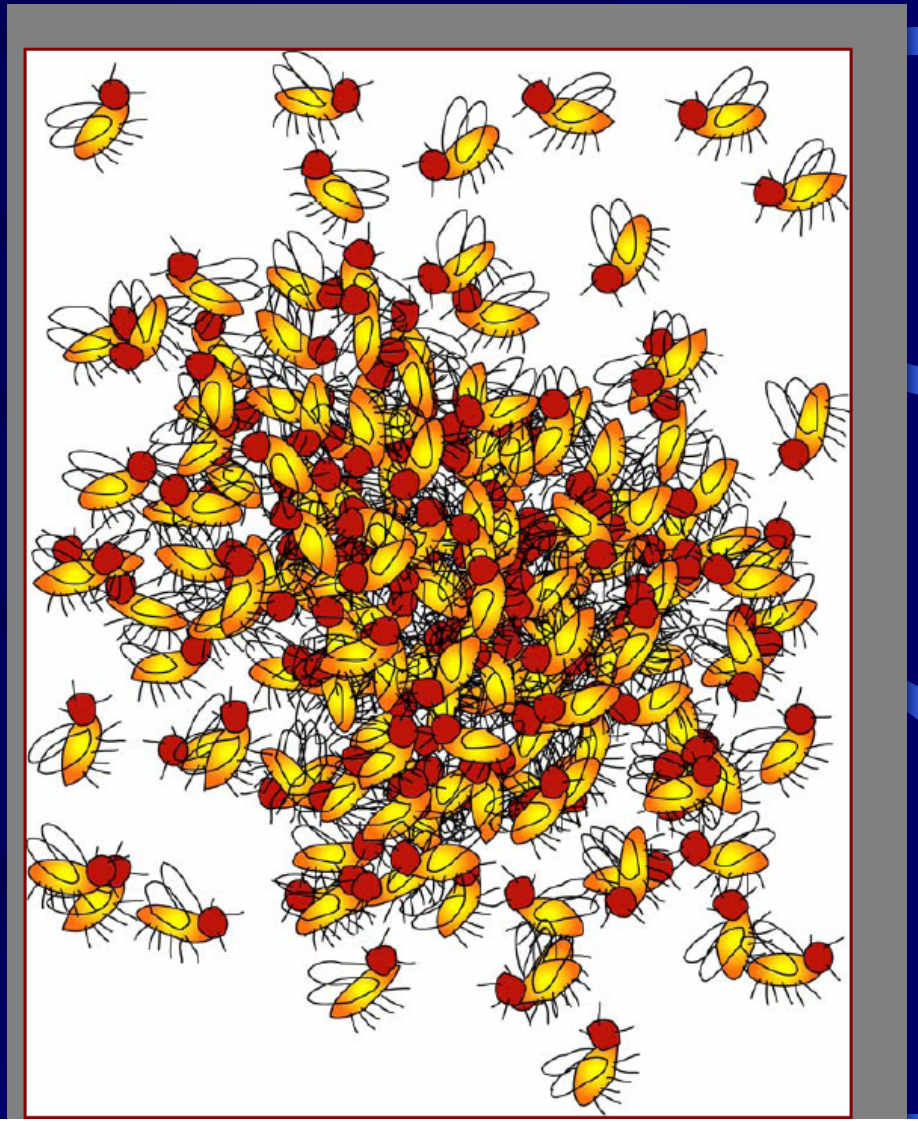
1. Initialize the population
2. Calculate the fitness of each individual in the population
3. Reproduce selected individuals to form a new population
4. Perform evolutionary operations such as *crossover* and *mutation* on the population
5. Loop to step 2 until some condition is met

Evolutionary Computation Paradigms

- Genetic algorithms (GAs) - John Holland
- Evolutionary programming (EP) - Larry Fogel
- Evolution strategies (ES) - I. Rechenberg
- Genetic programming (GP) - John Koza
- Particle swarm optimization (PSO) - Kennedy & Eberhart (1995)

SWARMS

- Coherence without choreography
- Particle swarms; ".. behavior of a single organism in a swarm is often insignificant but their collective and social behavior is of paramount importance"



Intelligent Swarm

- A population of interacting individuals that optimizes a function or goal by collectively adapting to the local and/or global environment
- Swarm intelligence ≅ collective adaptation
- A “swarm” is an apparently disorganized collection (population) of moving individuals that tend to cluster together while each individual seems to be moving in a random direction
- We also use “swarm” to describe a certain family of social processes

Introduction to Particle Swarm Optimization (PSO)

A concept for optimizing nonlinear functions..

- Has roots in artificial life and evolutionary computation
- Developed by Kennedy and Eberhart (1995)
- Simple in concept
- Easy to implement
- Computationally efficient
- Effective on a variety of problems

Evolution of PSO Concept and Paradigm

- Discovered through simplified social model simulation
- Related to bird flocking, fish schooling, and swarming theory
- Related to evolutionary computation; some similarities to genetic algorithms and evolution strategies
- Kennedy developed the “cornfield vector” for birds seeking food
- Bird flock became a swarm
- Expanded to multidimensional search
- Incorporated acceleration by distance
- Paradigm simplified

Features of Particle Swarm Optimization

- Population initialized by assigning random positions *and* velocities; particles representing the potential solutions are then *flown* through hyperspace.
- Each particle keeps track of its "best" (highest fitness) position in hyperspace.
- This is called "*pbest*" for an individual particle
- It is called "*gbest*" for the best in the population
- At each time step, each particle stochastically accelerates toward its *pbest* and *gbest* (or *lbest*).

Particle Swarm Optimization Process

1. Initialize population in hyperspace.
2. Evaluate fitness of individual particles.
3. Modify velocities based on previous best and global (or neighborhood) best.
4. Terminate on some condition.
5. Go to step 2.

PSO Velocity Update Equations

- Basic version:

$$v_{id} = w_i v_{id} + c_1 rand()(p_{id} - x_{id}) + c_2 Rand()(p_{gd} - x_{id})$$

$$x_{id} = x_{id} + v_{id}$$

where ***d*** is the dimension, ***c1*** and ***c2*** are positive constants,***rand*** and ***Rand*** are random functions, and ***w*** is the inertia weight.

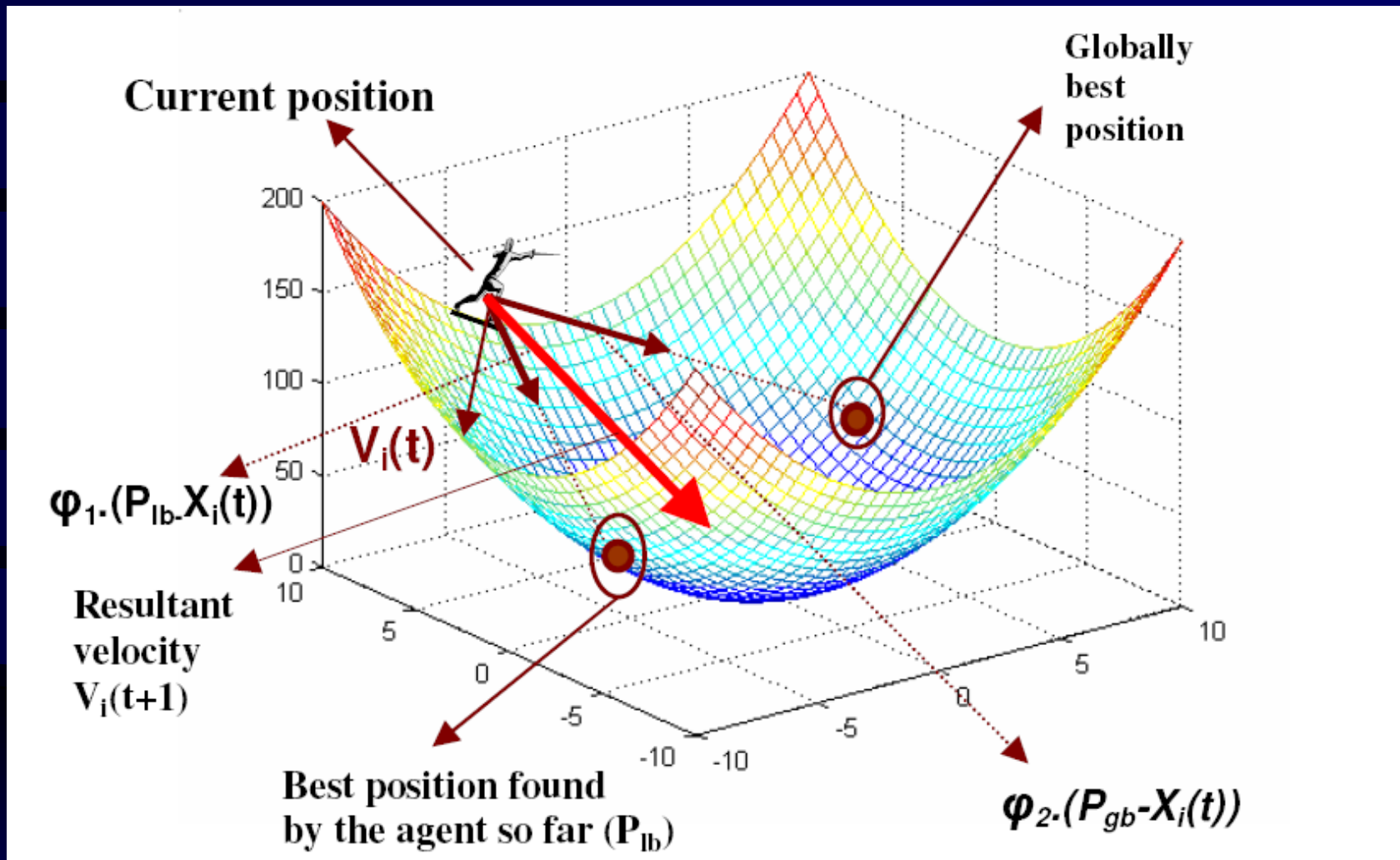


Fig. 4: Illustrating the velocity updating scheme of basic PSO

Basic PSO (*bPSO*)

In the basic PSO method, (*bPSO*), a swarm of particles fly through an *N*-dimensional search space where each particle represents a potential solution to the optimization problem. Each particle *a* in the swarm, $\xi = \{x_1,..,x_a,..,x_S\}$, is represented by the following characteristics:

$x_{a,j}(t)$: j^{th} dimensional component of the position of particle *a*, at time *t*

$v_{a,j}(t)$: j^{th} dimensional component of the velocity of particle *a*, at time *t*

$y_{a,j}(t)$: j^{th} dimensional component of the personal best (*pbest*) position of particle *a*, at time *t*

$\hat{y}_j(t)$: j^{th} dimensional component of the global best position of swarm, at time *t*

Let *f* denote the fitness function to be optimized. Without loss of generality assume that the objective is to find the minimum of *f* in *N* dimensional space. Then the personal best of particle *a* can be updated in iteration *t*+1 as,

bPSO ..

$$y_{a,j}(t+1) = \begin{cases} y_{a,j}(t) & if\ f(x_a(t+1)) > f(y_a(t)) \\ x_{a,j}(t+1) & else \end{cases} \Bigg\} j = 1,2,...,N \qquad (1)$$

