Differential Evolution

Kenneth V. Price

摘要：在介绍了对经典随机游动的讨论之后，本文介绍了差分进化（DE）全局数值优化算法的逐步开发。根据与不变性和对称性原则的一致性，控制参数的依赖程度，计算效率和对随机化的响应，对五种基本的DE策略进行了评估，每种策略都比它的上一种更为复杂。根据经验得出针对凸二次函数族的最佳控制参数设置。

Abstract. After an introduction that includes a discussion of the classic random walk, this paper presents a step-by-step development of the differential evolution (DE) global numerical optimization algorithm. Five fundamental DE strategies, each more complex than the last, are evaluated based on their conformance to invariance and symmetry principles, degree of control parameter dependence, computational efficiency and response to randomization. Optimal control parameter settings for the family of convex, quadratic functions are empirically derived.

1 Introduction

First proposed in 1997 by Storn and Price (this author), Differential Evolution (DE) is a simple yet effective algorithm for global (*multimodal*) optimization [1–10]. Although primarily designed for optimizing functions of continuous and discrete numerical variables, DE has also been applied with some success to combinatorial problems [11].

DE belongs to the class of *evolutionary algorithms* (EAs), so-called because they are population-based methods that rely on mutation, recombination and selection to evolve a collection of candidate solutions toward an optimal state. Like most EAs, DE exploits the population *via* recombination. DE does not, however, attempt to mimic natural searches, like those of ants [12], bees [13], the immune system [14], or those arising from social interaction [15]. Furthermore, only DE directly samples the population to drive mutation – a strategy that has many benefits.

Before introducing DE, the next section provides background by outlining the general global numerical optimization problem and by providing an example of how the classic random walk algorithm tries to solve it. The subsequent section introduces DE's unique *differential mutation* operator and a simple mutation-only DE algorithm, whose performance it contrasts with that of the random walk.

Kenneth V. Price
e-mail: kvprice@pacbell.net

Subsequent sections introduce recombination and a series of increasingly complex DE updating strategies, concluding with what has become known as “classic DE”. Algorithms with adaptive control variables, multiple difference vectors and hybrid methods are not discussed. For a recent summary of work in these and other areas, the reader is directed to [16, 17]. Instead, space limits this discussion to only the most basic DE strategies, some of which, however, are incorporated into these other algorithms.

Throughout, this chapter emphasizes how symmetry constrains algorithm design. Strategies are also judged by other criteria, like their degree of control parameter dependence, computational efficiency and response to randomization. Ultimately, the reader should gain new perspectives on basic DE strategies, learn how they are best applied and develop a better idea of each algorithm’s strengths and weaknesses.

2 Background

After stating the general global numerical optimization problem, this section briefly explores the classical random walk algorithm so that its methodology and performance can subsequently be contrasted with that of DE.

2.1 Problem Statement

Given a function $f(\mathbf{x})$, the general numerical optimization problem can be stated as:

求解最小化问题

$$\text{Find } \mathbf{x}^* \,\big|\, f(\mathbf{x}^*) \le f(\mathbf{x}) \;\forall\; \mathbf{x} \in M \subset \Re^D, \tag{1}$$

where M is the *feasible* space and both $\mathbf{x}$ and $\mathbf{x}^*$ are vectors of D, real-valued (floating-point) parameters. Parameter values for the vector $\mathbf{x} = \{x_1, x_2\ldots, x_D\}$ can be viewed as its coordinates in D-dimensional space, so the problem is to find the location in the feasible space at which the function reaches its minimum value. (To search for a maximum, replace “$\le$” with “$\ge$” in Eq. 1, or multiply $f(\mathbf{x})$ by -1).

2.2 The Random Walk

DE is a stochastic, direct search that can optimize functions based only on samples of their value at isolated, stochastically chosen locations. The direct search is particularly valuable when the objective function is not differentiable or has no analytical description. This next section describes the quintessential direct search with which DE shares much in common – the random walk.

The random walk begins at a single point $\mathbf{x}_g$ that has been chosen with random uniformity from M. The index g is a generation counter. Once the walk has been initialized, a *mutation* operator creates an adversarial *mutant vector* $\mathbf{v}_g$ that will compete to replace $\mathbf{x}_g$.

In real-parameter optimization, mutation usually means incrementing one or more parameters, not inverting random bits. The classic random walk mutates $\mathbf{x}_g$ by adding it to a random vector $\mathbf{n}_i(0,1)$ whose D components are *independent* samples of a normally distributed random variable whose mean and standard deviation are zero and one, respectively (Eq. 2). The *mutation scale factor F* adjusts the normal distribution's standard deviation, or "*step-size*". F：变异缩放因子

变异

$$\mathbf{v}_g = \mathbf{x}_g + F \cdot \mathbf{n}_i(0,1) \tag{2}$$

A *selection* operation (Eq. 3) then ensures that the fitter vector, i.e. the one with the lower objective function value, survives into the next generation. In particular, if $f(\mathbf{v}_g) \le f(\mathbf{x}_g)$, then $\mathbf{v}_g$ instead of $\mathbf{x}_g$ becomes the vector $\mathbf{x}_{g+1}$ that will be mutated in generation $g + 1$.

选择

$$\mathbf{x}_{g+1} = \begin{cases} \mathbf{v}_g & \text{if } f(\mathbf{v}_g) \le f(\mathbf{x}_g) \\ \mathbf{x}_g & \text{otherwise} \end{cases} \tag{3}$$

It is important to note that $\mathbf{v}_g$ *is accepted even if its function value equals that of its adversary* $\mathbf{x}_g$; otherwise, the walk will stagnate on flat functional landscapes.

There are a number of possible termination criteria [18], but for simplicity it is assumed that competitions continue until the number of generations reaches the preset maximum, $g_{\max}$.

3 Differential Mutation with Local Selection

Compared to the random walk, the differential mutation algorithm:

- Executes N walks in parallel: $\mathbf{x}_g \rightarrow \mathbf{x}_{i,g}$, $i = 1, 2\ldots, N$
- Replaces $\mathbf{n}_i(0,1)$ with a random vector difference: $\mathbf{n}_i(0,1) \rightarrow \mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}$

More particularly, the population consists of N, D-dimensional vectors $\mathbf{x}_{i,g} = [x_{1,i,g}, x_{2,i,g}, \ldots, x_{D,i,g}]$. Some recent DE variants allow the population size to vary during optimization [19], [20], but most hold N constant throughout, in part because population size is often dimension-dependent and the dimension of most problems does not change during optimization.

3.1 DE/target/1

In this simple DE algorithm, each population vector $\mathbf{x}_{i,g}$ competes against a mutant vector $\mathbf{v}_{i,g}$ once per generation for the right to remain in the population.

DE generates $\mathbf{v}_{i,g}$ by the process known as *differential mutation* (Eq. 4), which adds $\mathbf{x}_{i,g}$ to a randomly selected *vector difference* $\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}$ that has been scaled by F.

差分变异

$$\mathbf{v}_{i,g} = \mathbf{x}_{i,g} + F \cdot \left(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}\right), \quad r1, r2 \in \{1,2\ldots,N\}, \quad r1 \neq r2 \neq i\,. \tag{4}$$

Both $r1$ and $r2$ are randomly chosen population indices, except that $r1 \neq r2 \neq i$. The restriction $r1 \neq r2$ ensures that the perturbation will not be zero, while the restrictions $r1 \neq i$ and $r2 \neq i$ exclude the possibility that the differential mutation operation in Eq. 4 will degenerate into *line-recombination* (see section 5).

In DE parlance, $\mathbf{x}_{i,g}$ – the vector that is competing to remain in the population – is called the *target vector*, whereas $\mathbf{x}_{r1,g}$ and $\mathbf{x}_{r2,g}$ are known as *difference vectors*. The vector to which the scaled vector difference is added is known as the *base vector*. In this simplest of DE algorithms, $\mathbf{x}_{i,g}$ is both the base vector and the target vector, but this will not always be the case. Other base vector options and their impact on optimization are discussed in sections 4, 5 and 6.

Many (but not all) DE strategies have a shorthand moniker of the form: DE/x/y/z, where "x" is a string that describes how the base vector is chosen, "y" is the number of vector differences added to the base vector and "z" is a string that describes a recombination operation. Since it does not invoke recombination, the above algorithm is simply known as *DE/target/1*.

Like the random walk, a single comparison between the function values of $\mathbf{x}_{i,g}$ and its associated mutant vector $\mathbf{v}_{i,g}$ determines which becomes the i^{th} member of the next generation.

选择

$$\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{v}_{i,g} & \text{if} \quad f(\mathbf{v}_{i,g}) \leq f(\mathbf{x}_{i,g}) \\ \mathbf{x}_{i,g} & \text{otherwise} \end{cases} \tag{5}$$

Local selection refers to the present case in which the base and target vectors are the same, whereas *global selection* describes those cases in which the base and target vectors are distinct. Under local selection, each population vector evolves in isolation within its own niche as it competes against its own mutant.

Full pseudo-code for the DE/target/1 algorithm appears below, but for convenience and to save space, the remaining algorithms in this chapter appear only in shorthand notation, like this expression for the DE/target/1/ algorithm:

$$\mathbf{x}_{i,g} \;\; vs. \;\; \mathbf{x}_{i,g} + F \cdot \left(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}\right), \quad r1 \neq r2 \neq i \tag{6}$$

where by "*vs.*" it is understood that the mutant vector becomes the i^{th} member of the next generation if its function values is less than or equal to that of the target vector. A similar algorithm exists for functions defined by matrix variables:

$$\mathbf{X}_{i,g} \;\; vs. \;\; \mathbf{X}_{i,g} + F \cdot \left(\mathbf{X}_{r1,g} - \mathbf{X}_{r2,g}\right), \quad r1 \neq r2 \neq i \tag{7}$$

In this scenario, matrix addition/subtraction replaces vector addition/subtraction.

