Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives

Swagatam Das[1], Ajith Abraham[2], and Amit Konar[1]

[1] Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata 700032, India, swagatamdas19@yahoo.co.in, konaramit@yahoo.co.in

[2] Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology, Norway, ajith.abraham@ieee.org

Summary. Since the beginning of the nineteenth century, a significant evolution in optimization theory has been noticed. Classical linear programming and traditional non-linear optimization techniques such as Lagrange's Multiplier, Bellman's principle and Pontyagrin's principle were prevalent until this century. Unfortunately, these derivative based optimization techniques can no longer be used to determine the optima on rough non-linear surfaces. One solution to this problem has already been put forward by the evolutionary algorithms research community. Genetic algorithm (GA), enunciated by Holland, is one such popular algorithm. This chapter provides two recent algorithms for evolutionary optimization – well known as particle swarm optimization (PSO) and differential evolution (DE). The algorithms are inspired by biological and sociological motivations and can take care of optimality on rough, discontinuous and multimodal surfaces. The chapter explores several schemes for controlling the convergence behaviors of PSO and DE by a judicious selection of their parameters. Special emphasis is given on the hybridizations of PSO and DE algorithms with other soft computing tools. The article finally discusses the mutual synergy of PSO with DE leading to a more powerful global search algorithm and its practical applications.

1 Introduction

The aim of optimization is to determine the best-suited solution to a problem under a given set of constraints. Several researchers over the decades have come up with different solutions to linear and non-linear optimization problems. Mathematically an optimization problem involves a fitness function describing the problem, under a set of constraints representing the solution space for the problem. Unfortunately, most of the traditional optimization techniques are centered around evaluating the first derivatives to locate the optima on a given constrained surface. Because of the difficulties in evaluating

S. Das et al.: *Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives*, Studies in Computational Intelligence (SCI) **116**, 1–38 (2008)
www.springerlink.com

the first derivatives, to locate the optima for many rough and discontinuous optimization surfaces, in recent times, several derivative free optimization algorithms have emerged. The optimization problem, now-a-days, is represented as an intelligent search problem, where one or more agents are employed to determine the optima on a search landscape, representing the constrained surface for the optimization problem [1].

In the later quarter of the twentieth century, Holland [2], pioneered a new concept on evolutionary search algorithms, and came up with a solution to the so far open-ended problem to non-linear optimization problems. Inspired by the natural adaptations of the biological species, Holland echoed the Darwinian Theory through his most popular and well known algorithm, currently known as genetic algorithms (GA) [2]. Holland and his coworkers including Goldberg and Dejong, popularized the theory of GA and demonstrated how biological crossovers and mutations of chromosomes can be realized in the algorithm to improve the quality of the solutions over successive iterations [3]. In mid 1990s Eberhart and Kennedy enunciated an alternative solution to the complex non-linear optimization problem by emulating the collective behavior of bird flocks, particles, the boids method of Craig Reynolds and socio-cognition [4] and called their brainchild the particle swarm optimization (PSO) [4–8]. Around the same time, Price and Storn took a serious attempt to replace the classical crossover and mutation operators in GA by alternative operators, and consequently came up with a suitable differential operator to handle the problem. They proposed a new algorithm based on this operator, and called it differential evolution (DE) [9].

Both algorithms do not require any gradient information of the function to be optimized uses only primitive mathematical operators and are conceptually very simple. They can be implemented in any computer language very easily and requires minimal parameter tuning. Algorithm performance does not deteriorate severely with the growth of the search space dimensions as well. These issues perhaps have a great role in the popularity of the algorithms within the domain of machine intelligence and cybernetics.

2 Classical PSO

Kennedy and Eberhart introduced the concept of function-optimization by means of a particle swarm [4]. Suppose the global optimum of an n-dimensional function is to be located. The function may be mathematically represented as:

$$f(x_1, x_2, x_3, \ldots, x_n) = f(\vec{X})$$

where $\vec{x}$ is the search-variable vector, which actually represents the set of independent variables of the given function. The task is to find out such a $\vec{x}$, that the function value $f(\vec{x})$ is either a minimum or a maximum denoted by f^* in the search range. If the components of $\vec{x}$ assume real values then the

task is to locate a particular point in the n-dimensional hyperspace which is a continuum of such points.

Example 1. Consider the simplest two-dimensional sphere function given by

$$f(x_1, x_2) = f(\vec{X}) = x_1^2 + x_2^2,$$

if x_1 and x_2 can assume real values only then by inspection it is pretty clear that the global minima of this function is at $x_1 = 0,\ x_2 = 0$, i.e., at the origin (0, 0) of the search space and the minimum value is f(0, 0) = f* = 0. No other point can be found in the $x_1 - x_2$ plane at which value of the function is lower than f* = 0. Now the case of finding the optima is not so easy for some functions (an example is given below):

$$f(x_1, x_2) = x_1 \sin(4\pi x_2) - x_2 \sin(4\pi x_1 + \pi) + 1$$

This function has multiple peaks and valleys and a rough fitness landscape. A surface plot of the function is shown in Fig. 1. To locate the global optima quickly on such a rough surface calls for parallel search techniques. Here many agents start from different initial locations and go on exploring the search space until some (if not all) of the agents reach the global optimal position. The agents may communicate among themselves and share the fitness function values found by them.

PSO is a multi-agent parallel search technique. Particles are conceptual entities, which fly through the multi-dimensional search space. At any particular instant, each particle has a position and a velocity. The position vector of a particle with respect to the origin of the search space represents a trial solution of the search problem. At the beginning, a population of particles is initialized with random positions marked by vectors $\vec{x}_i$ and random velocities $\vec{v}_i$. The population of such particles is called a "swarm" S. A neighborhood

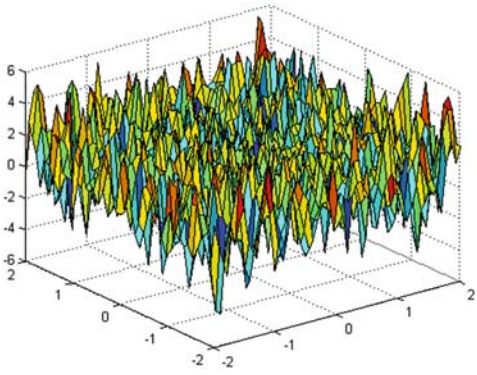

Fig. 1. Surface plot of the above-mentioned function

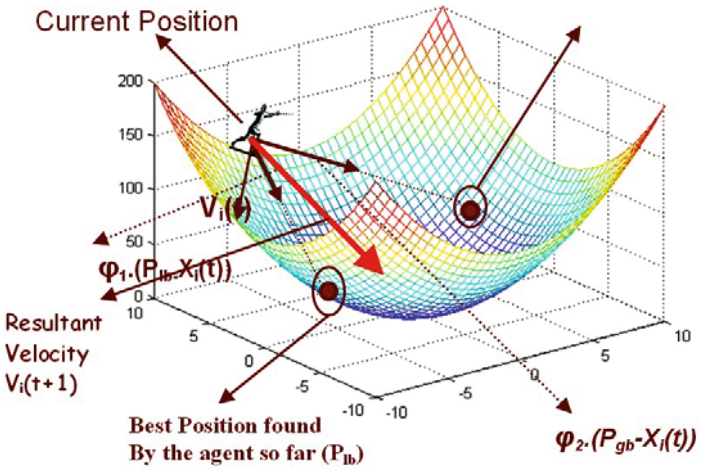

Fig. 2. Illustrating the dynamics of a particle in PSO

relation N is defined in the swarm. N determines for any two particles P_i and P_j whether they are neighbors or not. Thus for any particle P, a neighborhood can be assigned as $N(P)$, containing all the neighbors of that particle. Different neighborhood topologies and their effect on the swarm performance will be discussed later. However, a popular version of PSO uses $N = S$ for each particle. In this case, any particle has all the remaining particles in the swarm in its neighborhood. PSO dynamics is illustrated in Fig. 2.

Each particle P has two state variables viz., its current position $\vec{x}(t)$ and its current velocity $\vec{v}(t)$. It is also equipped with a small memory comprising its previous best position (one yielding the highest value of the fitness function found so far) $\vec{p}(t)$, i.e., personal best experience and the best $\vec{p}(t)$ of all $P \in N(P)$: $\vec{g}(t)$, i.e., the best position found so far in the neighborhood of the particle. When we set $N(P) = S$, $\vec{g}(t)$ is referred to as the globally best particle in the entire swarm. The PSO (PSO) scheme has the following algorithmic parameters:

(a) $V_{\max}$ or maximum velocity which restricts $\vec{V}_i(t)$ within the interval $[-V_{\max}, V_{\max}]$
(b) An inertial weight factor ω
(c) Two uniformly distributed random numbers φ_1 and φ_2 that respectively determine the influence of $\vec{p}(t)$ and $\vec{g}(t)$ on the velocity update formula.
(d) Two constant multiplier terms C_1 and C_2 known as "self-confidence" and "swarm confidence", respectively.

Initially the settings for $\vec{p}(t)$ and $\vec{g}(t)$ are $\vec{p}(0) = \vec{g}(0) = \vec{x}(0)$ for all particles. Once the particles are all initialized, an iterative optimization process begins, where the positions and velocities of all the particles are altered by

the following recursive equations. The equations are presented for the dth dimension of the position and velocity of the ith particle.

$$\left.\begin{aligned} V_{\mathrm{id}}(t+1) &= \omega \cdot v_{\mathrm{id}}(t) + C_1 \cdot \varphi_1 \cdot (P_{\mathrm{id}}(t) - x_{\mathrm{id}}(t)) + C_2 \cdot \varphi_2 \cdot (g_{\mathrm{id}}(t) - x_{\mathrm{id}}(t)) \\ x_{\mathrm{id}}(t+1) &= x_{\mathrm{id}}(t) + v_{\mathrm{id}}(t+1). \end{aligned}\right\} \quad (1)$$

The first term in the velocity updating formula represents the inertial velocity of the particle. "ω" is called the inertia factor. Venter and Sobeiski [10] termed C_1 as "self-confidence" and C_2 as "swarm confidence". These terminologies provide an insight from a sociological standpoint. Since the coefficient C_1 has a contribution towards the self-exploration (or experience) of a particle, we regard it as the particle's self-confidence. On the other hand, the coefficient C_2 has a contribution towards motion of the particles in global direction, which takes into account the motion of all the particles in the preceding program iterations, naturally its definition as "swarm confidence" is apparent. φ_1 and φ_2 stand for a uniformly distributed random number in the interval [0, 1]. After having calculated the velocities and position for the next time step $t + 1$, the first iteration of the algorithm is completed. Typically, this process is iterated for a certain number of time steps, or until some acceptable solution has been found by the algorithm or until an upper limit of CPU usage has been reached. The algorithm can be summarized in the following pseudo code:

The PSO Algorithm

Input: Randomly initialized position and velocity of the particles: $\vec{X}_i(0)$ and $\vec{V}_i(0)$
Output: Position of the approximate global optima $\vec{X}^*$

Begin
While terminating condition is not reached **do**
Begin
for i = 1 to number of particles
Evaluate the fitness:= $f(\vec{X}_i)$;
Update $\vec{p}_i$ and $\vec{g}_i$;
Adapt velocity of the particle using equations (1);
Update the position of the particle;
increase i;
end while
end

The swarm-dynamics has been presented below using a humanoid agent in place of a particle on the spherical fitness-landscape.