Since *gbest* is the index of the GB particle, then $\hat{y}(t) = y_{gbest}(t) = \min(y_1(t),..., y_S(t))$. Then for each iteration in a PSO process, positional updates are performed for each dimension component, $j \in \{1, N\}$ and for each particle, $a \in \{1, S\}$, as follows:

$$v_{a,j}(t+1) = w(t)\, v_{a,j}(t) + c_1 r_{1,j}(t)\left(y_{a,j}(t) - x_{a,j}(t)\right) + c_2 r_{2,j}(t)\left(\hat{y}_j(t) - x_{a,j}(t)\right) \qquad (2)$$

$$x_{a,j}(t+1) = x_{a,j}(t) + v_{a,j}(t+1)$$

bPSO (*termination criteria:{IterNo,* ε_C *,...},* $V_{\max}$)

1. For $\forall a \in \{1, S\}$ do:
   1.1. Randomize $x_a(0), v_a(0)$
   1.2. Let $y_a(0) = x_a(0)$
2. End For.
3. For $\forall t \in \{1, IterNo\}$ do:
   3.1. For $\forall a \in \{1, S\}$ do:
      3.1.1. Compute $y_a(t)$ using (1)
      3.1.2. If ($f(y_a(t)) < f(\hat{y}(t-1))$) then *gbest* = *a*
   3.2. End For.
   3.3. If the *termination criteria* is met, then **Stop**.
   3.4. For $\forall a \in \{1, S\}$ do:
      3.4.1. For $\forall j \in \{1, N\}$ do:
         3.4.1.1. Compute $v_{a,j}(t)$ using (2)
         3.4.1.2. if($\left|v_{a,j}(t)\right| > V_{\max}$) then clamp it to $\left|v_{a,j}(t)\right| = V_{\max}$
         3.4.1.3. Compute $x_{a,j}(t)$ using (2)
      3.4.2. End For.
   3.5. End For.
4. End For.

$D_{min} \leq N \leq D_{max}$

MD PSO Algorithm

- Instead of operating at a fixed dimension *N*, the MD PSO algorithm is designed to seek both positional and dimensional optima within a dimension range, (*Dmin*<*N*<*Dmax*).
- In order to accomplish this, each particle has two sets of components, each of which has been subjected to two independent and consecutive processes. The first one is a regular positional PSO, i.e. the traditional velocity updates and due positional shifts in *N* dimensional search (solution) space. The second one is a dimensional PSO, which allows the particle to navigate through dimensions.

MD PSO Algorithm (2)

- Accordingly, each particle keeps track of its last position, velocity and personal best position (*pbest*) in a particular dimension so that when it re-visits that the same dimension at a later time, it can perform its regular “positional” fly using this information.
- The dimensional PSO process of each particle may then move the particle to another dimension where it will remember its positional status and keep “flying” within the positional PSO process in this dimension, and so on.

MD PSO Algorithm (3)

- The swarm, on the other hand, keeps track of the *gbest* particles in all dimensions, each of which respectively indicates the best (global) position so far achieved and can thus be used in the regular velocity update equation for that dimension.
- Similarly the dimensional PSO process of each particle uses its personal best dimension in which the personal best fitness score has so far been achieved.
- Finally, the swarm keeps track of the global best dimension, *dbest*, among all the personal best dimensions.
- The *gbest* particle in *dbest* dimension represents the optimum solution and dimension, respectively.

MD PSO Algorithm (4)

In a MD PSO process at time (iteration) t, each particle a in the swarm, $\xi = \{x_1,..,x_a,..,x_S\}$, is represented by the following characteristics:

$xx_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the position of particle a, in dimension $xd_a(t)$

$vx_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the velocity of particle a, in dimension $xd_a(t)$

$xy_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the personal best (*pbest*) position of particle a, in dimension $xd_a(t)$

gbest(d) : Global best particle index in dimension d

$x\hat{y}_j^d(t)$: j^{th} component (dimension) of the global best position of swarm, in dimension d

$xd_a(t)$: Dimension component of particle a

$vd_a(t)$: Velocity component of dimension of particle a

$\tilde{xd}_a(t)$: Personal best dimension component of particle a

Let f denote the dimensional fitness function that is to be optimized within a certain dimension range, ($D_{\min} \le N \le D_{\max}$). Without loss of generality assume that the objective is to find the minimum (position) of f at the optimum dimension within a multi-dimensional search space. Assume that the particle a visits (back) the same dimension after T iterations (i.e. $xd_a(t) = xd_a(t+T)$), then the personal best position can be updated in iteration $t+T$ as follows,

$$xy_{a,j}^{xd_a(t+T)}(t+T) = \begin{Bmatrix} xy_{a,j}^{xd_a(t)}(t) & if\ f(xx_a^{xd_a(t+T)}(t+T)) > f(xy_a^{xd_a(t)}(t)) \\ xx_{a,j}^{xd_a(t+T)}(t+T) & else \end{Bmatrix} j = 1,2,...,N \quad (3)$$

Furthermore, the personal best dimension of particle a can be updated in iteration $t+1$ as follows,

$$\widetilde{xd}_a(t+1) = \begin{Bmatrix} \widetilde{xd}_a(t) & if\ f(xx_a^{xd_a(t+1)}(t+1)) > f(xy_a^{\widetilde{xd}_a(t)}(t)) \\ xd_a(t+1) & else \end{Bmatrix} \quad (4)$$

Recall that *gbest(d)* is the index of the global best particle at dimension d and let *S(d)* is total number of particles in dimension d, then $\hat{xy}^{dbest}(t) = xy_{gbest(dbest)}^{dbest}(t) = \min(xy_1^{dbest}(t),..,xy_S^{dbest}(t))$. For a particular iteration t, and for a particle $a \in \{1, S\}$, first the positional components are updated in its current dimension, $xd_a(t)$ and then the dimensional update is performed to determine its next ($t+1^{st}$) dimension, $xd_a(t+1)$. The positional update is performed for each dimension component, $j \in \{1, xd_a(t)\}$ as follows:

$$vx_{a,j}^{xd_a(t)}(t+1) = w(t)\,vx_{a,j}^{xd_a(t)}(t) + c_1 r_{1,j}(t)\left(xy_{a,j}^{xd_a(t)}(t) - xx_{a,j}^{xd_a(t)}(t)\right) + c_2 r_{2,j}(t)\left(\hat{xy}_j^{xd_a(t)}(t) - xx_{a,j}^{xd_a(t)}(t)\right) \quad (5)$$

$$xx_{a,j}^{xd_a(t)}(t+1) = xx_{a,j}^{xd_a(t)}(t) + vx_{a,j}^{xd_a(t)}(t+1)$$

MD PSO Algorithm (6)

Note that the particle's new position, $xx_a^{xd_a(t)}(t+1)$, will still be in the same dimension, $xd_a(t)$; however, the particle may fly to another dimension afterwards with the following dimensional update equations:

$$vd_a(t+1) = \left\lfloor vd_a(t) + c_1 r_1(t)\left(\tilde{xd}_a(t) - xd_a(t)\right) + c_2 r_2(t)\left(dbest - xd_a(t)\right)\right\rfloor \quad (6)$$
$$xd_a(t+1) = xd_a(t) + vd_a(t+1)$$

where $\lfloor . \rfloor$ is the *floor* operator. As in (2), we employ the inertia weight for positional velocity update; however, we experienced no benefit of using it for dimensional PSO, and hence we left it out of (6) for the sake of simplicity. To avoid exploding, along with the (positional) velocity limit $V_{\max}$, two more clamping are applied for dimensional PSO components, such as $\left|vd_{a,j}(t+1)\right| < VD_{\max}$ and the initial dimension range set by the user, $D_{\min} \le xd_a(t) \le D_{\max}$.

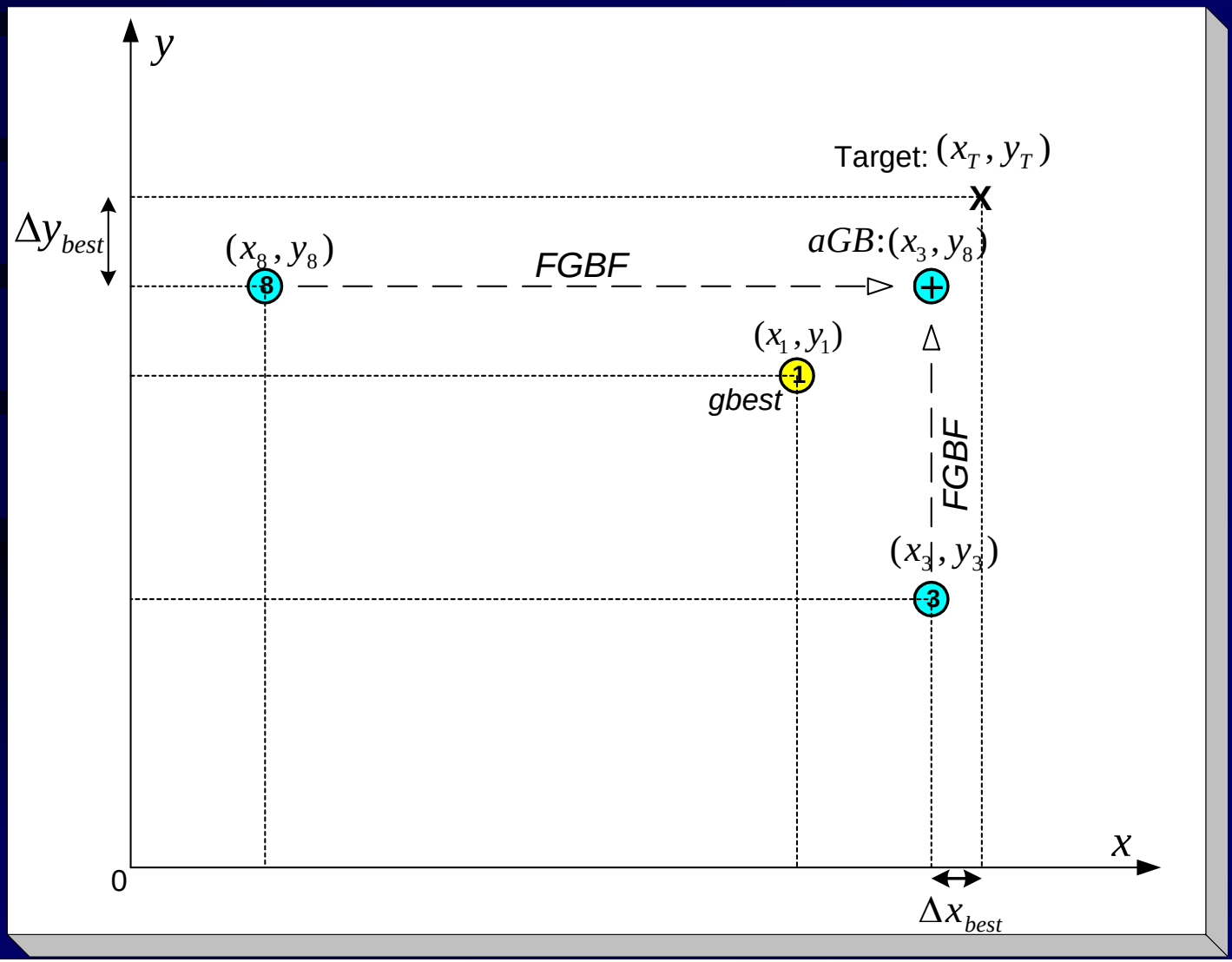
MD PSO (*termination criteria:{IterNo,* ε_C *, ...},* $V_{\max}$, $VD_{\max}$, $D_{\min}$, $D_{\max}$)