DE/target/1: Input: $F > 0$, $N > 2$, $g_{\max}$

```
Initialize the population with $N$ vectors sampled at random from $M$; $g = 0$.
while $(g \le g_{\max})$
    for $i = 1$ to $N$
        select $r1, r2 \in \{1,2...,N\}$ at random,   $r1 \ne r2 \ne i$
        $\mathbf{v}_{i,g} = \mathbf{x}_{i,g} + F \cdot (\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g})$

        $\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{v}_{i,g} & \text{if } f(\mathbf{v}_{i,g}) \le f(\mathbf{x}_{i,g}) \\ \mathbf{x}_{i,g} & \text{otherwise} \end{cases}$
    end for
    $g = g + 1$
end while
```

3.2 *A Quadratic Performance Comparison: RW vs. DE/target/1*

Three simple quadratic functions – the sphere, the ellipse and the rotated ellipse – serve to illustrate some of DE's major features. The sphere demonstrates that DE's performance on these functions – unlike that of the random walk – is scale-invariant. Because its performance is scale-invariant, DE can return highly accurate results. The ellipse, with its principle axes *aligned along coordinate axes*, shows that DE's performance is also invariant with respect to eccentricity. As a result, solution accuracy does not suffer when parameter sensitivities are different. Finally, the rotated ellipse with its axes *aligned along coordinate diagonals* reveals that *like* the random walk, the performance of some DE algorithms is rotationally invariant, even though the mechanism that enables the invariance for DE is different from that of the random walk. Rotational invariance ensures that the choice of coordinate system orientation does not impair DE's ability to compensate for different parameter sensitivities.

3.2.1 The Sphere: Scale-Invariant Performance

Equation 8 shows that the sphere function $f_1(\mathbf{x})$ is a sum of squares whose minimum value is $f_1(\mathbf{x}^*) = 0.0$ at $\mathbf{x}^* = \mathbf{0} = [0.0, 0.0\ldots, 0.0]$.

$$f_1(\mathbf{x}) = \sum_{j=1}^{D} x_j^2 \,. \tag{8}$$

Like the other two functions in this section, $f_1(\mathbf{x})$ is unimodal, so finding its minimum is not hard; what is of interest is the rate at which an optimization algorithm resolves $f_1(\mathbf{x})$'s optimum.

The following experiment with the ten-dimensional version of $f_1(\mathbf{x})$ compares the random walk's convergence rate to that of DE/target/1. All initial parameter values $x_{j,i,g=0}$ are uniformly distributed over the range $[x^{\text{low}}, x^{\text{high}}]$.

$$x_{j,i,g=0} = x^{\text{low}} + \mathrm{U}(0,1) \cdot \left(x^{\text{high}} - x^{\text{low}}\right); \quad j = 1,2...,D; \quad i = 1,2...,N. \tag{9}$$

In Eq. 9, U(0,1) is a random number generator that returns a uniformly distributed value in the range (0,1). In this experiment, $x_j^{\text{low}} = -100$, $x_j^{\text{high}} = 100$, $D = 10$ and $N = 20$ ($N = 1$ for the random walk).

During optimization, mutant vector parameters $v_{j,i,g}$ (and later in the text, *trial vector* parameters $u_{j,i,g}$) that fall outside this range are reset to a value randomly chosen to lie between the bound exceeded and the corresponding parameter value in $\mathbf{x}_{i,g}$.

$$v_{j,i,g} = \begin{cases} x^{\text{low}} + \mathrm{U}(0,1) \cdot \left(x_{j,i,g} - x^{\text{low}}\right) \text{ if } v_{j,i,g} < x^{\text{low}} \\ \\ x^{\text{high}} + \mathrm{U}(0,1) \cdot \left(x_{j,i,g} - x^{\text{high}}\right) \text{ if } v_{j,i,g} > x^{\text{high}} \end{cases} \tag{10}$$

As Fig. 1 (left) shows, the standard deviation F strongly affects the random walk's performance. Initially, large values produce more rapid convergence than small ones, but as the resolution of the search increases, steps must shrink to remain effective. Consequently, the random walk is efficient only over the limited range set by F.

By contrast, Fig. 1 (right) shows that when F falls within the right range, DE/target/1's performance on $f_1(\mathbf{x})$ is *scale-invariant*, i.e. the slope of the plot is the same regardless of the scale at which the measurement is taken. Each generation improves the population's average objective function value by a constant *factor*. As F decreases, convergence speed increases until $F < 0.4$ at which point steps become too small for the population to evolve with $N = 20$ (see sections 3.6.2 and 4.1 for more on the relation between F and N). Even though it maintains a population of candidate solutions, DE/target/1 ultimately resolves $f_1(\mathbf{x})$'s optimum faster than the random walk. Furthermore, its scale-invariant performance enables DE/target/1 to resolve $f_1(\mathbf{x})$'s optimum to the limit set by the software's floating-point precision (not shown).

3.2.2 The Ellipse: High Conditioning

The random walk's average step-size is not only constant over time, but also the same for each parameter. Typically, however, objective function values are more sensitive to changes in some variables than others. Such cases are common in engineering where parameters measure different physical quantities whose effects on the objective function may differ by orders of magnitude. For these *ill-conditioned* or *highly conditioned* functions, the large changes in function value caused by altering one variable can overwhelm the smaller changes produced when less sensitive variables are altered by the same amount. *Efficient optimization requires adapting step-sizes to each parameter so that they all generate the same average change in function value.*

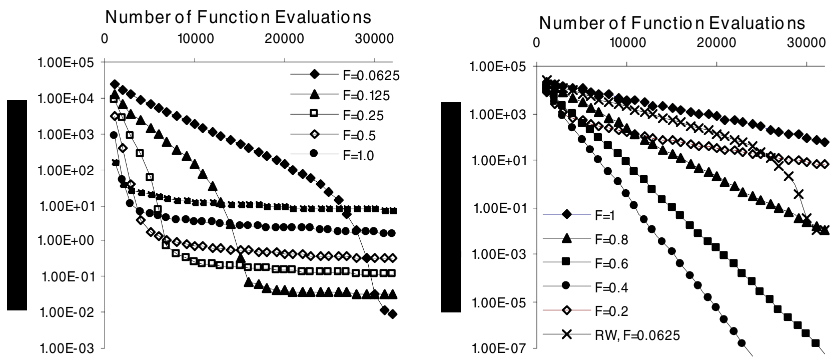

Fig. 1 (left) The random walk is efficient on $f_1(\mathbf{x})$ only over a short range defined by F, but DE/target/1 (right) is unaffected by the scale of the search when F • 0.4. Results are 100-trial averages and $D = 10$.

Because high eccentricity signifies high conditioning, the ellipsoid is an effective function for testing an algorithm's ability to adapt step-sizes to each parameter. For example, the *unimodal* benchmark function

$$f_2(\mathbf{x}) = \sum_{j=1}^{D} \left(jx_j\right)^2 \tag{11}$$

displays ellipsoidal contours (i.e., level lines/surfaces of constant function value) that are centered on the origin and whose principal axes coincide with the coordinate axes.

Because each parameter requires its own step-size, the simple random walk's performance on this function is even worse than it is on $f_1(\mathbf{x})$. For example, when steps are large enough to efficiently search the along the major axis of the ellipsoid, they are too large to efficiently search its minor axis. Similarly, when steps are small enough to efficiently search the ellipsoid's minor axis, they are too small to efficiently search its major axis.

Figure 2 (left) illustrates this performance hit as a convergence plot. Problem dimension ($D = 10$), parameter bounds (±100) and bound resetting (Eq. 10) are the same as for $f_1(\mathbf{x})$. Figure 2 (left) shows that when a single value for F scales step-sizes, the random walk exhibits a slow decline in efficiency. Curves flatten with time and have no "sweet spot" defined by F like the plots in Fig. 1 (left).

Figure 2 (right) plots DE's performance for $f_1(\mathbf{x})$, $f_2(\mathbf{x})$ and, for comparison, the random walk's best result for both functions. Whereas the random walk performs differently on $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ (and poorly on both), these two functions *are the same problem* for DE/target/1 except for the brief initial period required for the population to coalesce along functional contours. Its invariance to conditioning means that stretching the functional contours from spheres to ellipsoids does not significantly impact DE's performance.

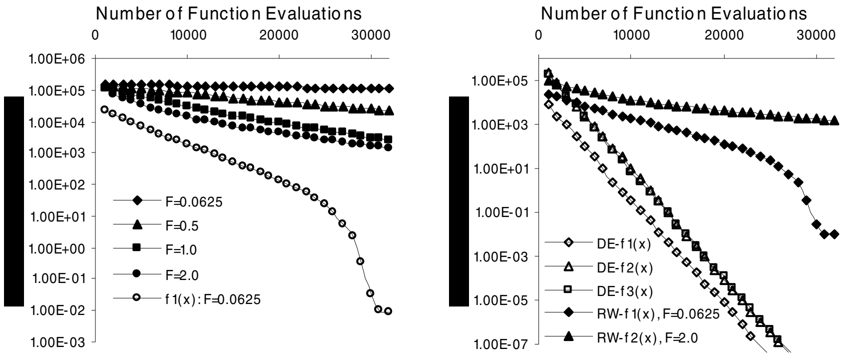

Fig. 2 (left) The random walk struggles on $f_2(\mathbf{x})$ because no one step-size is optimal for all axes. By contrast, Fig. 2 (right) shows that except for an initial adjustment period, $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$ (results overlap) are no harder for DE/target/1 than $f_1(\mathbf{x})$. All results are 100-trial averages; $D = 10$ and $F = 0.4$.