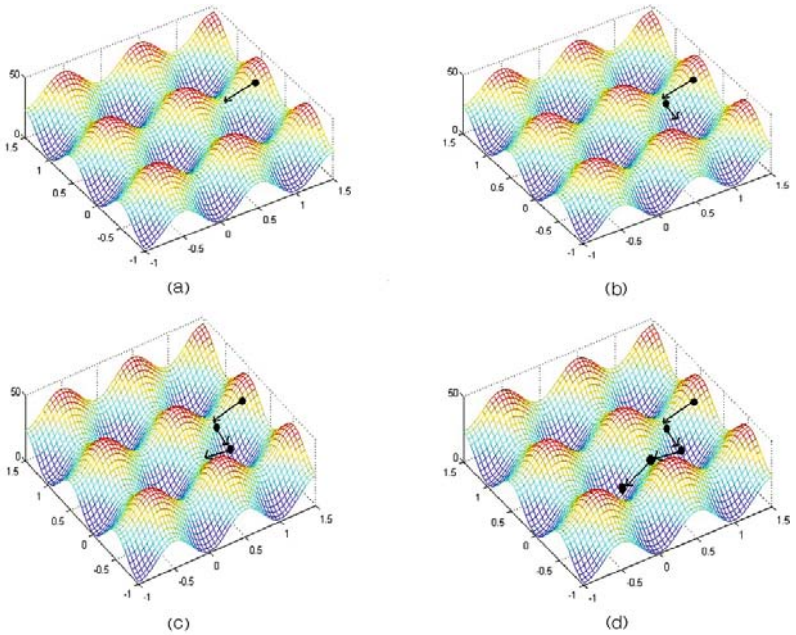

Fig. 3. Trajectory of the best particle for the 2-D Rastrigin function. (**a**) After 40 iterations (**b**) after 80 iterations (**c**) after 150 iterations (**d**) after 200 iterations

Example 2. Consider the two-dimensional function given by

$$f(\vec{x}) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2).$$

This function is known as the Rastrigin function [11] and has a global minimum value 0 at $x_1 = 0$ and $x_2 = 0$. A PSO is run to optimize the function. We used 30 particles and randomly initialized their positions and velocities in the interval $[-10, 10]$. We used the following parameter set-up: $C_1 = C_2 = 2.00$, $\omega = 0.729$, and the maximum particle velocity $V_{\max} = 10$. Figure 3 depicts the trajectory of the globally best particle towards the global minima of the function over different iterations.

3 Selection of Parameters for PSO

The main parameters of the canonical PSO model are ω, C_1, C_2, $V_{\max}$ and the swarm size S. The settings of these parameters determine how it optimizes the search-space. For instance, one can apply a general setting that gives reasonable results on most problems, but seldom is very optimal. Since the same parameter settings not at all guarantee success in different problems, we must have knowledge of the effects of the different settings, such that we can pick a suitable setting from problem to problem.

3.1 The Inertia Weight ω

The inertia weight ω controls the momentum of the particle: If $\omega << 1$, only little momentum is preserved from the previous time-step; thus quick changes of direction are possible with this setting. The concept of velocity is completely lost if $\omega = 0$, and the particle then moves in each step without knowledge of the past velocity. On the other hand, if ω is high (>1) we observe the same effect as when C_1 and C_2 are low: Particles can hardly change their direction and turn around, which of course implies a larger area of exploration as well as a reluctance against convergence towards optimum. Setting $\omega > 1$ must be done with care, since velocities are further biased for an exponential growth (see Fig. 2). This setting is rarely seen in PSO implementation, and always together with V_{max}. In short, high settings near 1 facilitate global search, and lower settings in the range [0.2, 0.5] facilitate rapid local search.

Eberhart and Shi have studied ω in several papers and found that "when V_{max} is not small ($\geq$3), an inertia-weight of 0.8 is a good choice" [12]. Although this statement is solely based on a single test function, the Schaffer f_6 function, this setting actually is a good choice in many cases. The authors have also applied an annealing scheme for the ω-setting of the PSO, where ω decreases linearly from $\omega = 0.9$ to $\omega = 0.4$ over the whole run [13]. They compared their annealing scheme results to results with $\omega = 1$ obtained by Angeline [14], and concluded a significant performance improvement on the four tested functions. The decreasing ω-strategy is a near-optimal setting for many problems, since it allows the swarm to explore the search-space in the beginning of the run, and still manages to shift towards a local search when fine-tuning is needed. This was called PSO-TVIW method (PSO with Time varying inertia weight) [15].

Finally, Eberhart and Shi devised an adaptive fuzzy PSO, where a fuzzy controller was used to control ω over time [16]. This approach is very interesting, since it potentially lets the PSO self-adapt ω to the problem and thus optimizes and eliminates a parameter of the algorithm. This saves time during the experimentation, since fine-tuning of ω is not necessary anymore. At each time-step, the controller takes the "Normalized Current Best Performance Evaluation" (NCBPE) and the current setting of ω as inputs, and it outputs a probabilistic change in ω.

3.2 The Maximum Velocity $V_{\max}$

The maximum velocity $V_{\max}$ determines the maximum change one particle can undergo in its positional coordinates during an iteration. Usually we set the full search range of the particle's position as the $V_{\max}$. For example, in case, a particle has position vector $\overrightarrow{x} = (x_1, x_2, x_3)$ and if $-10 \leq x_i \leq 10$ for $i = 1$, 2 and 3, then we set $V_{\max} = 20$. Originally, $V_{\max}$ was introduced to avoid explosion and divergence. However, with the use of constriction factor χ (to be discussed shortly) or ω in the velocity update formula, $V_{\max}$ to some

degree has become unnecessary; at least convergence can be assured without it [17]. Thus, some researchers simply do not use $V_{\max}$. In spite of this fact, the maximum velocity limitation can still improve the search for optima in many cases.

3.3 The Constriction Factor χ

In 2002, Clerc and Kennedy proposed an adaptive PSO model [17] that uses a new parameter 'χ' called the constriction factor. The model also excluded the inertia weight ω and the maximum velocity parameter $V_{\max}$. The velocity update scheme proposed by Clerc can be expressed for the dth dimension of ith particle as:

$$\left.\begin{aligned} V_{\mathrm{id}}(t+1) &= \chi[V_{\mathrm{id}}(t) + C_1 \cdot \varphi_1 \cdot (P_{\mathrm{id}}(t) - X_{\mathrm{id}}(t)) + C_2 \cdot \varphi_2 \cdot (g_{\mathrm{id}}(t) - X_{\mathrm{id}}(t))] \\ X_{\mathrm{id}}(t+1) &= X_{\mathrm{id}}(t) + V_{\mathrm{id}}(t+1), \end{aligned}\right\} \tag{2}$$

where,

$$\chi = \frac{2}{\left|4 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \text{ With } \quad \varphi = C_1 + C_2$$

Constriction coefficient results in the quick convergence of the particles over time. That is the amplitude of a particle's oscillations decreases as it focuses on the local and neighborhood previous best points. Though the particle converges to a point over time, the constriction coefficient also prevents collapse if the right social conditions are in place. The particle will oscillate around the weighted mean of p_{id} and p_{gd}, if the previous best position and the neighborhood best position are near each other the particle will perform a local search. If the previous best position and the neighborhood best position are far apart from each other, the particle will perform a more exploratory search (global search). During the search, the neighborhood best position and previous best position will change and the particle will shift from local search back to global search. The constriction coefficient method therefore balances the need for local and global search depending on what social conditions are in place.

3.4 The Swarm Size

It is quite a common practice in the PSO literature to limit the number of particles to limit the number of particles to the range 20–60 [12, 18]. Van den Bergh and Engelbrecht [19] have shown that though there is a slight improvement of the optimal value with increasing swarm size, a larger swarm increases the number of function evaluations to converge to an error limit. Eberhart and Shi [18] illustrated that the population size has hardly any effect on the performance of the PSO method.

3.5 The Acceleration Coefficients C_1 and C_2

A usual choice for the acceleration coefficients C_1 and C_2 is $C_1 = C_2 = 1.494$ [18]. However, other settings were also used in different papers. Usually C_1 equals to C_2 and ranges from [0, 4]. Ratnaweera et al. have recently investigated the effect of varying these coefficients with time in [20]. Authors adapted C_1 and C_2 with time in the following way:

$$\begin{aligned} C_1 &= (C_{1f} - C_{1i})\frac{iter}{MAXITER} + C_{1i} \\ C_2 &= (C_{2f} - C_{2i})\frac{iter}{MAXITER} + C_{2i} \end{aligned}, \tag{3}$$

where $C_{1i},\ C_{1f},\ C_{2i}$, and C_{2f} are constants, iter is the current iteration number and MAXITER is the number of maximum allowable iterations. The objective of this modification was to boost the global search over the entire search space during the early part of the optimization and to encourage the particles to converge to global optima at the end of the search. The authors referred this as the PSO-TVAC (PSO with time varying acceleration coefficients) method. Actually C_1 was decreased from 2.5 to 0.5 whereas C_2 was increased from 0.5 to 2.5.

4 The Neighborhood Topologies in PSO

The commonly used PSOs are either global version or local version of PSO [5]. In the global version of PSO, each particle flies through the search space with a velocity that is dynamically adjusted according to the particle's personal best performance achieved so far and the best performance achieved so far by all the particle. While in the local version of PSO, each particle's velocity is adjusted according to its personal best and the best performance achieved as far within its neighborhood. The neighborhood of each particle is generally defined as topologically nearest particles to the particle at each side. The global version of PSO also can be considered as a local version of PSO with each particle's neighborhood to be the whole population. It has been suggested that the global version of PSO converges fast, but with potential to converge to the local minimum, while the local version of PSO might have more chances to find better solutions slowly [5]. Since then, a lot of researchers have worked on improving its performance by designing or implementing different types of neighborhood structures in PSO. Kennedy [21] claimed that PSO with small neighborhoods might perform better on complex problems while PSO with large neighborhood would perform better for simple problems.

The k-best topology, proposed by Kennedy connects every particle to its k nearest particles in the topological space. With $k = 2$, this becomes the circle topology (and with $k =$ swarmsize-1 it becomes a gbest topology). The wheel topology, in which the only connections are from one central particle to

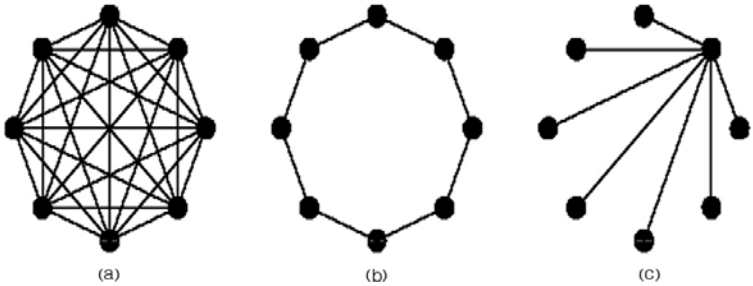

Fig. 4. (**a**) The fully connected org_best topology (**b**) The k-best nearest neighbor topology with $k = 2$ and (**c**) the wheel topology

the others (see Fig. 4). In addition, one could imagine a huge number of other topologies.

5 The Binary PSO

A binary optimization problem is a (normal) optimization problem, where the search space S is simply the set of strings of 0s and 1s of a given fixed length, n. A binary optimization algorithm solves binary optimization problems, and thus enables us to solve many discrete problems. Hence, faced with a problem-domain that we cannot fit into some sub-space of the real-valued n-dimensional space, which is required by the PSO, odds are that we can use a binary PSO instead. All we must provide is a mapping from this given problem-domain to the set of bit strings.