1. For $\forall a \in \{1, S\}$ do:
   - 1.1. Randomize $xd_a(0), vd_a(0)$
   - 1.2. Initialize $\widetilde{xd}_a(0) = xd_a(0)$
   - 1.3. For $\forall d \in \{D_{\min}, D_{\max}\}$ do:
     - 1.3.1. Randomize $xx_a^d(0)$, $xv_a^d(0)$
     - 1.3.2. Initialize $xy_a^d(0) = xx_a^d(0)$
   - 1.4. End For.
2. End For.
3. For $\forall t \in \{1, IterNo\}$ do:
   - 3.1. For $\forall a \in \{1, S\}$ do:
     - 3.1.1. If ($f(xx_a^{xd_a(t)}(t)) < f(xy_a^{xd_a(t)}(t-1))$) then do:
       - *3.1.1.1.* $xy_a^{xd_a(t)}(t) = xx_a^{xd_a(t)}(t)$
       - *3.1.1.2.* If ($f(xx_a^{xd_a(t)}(t)) < f(xy_{gbest(xd_a(t))}^{xd_a(t)}(t-1))$) then $gbest\,(xd_a(t))=a$
       - *3.1.1.3.* If ($f(xx_a^{xd_a(t)}(t)) < f(xy_a^{\widetilde{xd}_a(t-1)}(t-1))$) then $\widetilde{xd}_a(t) = xd_a(t)$
       - *3.1.1.4.* If ($f(xx_a^{xd_a(t)}(t)) < f(x\hat{y}^{dbest}(t-1))$) then $dbest = xd_a(t)$
     - 3.1.2. End If.
   - 3.2. End For.
   - 3.3. If the *termination criteria* are met, then **Stop**.
   - 3.4. For $\forall a \in \{1, S\}$ do:
     - 3.4.1. For $\forall j \in \{1, xd_a(t)\}$ do:
       - *3.4.1.1.* Compute $vx_{a,j}^{xd_a(t)}(t)$ using (5)
       - *3.4.1.2.* if($\left|vx_{a,j}^{xd_a(t)}(t)\right| > V_{\max}$) then clamp it to $\left|vx_{a,j}^{xd_a(t)}(t)\right| = V_{\max}$
       - *3.4.1.3.* Compute $xx_{a,j}^{xd_a(t)}(t)$ using (5)
     - 3.4.2. End For.
     - 3.4.3. Compute $vd_a(t)$ using (6)
     - 3.4.4. if($\left|vd_a(t+1)\right| > VD_{\max}$) then clamp it to $\left|vd_a(t+1)\right| = VD_{\max}$
     - 3.4.5. Compute $xd_a(t+1)$ using (6)
     - 3.4.6. if($xd_a(t+1) < D_{\min}$) then $xd_a(t+1) = D_{\min}$)
     - 3.4.7. if($xd_a(t+1) > D_{\max}$) then $xd_a(t+1) = D_{\max}$)
   - 3.5. End For.
4. End For.

Fractional GB formation (FGBF)

- Fractional GB formation (FGBF) is designed to avoid the premature convergence by providing a significant diversity obtained from a proper *fusion* of the swarm's best components (the individual dimension(s) of the current position of each particle in the swarm). At each iteration in a *bPSO* process, an artificial GB particle (*aGB*) is (fractionally) formed by selecting the most promising (or simply the best) particle (dimensional) components from the entire swarm.

FGBF (2)


y
Target: (x_T, y_T)
X
Δy_{best}
(x_8, y_8)
FGBF
$aGB:(x_3, y_8)$
(x_1, y_1)
gbest
FGBF
(x_3, y_3)
x
0
Δx_{best}

FGBF (3)

- Therefore, especially during the initial steps, the FGBF can be and most of the time, is a better alternative than the native *gbest* particle since it has the advantage of assessing each dimension of every particle in the swarm individually, and forming the *aGB* particle fractionally by using the most promising (or simply the best) components among them.
- This process naturally uses the available diversity among individual dimensional components and thus it can prevent swarm from trapping in local optima due to its ongoing and ever-varying FGBF operations. It maintains ongoing transitions (changes) towards the next (and better) *aGB* particle.
- At each iteration FGBF is performed after the assignment of the swarm's *gbest* particle (i.e. performed between steps 3.2 and 3.3 in the pseudo-code of *bPSO*) and the best one between the two will be the GB particle, which is used in swarm's velocity updates. In other words, the swarm will be guided only by the best (winner) GB particle at any time. Such a competitive GB selection is repeated each time a better *aGB* and/or *gbest* particle is obtained.

FGBF in MD PSO

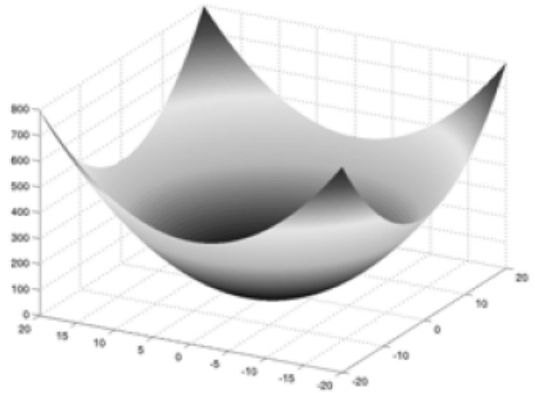
FGBF in MD PSO ($f(a, j)$)

1. Create a new *aGB* particle, $\{xx^{d}_{aGB,j}(t),\ xy^{d}_{aGB,j}(t)\}\ for\ \forall d \in \{D_{\min}, D_{\max}\}, \forall j \in \{1, d\}$
2. Let $a[j] = \arg \min_{a \in \{1,S\}\ j \in \{1, D_{\max}\}} (f(a, j))$ be the index array of particles yielding the minimum $f(a, j)$ for the j^{th} dimension.
3. For $\forall d \in \{D_{\min}, D_{\max}\}$ do:
   - 3.1. $xx^{d}_{aGB,j}(t) = xx^{d}_{a[j],j}(t)\ for\ \forall j \in \{1, d\}$
   - 3.2. If ($f(xx^{d}_{aGB}(t)) < f(xy^{d}_{aGB}(t-1))$) then $xy^{d}_{aGB}(t) = xx^{d}_{aGB}(t)$
   - 3.3. Else $xy^{d}_{aGB}(t) = xy^{d}_{aGB}(t-1)$
   - 3.4. If ($f(xy^{d}_{aGB}(t)) < f(xy^{d}_{gbest(d)}(t))$ then $xy^{d}_{gbest(d)}(t) = xy^{d}_{aGB}(t)$
4. End For.
5. Re-assign *dbest*: $dbest = \arg \min_{d \in \{D_{\min}, D_{\max}\}} (f(xy^{d}_{gbest(d)}(t))$

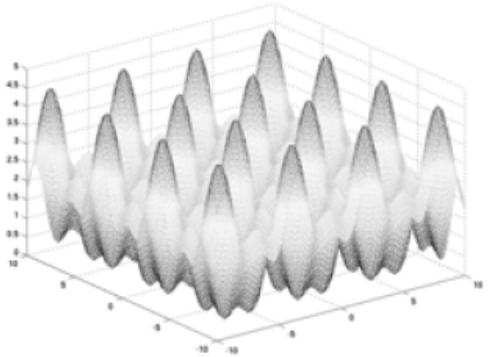
Experimental Results
1- Function Minimization

Table I: Benchmark functions with dimensional bias

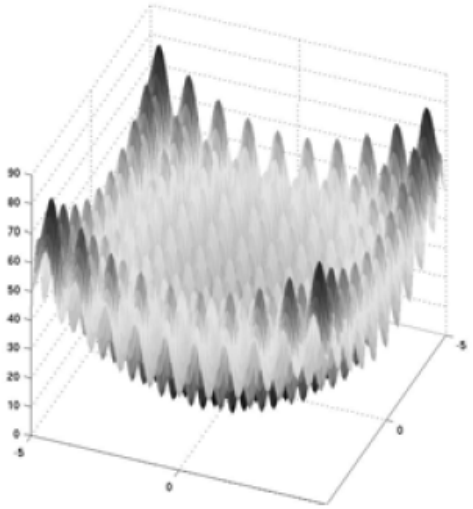
Function	Formula	Initial Range $\pm x_{max}$	Dimension Range $\{D_{min}, D_{max}\}$
Sphere	$F_1(x,d_0)=\left(\sum_{i=1}^{d} x_i^2\right)+(d-d_0)^4$	±150	{2, 100}
De Jong	$F_2(x,d_0)=\left(\sum_{i=1}^{d} i x_i^4\right)+(d-d_0)^4$	±50	{2, 100}
Rosenbrock	$F_3(x,d)=\left(\sum_{i=1}^{d} 100\,(x_{i+1}-x_i^2)^2+(x_i-1)^2\right)+(d-d_0)^4$	±50	{2, 100}
Rastrigin	$F_4(x,d_0)=\left(\sum_{i=1}^{d} 10+x_i^2-10Cos(2\pi x_i)\right)+(d-d_0)^4$	±50	{2, 100}
Griewank	$F_5(x,d_0)=\left(\frac{1}{4000}\sum_{i=1}^{d} x_i^2-\prod_{i=1}^{d} Cos\left(\frac{x_i}{\sqrt{i+1}}\right)\right)+0.2(d-d_0)^2$	±500	{2, 100}
Schwefel	$F_6(x,d_0)=\left(418.9829\,d+\sum_{i=1}^{d} x_i\,Sin\left(\sqrt{\lvert x_i\rvert}\right)\right)+40(d-d_0)^2$	±500	{2, 100}
Giunta	$F_7(x,d_0)=\left(\sum_{i=1}^{d} Sin\left(\frac{16}{15}x_i-1\right)+Sin^2\left(\frac{16}{15}x_i-1\right)+\frac{1}{50}Sin\left(4\left(\frac{16}{15}x_i-1\right)\right)+\frac{268}{1000}\right)+\sqrt{\lvert d-d_0\rvert}$	±500	{2, 100}