3.2.3 The Rotated Ellipse: Rotational Invariance

As long as the principal axes of the ellipsoid are aligned with coordinate axes – as they are for $f_2(\mathbf{x})$ – then the optimum can be efficiently located by independently adjusting the average step-size to the appropriate scale for each parameter of the objective function, i.e. parameters can be *independently optimized*. If, however, the ellipse is rotated with respect to the coordinate axes, then parameters become *dependent* (the equation of the ellipsoid in the new coordinate system will contain cross terms) [21]. Now, parameters cannot be optimized independently because the current best value for a given parameter depends on the values assigned to the remaining parameters.

One example of an ellipsoidal benchmark function whose principal axes align with coordinate system *diagonals* is Schwefel's ridge/valley:

$$f_3(\mathbf{x}) = \sum_{k=1}^{D} \left(\sum_{j=1}^{k} x_j \right)^2 \qquad (12)$$

Efficiently searching the major axis of this function's ellipsoidal contours now requires taking the longest steps along a coordinate diagonal. Taking this step will require incrementing all parameters simultaneously. Moreover, these mutations must be correlated, e.g. they all must be big steps of the right sign to generate a big step along the diagonal. Without correlating mutations, optimization will not be efficient when parameters are dependent [21].

Because all of DE/target/1's operations are vector-level, its performance is invariant under coordinate rotation. Consequently, DE/target/1's *good performance on highly conditioned functions is not affected by rotation.* Even though the ellipsoid is rotated, mutations are *automatically correlated.* Figure 2 (right) shows that despite $f_3(\mathbf{x})$ having dependent parameters, DE/target/1 performs as well on this rotated ellipsoid as it does on $f_2(\mathbf{x})$, so much so that the two plots are barely distinguishable. The next section looks at the phenomenon that enables DE to adapt not only to coordinate system rotation, but also to scaling and conditioning.

3.3 Contour Matching

DE automatically compensates for disparities in parameter sensitivity by exploiting the tendency of a population to become distributed along functional contours, i.e. *contour matching*. Contour matching occurs because poor solutions typically show greater improvement per generation than do the population's better solutions (which may not have much further to improve). Since the poorer solutions improve more quickly than the better ones, the population tends to concentrate in a comparatively narrow band of function values, thus revealing the function's contours, which typically contain the information needed to efficiently optimize the function.

For the family of convex quadratic functions, contour matching solves the problems posed by high eccentricity and rotation. For example, the plot on the left of Fig. 3 shows one of $f_3(\mathbf{x})$'s elliptical contours along with a twenty-member population after it has evolved under the action of DE/target/1 for 220 generations. The graph on the right plots the endpoints of the $20 \cdot 19 = 380$ non-zero difference vectors that the population generates. Visually, it is clear that both the population and its distribution of difference vectors not only mimic the eccentricity of the elliptical contour, but also its orientation. As a result of contour matching, step-sizes and orientations become well adapted to searching the region of interest.

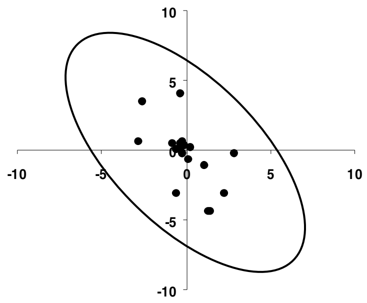

Fig. 3 Contour matching. The figure on the left shows one of $f_3(\mathbf{x})$'s elliptical contours and a population of twenty, two-dimensional vectors after they have evolved 220 generations under DE/target/1. The distribution of difference vectors that they generate is on the right. Dots locate each vector's head. For clarity, the difference vectors have been centered on the origin, but should be thought of as being centered on population vectors. Notice that the difference distribution is sparse, but center-symmetric.

3.4 Unbiased Mutant Distributions

The distribution of difference vectors, i.e. DE's mutation distribution, is *point-* or *center-symmetric*. Each point in Fig. 3 (right) has an equally probable, diametrically opposed counterpart that is equidistant from the distribution's center (the origin in this case). This is because ($\mathbf{x}_{r1} - \mathbf{x}_{r2}$) and ($\mathbf{x}_{r2} - \mathbf{x}_{r1}$) are equally probable, of equal magnitude, but oppositely directed. This *center-symmetry condition* can be stated mathematically as:

$$\forall \Delta\mathbf{x} \quad p(\Delta\mathbf{x}) = p(-\Delta\mathbf{x}), \tag{13}$$

where $\Delta\mathbf{x}$ is a sample of the distribution and $p(\Delta\mathbf{x})$ is its probability.

Center-symmetry ensures that the distribution of vector differences is free of *drift bias* [22]. For example, if one difference vector does not have an equally probable and oppositely directed counterpart, then the mutation distribution becomes skewed in that difference vector's direction. Instead of being zero, the mutation distribution's expected value (which is a simple average, since all allowed difference vectors are equally probable), becomes non-zero, i.e. biased. This bias is hard to justify, especially if nothing is known about the objective function. Since DE is to be a "black box" algorithm that operates without special knowledge of the function it is optimizing, its mutation distribution should be – and is – unbiased.

3.5 Randomizing the Mutation Scale Factor F

As Fig. 3 (right) suggests, the distribution of possible mutants is both finite and discrete when F is held constant. Randomizing F vastly increases the pool of possible mutants, which can mitigate the risk of the population *stagnating*, i.e. ceasing to evolve because none of the possible mutants is acceptable [23]. Randomization also ensures that vector differences' least significant digits do not become zero as the limit of the mantissa' precision is reached [1, p. 52].

3.5.1 Choosing a Distribution

Making F a random variable requires choosing a probability distribution function, its characteristic parameters and the frequency with which to sample new values for F. DE algorithms have appeared that sample F from normal [24], Cauchy [25], Lévy [26], Laplace [27], uniform [3, p. 89] and chaotic distributions [28]. None is consistently the best performer, although with their long "tails" the Cauchy and Lévy distributions may have some advantages when it comes to multimodal optimization. Generally speaking, the choice of probability distribution function is not critical, i.e. rarely has the choice of distribution alone made the difference between success and failure. This is perhaps not surprising, since holding F constant is usually effective.

This chapter randomizes F by sampling it from a normal distribution (Eq. 14). In this approach, F moderates the normal distribution's *standard deviation*, not its average.

$$F_i = F \cdot n_i(0,1)\,. \tag{14}$$

For comparison, this chapter also randomizes the scaling factor with a lognormal distribution whose *average* value is F (Eq. 15).

$$F_i = F \cdot \exp(n_i(0,1) - 0.5)\,. \tag{15}$$

3.5.2 Dither and Jitter

Perhaps more important to algorithm performance than the probability distribution function is the frequency with which new values for F are sampled. Unlike generating F anew for each vector (known as *dither*) or anew each generation, choosing F anew for each parameter (known as *jitter*) introduces artifacts into the search that reflect the choice of coordinate system. More specifically, the performance of the algorithm depends not just on the function, but also on the orientation of the coordinate axes with respect to the functional landscape [1 pp. 81–87], [10, pp. 89–94]. Consequently, dithering offers the greatest amount of diversity without causing DE's performance to become rotationally dependent.

3.6 Computational Efficiency

Computational complexity or "optimality" measures how a problem's size affects its difficulty. In this chapter, dimension (i.e. the number of variables in the objective function) measures a problem's size. Problem difficulty, however, is not so easy to characterize.

3.6.1 Success Performance

One performance measure that balances the conflicting objectives of speed and reliability is the *success performance* (*SP*) (Eq. 16), which is defined as the average number of function evaluations (*FE*) per success *s*, divided by the probability of success, which is estimated as the fraction of successful trials *s*/*t*, where *t* is the number of trials.

$$SP = \frac{\langle FE \rangle}{p_{success}} = \frac{\frac{1}{s}\sum_{k=1}^{s} FE_k}{s/t} = \frac{\sum_{k=1}^{s} FE_k}{s^2/t} \tag{16}$$

Since the optima of most benchmark functions are already known, *success* can be defined as any optimization trial that reaches the optimum to within a preset tolerance and within the allotted number of function evaluations. The optimum function value plus the tolerance is often called the *value-to-reach* (*VTR*) and it is chosen so that any algorithm that attains the *VTR* has already found the basin of attraction containing the global optimum. Presumably, refining the solution beyond the *VTR* is trivial. If there are no successes (i.e. if the *VTR* is not reached), then the success performance is not defined. The success performance measure embodies the idea that two algorithms perform equally if, for example, one algorithm is half as reliable but converges twice as fast as does the other. With a sufficient number of CPU cycles, both algorithms are equally likely to succeed. The *VTR* for the experiments in this chapter is 10^{-6}.

3.6.2 Computational Complexity of DE/target/1 on the Quadratic Family

Computational complexity is a property of an algorithm-function combination. The following experiment with DE/target/1 and $f_1(\mathbf{x})$ looks for the population size that gives best success performance for each combination of $F = [0.1, 0.15\ldots, 0.7]$ and $D = [1, 2\ldots, 30]$. The results, which are 100-trial averages, appear in Fig. 4. For clarity, only the results for which F was a multiple of 0.1 are plotted. Figure 4 (left), shows how *SP* grows with dimension.