The binary PSO model was presented by Kennedy and Eberhart, and is based on a very simple modification of the real-valued PSO [22]. As with the original PSO, a fitness function f must be defined. In this case, it maps from the n-dimensional binary space B^n (i.e., bit strings of length n) to the real numbers: $f : \mathrm{B}^n \rightarrow \Re^n$. In the binary PSO the positions of the particles naturally must belong to B^n in order to be evaluated by f. Surprisingly, the velocities till belong to $V = [-V_{\max}, V_{\min}] \subset \Re$. This is obtained by changing the position update formula (1) and leaving the velocity update formula unchanged. Now, the ith coordinate of each particle's position is a bit, which state is given by

$$\begin{aligned} X_i(t+1) = 1 \;\; &\text{if} \;\; \rho < s(\vec{V}_i(t)), \\ = 0 \;\; &\text{otherwise}, \end{aligned} \tag{4}$$

where ρ is a random and uniformly selected number in [0, 1], which is resampled for each assignment of $\vec{X}_i(t)$. S is a sigmoid function that maps from the real numbers into [0, 1]. It has the properties that $S(0) = \frac{1}{2}$ and $S(x) \rightarrow 0$ as $x \rightarrow \infty$. It is mathematically formulated as

$$S(x) = \frac{1}{1 + e^{-x}}$$

The position $\vec{X}_i(t)$ has changed from being a point in real-valued space to being a bit-string, and the velocity $\vec{V}_i(t)$ has now become a probability for $\vec{X}_i(t)$ to be 1or 0. If $\vec{V}_i(t) = 0$ the probability for the outcome $\vec{X}_i(t+1) = 1$ will be 50%. On the other hand, if $\vec{V}_i(t) > 0$ the probability for $\vec{X}_i(t+1) = 1$ will be above 50%, and if $\vec{V}_i(t) < 0$ a probability below 50%. It is not quite clear, at least intuitively, how and why this changed approach will work. In particular, to the best of our knowledge, there has not been published any papers concerning with the changed meaning of the ω parameter in the binary PSO. The discussion on the meaning of ω above must therefore be regarded as a novel discovery and an independent research result in its own right. In conclusion, since a binary version of the PSO is a key point to practical and commercial use of the PSO in discrete problems solving, this topic definitely needs a lot more attention in the coming years.

6 Hybridization of PSO with Other Evolutionary Techniques

A popular research trend is to merge or combine the PSO with the other techniques, especially the other evolutionary computation techniques. Evolutionary operators like selection, crossover and mutation have been applied into the PSO. By applying selection operation in PSO, the particles with the best performance are copied into the next generation; therefore, PSO can always keep the best-performed particles [14]. By applying crossover operation, information can be swapped between two individuals to have the ability to "fly" to the new search area as that in evolutionary programming and GAs [23]. Among the three evolutionary operators, the mutation operators are the most commonly applied evolutionary operators in PSO. The purpose of applying mutation to PSO is to increase the diversity of the population and the ability to have the PSO to escape the local minima [24–27]. One approach is to mutate parameters such as χ, C_1 and C_2, the position of the neighborhood best [26], as well as the inertia weight [25]. Another approach is to prevent particles from moving too close to each other so that the diversity could be maintained and therefore escape from being trapped into local minima. In [25], the particles are relocated when they are too close to each other. In [24, 27], collision-avoiding mechanisms are designed to prevent particle from colliding with each other and therefore increase the diversity of the population. In addition to incorporating evolutionary operations into PSO, different approaches to combine PSO with the other evolutionary algorithms have been reported. Robinson et al. [28] obtained better results by applying PSO first followed by applying GA in their profiled corrugated horn antenna optimization problem. In [29], either PSO algorithm or GA or hill climbing search algorithm can be applied to a different sub-population of individuals which each individual is dynamically assigned to according to some pre-designed rules. In [30], ant

colony optimization is combined with PSO. A list of best positions found so far is recorded and the neighborhood best is randomly selected from the list instead of the current neighborhood best.

In addition, non-evolutionary techniques have been incorporated into PSO. In [31], a cooperative particle swarm optimizer (CPSO) is implemented. The CPSO employs cooperative behavior to significantly improve the performance of the original PSO algorithm through using multiple swarms to optimize different components of the solution vector cooperatively. The search space is partitioned by splitting the solutions vectors into smaller vector. For example, a swarm with n-dimensional vector is partitioned into n swarms of one-dimensional vectors with each swarm attempting to optimize a single component of the solution vector. A credit assignment mechanism needs to be designed to evaluate each particle in each swarm. In [23], the population of particles is divided into subpopulations which would breed within their own sub-population or with a member of another with some probability so that the diversity of the population can be increased. In [32], deflection and stretching techniques as well as a repulsion technique.

7 The Differential Evolution (DE)

In 1995, Price and Storn proposed a new floating point encoded evolutionary algorithm for global optimization and named it DE [9] owing to a special kind of differential operator, which they invoked to create new offspring from parent chromosomes instead of classical crossover or mutation. Easy methods of implementation and negligible parameter tuning made the algorithm quite popular very soon. In the following section, we will outline the classical DE and its different versions in sufficient details.

7.1 Classical DE – How Does it Work?

Like any other evolutionary algorithm, DE also starts with a population of NP D-dimensional search variable vectors. We will represent subsequent generations in DE by discrete time steps like $t = 0, 1, 2, \ldots, t,\ t + 1$, etc. Since the vectors are likely to be changed over different generations we may adopt the following notation for representing the ith vector of the population at the current generation (i.e., at time $t = t$) as

$$\vec{X}_i(t) = [x_{i,1}(t), x_{i,2}(t), x_{i,3}(t) \ldots\ldots x_{i,D}(t)].$$

These vectors are referred in literature as "genomes" or "chromosomes". DE is a very simple evolutionary algorithm.

For each search-variable, there may be a certain range within which value of the parameter should lie for better search results. At the very beginning

of a DE run or at $t = 0$, problem parameters or independent variables are initialized somewhere in their feasible numerical range. Therefore, if the jth parameter of the given problem has its lower and upper bound as x_j^L and x_j^U, respectively, then we may initialize the jth component of the ith population members as

$$x_{i,j}(0) = x_j^L + rand\ (0,1) \cdot (x_j^U - x_j^L),$$

where rand (0,1) is a uniformly distributed random number lying between 0 and 1.

Now in each generation (or one iteration of the algorithm) to change each population member $\vec{X}_i(t)$ (say), a Donor vector $\vec{V}_i(t)$ is created. It is the method of creating this donor vector, which demarcates between the various DE schemes. However, here we discuss one such specific mutation strategy known as DE/rand/1. In this scheme, to create $\vec{V}_i(t)$ for each ith member, three other parameter vectors (say the r_1, r_2, and r_3th vectors) are chosen in a random fashion from the current population. Next, a scalar number F scales the difference of any two of the three vectors and the scaled difference is added to the third one whence we obtain the donor vector $\vec{V}_i(t)$. We can express the process for the jth component of each vector as

$$v_{i,j}(t+1) = x_{r1,j}(t) + F \cdot (x_{r2,j}(t) - x_{r3,j}(t)) \ldots\ldots \tag{5}$$

The process is illustrated in Fig. 5. Closed curves in Fig. 5 denote constant cost contours, i.e., for a given cost function f, a contour corresponds to $f(\vec{X}) =$ constant. Here the constant cost contours are drawn for the Ackley Function.

Next, to increase the potential diversity of the population a crossover scheme comes to play. DE can use two kinds of cross over schemes namely "Exponential" and "Binomial". The donor vector exchanges its "body parts", i.e., components with the target vector $\vec{X}_i(t)$ under this scheme. In "Exponential" crossover, we first choose an integer n randomly among the numbers [0, D–1]. This integer acts as starting point in the target vector, from where the crossover or exchange of components with the donor vector starts. We also choose another integer L from the interval [1, D]. L denotes the number of components; the donor vector actually contributes to the target. After a choice of n and L the trial vector:

$$\vec{U}_i(t) = [u_{i,1}(t), u_{i,2}(t) \ldots u_{i,D}(t)] \tag{6}$$

is formed with

$$\begin{aligned} u_{i,j}(t) &= v_{i,j}(t) \quad \text{for} \quad j =< n >_D, < n+1 >_D, \ldots < n-L+1 >_D \\ &= x_{i,j}(t), \end{aligned} \tag{7}$$

where the angular brackets $<>_D$ denote a modulo function with modulus D. The integer L is drawn from [1, D] according to the following pseudo code.

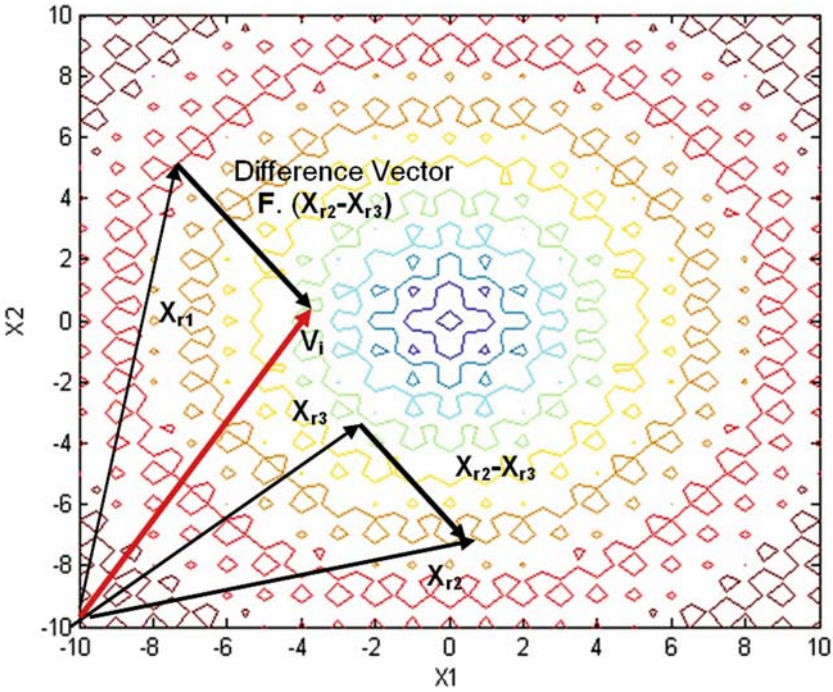

Fig. 5. Illustrating creation of the donor vector in 2-D parameter space (The constant cost contours are for two-dimensional Ackley Function)

```
L = 0;
do
{
   L=L+1;
} while (rand (0, 1) < CR) AND (L<D));
```

Hence in effect probability $(L > m) = (\mathrm{CR})^{\mathrm{m}-1}$ for any m > 0. CR is called "Crossover" constant and it appears as a control parameter of DE just like F. For each donor vector V, a new set of n and L must be chosen randomly as shown above. However, in "Binomial" crossover scheme, the crossover is performed on each of the D variables whenever a randomly picked number between 0 and 1 is within the CR value. The scheme may be outlined as

$$\begin{aligned} u_{i,j}(t) &= v_{i,j}(t) \quad \text{if rand}\,(0,1) < \mathrm{CR}, \\ &= x_{i,j}(t) \quad \text{else}\ldots\ldots \end{aligned} \tag{8}$$

In this way for each trial vector $\vec{X}_i(t)$ an offspring vector $\vec{U}_i(t)$ is created. To keep the population size constant over subsequent generations, the next step of the algorithm calls for "selection" to determine which one of the target vector and the trial vector will survive in the next generation, i.e., at time

$t = t + 1$. DE actually involves the Darwinian principle of "Survival of the fittest" in its selection process which may be outlined as

$$\begin{aligned} \vec{X}_i(t+1) &= \vec{U}_i(t) \quad \text{if} \quad f(\vec{U}_i(t)) \leq f(\vec{X}_i(t)), \\ &= \vec{X}_i(t) \quad \text{if} \quad f(\vec{X}_i(t)) < f(\vec{U}_i(t)), \ldots \end{aligned} \tag{9}$$

where $f()$ is the function to be minimized. So if the new trial vector yields a better value of the fitness function, it replaces its target in the next generation; otherwise the target vector is retained in the population. Hence the population either gets better (w.r.t. the fitness function) or remains constant but never deteriorates. The DE/rand/1 algorithm is outlined below:

Procedure DE

Input: Randomly initialized position and velocity of the particles: $\vec{x}_i(0)$

Output: Position of the approximate global optima $\vec{X}^*$

Begin

Initialize population;
Evaluate fitness;
For i = 0 to max-iteration **do**
Begin
Create Difference-Offspring;
Evaluate fitness;
If an offspring is better than its parent
Then replace the parent by offspring in the next generation;
End If;
End For;
End.