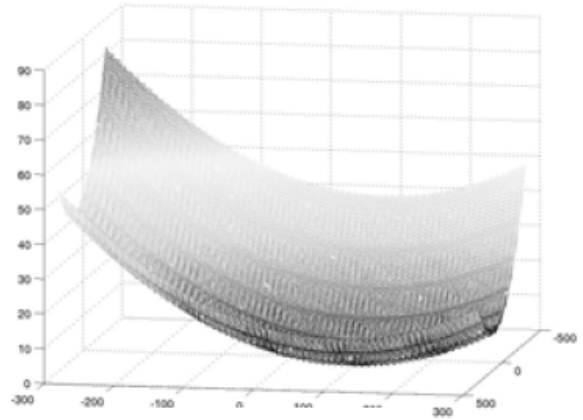
a) Sphere



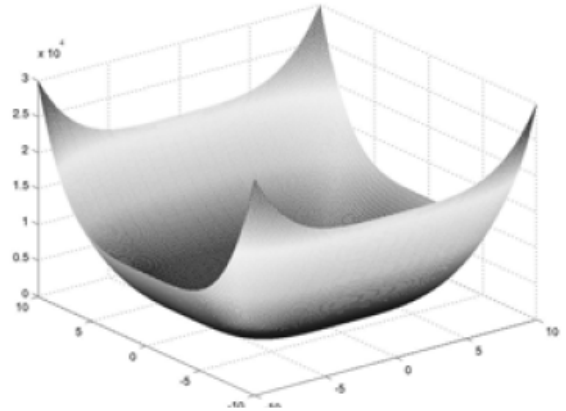
b) Giunta



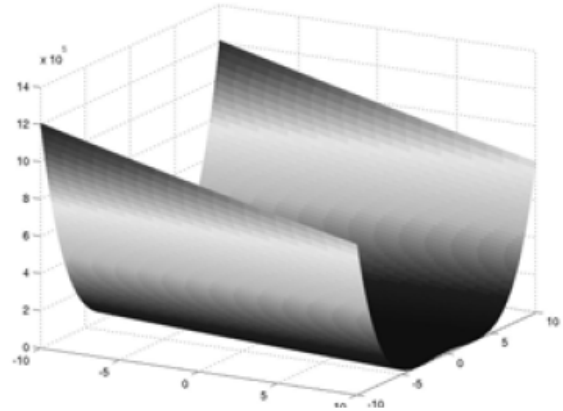
c) Rastrigin



d) Griewank



e) DeJong



f) Rosenbrock

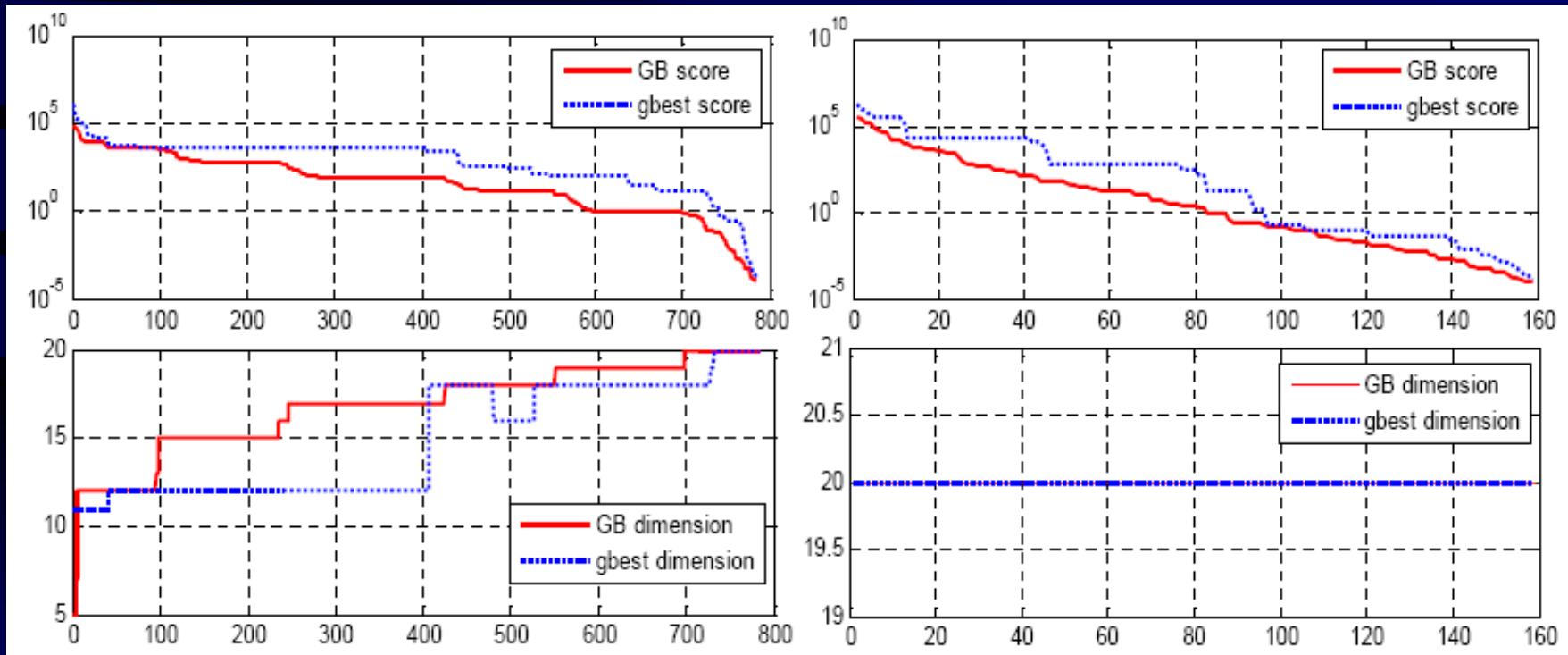


Figure 3: Fitness score (top in log-scale) and dimension (bottom) plots vs. iteration number for MD PSO (left) and *bPSO* (right) operations both of which run over *De Jong* function.

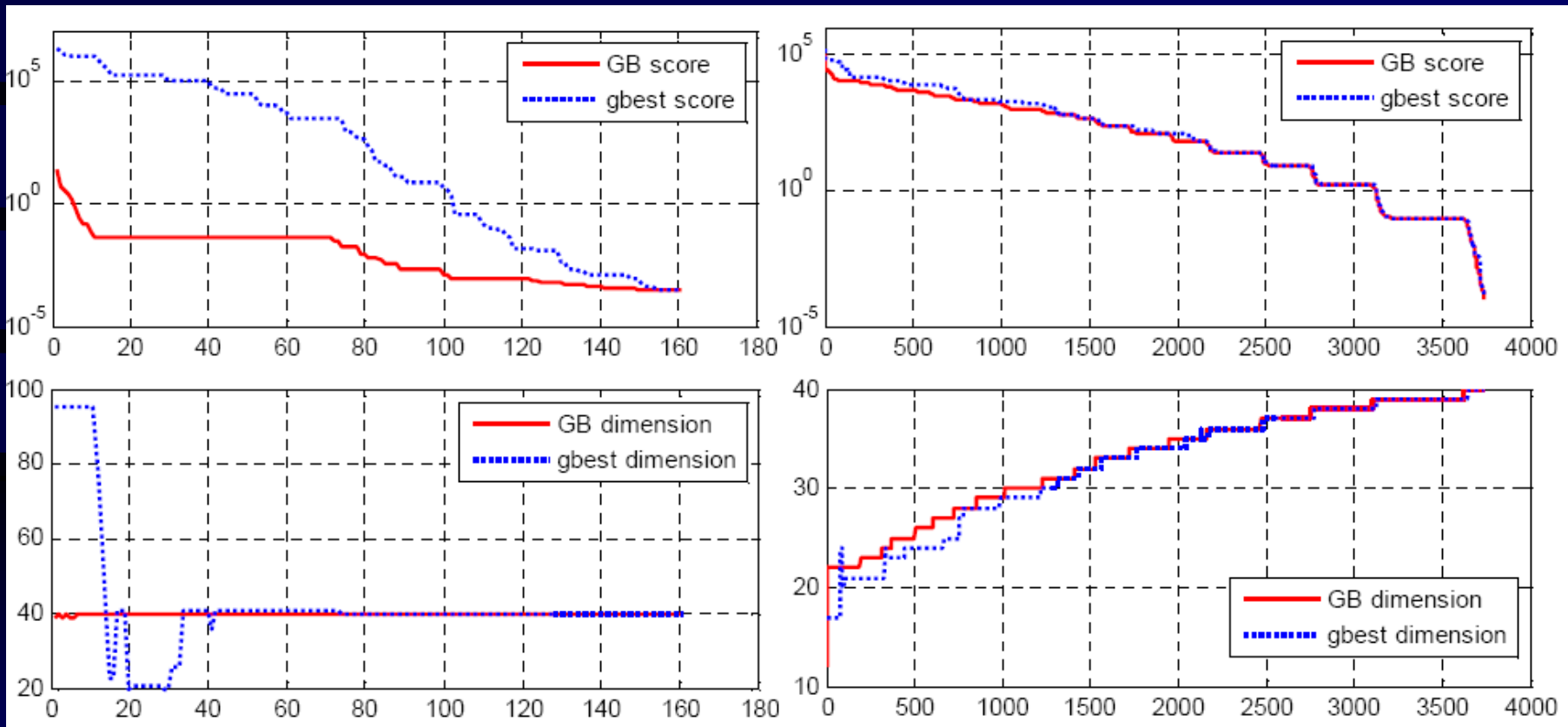


Figure 4: Fitness score (top in log-scale) and dimension (bottom) plots vs. iteration number for a MD PSO run over *Sphere* function with (left) and without (right) FGBF.

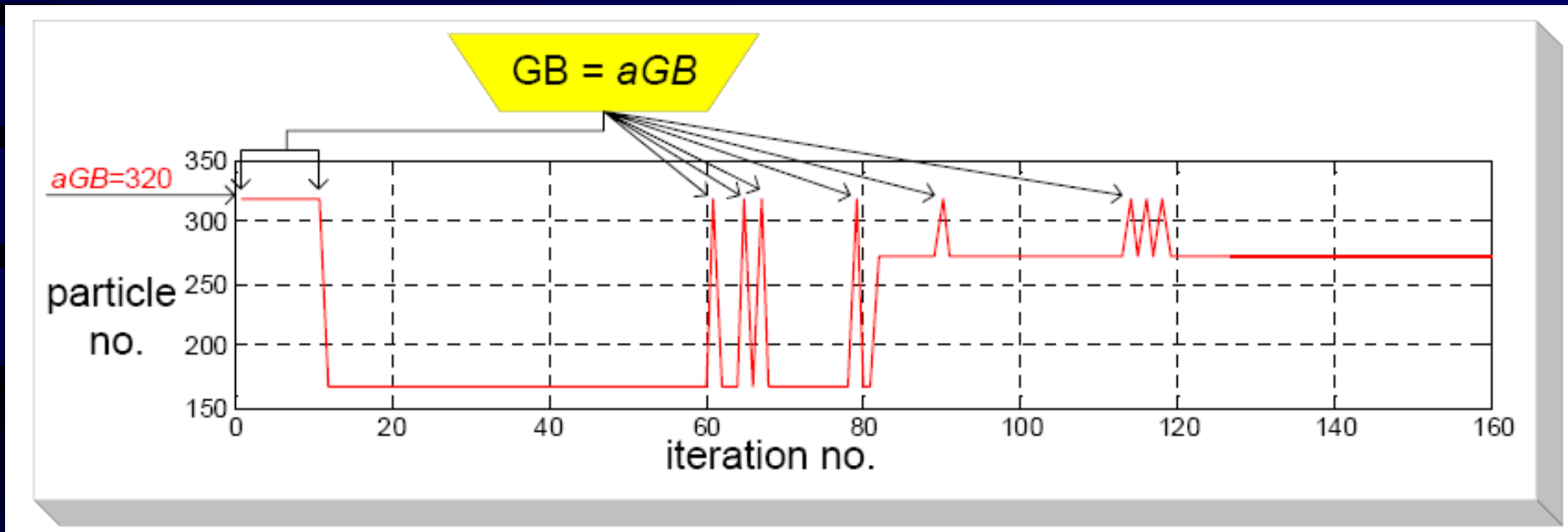


Figure 5: Particle index plot for the MD PSO with FGBF operation shown in Figure 4.