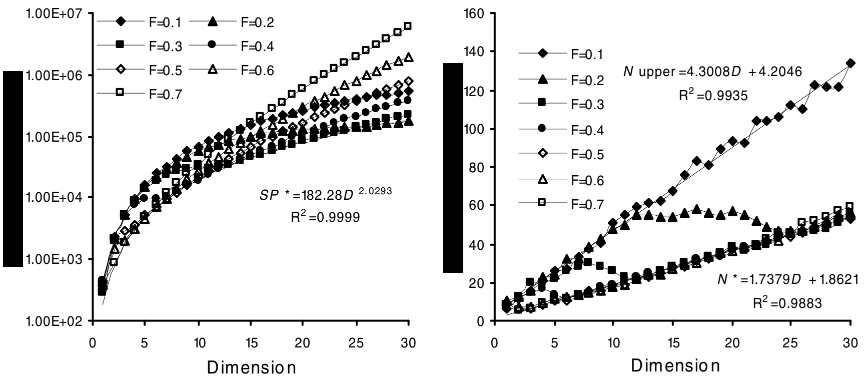

Fig. 4 (left) *SP vs. D* over a range of values for *F*. *N* is optimally chosen. The lower envelope of this collection of curves shows that the best *SP* at a given dimension, i.e. *SP**, is proportional to the D^2 when both *F* and *N* are optimal. Figure 4 (right) shows that when *F* is greater than *F**, *N* grows linearly with dimension *independent of F*.

At low dimension, the larger *F*-values in this range are the most effective, but as *D increases*, the optimal value for *F decreases*. Figure 5 plots, as a function of dimension, the values for *F* that produced the lowest *SP* at each dimension, i.e. *SP**, revealing that the optimal value for *F*, i.e. *F**, *approximately* decreases in proportion to $1/\sqrt{D}$.

$$F^* = 1.44D^{-0.5534} \qquad (17)$$

A similar result was reported in [29], where $F^* = 1.3149D^{-0.5242}$ was shown to be optimal for $f_1(\mathbf{x})$, $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$. The exact value for F^* is not critical, so for convenience, the experiments that follow assume that $F^* = 1.3/\sqrt{D}$.

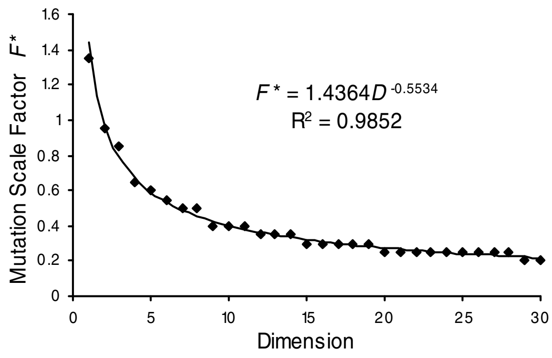

Fig. 5 The best value for *F*, i.e. *F**, approximately varies inversely with the square root of *D*.

A power-law trend-line through *SP** at each dimension shows that under the action of the DE/target/1 algorithm with $F = F^*$, *SP** grows in *proportion to the square of the dimension*, or in "Big *O*" notation: $f_1(\mathbf{x}) = O(D^2)$.

$$f_1(\mathbf{x}): SP^* = 182D^{2.03}, \quad F = F^*. \tag{18}$$

Ideally, the dependence of *SP** on dimension would be linear, but results are often considered to be acceptable if growth is low-order polynomial, since the time to execute an exhaustive grid-search grows exponentially with dimension. A stochastic EA that exploits this function's decomposability should solve it in $O(D\log D)$ time [21], so for DE/target/1 to solve it in $O(D^2)$ time *without* exploiting decomposability is quite good. Multi-modal functions, however, should not be expected to be solved as efficiently.

Figure 4 (right) plots the value of *N**, i.e. the value of *N*, that gave the lowest *SP* for each combination of *F* and *D*. For clarity, only the results for which *F* was a multiple of 0.1 are plotted. A trend line shows that the growth of *N** is linear with dimension as long as $F \geq F^*$.

$$f_1(\mathbf{x}): N^* = 1.74D + 1.9, \quad F \geq F^*. \tag{19}$$

Figure 4 (right) also shows that when $F < F^*$, the best population size can increase up to a second limit, roughly defined as $N_{\text{upper}} = 4.3D + 4.2$.

The overlap of curves along the bottom line in Fig. 4 (right) shows that *N** is independent of *F* as long as $F \geq F^*$. This *control parameter independence* is an important feature of DE/target/1, because it means that *F* can be adjusted through this range without requiring a corresponding adjustment in *N*, i.e. that independent features of the algorithm can be independently controlled.

3.6.3 Randomization's Impact on Computational Efficiency

One consequence of the independence of *F* and *N* is that randomizing *F* does not significantly impact DE/target/1's computational efficiency. Table 1 shows the effect that randomization has on both *SP** and *N**. A best fit power-law trend-line (which excludes the anomalous result for $D = 1$) shows that *neither the normal nor the lognormal distribution significantly alters the computational complexity (exponent) of the* $f_1(\mathbf{x})$*–DE/target/1 combination* compared to when *F* is held constant. Both distributions do, however, slow convergence by a comparatively small constant factor, but the better speed associated with holding *F* constant should not be construed as being typical, because randomization often proves to be faster when the objective function is multimodal.

Table 1 The effect that randomizing *F* has on *SP** and *N** for DE/target/1 ($1 < D \leq 30$).

F	SP^*	N^*
$F = F^* = 1.3/\sqrt{D}$	$182D^{2.03}$	$1.75D + 1.8$
$F_i = F^* \cdot n_i(0,1)$	$318D^{1.99}$	$2.22D + 5.5$
$F_i = F^* \cdot \exp(n_i(0,1) - 0.5)$	$343D^{2.01}$	$2.47D + 3.6$

Table 1 also summarizes the effect that randomizing *F* has on *N**. Randomization does not affect the linear dependence of *N** on *D*, but it does incur slightly larger populations when compared to holding *F* constant, probably to compensate for generating a significant fraction of *F*-values that are less than *F**. Like the control parameter independence that *N* and *F* display, the insensitivity of DE/target/1's efficiency to randomizing *F* is a valuable characteristic that not all strategies share.

4 Differential Mutation with Global Selection

Global selection refers to the case in which the base and target vectors are distinct. When selection is global, the target vector competes against another population member (or composite vector) that has been mutated.

4.1 DE/rand/1

In DE/rand/1 (Eq. 20), the base vector $\mathbf{x}_{r0,g}$ is randomly chosen from the population and – except for being mutually distinct from the base vector and from each other – so too are the two difference vectors $\mathbf{x}_{r1,g}$ and $\mathbf{x}_{r2,g}$.

$$\mathbf{x}_{i,g} \text{ vs. } \mathbf{x}_{r0,g} + F \cdot \left(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}\right), \quad r0 \neq r1 \neq r2 . \tag{20}$$

To prevent *premature convergence*, i.e. convergence to a sub-optimal solution, the population size for DE/rand/1 must *increase* as *F decreases*. In the DE/rand/1 algorithm, *N* and *F* are *dependent control parameters*. Figure 6 illustrates this dependence as it occurs for the ten-dimensional version of $f_1(\mathbf{x})$, showing that *N** *grows super-exponentially* as *F* decreases. The reason for this dependence, detailed in the next section, is the presence of bias in DE/rand/1's selection procedure.

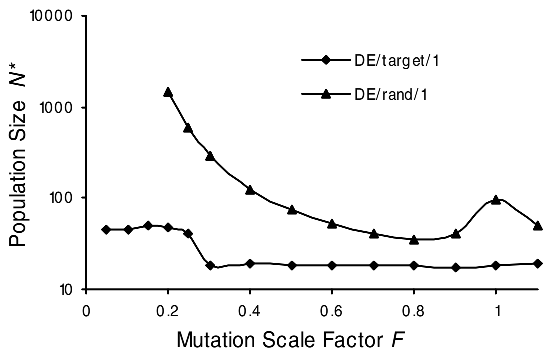

Fig. 6 This semi-log plot shows that the optimal population size N^* depends strongly on F when DE/rand/1 optimizes $f_1(\mathbf{x})$. DE/rand/1 also displays anomalous behavior when $F = 1.0$.

4.2 Unbiased Selection

The difference between the target vector and the mutant vector can be thought of as a possible *selection move*, i.e. a difference vector that *potentially* transfers the i^{th} vector to the location of the mutant vector. In the DE/target/1 algorithm, mutants are center-symmetrically distributed about the target vector, so the sum of all selection moves is null. As a result, DE/target/1 does not exhibit a selection bias. By contrast, mutants in the DE/rand/1 algorithm are center-symmetrically distributed about a *randomly chosen vector*, not the target vector. If the set of all possible selection moves is averaged, the result will be a *drift vector* pointing in the direction of the population's centroid, or mean location [22].

As before, this bias is hard to justify theoretically, but perhaps the best reason for eliminating selection drift bias is that the tendency to coalesce around the population's centroid must be counteracted by over-inflating populations to maintain reliable performance. As the next section shows, these large populations degrade DE/rand/1's computation efficiency compared to that of DE/target/1.

4.3 Computational Efficiency

When F becomes too small, the increased population size needed to keep evolution robust slows convergence more than decreasing F speeds it. Consequently, as the dimension of $f_1(\mathbf{x})$ increases, the population size needed to support evolution with $F = F^*$ becomes so large that convergence is slower what can be achieved with $F > F^*$ and smaller populations. As a result, DE/rand/1's optimal scale factor for $f_1(\mathbf{x})$ is not F^*, but *approximately* $F = 0.5$ (somewhat higher at low D). Furthermore, this value is virtually *independent of dimension* (Fig. 7 (left)).