Example 3. This example illustrates the complete searching on the fitness landscape of a two-dimensional sphere function by a simple DE. Sphere is perhaps one of the simplest two-dimensional functions and has been chosen to provide easy visual depiction of the search process. The function is given by

$$f(\vec{x}) = x_1^2 + x_2^2.$$

As can be easily perceived, the function has only one global minima $f^* = 0$ at $X^* = [0,\ 0]^{\mathrm{T}}$. We start with a randomly initialized population of five vectors in the search range $[-10, 10]$. Initially, these vectors are given by

$$\begin{aligned} X_1(0) &= [5, -9]^{\mathrm{T}} \\ X_2(0) &= [6, 1]^{\mathrm{T}} \\ X_3(0) &= [-3, 5]^{\mathrm{T}} \\ X_4(0) &= [-7, 4]^{\mathrm{T}} \\ X_5(0) &= [6, 7]^{\mathrm{T}} \end{aligned}$$

Figures 6–9 illustrate the initial orientation of the search variable vectors in the two-dimensional $X_1 - X_2$ space. The concentric circular lines are the *constant cost contours* of the function, i.e., locus in the $X_1 - X_2$ plane. Now following the mutation and recombination schemes as presented in expressions (7) and

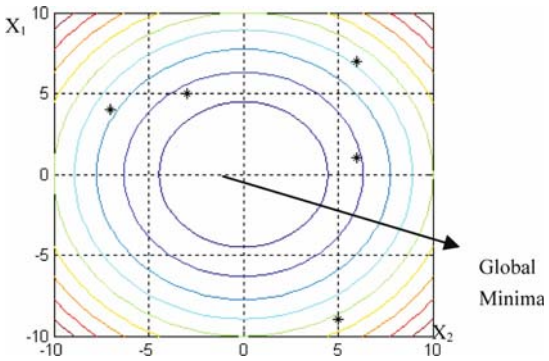

Fig. 6. Orientation of the initial solutions in the two-dimensional search space

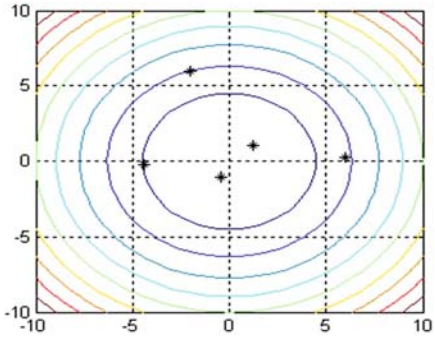

Fig. 7. Orientation of the population members after five iterations

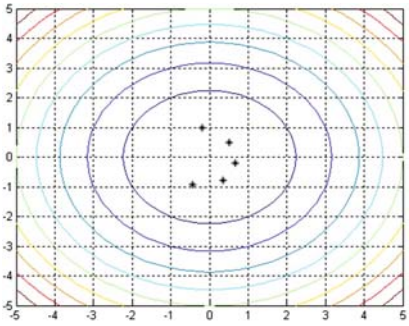

Fig. 8. Orientation of the population members after 10 iterations

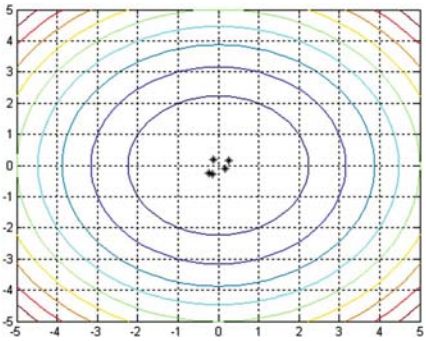

Fig. 9. Orientation of the population members after 20 iterations

(8), we form five donor vectors and then create five offspring vectors for time $t = 1$. Next we apply the selection method described by (9) and evolve the entire population at time $t = 1$. These steps have been summarized in Table 1.

7.2 The Complete DE Family of Storn and Price

Actually, it is the process of mutation, which demarcates one DE scheme from another. In the former section, we have illustrated the basic steps of a simple DE. The mutation scheme in (5) uses a randomly selected vector $\vec{X}_{r1}$ and only one weighted difference vector $F \cdot (\vec{X}_{r2} - \vec{X}_{r3})$ is used to perturb it. Hence, in literature the particular mutation scheme is referred to as DE/rand/1. We can now have an idea of how different DE schemes are named. The general convention used, is DE/x/y. DE stands for DE, x represents a string denoting the type of the vector to be perturbed (whether it is randomly selected or it is the best vector in the population with respect to fitness value) and y is the number of difference vectors considered for perturbation of x. Below we outline the other four different mutation schemes, suggested by Price et al. [33].

Scheme DE/rand to best/1

DE/rand to best/1 follows the same procedure as that of the simple DE scheme illustrated earlier. The only difference being that, now the donor vector, used to perturb each population member, is created using any two randomly selected member of the population as well as the best vector of the current generation (i.e., the vector yielding best suited objective function value at $t = t$). This can be expressed for the ith donor vector at time $t = t + 1$ as

$$\vec{V}_i(t+1) = \vec{X}_i(t) + \lambda \cdot (\vec{X}_{best}(t) - \vec{X}_i(t)) + F \cdot (\vec{X}_{r_2}(t) - \vec{X}_{r3}(t)) \qquad (10)$$

where λ is another control parameter of DE in [0, 2], $X_i(t)$ is the target vector and $\vec{X}_{best}(t)$ is the best member of the population regarding fitness at current

Table 1. Evolution of the population from $t = 0$ to $t = 1$ in Example 3

Population at $t = 0$	Fitness at $t = 0$	Donor vector at $t = 1$	Offspring vector at $t = 1$	Fitness of offspring at $t = 1$	Evolved population at $t = 1$
$X_1(0) = [2, -1]$	5	$V_1(1) = [-0.4, 10.4]$	$T_1(1) = [-0.4, -1]$	1.16	$X_1(1) = [-0.4, -1]$
$X_2(0) = [6, 1]$	37	$V_2(1)X_1(0) = [1.2, -0.2]$	$T_2(1) = [1.2, 1]$	2.44	$X_2(1) = [1.2, 1]$
$X_3(0) = [-3, 5]$	34	$V_3(1) = [-4.4, -0.2]$	$T_3(1) = [-4.4, -0.2]$	19.4	$X_3(1) = [-4.4, -0.2]$
$X_4(0) = [-2, 6]$	40	$V_4(1) = [9.2, -4.2]$	$T_4(1) = [9.2, 6]$	120.64	$X_4(1) = [-2, 6]$
$X_5(0) = [6, 7]$	85	$V_5(1) = [5.2, 0.2]$	$T_5(1) = [6, 0.2]$	36.04	$X_5(1) = [6, 0.2]$

time step $t = t$. To reduce the number of control parameters a usual choice is to put $\lambda = F$.

Scheme DE/best/1

In this scheme everything is identical to DE/rand/1 except the fact that the trial vector is formed as

$$\vec{V}_i(t+1) = \vec{X}_{best}(t) + F \cdot (\vec{X}_{r1}(t) - \vec{X}_{r2}(t)), \tag{11}$$

here the vector to be perturbed is the best vector of the current population and the perturbation is caused by using a single difference vector.

Scheme DE/best/2

Under this method, the donor vector is formed by using two difference vectors as shown below:

$$\vec{V}_i(t+1) = \vec{X}_{best}(t) + F \cdot (\vec{X}_{r1}(t) + \vec{X}_{r2}(t) - \vec{X}_{r3}(t) - \vec{X}_{r4}(t)). \tag{12}$$

Owing to the central limit theorem the random variations in the parameter vector seems to shift slightly into the Gaussian direction which seems to be beneficial for many functions.

Scheme DE/rand/2

Here the vector to be perturbed is selected randomly and two weighted difference vectors are added to the same to produce the donor vector. Thus for each target vector, a totality of five other distinct vectors are selected from the rest of the population. The process can be expressed in the form of an equation as

$$\vec{V}_i(t+1) = \vec{X}_{r1}(t) + F_1 \cdot (\vec{X}_{r2}(t) - \vec{X}_{r3}(t)) + F_2 \cdot (\vec{X}_{r4}(t) - \vec{X}_{r5}(t)) \tag{13}$$

Here F_1 and F_2 are two weighing factors selected in the range from 0 to 1. To reduce the number of parameters we may choose $F_1 = F_2 = F$.

Summary of all Schemes

In 2001 Storn and Price [2] suggested total ten different working strategies of DE and some guidelines in applying these strategies to any given problem. These strategies were derived from the five different DE mutation schemes outlined above. Each mutation strategy was combined with either the

"exponential" type crossover or the "binomial" type crossover. This yielded $5 \times 2 = 10$ DE strategies, which are listed below.

1. DE/best/1/exp
2. DE/rand/1/exp
3. DE/rand-to-best/1/exp
4. DE/best/2/exp
5. DE/rand/2/exp
6. DE/best/1/bin
7. DE/rand/1/bin
8. DE/rand-to-best/1/bin
9. DE/best/2/bin
10. DE/rand/2/bin

The general convention used above is again DE/x/y/z, where DE stands for DE, x represents a string denoting the vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z stands for the type of crossover being used (exp: exponential; bin: binomial).

7.3 More Recent Variants of DE

DE is a stochastic, population-based, evolutionary search algorithm. The strength of the algorithm lies in its simplicity, speed (how fast an algorithm can find the optimal or suboptimal points of the search space) and robustness (producing nearly same results over repeated runs). The rate of convergence of DE as well as its accuracy can be improved largely by applying different mutation and selection strategies. A judicious control of the two key parameters namely the scale factor F and the crossover rate CR can considerably alter the performance of DE. In what follows we will illustrate some recent modifications in DE to make it suitable for tackling the most difficult optimization problems.

DE with Trigonometric Mutation

Recently, Lampinen and Fan [34] has proposed a trigonometric mutation operator for DE to speed up its performance. To implement the scheme, for each target vector, three distinct vectors are randomly selected from the DE population. Suppose for the ith target vector $\vec{X}_i(t)$, the selected population members are $\vec{X}_{r1}(t)$, $\vec{X}_{r2}(t)$ and $\vec{X}_{r3}(t)$. The indices r_1, r_2 and r_3 are mutually different and selected from $[1, 2, \ldots, N]$ where N denotes the population size. Suppose the objective function values of these three vectors are given by, $f(\vec{X}_{r1}(t))$, $f(\vec{X}_{r2}(t))$ and $f(\vec{X}_{r3}(t))$. Now three weighing coefficients are formed according to the following equations:

$$p' = \left|f(\vec{X}_{r1})\right| + \left|f(\vec{X}_{r2})\right| + \left|f(\vec{X}_{r3})\right|, \tag{14}$$

$$p_1 = \left|f(\vec{X}_{r1})\right| \Big/ p', \tag{15}$$

$$p_2 = \left|f(\vec{X}_{r2})\right| \Big/ p', \tag{16}$$

$$p_3 = \left|f(\vec{X}_{r3})\right| \Big/ p'. \tag{17}$$

Let rand (0, 1) be a uniformly distributed random number in (0, 1) and Γ be the trigonometric mutation rate in the same interval (0, 1). The trigonometric mutation scheme may now be expressed as

$$\begin{aligned} \vec{V}_i(t+1) = {} & (\vec{X}_{r1} + \vec{X}_{r2} + \vec{X}_{r3})/3 + (p_2 - p_1) \cdot (\vec{X}_{r1} - \vec{X}_{r2}) \\ & + (p_3 - p_2) \cdot (\vec{X}_{r2} - \vec{X}_{r3}) + (p_1 - p_3) \cdot (\vec{X}_{r3} - \vec{X}_{r1}) \\ & \qquad\qquad\qquad\qquad\qquad\qquad \text{if rand}\,(0,1) < \Gamma \\ \vec{V}_i(t+1) = {} & \vec{X}_{r1} + F \cdot (\vec{X}_{r2} + \vec{X}_{r3}) \qquad \text{else.} \end{aligned} \tag{18}$$

Thus, we find that the scheme proposed by Lampinen et al. uses trigonometric mutation with a probability of Γ and the mutation scheme of DE/rand/1 with a probability of $(1 - \Gamma)$.