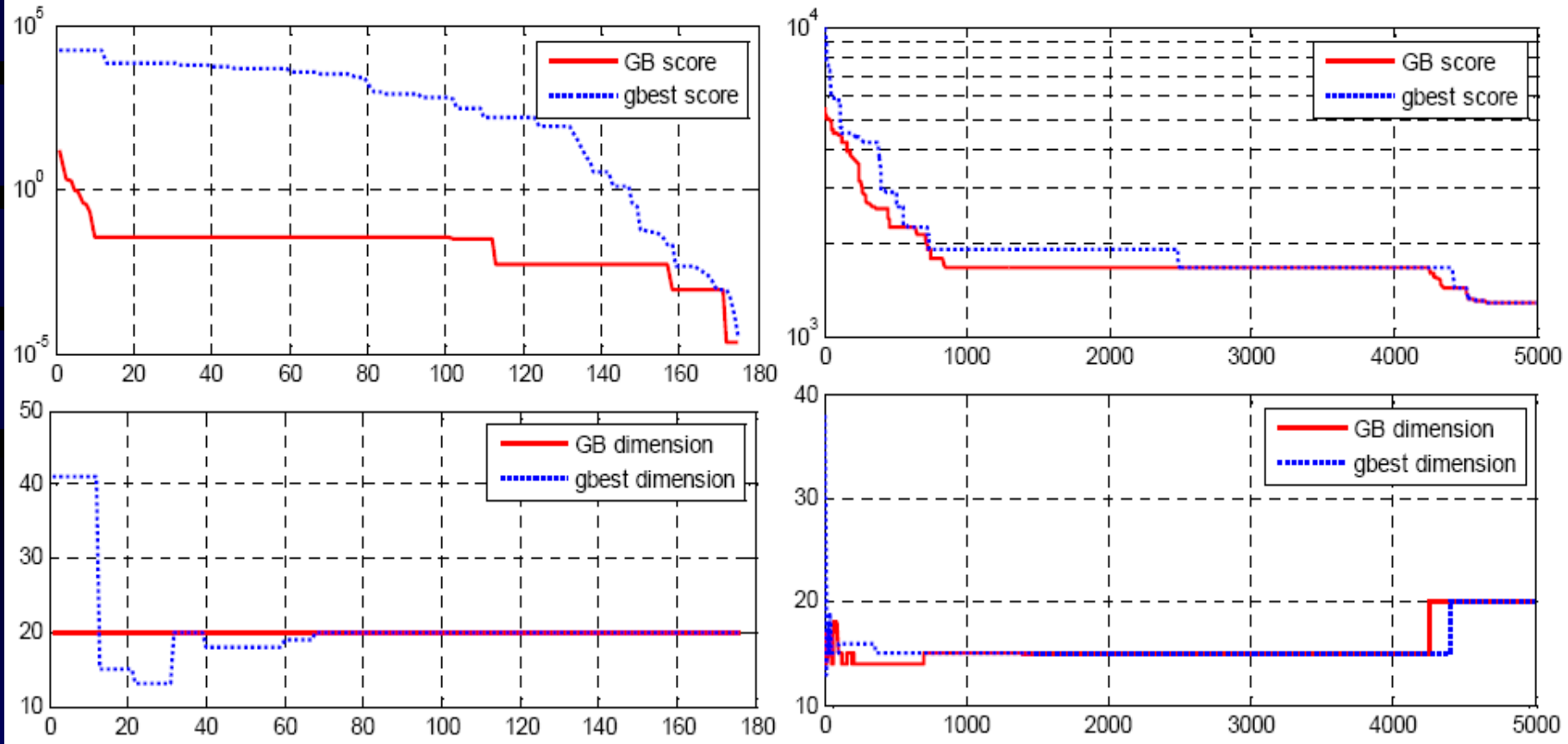


Figure 6: Fitness score (top in log-scale) and dimension (bottom) plots vs. iteration number for a MD PSO run over *Schwefel* function with (left) and without (right) FGBF.

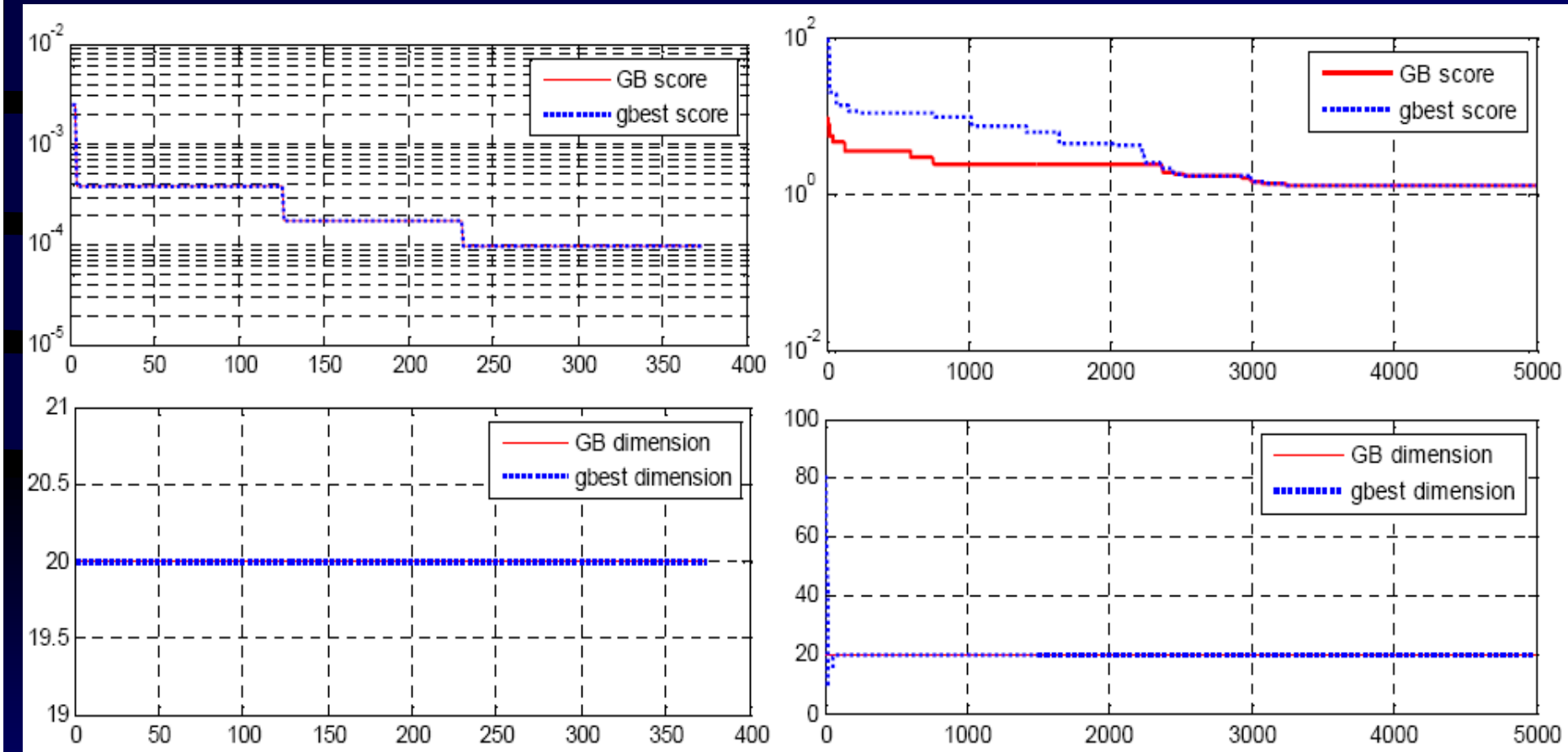


Figure 7: Fitness score (top in log scale) and dimension (bottom) plots vs. iteration number for a MD PSO run over *Giunta* function with (left) and without (right) FGBF.