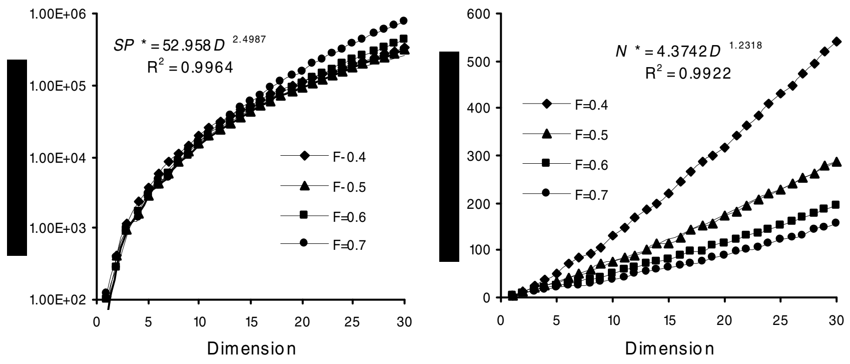

Fig. 7 (left) This plot of *SP vs.* the dimension of $f_1(\mathbf{x})$ shows that $F = 0.5$ is close to optimal regardless of dimension. Fig. 7 (right) Population growth is non-linear. Results are 100-trial averages.

Even at its optimal setting $F = 0.5$, DE/rand/1 exhibits non-linear population growth, which ultimately degrades its performance when compared to DE/target/1 (Fig. 7 (right)). Although DE/rand/1 is initially faster than DE/target/1, their roles are reversed once $D > 16$ because the complexity of $f_1(\mathbf{x})$ under DE/rand/1 with $F = 0.5$ is $O(D^{2.50})$, which is worse than DE/target/1's $O(D^{2.03})$ complexity at $F = F^*$.

4.4 Randomization's Impact on Computational Efficiency

Setting $F_i = F \cdot n_i(0,1)$ is effective for DE/target/1 because the normal distribution's many small perturbations do not dramatically affect N^*. The normal distribution, however, is inappropriate for DE/rand/1 because it requires inflating population sizes to compensate for the many small values of F that it generates. As the previous subsection showed, these large populations degrade DE/rand/1's efficiency. Consequently, distributions for randomizing global selection models should not generate many values less than 0.5.

For example, Table 2 tabulates the effect that several different distributions have DE/rand/1's performance. Holding F constant is still the most efficient technique. Among randomized strategies, distributing F_i with random uniformity over the range 0.5 and 1.0 achieved a lower complexity than either the normal or lognormal distribution functions because it succeeds with a smaller population. Of course, if randomization does not generate any values less than 0.5, then the average value for F will be greater than 0.5, making DE/rand/1's performance less than optimal (on the quadratics). Compared to when F is held constant, *randomization increases the complexity exponent* of the DE/rand/1$-f_1(\mathbf{x})$ combination. By contrast, randomizing F with the normal and lognormal distributions did not significantly change DE/target/1's complexity exponent. One way to speed-up DE/target/1 while retaining its computational efficiency and its freedom from selection bias is to combine it with DE/rand/1 *via* recombination.

Table 2 DE/rand/1: The effect that randomizing F has on SP^* and N^* ($1 < D \leq 30$).

F	SP^*	N^*
$F = 0.5$	$53.0D^{2.50}$	$4.37D^{1.23}$
$F_i = 0.5 \cdot n_i(0,1)$	$56.8D^{2.89}$	$5.70D^{1.63}$
$F_i = 0.5 \cdot \exp(n_i(0,1) - 0.5)$	$61.2D^{2.98}$	$4.70D^{1.73}$
$F_i = \mathrm{U}(0.5,1.0)$	$33.4D^{2.86}$	$3.04D^{1.18}$

5 Differential Mutation and Line-Recombination

This section looks at an algorithm that bridges the gap between DE/target/1 and DE/rand/1. In the DE/target-to-rand/1 algorithm, both mutation *and* recombination contribute to each new vector's creation. In this and the remaining strategies, the target vector $\mathbf{x}_{i,g}$ competes composite against a *trial vector* $\mathbf{u}_{i,g}$ not just a mutant $\mathbf{v}_{i.g}$.

5.1 DE/target-to-rand/1

In DE/target-to-rand/1, the trial vector $\mathbf{u}_{i,g}$ is a differentially mutated line-recombinant, i.e., a line-recombinant base vector to which a scaled difference vector is added.

$$\mathbf{x}_{i,g} \; vs. \; \mathbf{u}_{i,g} = \mathbf{x}_{i,g} + K \cdot \left(\mathbf{x}_{r0,g} - \mathbf{x}_{i,g}\right) + F \cdot \left(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}\right), \quad r0 \neq r1 \neq r2 \neq i \tag{21}$$

The term "line-recombination" refers to the fact that each value of the *coefficient of recombination K* corresponds to a point on the line defined by $\mathbf{x}_{i,g}$ and $\mathbf{x}_{r0,g}$. The effect of K is to control the *intensity* of the recombination operation.

Setting $K = 0$ yields the DE/target/1algorithm in which recombination plays no role, while $K = 1$ reproduces the DE/rand/1 algorithm. The fact that $K = 1$ reproduces the DE/rand/1 algorithm makes clear that global selection can alternatively be viewed as a special case of line-recombination in the sense that in both operations the target vector competes against a randomly chosen base vector.

In DE algorithms that employ both differential mutation and line-recombination, F and K are often set equal to one another, but other than to minimize the algorithm's tuning effort, there are good reasons why F and K *should not, in general, be equal.* For example, a mutation operation with $F = 1$ (and $K = 0$) generates a cloud of mutants, none of which is likely to already be a population member. By contrast, the cloud of recombinants generated when $K = 1$ (and $F = 0$) is just the rest of the population. Because F and K control operations with different dynamics, they ought to be independently adjustable.

DE/target-to-rand/1 requires one more random vector ($\mathbf{x}_{r0,g}$) than mutation-only schemes to ensure that mutation and recombination are independent operations. Even so, F and K *exhibit a mild co-dependence* in this algorithm. For example, when $K = 0$, F^* is the most effective value for the quadratics. On the other hand, when $K = 1$, the best value for F is approximately 0.5. Thus, the value chosen for

K affects the best value for F. This residual interaction between mutation and recombination is probably inevitable as long as they are performed simultaneously.

5.2 Unbiased Recombinant Distributions

As was the case for the distribution of mutants, the set of possible trial vectors must be center-symmetrically distributed about the target vector; otherwise, there will be an unwarranted residual bias that can cause the population to respond to influences that are artifacts of the generating process. Since $\mathbf{x}_{i,g} + F\cdot(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g})$ and $\mathbf{x}_{i,g} + F\cdot(\mathbf{x}_{r2,g} - \mathbf{x}_{r1,g})$ are equally probable when difference vectors are randomly chosen, F can remain constant without jeopardizing the center-symmetry of the mutant distribution. Because only one difference vector is randomly chosen during (two-vector) line-recombination, *K must be distributed center-symmetrically about zero* to ensure that each recombinant $\mathbf{x}_{i,g} + K\cdot(\mathbf{x}_{r0,g} - \mathbf{x}_{i,g})$ has an equally probable counterpart $\mathbf{x}_{i,g} - K\cdot(\mathbf{x}_{r0,g} - \mathbf{x}_{i,g})$. Thus, if K_i is drawn from a center-symmetric distribution, each mutated recombinant acquires an equally probable counterpart. For example, $\mathbf{x}_{i,g} + K_i\cdot(\mathbf{x}_{r0,g} - \mathbf{x}_{i,g}) + F\cdot(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g})$ and $\mathbf{x}_{i,g} - K_i\cdot(\mathbf{x}_{r0,g} - \mathbf{x}_{i,g}) + F\cdot(\mathbf{x}_{r2,g} - \mathbf{x}_{r1,g})$ become center-symmetric pairs.

5.3 Randomizing the Coefficient of Recombination K

Since K must be distributed to avoid generating a bias, it makes sense to randomize it by sampling values from a center-symmetric distribution. K cannot be the distribution's average value; otherwise the distribution's center would shift from the target vector and induce a bias. Instead, K can control recombination intensity by scaling the standard deviation of a predefined, probability distribution. In this study, the normal distribution generates values for K, but any symmetric distribution centered on zero would suffice.

$$K_i = K \cdot (n_i(0,1)) \tag{22}$$

As was the case when randomizing F, if K is chosen more often than once per vector, the distribution of recombinants becomes rotationally dependent, i.e. rotating the coordinate system alters the distribution of recombinants, thus changing the algorithm's performance. Consequently, dithering K, i.e. sampling a new value once per vector, gives the maximum amount of variety without jeopardizing DE's ability to solve highly conditioned problems with dependent parameters.

5.4 Computational Efficiency

Sampling K_i from a distribution centered on zero implies the local selection model, so it is natural to set $F = F^*$ and ask for what value of K, K^*, does DE/target-to-rand/1 yield its best success performance on $f_1(\mathbf{x})$? In other words, how big can K become before the increase in population size required to maintain reliability begins to slow convergence more than recombination speeds it? Figure 8 (left) provides the answer by plotting *SP* as a function of dimension with $K_i = K\cdot n_i(0,1)$ and with K sampled in increments of 0.0125 over the interval [0.0, 0.15] and in increments of 0.05 over the interval [0.2, 0.4]. For clarity, only a few of these results are plotted.