DERANDSF (DE with Random Scale Factor)

In the original DE [9] the difference vector $(\vec{X}_{r1}(t) - \vec{X}_{r2}(t))$ is scaled by a constant factor "F". The usual choice for this control parameter is a number between 0.4 and 1. We propose to vary this scale factor in a random manner in the range (0.5, 1) by using the relation

$$F = 0.5^*(1 + \text{rand}\ (0, 1)), \tag{19}$$

where rand (0, 1) is a uniformly distributed random number within the range [0, 1]. We call this scheme DERANDSF (DE with Random Scale Factor) [35]. The mean value of the scale factor is 0.75. This allows for stochastic variations in the amplification of the difference vector and thus helps retain population diversity as the search progresses. Even when the tips of most of the population vectors point to locations clustered near a local optimum due to the randomly scaled difference vector, a new trial vector has fair chances of pointing at an even better location on the multimodal functional surface. Therefore, the fitness of the best vector in a population is much less likely to get stagnant until a truly global optimum is reached.

DETVSF (DE with Time Varying Scale Factor)

In most population-based optimization methods (except perhaps some hybrid global-local methods) it is generally believed to be a good idea to encourage

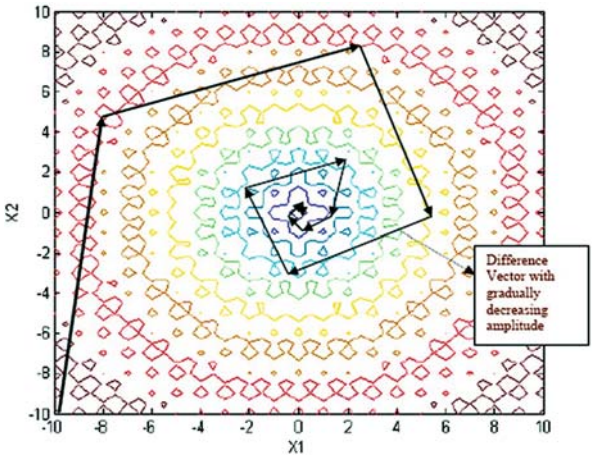

Fig. 10. Illustrating DETVSF scheme on two-dimensional cost contours of Ackley function

the individuals (here, the tips of the trial vectors) to sample diverse zones of the search space during the early stages of the search. During the later stages it is important to adjust the movements of trial solutions finely so that they can explore the interior of a relatively small space in which the suspected global optimum lies. To meet this objective we reduce the value of the scale factor linearly with time from a (predetermined) maximum to a (predetermined) minimum value:

$$R = (R_{\max} - R_{\min})^*(MAXIT - iter)/MAXIT \tag{20}$$

where $F_{\max}$ and $F_{\min}$ are the maximum and minimum values of scale factor F, *iter* is the current iteration number and *MAXIT* is the maximum number of allowable iterations. The locus of the tip of the best vector in the population under this scheme may be illustrated as in Fig. 10. The resulting algorithm is referred as DETVSF (DE with a time varying scale factor) [35].

DE with Local Neighborhood

Only in 2006, a new DE-variant, based on the neighborhood topology of the parameter vectors was developed [36] to overcome some of the disadvantages of the classical DE versions. The authors in [36] proposed a neighborhood-based local mutation operator that draws inspiration from PSO. Suppose we have a DE population $P = [\vec{X}_1, \vec{X}_2, \ldots, \vec{X}_{N_p}]$ where each $\vec{X}_i$ $(i = 1,\ 2,\ \ldots,\ N_p)$ is a D-dimensional vector. Now for every vector $\vec{X}_i$ we define a neighborhood of radius k, consisting of vectors $\vec{X}_{i-k}, \ldots . \vec{X}_i, \ldots . \vec{X}_{i+k}$. We assume the vectors to be organized in a circular fashion such that two immediate neighbors of vector $\vec{X}_1$ are $\vec{X}_{N_p}$ and $\vec{X}_2$. For each member of the population a local mutation is created by employing the fittest vector in the neighborhood of

that member and two other vectors chosen from the same neighborhood. The model may be expressed as:

$$\vec{L}_i(t) = \vec{X}_i(t) + \lambda' \cdot (\vec{X}_{nbest}(t) - \vec{X}_i(t)) + F' \cdot (\vec{X}_p(t) - \vec{X}_q(t)) \tag{21}$$

where the subscript *nbest* indicates the best vector in the neighborhood of $\overrightarrow{X}_i$ and $p, q \in (i-k, i+k)$. Apart from this, we also use a global mutation expressed as:

$$\vec{G}_i(t) = \vec{X}_i(t) + \lambda \cdot (\vec{X}_{best}(t) - \vec{X}_i(t)) + F \cdot (\vec{X}_r(t) - \vec{X}_s(t)), \tag{22}$$

where the subscript *best* indicates the best vector in the entire population, and $r, s \in (1, N_p)$. Global mutation encourages exploitation, since all members (vectors) of a population are biased by the same individual (the population best); local mutation, in contrast, favors exploration, since in general different members of the population are likely to be biased by different individuals. Now we combine these two models using a time-varying scalar weight $w \in (0, 1)$ to form the actual mutation of the new DE as a weighted mean of the local and the global components:

$$\vec{V}_i(t) = w \cdot \vec{G}_i(t) + (1 - w) \cdot \vec{L}_i(t). \tag{23}$$

The weight factor varies linearly with time as folows:

$$w = w_{\min} + (w_{\max} - w_{\min}) \cdot \left(\frac{iter}{MAXIT} \right), \tag{24}$$

where *iter* is the current iteration number, *MAXIT* is the maximum number of iterations allowed and $w_{\max}$, $w_{\min}$ denote, respectively, the maximum and minimum value of the weight, with $w_{\max}, w_{\min} \in (0, 1)$. Thus the algorithm starts at $iter = 0$ with $w = w_{\min}$ but as *iter* increases towards *MAXIT*, w increases gradually and ultimately when $iter = MAXIT$ w reaches $w_{\max}$. Therefore at the beginning, emphasis is laid on the local mutation scheme, but with time, contribution from the global model increases. In the local model attraction towards a single point of the search space is reduced, helping DE avoid local optima. This feature is essential at the beginning of the search process when the candidate vectors are expected to explore the search space vigorously. Clearly, a judicious choice of $w_{\max}$ and $w_{\min}$ is necessary to strike a balance between the exploration and exploitation abilities of the algorithm. After some experimenting, it was found that $w_{\max} = 0.8$ and $w_{\min} = 0.4$ seem to improve the performance of the algorithm over a number of benchmark functions.

8 A Synergism of PSO and DE – Towards a New Hybrid Evolutionary Algorithm

Das et al. proposed a new scheme of adjusting the velocities of the particles in PSO with a vector differential operator borrowed from the DE family [37]. The canonical PSO updates the velocity of a particle using three terms. These

include a previous velocity term that provides the particle with the necessary momentum, a *social* term that indicates how the particle is stochastically drawn towards the globally best position found so far by the entire swarm, and finally a *cognitive* term that reflects the personal thinking of the particle, i.e., how much it is drawn towards the best position so far encountered in its own course. In the proposed scheme the cognitive term is omitted; instead the particle velocities are perturbed by a new term containing the weighted difference of the position vectors of any two distinct particles randomly chosen from the swarm. This differential velocity term is inspired by the DE mutation scheme and hence the authors name this algorithm as PSO-DV (particle swarm with differentially perturbed velocity). A *survival of the fittest* mechanism has also been incorporated in the swarm following the greedy selection scheme of DE.

8.1 The PSO-DV Algorithm

PSO-DV introduces a differential operator (borrowed from DE) in the velocity-update scheme of PSO. The operator is invoked on the position vectors of two randomly chosen particles (population-members), not on their individual best positions. Further, unlike the PSO scheme, a particle is actually shifted to a new location only if the new location yields a better fitness value, i.e., a selection strategy has been incorporated into the swarm dynamics. In the proposed algorithm, for each particle i in the swarm two other distinct particles, say j and k ($i \neq j \neq k$), are selected randomly. The difference between their positional coordinates is taken as a difference vector:

$$\vec{\delta} = \vec{X}_k - \vec{X}_j.$$

Then the dth velocity component ($1 < d < n$) of the target particle i is updated as

$$\left.\begin{array}{ll} V_{\mathrm{id}}(t+1) = \omega \cdot V_{\mathrm{id}}(t) + \beta \cdot \delta_{\mathrm{d}} + C_2 \cdot \varphi_2 \cdot (P_{\mathrm{gd}} - jX_{\mathrm{id}}(t)), & \text{if } \mathrm{rand}_{\mathrm{d}}(0,1) \leq \mathrm{CR}. \\ = V_{\mathrm{id}}(t), & \text{otherwise}, \end{array}\right\} \tag{25}$$

where CR is the crossover probability, δ_{d} is the dth component of the difference vector defined earlier, and β is a scale factor in [0, 1]. In essence the cognitive part of the velocity update formula in (1) is replaced with the vector differential operator to produce some additional exploration capability. Clearly, for CR ≤ 1, some of the velocity components will retain their old values. Now, a new trial location Tr_i is created for the particle by adding the updated velocity to the previous position X_i:

$$\vec{T}r_i = \vec{X}_i(t) + \vec{V}_i(t+1). \tag{26}$$

The particle is placed at this new location only if the coordinates of the location yield a better fitness value. Thus if we are seeking the minimum

of an n-dimensional function $f(\vec{X})$, then the target particle is relocated as follows:

$$\begin{aligned} \vec{X}_i(t+1) &= \vec{T}r_i \qquad \text{if } f(\vec{T}r_i) \le f(\vec{X}_i(t)), \\ \vec{X}_i(t+1) &= \vec{X}_i(t) \qquad \text{Otherwise,} \end{aligned} \tag{27}$$

Therefore, every time its velocity changes, the particle either moves to a better position in the search space or sticks to its previous location. The current location of the particle is thus the best location it has ever found. In other words, unlike the classical PSO, in the present scheme, P_{lid} always equals X_{id}. So the cognitive part involving $|\mathrm{P}_{lid} - X_{id}|$ is automatically eliminated in our algorithm. If a particle gets stagnant at any point in the search space (i.e., if its location does not change for a predetermined number of iterations), then the particle is shifted by a random mutation (explained below) to a new location. This technique helps escape local minima and also keeps the swarm "moving":

If $(\vec{X}_i(t) = \vec{X}_i(t+1) = \vec{X}_i(t+2) = \ldots. = \vec{X}_i(t+N))$ and $f(\vec{X}_i(t+N))$ **then**

for (r = 1 to n)

$$X_{\mathrm{ir}}(t+N+1) = X_{\min} + \mathrm{rand_r}\,(0,1)^*(X_{\max} - X_{\min}), \tag{28}$$

where f* is the global minimum of the fitness function, N is the maximum number of iterations up to which stagnation can be tolerated and $(X_{\max}, X_{\min})$ define the permissible bounds of the search space. The scheme is conceptually outlined in Fig. 11. The pseudo-code for this algorithm is illustrated as follows:

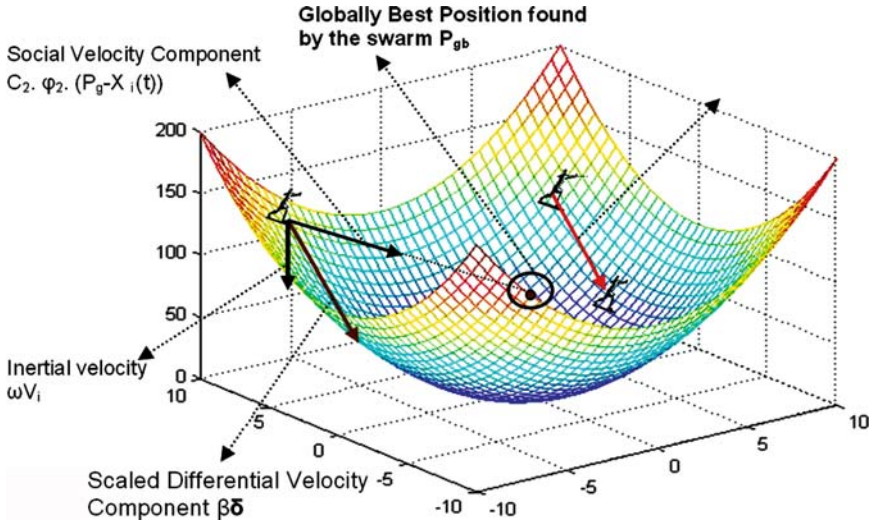

Fig. 11. Illustrating PSO-DV on a two-dimensional function surface

Procedure PSO-DV

begin
initialize population;
  while stopping condition not satisfied do
    for $i = 1$ to no_of_particles
    evaluate fitness of particle;
    update P_{gd};
    select two other particles j and k ($i \neq j \neq k$) randomly;
    construct the difference vector as $\vec{\delta} = \vec{X}_k - \vec{X}_j$;
    for $d = 1$ to no_of_dimensions
      if $\mathrm{rand_d}\ (0, 1) \leq \mathrm{CR}$
        $V_{\mathrm{id}}(t+1) = \omega \cdot V_{\mathrm{id}}(t) + \beta \cdot \delta_{\mathrm{d}} + C_2 \cdot \varphi_2 \cdot (P_{\mathrm{gd}} - X_{\mathrm{id}}(t))$;
      else $V_{\mathrm{id}}(t+1) = V_{\mathrm{id}}(t)$;
      endif
    endfor
    create trial location as $\vec{T}r_i = \vec{X}_i(t) + \vec{V}_i(t+1)$;
      if $(f(\vec{T}r_i) \leq f(\vec{X}_i(t)))$ **then** $\vec{X}_i(t+1) = \vec{T}r$;
      else $\vec{X}_i(t+1) = \vec{X}_i(t)$;
      endif
    endfor
    for $i = 1$ to no_of_particles
      if X_{i} stagnates for N successive generations
            for $r = 1$ to no_of_dimensions
            $X_{\mathrm{ir}}(t+1) = X_{\min} + \mathrm{rand_r}\,(0,1)^*(X_{\max} - X_{\min})$
            end for
      end if
    end for
  end while
end

9 PSO-DV Versus Other State-of-the-Art Optimizers

In this section we provide performance comparison between PSO-DV and four variants of the PSO algorithm and the classical DE algorithm over a test-bed of five well known benchmark functions. In Table 2, n represents the number of dimensions (we used $n = 25$, 50, 75 and 100). The first two test functions are uni-modal, having only one minimum. The others are multi-modal, with a considerable number of local minima in the region of interest. All benchmark functions except f_6 have the global minimum at the origin or very near to the origin [28]. For Shekel's foxholes (f_6), the global minimum is at $(-31.95,\ -31.95)$ and $f_6\ (-31.95,\ -31.95) \approx 0.998$, and the function has only two dimensions. An asymmetrical initialization procedure has been used here following the work reported in [29].

Table 2. Benchmark functions used

Function	Mathematical representation
Sphere function	$f_1(x) = \sum_{i=1}^{n} x_i^2$
Rosenbrock	$f_2(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
Rastrigin	$f_3(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$
Griewank	$f_4(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Ackley	$f_5(x) = -20\exp\left(-0.2\sqrt{(\frac{1}{n}\sum_{i=1}^{n} x_i^2)}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e$
Shekel's Foxholes	$f_6(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}\right]^{-1}$

Simulations were carried out to obtain a comparative performance analysis of the new method with respect to: (a) canonical PSO, (b) PSO-TVIW [18,37], (c) MPSO-TVAC [20], (d) HPSO-TVAC [20], and (e) classical DE. Thus a total of six algorithms were considered – one new, the other five existing in the literature. For a given function of a given dimension, 50 independent runs of each of the six algorithms were executed, and the average best-of-run value and the standard deviation were obtained. Different maximum generations ($G_{\max}$) were used according to the complexity of the problem. For all benchmarks (excluding Schaffer's f_6 and f_7) the stopping criterion was set as reaching a fitness of 0.001. However, for Shekel's Foxholes function (f_7) it was fixed at 0.998. Table 3 compares the algorithms on the quality of the best solution. The mean and the standard deviation (within parentheses) of the best-of-run solution for 50 independent runs of each of the six algorithms are presented in Table 3. Missing values of standard deviation in this table indicate a zero standard deviation. The best solution in each case has been shown in bold. Table 4 shows results of unpaired t-tests between the best algorithm and the second best in each case (standard error of difference of the two means, 95% confidence interval of this difference, the t value, and the two-tailed P value). For all cases in Table 4, sample size = 50 and degrees of freedom = 98. It is interesting to see from Tables 3 and 4 that the proposed method performs equally or better than other algorithms in a statistically meaningful way.

In Fig. 12, we have graphically presented the rate of convergence of all the methods for two most difficult functions Rosenbrock (f_2) and Ackley (f_5) functions (in 30 dimensions). We do not provide convergence graphs for all the functions in order to save space.

Table 3. Average and standard deviation of the best-of-run solution obtained for 50 runs of each of the six different methods

F	Dim	N	$G_{\max}$	Average (standard deviation)					
				PSO	PSO-TVIW	MPSO-TVAC	HPSO-TVAC	DE	PSO-DV
F_1	10	50	1000	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**
	20	100	2000	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**
	30	150	4500	**0.001**	**0.001**	**0.001**	**0.0001**	**0.001**	**0.001**
F_2	10	75	3000	21.705 (40.162)	16.21 (14.917)	1.234 (4.3232)	1.921 (4.330)	2.263 (4.487)	**0.0063 (0.0561)**
	20	150	4000	52.21 (148.32)	42.73 (72.612)	22.732 (30.638)	20.749 (12.775)	18.934 (9.453)	**0.0187 (0.554)**
	30	250	5000	77.61 (81.172)	61.78 (48.933)	34.229 (36.424)	10.414 (44.845)	6.876 (1.688)	**0.0227 (0.182)**
F_3	10	50	3000	2.334 (2.297)	2.1184 (1.563)	1.78 (2.793)	0.039 (0.061)	0.006 (0.0091)	**0.0014 (0.0039)**
	20	100	4000	13.812 (3.491)	16.36 (4.418)	11.131 (0.91)	0.2351 (0.1261)	0.0053 (0.0032)	**0.0028 (0.0017)**
	30	150	5000	6.652 (21.811)	24.346 (6.317)	50.065 (21.139)	1.903 (0.894)	0.099 (0.112)	**0.0016 (0.277)**
F_4	10	50	2500	0.1613 (0.097)	0.092 (0.021)	**0.00561 (0.047)**	0.057 (0.045)	0.054 (0.0287)	0.024 (0.180)
	20	100	3500	0.2583 (0.1232)	0.1212 (0.5234)	0.0348 (0.127)	0.018 (0.0053)	0.019 (0.0113)	**0.0032 (0.0343)**
	30	150	5000	0.0678 (0.236)	0.1486 (0.124)	0.0169 (0.116)	0.023 (0.0045)	0.005 (0.0035)	**0.0016 (0.0022)**
F_5	10	50	2500	0.406 (1.422)	0.238 (1.812)	0.169 (0.772)	0.0926 (0.0142)	**0.00312 (0.0154)**	0.00417 (0.1032)
	20	100	3500	0.572 (3.094)	0.318 (1.118)	0.537 (0.2301)	0.117 (0.025)	0.029 (0.0067)	**0.0018 (0.028)**
	30	150	5000	1.898 (2.598)	0.632 (2.0651)	0.369 (2.735)	0.068 (0.014)	0.0078 (0.0085)	**0.0016 (0.0078)**
F_6	2	40	1000	1.235 (2.215)	1.239 (1.468)	1.321 (2.581)	1.328 (1.452)	1.032 (0.074)	**0.9991 (0.0002)**

Table 4. Results of unpaired t-tests on the data of Table 3

Fn, dim	Std. err.	t	95% Conf. intvl	Two-tailed P	Significance
f_2, 10	0.612	3.1264	(−3.129, −0.699)	0.0023	Very significant
f_2, 20	1.339	14.1249	(−21.573, −16.258)	<0.0001	**Extremely significant**
f_2, 30	0.054	1.3981	(−0.184, 0.032)	0.1653	Not significant
f_3, 10	0.004	6.3954	(−0.037, −0.019)	<0.0001	**Extremely significant**
f_3, 20	0.000	0.000	(−0.000421, 0.000421)	1.0000	Not significant
f_3, 30	0.042	2.3051	(−0.181, −0.014)	0.0233	Significant
f_4, 10	0.026	0.6990	(−0.071, 0.034)	0.4862	Not significant
f_4, 20	0.005	2.6781	(−0.023, −0.003)	0.0087	Very significant
f_4, 30	0.000	5.3112	(−0.0027, −0.0012)	<0.0001	**Extremely significant**
f_5, 10	0.015	0.0712	(−0.030, 0.028)	0.9434	Not significant
f_5, 20	0.004	9.3541	(−0.045, −0.03)	<0.0001	**Extremely significant**
f_5, 30	0.001	4.0532	(−0.009, −0.003)	<0.0001	**Extremely significant**
f_7, 2	2.615	0.0241	(−5.251, 5.125)	0.9809	Not significant

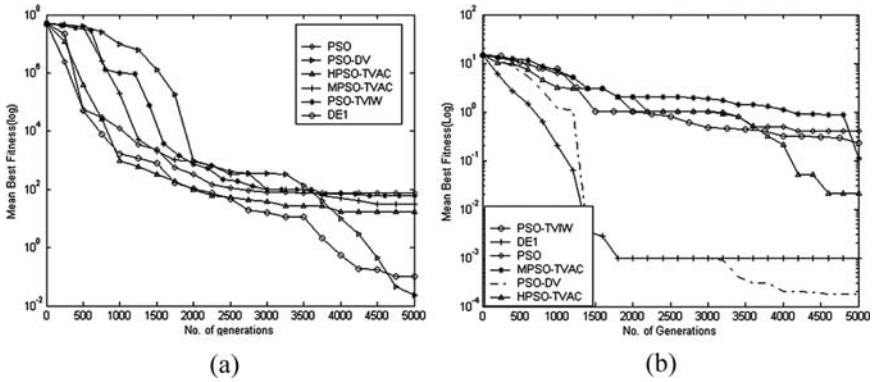

Fig. 12. Variation of the mean best value with time (all the graphs are for dimension = 30 except for Shekel's Foxholes (f_6) which is 2D) (**a**) Rosenbrock Function (f_2) (**b**) Ackley Function (f_5)

10 Applications

PSO has been applied to solve many interesting problems including a lot of practical application problems. It has been applied to evolve weights and structure of neural networks [38–40], analyze human tremor [41], register 3D-to-3D biomedical image [42], play games [43], control reactive power and voltage [44, 45], pattern recognition [46], etc. Generally speaking, PSO can be applied to solve most optimization problems and problems that can be converted to search or optimization problems.

Differential evolution (DE) has successfully been applied to many artificial and real optimization problems such as aerodynamic shape optimization

[47], automated mirror design [48], optimization of radial active magnetic bearings [49], and optimization of fermentation by using a high ethanol-tolerance yeast [50]. A DE based neural network-training algorithm was first introduced in [51]. In [52] the method's characteristics as a global optimizer were compared to other neural network training methods. Das et al. in [53] have compared the performance of some variants of the DE with other common optimization algorithms like PSO, GA, etc. in context to the partitional clustering problem and concluded in their study that DE rather than GAs should receive primary attention in such partitional cluster algorithms.