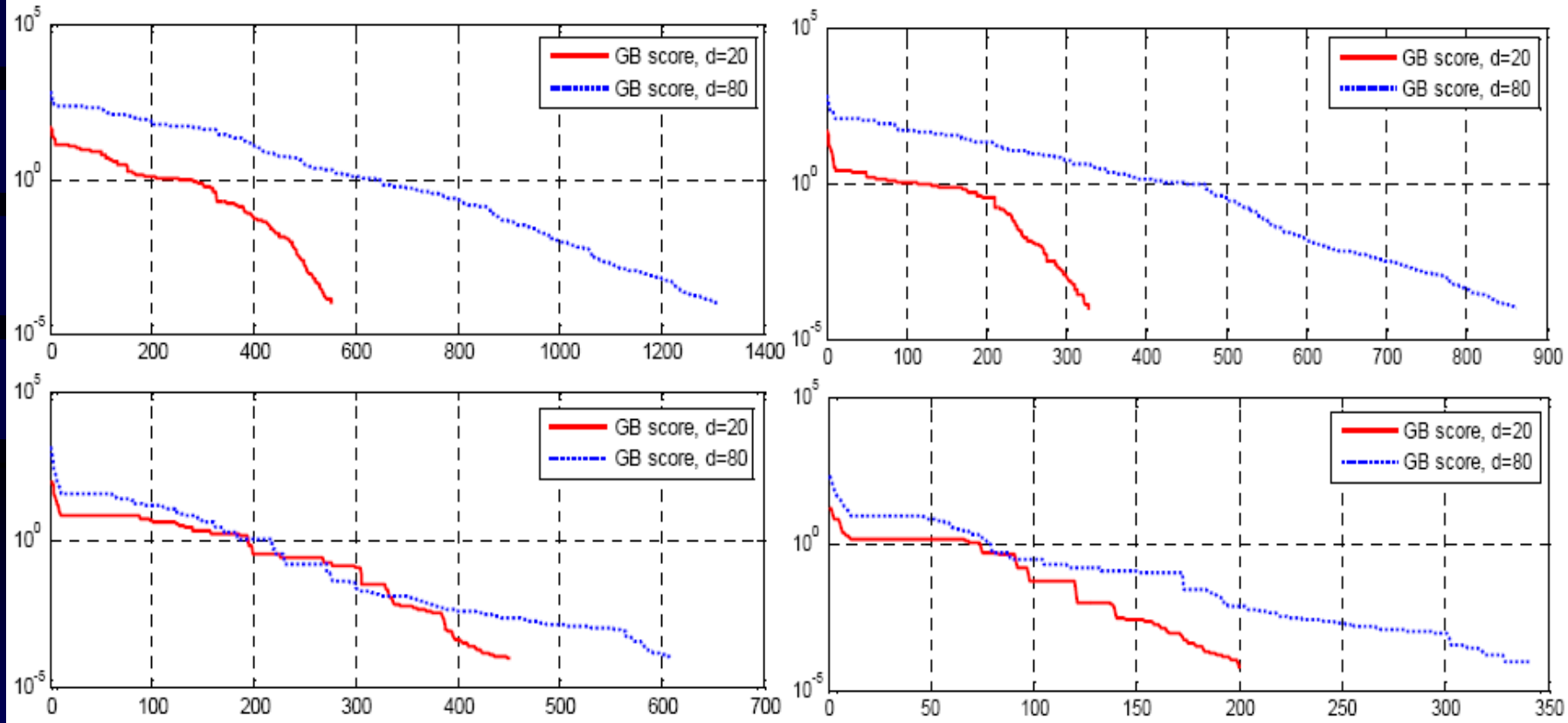


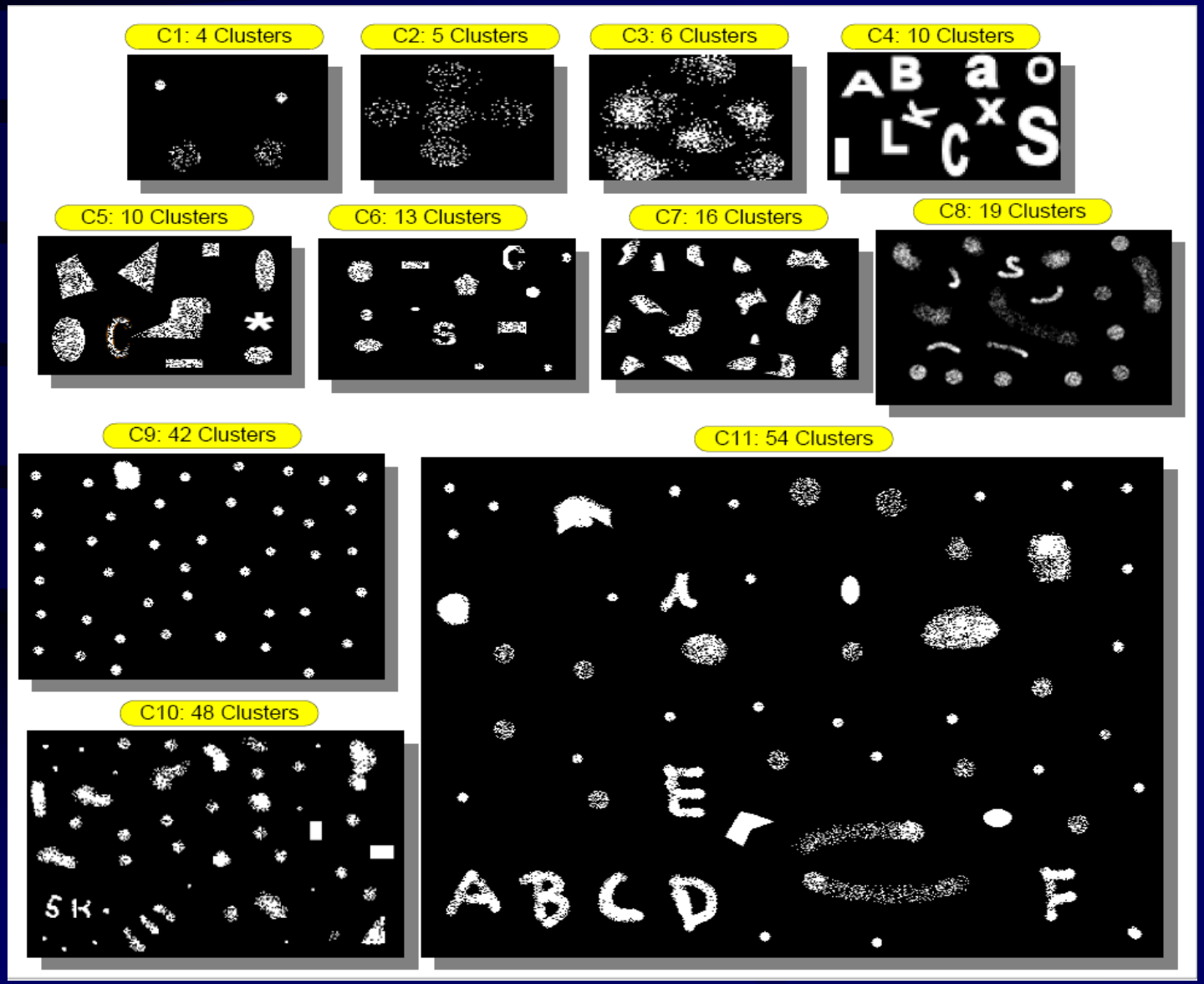
Figure 8: MD PSO with FGBF operation over *Griewank* (top) and *Rastrigin* (bottom) functions with $d_0 = 20$(red) and $d_0 = 80$(blue) using the swarm size, *S*=80 (left) and *S*=320 (right).

Table II: Statistical results from 100 runs over 7 benchmark functions

Functions			Standalone MD PSO						MD PSO with FGBF					
			Score		Iter. No		*dbest*		Score		Iter. No		*dbest*	
	S	d_0	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Sphere	160	20	0	0	1475	117	20	0	0	0	166	27	20	0
		50	0	0	4605	280	50	0	0	0	172	37	50	0
		80	45.066	61.23	4659	1046	78.5	1.234	0	0	169	34	80	0
	320	20	0	0	1024	58	20	0	0	0	133	16	20	0
		50	0	0	3166	631	50	0	0	0	131	25	50	0
		80	5.949	7.584	4712	626	79	0.858	0	0	126	24	80	0
	640	20	0	0	932	161	20	0	0	0	93	15	20	0
		50	0	0	2612	701	50	0	0	0	101	15	50	0
		80	0.317	0.462	4831	343	79.7	0.47	0	0	95	15	80	0
De Jong	160	20	0	0	1047	74	20	0	0	0	6	1	20	0
		50	0.705	0.462	5000	0	49.3	0.47	0	0	18	28	50	0
		80	5184.4	1745.7	5000	0	71.65	0.745	0	0	102	24	80	0
	320	20	0	0	833	66	20	0	0	0	4	1	20	0
		50	0	0	4004	166	50	0	0	0	6	1	50	0
		80	811.02	323.37	5000	0	74.85	0.587	0	0	19	20	80	0
	640	20	0	0	643	69	20	0	0	0	2	1	20	0
		50	0	0	3184	145	50	0	0	0	4	1	50	0
		80	67.452	24.75	5000	0	77.3	0.47	0	0	7	1	80	0
Rosenbrock	160	20	0.0008	0.0004	4797	903	20	0	0	0	1619	1785	20	0
		50	0.009	0.003	5000	0	50	0	0	0	398	149	50	0
		80	7.617	7.834	500	0	78.75	0.786	0	0	367	98	80	0
	320	20	0.001	0.002	4317	1221	20	0	0	0	325	239	20	0
		50	0.009	0.004	5000	0	50	0	0	0	257	33	50	0
		80	0.49	0.495	5000	0	79.65	0.587	0	0	258	47	80	0
	640	20	0.0009	0.002	4560	982	20	0	0	0	160	37	20	0
		50	0.006	0.003	5000	0	50	0	0	0	193	58	50	0
		80	0.018	0.005	5000	0	80	0	0	0	189	23	80	0
Rastrigin	160	20	2.137	0.565	5000	0	19.35	0.49	0	0	304	40	20	0
		50	24.051	5.583	5000	0	48.2	0.41	0	0	392	49	50	0
		80	212.375	165.196	5000	0	77.15	1.871	0	0	375	55	80	0
	320	20	1.506	0.531	5000	0	19.35	0.489	0	0	228	21	20	0
		50	9.788	4.181	5000	0	48.95	0.224	0	0	277	47	50	0
		80	77.317	31.909	5000	0	77.45	0.605	0	0	272	65	80	0
	640	20	0.96	0.527	5000	0	19.501	0.513	0	0	164	15	20	0
		50	7.308	2.581	5000	0	49.3	0.657	0	0	206	30	50	0
		80	28.709	8.399	5000	0	78.3	0.933	0	0	195	47	80	0

Griewank	160	20	3.262	14.485	4163	1486	19.101	4.025	0	0	438	78	20	0
		50	0.232	0.271	4779	673	49.2	0.696	0	0	725	36	50	0
		80	9.499	5.731	5000	0	73.9	2.435	0	0	1029	57	80	0
	320	20	0.019	0.016	4219	1601	20	0	0	0	395	76	20	0
		50	0.007	0.009	4537	454	50	0	0	0	618	34	50	0
		80	3.059	1.857	5000	0	76.45	1.099	0	0	866	43	80	0
	640	20	0.017	0.017	4205	1638	20	0	0	0	325	74	20	0
		50	0.016	0.02	4306	877	50	0	0	0	531	37	50	0
		80	0.675	0.385	5000	0	78.25	0.55	0	0	710	44	80	0
Schwefel	160	20	1432.3	336.76	5000	0	16.30	1.418	0	0	215	33	20	0
		50	6036.2	597.33	5000	0	45.45	1.234	0	0	199	29	50	0
		80	11304	1489.8	5000	0	74.7	2.226	0	0	161	35	80	0
	320	20	1261.7	320.709	5000	0	16.20	1.508	0	0	168	36	20	0
		50	5288.8	576.132	5000	0	45.55	1.394	0	0	146	24	50	0
		80	8882.8	1434.3	5000	0	75.8	1.814	0	0	120	22	80	0
	640	20	946.612	264.635	5000	0	16.85	1.268	0	0	121	21	20	0
		50	4412.2	921.062	5000	0	45.2	2.067	0	0	103	15	50	0
		80	8032.1	1180.9	5000	0	75.2	2.447	0	0	85	12	80	0
Giunta	160	20	1.488	0.69	5000	0	20	0	0	0	793	626	20	0
		50	0.776	0.207	5000	0	50	0	0	0	699	281	50	0
		80	1.131	0.14	5000	0	80	0	0	0	863	293	80	0
	320	20	1.003	0.582	5000	0	20	0	0	0	128	92	20	0
		50	0.543	0.127	5000	0	50	0	0	0	283	113	50	0
		80	1.140	0.756	5000	0	80	0	0	0	456	115	80	0
	640	20	0.675	0.517	5000	0	20	0	0	0	1	1	20	0
		50	0.418	0.140	5000	0	50	0	0	0	4	4	50	0
		80	0.789	0.145	5000	0	80	0	0	0	20	6	80	0