Although it may be difficult to see, the curves in Fig. 8 (left) that are associated with $K > 0$ start *below* the line for $K = 0$, which represents the mutation-only results for the DE/target/1 algorithm. The curves for N^* plotted in Fig. 8 (right) show a similar behavior. Initially, a small amount of recombination actually decreases N^* and this allows recombination to solve the function faster than mutation alone. A best fit rend line through the best SP at each dimension shows that $SP^* = 83.7D^{1.99}$.

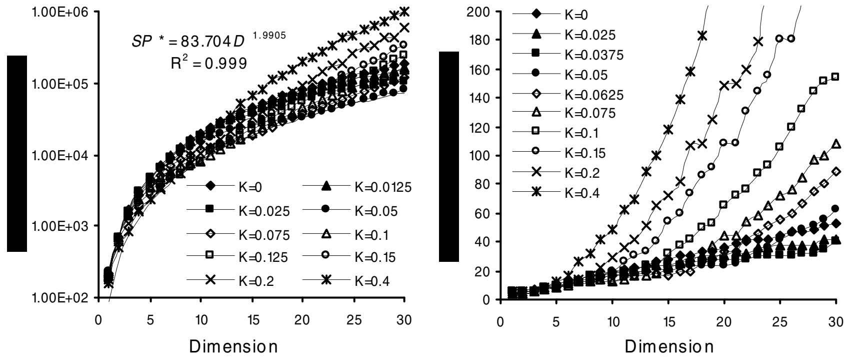

Fig. 8 *SP* and *N* as functions of dimension for DE/target-to-rand/1.

As D increases, any constant value for K eventually becomes too large for computation to be as efficient as mutation alone because population growth becomes nonlinear (Fig. 8 (right)). For DE/target-to-rand/1 to remain efficient, K must decrease as D increases. The data in Fig. 9 show that for $f_1(\mathbf{x})$ with $F = F^*$, the best value for K, i.e. K^*, is roughly equal to $1.3/D$.

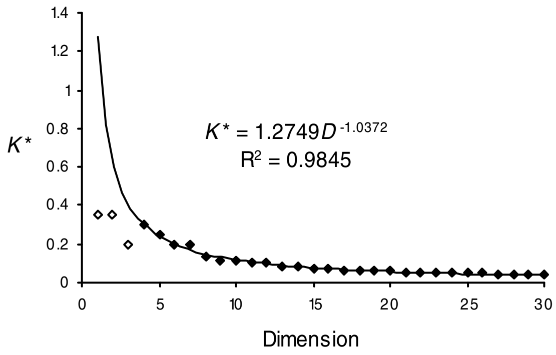

Fig. 9 Ignoring the first three data points as anomalous, K^* is inversely proportional to D.

When run at its optimal settings $F = F^*$ and $K_i = K^* \cdot n_i(0,1)$, DE/target-to-rand/1 solves $f_1(\mathbf{x})$ (and by extension, the elliptical family of functions) about twice as fast as DE/target/1 (Table 3). Table 3 also shows that the performance disparity between the two algorithms grows when F is randomized with a normal distribution, with DE/target-to-rand/1 being roughly three times faster than the comparable randomized DEtarget/1 algorithm. Perhaps surprisingly, sampling F from a lognormal distribution whose average is F^* raised DE/target-to-rand/1's complexity above $O(D^{2.3})$. Since randomizing F with a lognormal distribution did not raise the complexity of the DE/target/1 algorithm, its effect on DE/target-to-rand/1's complexity is likely due to the control parameter interaction between F and K alluded to at the end of section 5.1.

Although the DE/target-to-rand/1 algorithm does not improve on the $O(D^2)$ complexity demonstrated by DE/target/1, it nevertheless improves algorithm speed by lowering the leading factor of the complexity term. These results confirm the traditional view that *recombination does not reduce an algorithm's complexity exponent*, although it can speed execution by a constant factor [30]. The next section looks at an alternative way to drive evolution that interleaves mutation and recombination.

Table 3 DE/target/1 *vs*. DE/target-to-rand/1: Comparing the effect that randomizing F has on SP^* and N^* ($2 < D \le 30$).

F	DE/target/1		DE/target-to-rand/1, $K = K^*$	
	SP^*	N^*	SP^*	N^*
$F = F^* = 1.3/\sqrt{D}$	$180D^{2.03}$	$1.74D + 1.8$	$86.4D^{2.00}$	$1.67D + 1.1$
$F_i = F^* \cdot n_i(0,1)$	$318D^{1.99}$	$2.1D + 5.1$	$99.0D^{2.04}$	$1.63D + 1.3$
$F_i = F^* \cdot \exp(n_i(0,1) - 0.5)$	$343D^{2.01}$	$2.4D + 3.6$	$70.7D^{2.31}$	$4.42D - 4$

6 Differential Mutation or Line Recombination

When mutation and recombination are performed simultaneously, F and K become mildly dependent control parameters. Alternating between mutation and recombination can reduce this dependence.

6.1 DE/target/1/or_line

In DE/target/1/or_line, trial vectors are *either* mutants *or* line-recombinants. The basic "either/or" algorithm in Eq. 19 samples K from a normal distribution, but as before, any center-symmetric distribution would suffice.

$$\mathbf{x}_{i,g} \ vs. \ \mathbf{u}_{i,g} = \mathbf{x}_{i,g} + \begin{cases} K \cdot n_i(0,1) \cdot \left(\mathbf{x}_{r1,g} - \mathbf{x}_{i,g}\right) \text{ if } \mathrm{U}_i(0,1) \le P_\chi \\ F \cdot \left(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}\right) \text{ otherwise} \end{cases} \quad (K = 1) \qquad (23)$$

The control variable P_χ is the probability that the trial vector $\mathbf{u}_{i,g}$ will be a line recombinant. With P_χ as another parameter to tune, this algorithm is more complex than DE/target-to-rand/1. What is worse, K does not independently control recombination, but now shares that role with P_χ. Consequently, there is some *duplication of effort* because there are now two, albeit distinct, control mechanisms for recombination. To avoid having two variables control recombination, this algorithm sets $K = 1$ so that $K_i = n_i(0,1)$, leaving P_χ as the only control variable for recombination.

6.2 Computational Efficiency

To determine how effectively DE/target/1/or_line optimizes $f_1(\mathbf{x})$, both SP and N^* were computed for all combinations of $D = [1, 2.., 30]$ and P_χ, where P_χ was sampled from the range [0.0, 0.1] in increments of 0.01, from [0.125, 0.3] in increments of 0.025 and from [0.35, 1.0] in increments of 0.05. Figure 10 (left) plots some of these results, showing that DE/target/1/line performs very similarly to DE/target-to-rand/1. Initially, increasing P_χ lowers both N and SP compared to mutation alone ($P_\chi = 0$). Fitting a power-law trend-line through the best SP at each dimension gives $SP^* = 92.4D^{2.025}$. This is very close to the equivalent result for DE/target-to-rand/1, which was $SP^* = 86.4D^{2.00}$. The improved speed over mutation alone once again confirms the traditional view that recombination is a mechanism to speed convergence that nevertheless does not improve the complexity exponent.

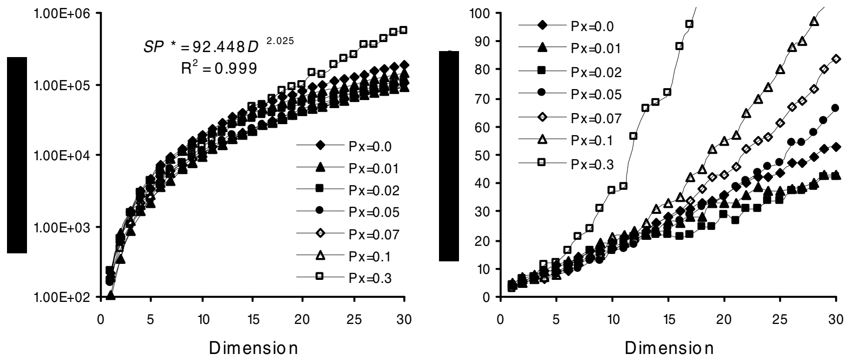

Fig. 10 SP and N^* for the $f_1(\mathbf{x})$–DE/target/1/or_line combination. SP^* in Fig. 10 (left) is derived from a power-law trend line through the best SP at each dimension.

Figure 10 (right) shows that if P_χ is held constant as D increases, then N^* eventually grows nonlinearly, which in turn impacts DE/target/or_line's computational efficiency. For DE/target/1/or_line to achieve its best possible performance on $f_1(\mathbf{x})$, P_χ must decrease as D increases and the data in Fig. 11 suggest that $P_\chi^* = 1/D$ is very near optimal when $F = F^*$ and $K = n_i(0,1)$. Perhaps surprisingly, this is the same functional dependence on D that K^* displayed in DE/target-to-rand/1.

Table 4 shows that when DE/target/1/or_line runs at its optimal setting ($F = F^*$, $P_\chi^* = 1/D$), population growth is linear and the success performance complexity at $SP^* = 106.2D^{1.99}$ is very close to the best performance $SP^* = 83.7D^{1.99}$ previously mentioned.

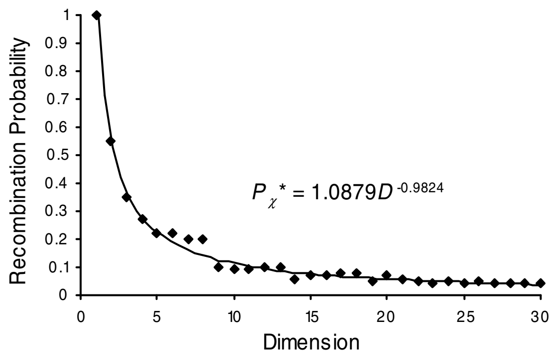

Fig. 11 This plot of the value of P that resulted in the lowest SP at the given dimension shows that $P = 1/D$ is very nearly optimal.