In this section we describe a simple application of the aforementioned algorithms to the design of two dimensional IIR filters [54]. In signal processing, the function of a filter is to remove unwanted parts of the signal, such as random noise, or to extract useful parts of the signal, such as the components lying within a certain frequency range. There are two main kinds of filter, *analog* and *digital*. They are quite different in their physical makeup and in how they work. An analog filter uses analog electronic circuits made up from components such as resistors, capacitors and op-amps to produce the required filtering effect. Such filter circuits are widely used in such applications as noise reduction, video signal enhancement, graphic equalisers in hi-fi systems, and many other areas. A digital filter uses a digital processor to perform numerical calculations on sampled values of the signal. The processor may be a general-purpose computer such as a PC, or a specialised DSP (digital signal processor) chip.

Digital filters are broadly classified into two main categories namely, FIR (*finite impulse response*) filters and IIR (*infinite impulse response*) filters. The impulse response of a digital filter is the output sequence from the filter when a unit impulse is applied at its input. (A unit impulse is a very simple input sequence consisting of a single value of 1 at time $t = 0$, followed by zeros at all subsequent sampling instants). An FIR filter is one whose impulse response is of finite duration. The output of such a filter is calculated solely from the current and previous input values. This type of filter is hence said to be *non-recursive*. On the other hand, an IIR filter is one whose impulse response (theoretically) continues for ever in time. They are also termed as *recursive* filters. The current output of such a filter depends upon previous output values. These, like the previous input values, are stored in the processor's memory. The word recursive literally means "running back", and refers to the fact that previously-calculated output values go back into the calculation of the latest output. The recursive (previous output) terms feed back energy into the filter input and keep it going.

In our work [54, 55] the filter design is mainly considered from a frequency domain perspective. Frequency domain filtering consists in first, taking the fourier transform of the two-dimensional signal (which may be the pixel intensity value in case of a gray-scale image), then multiplying the frequency domain signal by the transfer function of the filter and finally inverse trans-

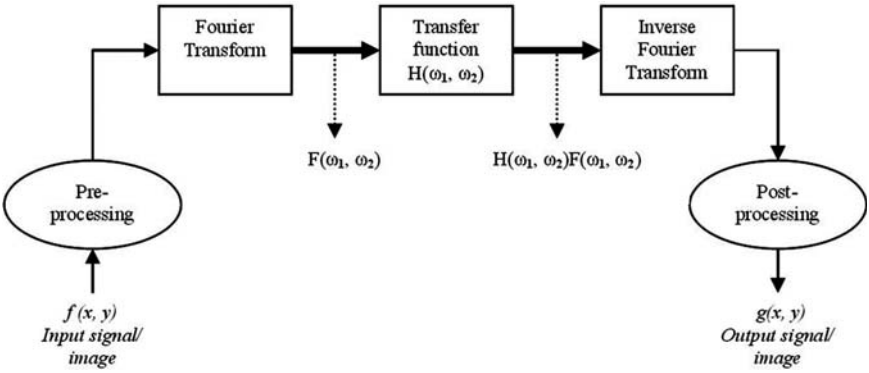

Fig. 13. Scheme of filtering with 2-D digital filter

forming the product in order to get the output response of the filter. The scheme is illustrated in Fig. 13.

Let the general prototype 2-D transfer function for the digital filter be

$$H(z_1, z_2) = H_0 \frac{\sum_{i=0}^{N} \sum_{j=0}^{N} p_{ij} z^i z^j}{\prod_{k=1}^{N} (1 + q_k z_1 + r_k z_2 + s_k z_1 \cdot z_2)} \tag{29}$$

It is a general practice to take $p_{00} = 1$ (by normalizing p_{ij}'s with respect to the value of p_{00}). Also, let us assume that the user-specified amplitude response of the filter to be designed is M_{d} which is obviously a function of digital frequencies ω_1 and ω_2 ($\omega_1,\ \omega_2\ \epsilon\ [0, \pi]$). Now the main design problem is to determine the coefficients in the numerator and denominator of (29) in such a fashion that $H(z_1,\ z_2)$ follows the desired response M_{d} ($\omega_1,\ \omega_2$) as closely as possible. Such an approximation of the desired response can be achieved by minimizing

$$J(p_{ij}, q_k, r_k, s_k, H_0) = \sum_{n_1=0}^{N_1} \sum_{n_2=0}^{N_2} [|M(\omega_1, \omega_2)| - M_d(\omega_1, \omega_2)]^b, \tag{30}$$

where

$$M(\omega_1, \omega_2) = H(z_1, z_2) \quad \Big|_{\substack{z_1 = e^{j\omega_1} \\ z_2 = e^{j\omega_2}}} \tag{31}$$

and

$$\omega_1 = (\pi/N_1)_{n_1};$$
$$\omega_2 = (\pi/N_2)_{n_2};$$

and b is an even positive integer (usually $b = 2$ or 4). Equation (30) can be restated as

$$J = \sum_{n_1=0}^{N_1} \sum_{n_2=0}^{N_2} \left[\left| M\left(\frac{\pi n_1}{N_1}, \frac{\pi n_2}{N_2}\right) \right| - M_d\left(\frac{\pi n_1}{N_1}, \frac{\pi n_2}{N_2}\right) \right]^b. \tag{32}$$

Here the prime objective is to reduce the difference between the desired and actual amplitude responses of the filter at $N_1 \cdot N_2$ points. For BIBO (bounded input bounded output) stability the prime requirement is that the z-plane poles of the filter transfer function should lie within the unit circle. Since the denominator contains only first degree factors, we can assert the stability conditions as

$$|q_k + r_k| - 1 < s_k < 1 - |q_k - r_k|, \tag{33}$$

where $k = 1,\ 2, \ldots, N$.

We solve the above constrained minimization problems using a binary coded GA [56], PSO, DE and PSO-DV and observe that the synergistic PSO-DV performs best as compared to the other competitive algorithms over this problem.

To judge the accuracy of the algorithms, we run all of them for a long duration upto 100,000 FEs. Each algorithm is run independently (with a different seed for the random number generator in every run) for 30 times and the mean best J value obtained along with the standard deviations have been repored for the design problem (32) in Table 5. Figures 14 and 15 illustrate the frequency responses of the filters. The notation J_b has been used to denote four sets of experiments performed with the value of J obtained using exponent $b = 1$, 2, 4 and 8. Table 6 summarizes the results of the unpaired t-test on the J values (standard error of difference of the two means, 95% confidence interval of this difference, the t value, and the two-tailed P value) between the best and next-to-best results in Table 4. For all cases in Table 6, sample size = 30 and number of degrees of freedom = 58.

The frequency response of the various filters deigned by the above mentioned algorithms are shown below.

Table 5. Mean value and standard deviations of the final results (J values) with exponent $p = 1$, 2, 4, 8 after 100,000 FEs (mean of 20 independent runs of each of the competitor algorithms)

Value of J for different exponents	PSO-DV	PSO	DE	Binary GA in [56]
J_1	$\mathbf{61.7113 \pm 0.0054}$	98.5513 ± 0.0327	95.7113 ± 0.0382	96.7635 ± 0.8742
J_2	$\mathbf{9.0215 \pm 0.0323}$	11.9078 ± 0.583	10.4252 ± 0.0989	10.0342 ± 0.0663
J_4	$\mathbf{0.5613 \pm 0.00054}$	0.9613 ± 0.0344	0.5732 ± 0.0024	0.6346 ± 0.0154
J_8	0.0024 ± 0.001	0.2903 ± 0.0755	$\mathbf{0.0018 \pm 0.0006}$	0.0091 ± 0.0014

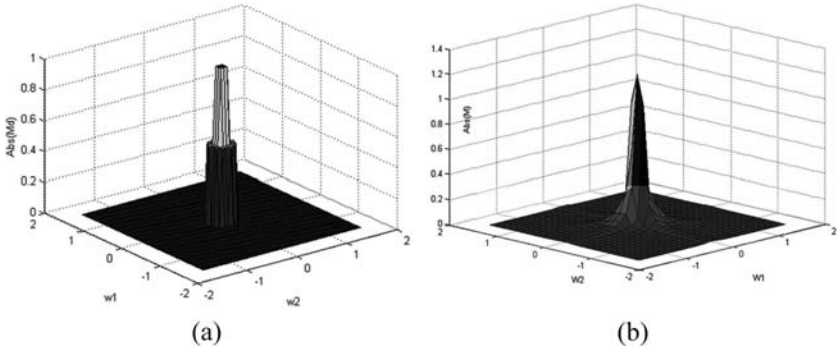

Fig. 14. (**a**) Ideal desired filter response (**b**) Frequency response of the filter designed by PSO-DV

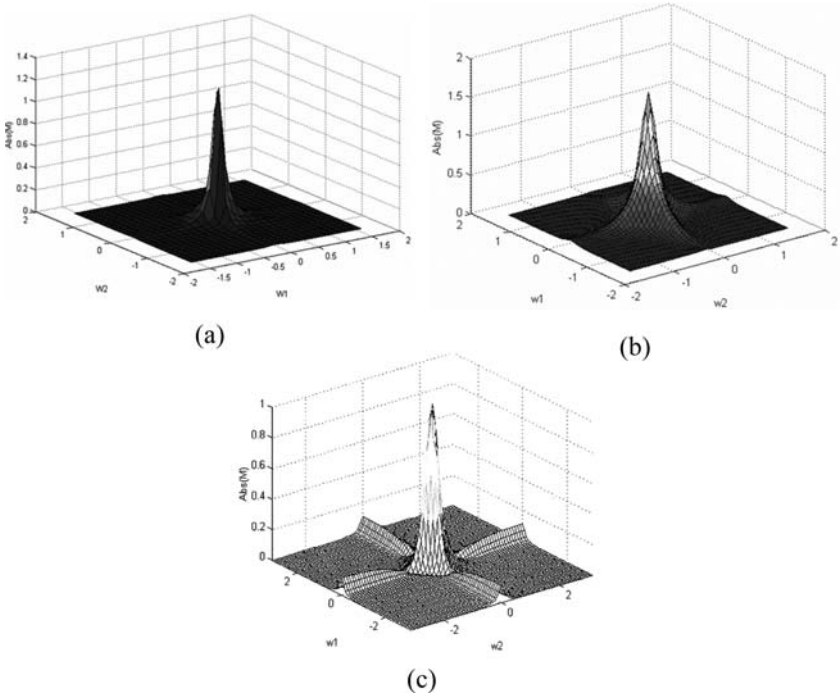

Fig. 15. (**a**) Frequency response of the filter designed by PSO (**b**) Frequency response of the filter designed by DE (**c**) Frequency response of the filter designed by a binary GAv [56]

Table 6. Results of unpaired t-tests on the data of Table 5

Cost function	Std. err	t	95% Conf. intvl	Two-tailed P	Significance
J_1	0.195	179.313	−35.447 to −34.656	<0.0001	**Extremely significant**
J_2	0.221	6.3440	−1.8516 to −0.9557	<0.0001	**Extremely significant**
J_4	0.030	3.1041	−0.1518 to −0.0319	0.0036	**Very significant**
J_8	0.000	1.9551	−0.0000213 to 0.0012213	0.0580	Not significant

11 Conclusions

Search and optimization problems are ubiquitous through the various realms of science and engineering. This chapter has provided a comprehensive overview of two promising optimization algorithms, which are currently gaining popularity for their greater accuracy, faster convergence speed and simplicity. One of these algorithms, known as PSO mimics the behavior of a group of social insects in multi-agent cooperative search problems. The latter one called DE (DE) is a deviant variety of GA, which attempts to replace the crossover operator in GA by a special type of differential operator for reproducing offspring in the next generation.