Data Clustering



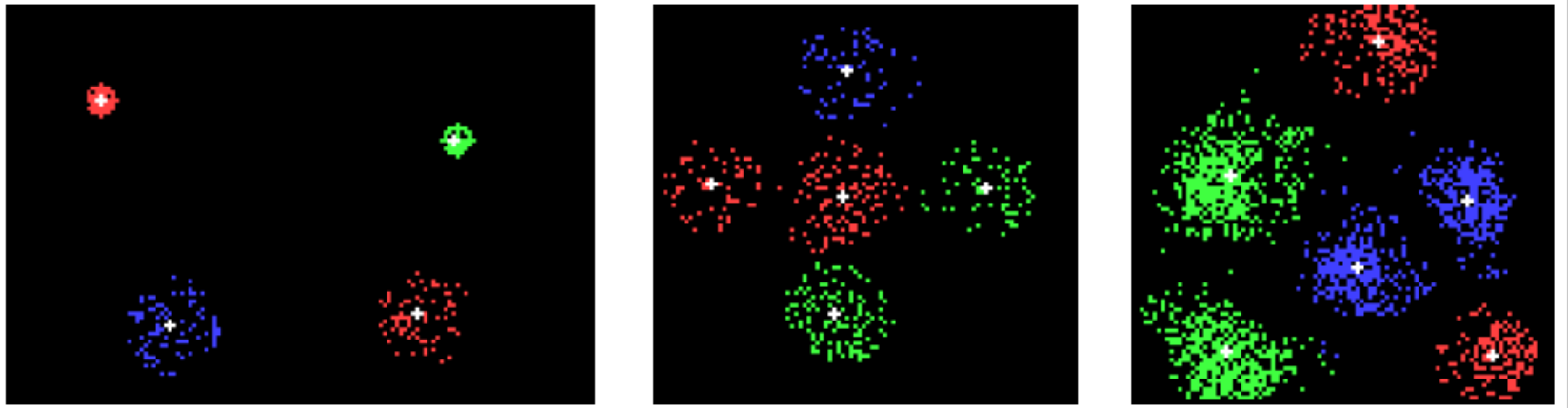


Figure 10: The standalone MD PSO (and *bPSO*) clustering for data spaces C1-3 shown in Figure 10.

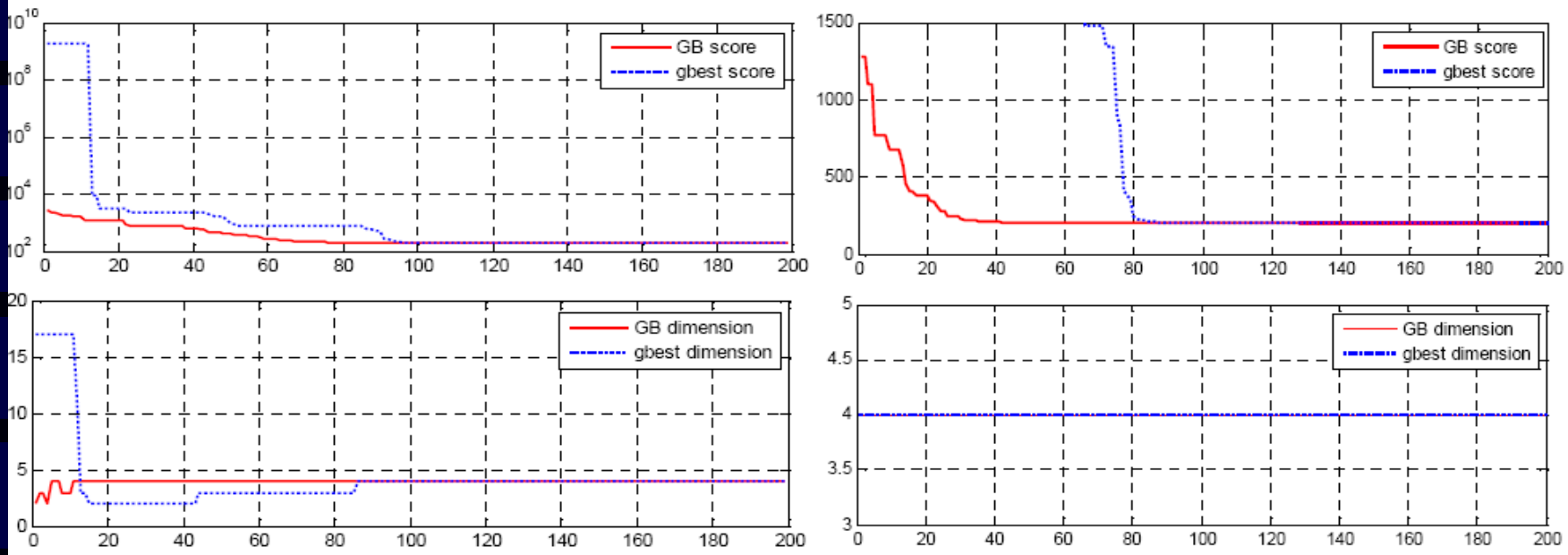


Figure 11: Fitness score (top) and dimension (bottom) plots vs. iteration number for MD PSO (left) and *bPSO* (right) clustering both of which run over C1.

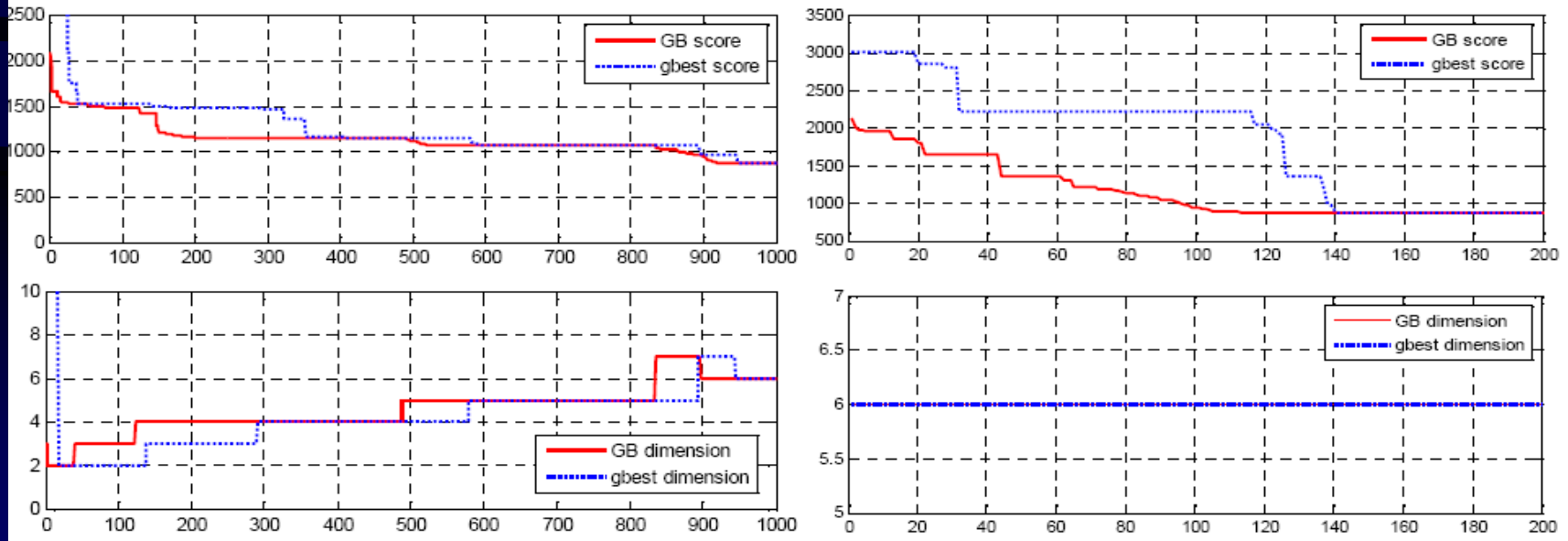


Figure 12: Fitness score (top) and dimension (bottom) plots vs. iteration number for MD PSO (left) and *bPSO* (right) clustering both of which run over C3.

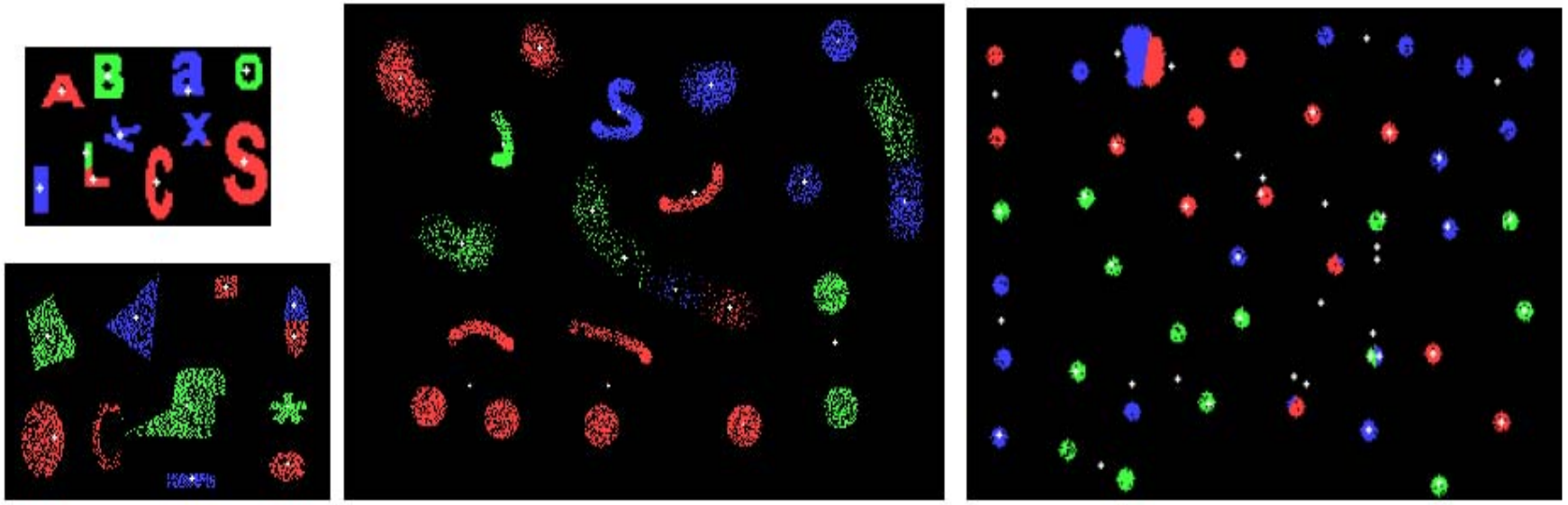


Figure 13: Erroneous *bPSO* clustering over data spaces C4, C5, C6 and C9 shown in Figure 10.