Table 4 DE/target-to-rand/1 *vs*. DE/target/1/or_line: Comparing the effect that randomizing F has on SP^* and N^* ($2 < D \leq 30$).

F	DE/target-to-rand/1, $K = K^*$		DE/target/1/or_line, $P = P^*$	
	SP^*	N^*	SP^*	N^*
$F = F^*$	$86.4D^{2.00}$	$1.67D + 1.1$	$106.2D^{1.99}$	$1.68D + 2.1$
$F_i = F^* \cdot n_i(0,1)$	$99.0D^{2.04}$	$1.63D + 1.3$	$99.2D^{2.09}$	$2.01D + 1.3$
$F_i = F^* \cdot \exp(n_i(0,1) - 0.5)$	$70.7D^{2.31}$	$4.42D - 4$	$106.7D^{2.09}$	$2.14D + 0.8$

Table 4 also shows how randomizing F affects DE/target/1/or_line's performance. For an easier comparison with DE/target-to-rand/1, Table 4 duplicates data from Table 3. Both algorithms perform similarly when F is either held constant or sampled from a normal distribution, but results diverge once F is sampled from a lognormal distribution. When compared to DE/target-to-rand/1, the complexity of the lognormal result for DE/target/1/or_line is significantly less, being the same as that for the normal distribution. The higher complexity of DE/target-to-rand/1 when F is lognormally distributed is likely due to control parameter dependence between F and K, so it seems fair to assume that the lower complexity posted by DE/target/1/or_line with the same distribution indicates that there is *less control parameter dependence* between F and P_χ than there is between F and K in DE/target-to-rand/1.

7 Differential Mutation and Discrete Recombination

Classic DE refers to the algorithm DE/rand/1/bin. Classic DE mutates a randomly selected the base vector with a single, randomly chosen vector difference. The term "bin" indicates that classic DE recombines vectors via *binomial crossover*, so-called because the number of parameters inherited by the trial vector is binomially distributed. This form of recombination is perhaps more commonly known as *uniform discrete crossover.*

7.1 DE/rand/1/bin: Classic DE

The previous DE algorithms in this chapter rely on vector-level operations, but classic DE's discrete crossover draws *each trial vector parameter* from *either* the target vector $\mathbf{x}_{i,g}$, *or* a randomly chosen base vector $\mathbf{x}_{r1,g}$ that has been mutated by the addition of a scaled random vector difference (Eq. 24).

$$\mathbf{v}_{i,g} = \mathbf{x}_{r0,g} + F \cdot \left(\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}\right)$$

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } \mathrm{U}(0,1) \le Cr \text{ or } j = j_{\text{rand}} \\ x_{j,i,g} & \text{otherwise} \end{cases} \tag{24}$$

$$r0 \ne r1 \ne r2 \ne i, \quad j_{\text{rand}} \in \{1,2...,D\}, \quad Cr \in [0,1]$$

To ensure that $\mathbf{v}_{i,g}$ is a mutant and not a line-recombinant, $r0$, $r1$ and $r2$ must be distinctly different indices.

During crossover, the trial vector $\mathbf{u}_{i,g}$ always inherits one randomly chosen parameter $v_{j\text{rand},i,g}$ from the random mutant $\mathbf{v}_{i,g}$ so that it will differ by at least one parameter value from its adversary, the target vector $\mathbf{x}_{i,g}$. For the remaining $D - 1$ parameters, the *crossover control variable Cr* sets the probability that the trial vector inherits a parameter from the random mutant. In classic DE, the target and mutant vectors occupy opposing corners of a D-dimensional hypercube whose remaining vertices locate other possible trial vectors.

7.2 Computational Efficiency

In DE/rand/1bin, Cr alone controls recombination. When $Cr = 1$, DE/rand/1/bin reduces to DE/rand/1 because the trial vector draws all of its parameters from a random mutant. In this limiting case, the optimal value for F, computational complexity of SP^* and growth rate for N^* are the same as they are for DE/rand/1. Furthermore, classic DE also inherits DE/rand/1's biases, dependencies and response to randomizing F.

To illustrate the role of Cr in optimization, SP and N^* were computed for every combination of $D = [1, 2..., 30]$ and $Cr = [0.0, 0.1..., 1.0]$. Since classic DE resembles DE/rand/1 in the limiting case $Cr = 1.0$, F was set to that algorithm's optimal value, i.e. $F = 0.5$. The results, only some of which are plotted in Fig. 12, show that the smallest values for Cr are by far the most effective on $f_1(\mathbf{x})$. Although $Cr = 0.0$ was the best setting over the full range of dimensions, any value of Cr from the range [0.0, 0.5] was nearly as effective. Once $Cr \geq 0.7$, however, the computational complexity begins to grow quickly, up to the limit established by DE/rand/1: $SP^* = 53.0D^{2.5}$. At $Cr = 0.0$, however, SP^* complexity drops to just $92.1D^{1.36}$ and as long as $Cr \leq 0.9$, N^* barely grows at all with dimension, though it is still dependent on Cr.

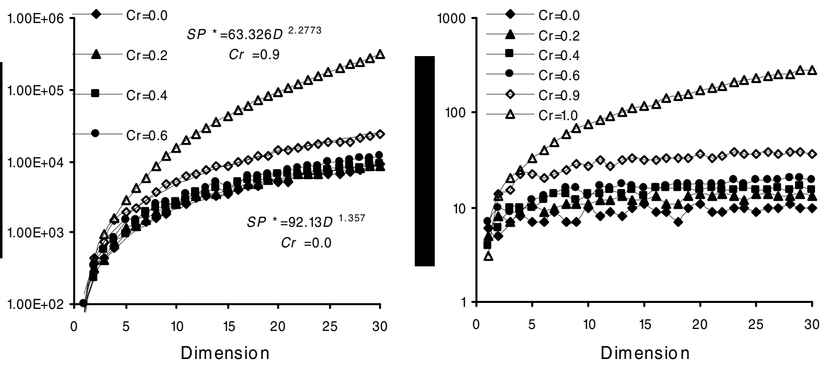

Fig. 12 *SP* and N^* as functions of both Cr and dimension

Discrete recombination dramatically enhances computational efficiency by exploiting $f_1(\mathbf{x})$'s decomposability. As first mentioned in section 3.2.3, the parameters of decomposable functions can be independently optimized one at a time. Unlike the vector-based operations described in sections 2–6, discrete recombination with low values of Cr is an effective way for DE to exploit a function's decomposability because it only changes a few parameters per function evaluation.

Of course if a function is decomposable, then it should be possible to find the solution in linear $O(D)$ time, which is better than the $O(D^{1.36})$ time for DE's low-Cr strategy. Most functions of interest, however, are not decomposable and for those problems, the low-Cr strategy becomes a liability [21]. A comparison between $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$ illustrates this point.

Figure 13 plots *SP* as a function of Cr = [0.0, 0.1..., 1.0] for the ten-dimensional versions of $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$. The data confirm that the most effective way to optimize $f_2(\mathbf{x})$'s independent parameters is with $Cr = 0.0$, i.e. by changing just one parameter value per function evaluation. The data also confirm the futility of trying to optimize $f_3(\mathbf{x})$'s dependent parameters with the same setting. Compared to $Cr = 0.0$, even a small increase in Cr gives significantly better performance on $f_3(\mathbf{x})$ because then it becomes possible to change more than one parameter per function evaluation. Once $Cr = 1.0$, classic DE degenerates into the rotationally invariant DE/rand/1 algorithm, which optimizes both $f_2(\mathbf{x})$ and $f_1(\mathbf{x})$ with nearly the same effort, thus providing further evidence that rotationally invariant strategies do not distinguish between these two functions.

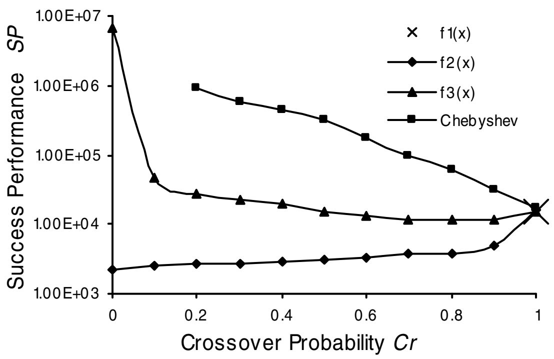

Fig. 13 Low-*Cr* strategies are only effective on decomposable functions. At *Cr* = 1, classic DE optimizes the ten-dimensional versions of $f_1(\mathbf{x})$, $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$ and the nine-dimensional Chebyshev polynomial fitting problem in about the same number of function evaluations.

Figure 13 also plots *SP* for the nine-dimensional Chebyshev polynomial fitting problem (see [4] for details). This normally very difficult, highly conditioned, parameter-dependent multimodal problem more closely resembles a real-world task than do either $f_2(\mathbf{x})$ or $f_3(\mathbf{x})$. Although almost intractable when $Cr < 0.2$, once $Cr = 1.0$, this problem is as easy as $f_1(\mathbf{x})$, $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$. Because interesting problems are seldom decomposable, $Cr = 0.9$ is classic DE's recommended default setting.