The chapter explores several schemes for controlling the convergence behaviors of PSO and DE by a judicious selection of their parameters. It also focuses on the hybridizations of these algorithms with other soft computing tools. It finally discusses the mutual synergy of PSO with DE leading to a more powerful global search algorithm. Applications of the algorithms to diverse domains of engineering problems have been surveyed. The chapter elaborates one such application of PSO, DE and their variants to the design of IIR digital filters in greater details.

The present article reveals that a significant progress has been made in the field of swarm intelligence and evolutionary computing in the past 10 years. In recent times, a symbiosis of swarm intelligence with other soft computing algorithms has opened up new avenues for the next generation computing systems. Engineering search and optimization problems including pattern recognition, bioinformatics and machine intelligence will find new dimensions in the light of hybridization of swarm intelligence with other algorithms.

References

1. Konar A (2005), Computational Intelligence: Principles, Techniques and Applications, Springer, Berlin Heidelberg New York.
2. Holland JH (1975), Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor.

3. Goldberg DE (1975), Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA.
4. Kennedy J, Eberhart R and Shi Y (2001), Swarm Intelligence, Morgan Kaufmann, Los Altos, CA.
5. Kennedy J and Eberhart R (1995), Particle Swarm Optimization, In Proceedings of IEEE International Conference on Neural Networks, pp. 1942–1948.
6. Storn R and Price K (1997), Differential Evolution – A Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces, Journal of Global Optimization, 11(4), 341–359.
7. Venter G and Sobieszczanski-Sobieski J (2003), Particle Swarm Optimization, AIAA Journal, 41(8), 1583–1589.
8. Yao X, Liu Y, and Lin G (1999), Evolutionary Programming Made Faster, IEEE Transactions on Evolutionary Computation, 3(2), 82–102.
9. Shi Y and Eberhart RC (1998), Parameter Selection in Particle Swarm Optimization, Evolutionary Programming VII, Springer, Lecture Notes in Computer Science 1447, 591–600.
10. Shi Y and Eberhart RC (1999), Empirical Study of Particle Swarm Optimization, In Proceedings of the 1999 Congress of Evolutionary Computation, vol. 3, IEEE Press, New York, pp. 1945–1950.
11. Angeline PJ (1998), Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences, Evolutionary Programming VII, Lecture Notes in Computer Science 1447, Springer, Berlin Heidelberg New York, pp. 601–610.
12. Shi Y and Eberhart RC (1998), A Modified Particle Swarm Optimiser, IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4–9.
13. Shi Y and Eberhart RC (2001), Fuzzy Adaptive Particle Swarm Optimization, In Proceedings of the Congress on Evolutionary Computation 2001, Seoul, Korea, IEEE Service Center, IEEE (2001), pp. 101–106.
14. Clerc M and Kennedy J (2002), The Particle Swarm – Explosion, Stability, and Convergence in a Multidimensional Complex Space, IEEE Transactions on Evolutionary Computation, 6(1), 58–73.
15. Eberhart RC and Shi Y (2000), Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization, In Proceedings of IEEE International Congress on Evolutionary Computation, vol. 1, pp. 84–88.
16. van den Bergh F and Engelbrecht PA (2001), Effects of Swarm Size on Cooperative Particle Swarm Optimizers, In Proceedings of GECCO-2001, San Francisco, CA, pp. 892–899.
17. Ratnaweera A, Halgamuge SK, and Watson HC (2004), Self-Organizing Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients, IEEE Transactions on Evolutionary Computation, 8(3), 240–255.
18. Kennedy J (1999), Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance, In Proceedings of the 1999 Congress of Evolutionary Computation, vol. 3, IEEE Press, New York, pp. 1931–1938.
19. Kennedy J and Eberhart RC (1997), A Discrete Binary Version of the Particle Swarm Algorithm, In Proceedings of the 1997 Conference on Systems, Man, and Cybernetics, IEEE Service Center, Piscataway, NJ, pp. 4104–4109.
20. Løvbjerg M, Rasmussen TK, and Krink T (2001), Hybrid Particle Swarm Optimizer with Breeding and Subpopulations, In Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001).

21. Krink T, Vesterstrøm J, and Riget J (2002), Particle Swarm Optimization with Spatial Particle Extension, In Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2002).
22. Løvbjerg M and Krink T (2002), Extending Particle Swarms with Self-Organized Criticality, In Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002).
23. Miranda V and Fonseca N (2002), EPSO – Evolutionary Particle Swarm Optimization, a New Algorithm with Applications in Power Systems, In Proceedings of IEEE T&D AsiaPacific 2002 – IEEE/PES Transmission and Distribution Conference and Exhibition 2002: Asia Pacific, Yokohama, Japan, vol. 2, pp. 745–750.
24. Blackwell T and Bentley PJ (2002), Improvised Music with Swarms. In Proceedings of IEEE Congress on Evolutionary Computation 2002.
25. Robinson J, Sinton S, and Rahmat-Samii Y (2002), Particle Swarm, Genetic Algorithm, and Their Hybrids: Optimization of a Profiled Corrugated Horn Antenna, In Antennas and Propagation Society International Symposium, 2002, vol. 1, IEEE Press, New York, pp. 314–317.
26. Krink T and Løvbjerg M (2002), The Lifecycle Model: Combining Particle Swarm Optimization, Genetic Algorithms and Hill Climbers, In Proceedings of PPSN 2002, pp. 621–630.
27. Hendtlass T and Randall M (2001), A Survey of Ant Colony and Particle Swarm Meta-Heuristics and Their Application to Discrete Optimization Problems, In Proceedings of the Inaugural Workshop on Artificial Life, pp. 15–25.
28. vandenBergh F and Engelbrecht A (2004), A Cooperative Approach to Particle Swarm Optimization, IEEE Transactions on Evolutionary Computation 8(3), 225–239.
29. Parsopoulos KE and Vrahatis MN (2004), On the Computation of All Global Minimizers Through Particle Swarm Optimization, IEEE Transactions on Evolutionary Computation, 8(3), 211–224.
30. Price K, Storn R, and Lampinen J (2005), Differential Evolution – A Practical Approach to Global Optimization, Springer, Berlin Heidelberg New York.
31. Fan HY, Lampinen J (2003), A Trigonometric Mutation Operation to Differential Evolution, International Journal of Global Optimization, 27(1), 105–129.
32. Das S, Konar A, Chakraborty UK (2005), Two Improved Differential Evolution Schemes for Faster Global Search, ACM-SIGEVO Proceedings of GECCO' 05, Washington D.C., pp. 991–998.
33. Chakraborty UK, Das S and Konar A (2006), DE with Local Neighborhood, In Proceedings of Congress on Evolutionary Computation (CEC 2006), Vancouver, BC, Canada, IEEE Press, New York.
34. Das S, Konar A, Chakraborty UK (2005), Particle Swarm Optimization with a Differentially Perturbed Velocity, ACM-SIGEVO Proceedings of GECCO' 05, Washington D.C., pp. 991–998.
35. Salerno J (1997), Using the Particle Swarm Optimization Technique to Train a Recurrent Neural Model, IEEE International Conference on Tools with Artificial Intelligence, pp. 45–49.
36. van den Bergh F (1999), Particle Swarm Weight Initialization in Multi-Layer Perceptron Artificial Neural Networks, Development and Practice of Artificial Intelligence Techniques, Durban, South Africa, pp. 41–45.

37. He Z, Wei C, Yang L, Gao X, Yao S, Eberhart RC, and Shi Y (1998), Extracting Rules from Fuzzy Neural Network by Particle Swarm Optimization, In Proceedings of IEEE Congress on Evolutionary Computation (CEC 1998), Anchorage, Alaska, USA.
38. Eberhart RC and Hu X (1999), Human Tremor Analysis Using Particle Swarm Optimization, In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), Washington D.C., pp. 1927–1930.
39. Wachowiak MP, Smolíková R, Zheng Y, Zurada MJ, and Elmaghraby AS (2004), An Approach to Multimodal Biomedical Image Registration Utilizing Particle Swarm Optimization, IEEE Transactions on Evolutionary Computation, 8(3), 289–301.
40. Messerschmidt L and Engelbrecht AP (2004), Learning to Play Games Using a PSO-Based Competitive Learning Approach, IEEE Transactions on Evolutionary Computation 8(3), 280–288.
41. Yoshida H, Kawata K, Fukuyama Y, Takayama S, and Nakanishi Y (2000), A Particle Swarm Optimization for Reactive Power and Voltage Control Considering Voltage Security Assessment, IEEE Transactions on Power Systems, 15(4), 1232–1239.
42. Abido MA (2002), Optimal Design of Power System Stabilizers Using Particle Swarm Optimization, IEEE Transactions on Energy Conversion, 17(3), 406–413.
43. Paterlini S and Krink T (2006), Differential Evolution and Particle Swarm Optimization in Partitional Clustering, Computational Statistics and Data Analysis, vol. 50, 1220–1247.
44. Rogalsky T, Kocabiyik S and Derksen R (2000), Differential Evolution in Aerodynamic Optimization, Canadian Aeronautics and Space Journal, 46(4), 183–190.
45. Doyle S, Corcoran D, and Connell J (1999), Automated Mirror Design Using an Evolution Strategy, Optical Engineering, 38(2), 323–333.
46. Stumberger G, Dolinar D, Pahner U, and Hameyer K (2000), Optimization of Radial Active Magnetic Bearings Using the Finite Element Technique and Differential Evolution Algorithm, IEEE Transactions on Magnetics, 36(4), 1009–1013.
47. Wang FS and Sheu JW (2000), Multi-Objective Parameter Estimation Problems of Fermentation Processes Using High Ethanol Tolerance Yeast, Chemical Engineering Science, 55(18), 3685–3695.
48. Masters T and Land W (1997), A New Training Algorithm for the General Regression Neural Network," Proceedings of Computational Cybernetics and Simulation, Organized by IEEE Systems, Man, and Cybernetics Society, 3, 1990–1994.
49. Zelinka I and Lampinen J (1999), An Evolutionary Learning Algorithms for Neural Networks, In Proceedings of Fifth International Conference on Soft Computing, MENDEL'99, pp. 410–414.
50. Das S, Abraham A, and Konar A (2007), Adaptive Clustering Using Improved Differential Evolution Algorithm, IEEE Transactions on Systems, Man and Cybernetics – Part A, IEEE Press, New York, USA.
51. Das S and Konar A (2007), A Swarm Intelligence Approach to the Synthesis of Two-Dimensional IIR Filters, Engineering Applications of Artificial Intelligence, 20(8), 1086–1096. http://dx.doi.org/10.1016/j.engappai.2007.02.004

52. Das S and Konar A (2006), Two-Dimensional IIR Filter Design with Modern Search Heuristics: A Comparative Study, International Journal of Computational Intelligence and Applications, 6(3), Imperial College Press.
53. Mastorakis N, Gonos IF, Swamy MNS (2003), Design of Two-Dimensional Recursive Filters Using Genetic Algorithms, IEEE Transactions on Circuits and Systems, 50, 634–639.
54. Liu H, Abraham A, and Clerc M (2007), Chaotic Dynamic Characteristics in Swarm Intelligence, Applied Soft Computing Journal, Elsevier Science, 7(3), 1019–1026.
55. Abraham A, Liu H, and Chang TG (2006), Variable Neighborhood Particle Swarm Optimization Algorithm, Genetic and Evolutionary Computation Conference (GECCO-2006), Seattle, USA, Late Breaking Papers, CD Proceedings, Jörn Grahl (Ed.).
56. Abraham A, Das S, and Konar A (2007), Kernel Based Automatic Clustering Using Modified Particle Swarm Optimization Algorithm, 2007 Genetic and Evolutionary Computation Conference, GECCO 2007, ACM Press, Dirk Thierens et al. (Eds.), ISBN 978-1-59593-698-1, pp. 2–9.