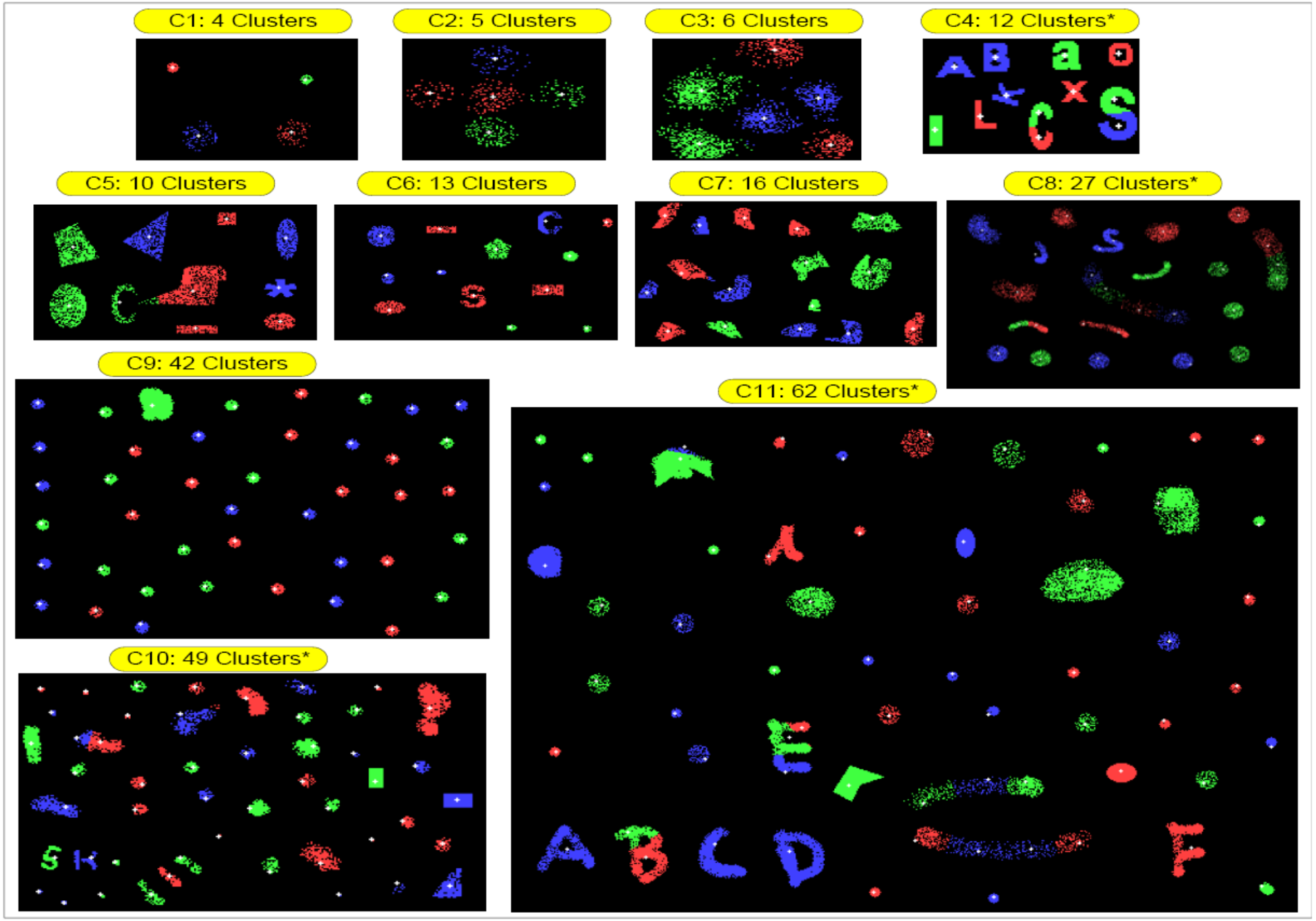


Figure 14: Typical clustering results via MD PSO with FGBF. Over-clustered samples are indicated with *.

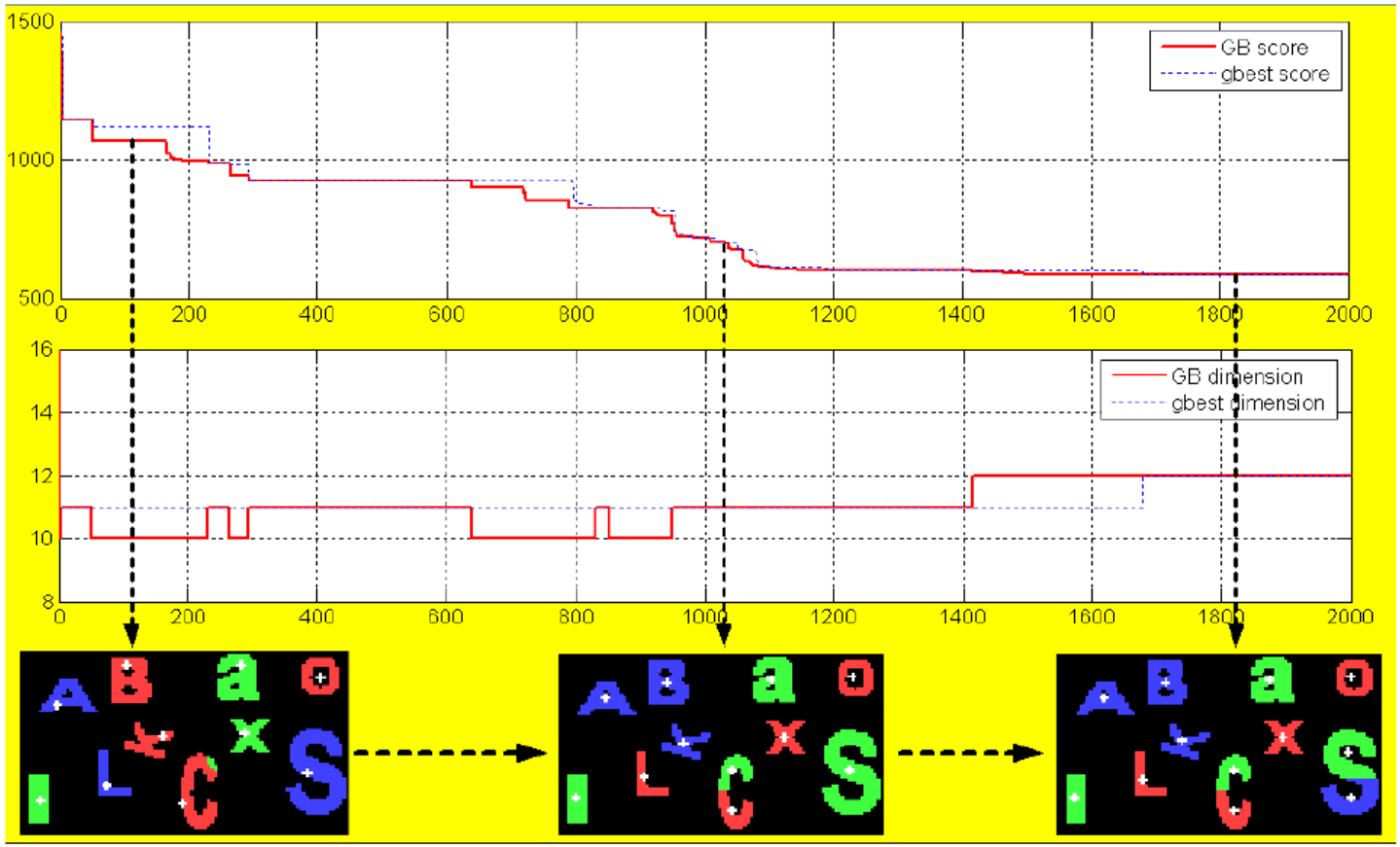


Figure 15: Fitness score (top) and dimension (bottom) plots vs. iteration number for a MD PSO with FGBF clustering operation over C4. 3 clustering snapshots at iterations 105, 1050 and 1850, are presented below.

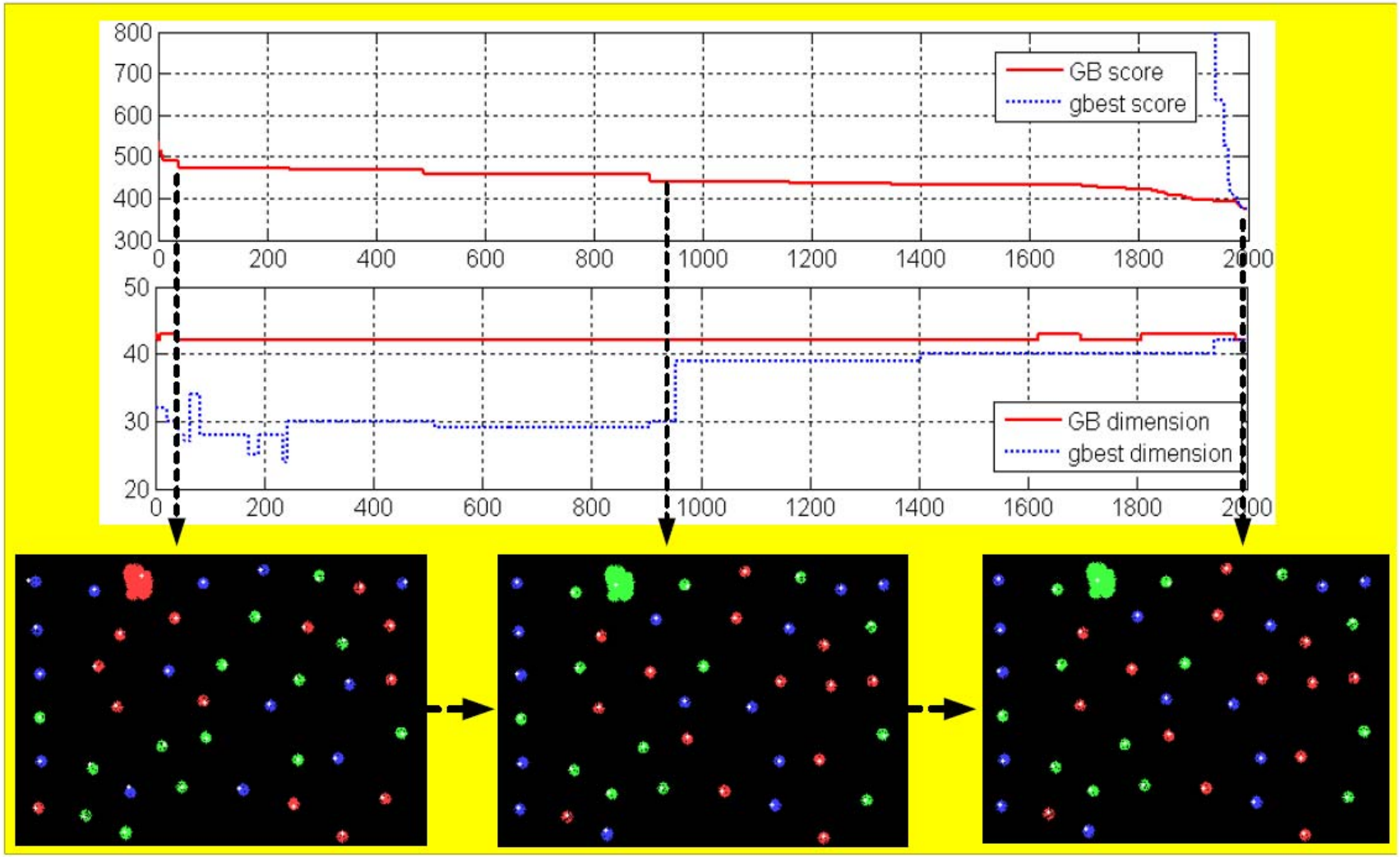


Figure 16: Fitness score (top) and dimension (bottom) plots vs. iteration number for a MD PSO with FGBF clustering operation over C9. 3 clustering snapshots at iterations 40, 950 and 1999, are presented below.