8 Conclusion

This chapter began by comparing DE/target/1 to the classic random walk to show that DE's performance on elliptical functions is invariant to their scale, eccentricity and orientation. After identifying DE's ability to exploit contour matching as the reason for its invariant performance, experiments confirmed the optimal values for *F* and *N* first reported in [29] for both DE/target/1 and DE/rand/1. Subsequent sections introduced three additional strategies, each of which includes a recombination operator. The next section tabulates the optimal control parameter settings for all five strategies and provides some guidance on which settings are likely to be most effective for multimodal optimization. The subsequent section summarizes algorithm performance in light of the criteria set forth in the Introduction.

8.1 Optimal Control Parameter Values

The values listed in Table 5, while effective for local optimization, are only a starting point when deciding control parameter values for multimodal problems. For example, DE should not be expected to optimize multimodal functions with populations smaller than those in Table 5. Similarly, for DE/target/1, DE/target-to-rand/1 and DE/target/1/or_line, an effective value for *F* will most likely lie in the interval [*F**, 1.0], with values from the upper end of this range often proving to be the most effective on multimodal problems. For DE/rand/1 and DE/rand/1/bin, F will typically lie between 0.5 and 1.0. For the line-recombination control parameters K and P_χ, this author has found $K = 2.6/N$ and $P_\chi = 2/N$ to be better starting points for multimodal optimization than $1.3/D$ and $1/D$, respectively. For classic DE, $Cr = 0.0$ is the best choice if the objective function is decomposable, but $Cr = 0.9$ is a better choice for parameter-dependent functions.

Table 5 Optimal control parameter settings for convex, quadratic functions. K multiplies a sample from a normal distribution, i.e. $K_i = K \cdot n_i(0,1)$. Values in parentheses were held constant while both N and the remaining control variable were optimized.

Algorithm	Approximate Optimal Control Parameter Values				
	F	N	K	P_χ	Cr
DE/target/1	$1.3/\sqrt{D}$	$1.7D$	–	–	–
DE/rand/1	0.5	$4.4D^{1.23}$	–	–	–
DE/target-to-rand/1	$(1.3/\sqrt{D})$	$1.7D$	$1.3/D$	–	–
DE/target/1/or_line	$(1.3/\sqrt{D})$	$1.7D$	(1.0)	$1/D$	–
DE/rand/1/bin	(0.5)	10	–	–	0.0 $f_1(\mathbf{x}), f_2(\mathbf{x})$
	(0.5)	40 ($D > 15$)	–	–	0.9 $f_3(\mathbf{x})$

8.2 Strategy Evaluation

Of the algorithms that do not exploit functional decomposability, DE/target/1, DE/target-to-rand/1 and DE/target/1/or_line were the most efficient strategies. Because their line-recombination operators are bias-free, DE/target/1/or_line and DE/target/1/or_line were faster than DE/trarget/1 by constant factors. Furthermore, all three of these strategies slowed, but otherwise retained their good efficiency when F was sampled from a normal distribution. Randomizing F with a lognormal distribution indicated that F and K in DE/target-to-rand/1 interact more strongly than do F and P_χ in DE/target/1/or_line. Consequently, DE/target/1/or_line's low level of control parameter interaction probably makes it easier to "tune" and better suited for adaptive algorithms like those in [31–33] because its *control* parameter values can be independently optimized.

The three algorithms mentioned above performed well, in part, because their design is such that F and N are only weakly dependent. By contrast, selection bias and the strong dependence of N on F exhibited by both classic DE (at $Cr = 0.9$) and DE/rand/1 raised their complexities significantly above $O(D^2)$, which was further degraded when F was randomized.

While the success of DE/rand/1/bin is well documented, DE/target/1/or_line performed better on the parameter-dependent elliptical function, was free of drift bias, responded well to randomizing and exhibited minimal control parameter interaction, all while retaining performance that is invariant with respect to scale, eccentricity and, unlike classic DE, objective function orientation.

Acknowledgments. The author wishes to thank The Handbook of Optimization's editors, Prof. Ivan Zelinka, Prof. Vaclav Snasel and Dr. Ajith Abraham. Gratitude is also extended to Dr. Donald Davendra.

References

[1] Storn, R., Price, K.V.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359 (1997)

[2] Price, K.V., Storn, R.: Differential evolution. Dr. Dobb's Journal 264, 18–24 (1997)

[3] Price, K.V., Storn, R., Lampinen, J.: Differential Evolution – A Practical Approach to Global Optimization. Springer, Heidelberg (2001)

[4] Storn, R.: Differential evolution web site, `http://www.icsi.berkeley.edu/~storn`

[5] Qing, A.: Differential Evolution: Fundamentals and Applications in Electrical Engineering. John Wiley and Sons, Singapore (2009)

[6] Feoktistov, V.: Differential evolution. In: Search of Solutions. Springer Science + Business Media, LLC (2006)

[7] Zhang, J., Sanderson, A.: Adaptive Differential Evolution: A Robust Approach to Multimodal Problem Optimization. Springer, Heidelberg (2009)

[8] Price, K.V., et al.: Differential evolution, Part Two. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 79–158. McGraw-Hill, Berkshire (1999)

[9] Das, S., Suganthan, P.N., Coello Coello, C.A.: Special issue on differential evolution. IEEE Transactions on Evolutionary Computation 15(1) (2011)

[10] Rönkkönen, J.: Continuous multimodal global optimization with differential evolution-based methods. Doctoral Thesis, Lappeenranta University of Technology, Lappeenranta, Finland (2009)

[11] Onwubolu, G., Davendra, D.: Differential Evolution: A Handbook for Global Permutation-based Combinatorial Optimization. Springer, Heidelberg (2009)

[12] Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man and Cybernetics: Part B 26(1), 29–41 (1996)

[13] Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department (2005)

[14] Dasgupta, D., Attoh-Okine, N.: Immunity-based systems: a survey. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Orlando, Florida, October 12-15, vol. 1, pp. 369–374 (1997)

[15] Eberhart, R.C., Shi, Y.: Special issue on particle swarm optimization. IEEE Transactions on Evolutionary Computation 8(3) (2004)

[16] Das, S., Suganthan, P.N.: Differential evolution: a survey of the state-of-the-art. IEEE Transactions on Evolutionary Computation 15(1), 4–31 (2011)

[17] Neri, F., Tirronen, V.: Recent advances in differential evolution: a survey and experimental analysis. Artif. Intell. Rev. 33(1-2), 61–106 (2010)

[18] Zeilinski, K., Laur, R.: Stopping criteria for differential evolution in constrained single-objective optimization. In: Chakraborty, U. (ed.) Advances in Differential Evolution, pp. 111–138. Springer, Heidelberg (2008)

[19] Wang, H., Rahnamayan, S., Wu, Z.: Adaptive differential evolution with variable population size for solving high dimensional problems. In: Proceedings of the 2011 IEEE Congress on Evolutionary Computation, New Orleans, June 5-8, pp. 2626–2632 (2011)

[20] Zhang, C., Chen, J., Xin, B., Cai, T., Chen, C.: Differential evolution with adaptive population size combining lifetime and extinction mechanisms. In: Proceedings of the Eight Asian Control Conference, Kaohsiung, May 15-18, pp. 1221–1226 (2011)

[21] Salomon, R.: Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions: a survey of some theoretical and practical aspects of genetic algorithms. Biosystems 39(3), 263–278 (1996)

[22] Price, K.V.: Eliminating drift bias from the differential evolution algorithm. In: Chakraborty, U. (ed.) Advances in Differential Evolution, pp. 33–88. Springer, Heidelberg (2008)

[23] Lampinen, J., Zelinka, I.: On stagnation of the differential evolution algorithm. In: Proceedings of the Sixth International Mendel Conference on Soft Computing, pp. 76–83 (2000)

[24] Abbass, H.: The self-adaptive Pareto differential evolution algorithm. In: Proceedings of the Congress on Evolutionary Computation, vol. 1, pp. 831–836 (2002)

[25] Thangraj, R., Pant, M., Abraham, A., Deep, K.: Differential evolution using a localized Cauchy mutation operator. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Istanbul, October 10-13, pp. 3710–3716 (2010)

[26] Fu, X., Yu, J.: A hybrid algorithm based on extremal optimization with adaptive Levy mutation and differential evolution and application. In: Proceedings of the Fifth International Conference on Natural Computation, Tianjian, China, August 14-16, vol. 1, pp. 12–16 (2009)

[27] Pant, M., Thangaraj, R., Abraham, A., Grosan, C.: Differential evolution with Laplace mutation operator. In: Proceedings of the 2009 IEEE Congress on Evolutionary Computation, Trondheim, May 18-21, pp. 2841–2849 (2009)

[28] Liu, G., Li, Y., Nie, X., Sun, Y.: Improving clustering-based differential evolution with chaotic sequences and new mutation operator. International Journal of Advancements in Computing Technology 3(6), 276–286 (2011)

[29] Price, K.V., Rönkkönen, J.: Comparing the unimodal scaling performance of global and local selection in a mutation-only algorithm. In: Proceedings of the 2006 World Congress on Computational; Intelligence, Vancouver, July 16-21, pp. 7387–7394 (2006)

[30] Mühlenbein, H., Schlierkamp-Voosen, D.: Predictive models for the breeder genetic algorithm I. Evolutionary Computation 1(1), 25–50 (1993)

[31] Zhang, J., Sanderson, A.C.: JADE: adaptive differential evolution with optional external archive. IEEE Transactions on Evolutionary Computation 13(5), 945–958 (2009)

[32] Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Transaction on Evolutionary Computation 13(2), 398–417 (2009)

[33] Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10(6), 646–657 (2006)