Chapter 13

Differential Evolution

Differential evolution (DE) is a stochastic, population-based search strategy developed by Storn and Price [696, 813] in 1995. While DE shares similarities with other evolutionary algorithms (EA), it differs significantly in the sense that distance and direction information from the current population is used to guide the search process. Furthermore, the original DE strategies were developed to be applied to continuous-valued landscapes.

This chapter provides an overview of DE, organized as follows: Section 13.1 discusses the most basic DE strategy and illustrates the method of adaptation. Alternative DE strategies are described in Sections 13.2 and 13.3. Section 13.4 shows how the original DE can be applied to discrete-valued and binary-valued landscapes. A number of advanced topics are covered in Section 13.5, including multi-objective optimization (MOO), constraint handling, and dynamic environments. Some applications of DE are summarized in Section 13.6.

13.1 Basic Differential Evolution

For the EAs covered in the previous chapters, variation from one generation to the next is achieved by applying crossover and/or mutation operators. If both these operators are used, crossover is usually applied first, after which the generated offspring are mutated. For these algorithms, mutation step sizes are sampled from some probability distribution function. DE differs from these evolutionary algorithms in that

- mutation is applied first to generate a trial vector, which is then used within the crossover operator to produce one offspring, and
- mutation step sizes are not sampled from a prior known probability distribution function.

In DE, mutation step sizes are influenced by differences between individuals of the current population.

Section 13.1.1 discusses the concept of difference vectors, used to determine mutation step sizes. The mutation, crossover, and selection operators are described in Sections 13.1.2 to 13.1.4. Section 13.1.5 summarizes the DE algorithm, and control parameters are discussed in Section 13.1.6. A geometric illustration of the DE variation approach is given in Section 13.1.7.

13.1.1 Difference Vectors

The positions of individuals provide valuable information about the fitness landscape. Provided that a good uniform random initialization method is used to construct the initial population, the initial individuals will provide a good representation of the entire search space, with relatively large distances between individuals. Over time, as the search progresses, the distances between individuals become smaller, with all individuals converging to the same solution. Keep in mind that the magnitude of the initial distances between individuals is influenced by the size of the population. The more individuals in a population, the smaller the magnitude of the distances.

Distances between individuals are a very good indication of the diversity of the current population, and of the order of magnitude of the step sizes that should be taken in order for the population to contract to one point. If there are large distances between individuals, it stands to reason that individuals should make large step sizes in order to explore as much of the search space as possible. On the other hand, if the distances between individuals are small, step sizes should be small to exploit local areas. It is this behaviour that is achieved by DE in calculating mutation step sizes as weighted differences between randomly selected individuals. The first step of mutation is therefore to first calculate one or more difference vectors, and then to use these difference vectors to determine the magnitude and direction of step sizes.

Using vector differentials to achieve variation has a number of advantages. Firstly, information about the fitness landscape, as represented by the current population, is used to direct the search. Secondly, due to the central limit theorem [177], mutation step sizes approaches a Gaussian (Normal) distribution, provided that the population is sufficiently large to allow for a good number of difference vectors [811].[1] The mean of the distribution formed by the difference vectors are always zero, provided that individuals used to calculate difference vectors are selected uniformly from the population [695, 164]. Under the condition that individuals are uniformly selected, this characteristic follows from the fact that difference vectors $(\mathbf{x}_{i_1} - \mathbf{x}_{i_2})$ and $(\mathbf{x}_{i_2} - \mathbf{x}_{i_1})$ occur with equal frequency, where $\mathbf{x}_{i_1}$ and $\mathbf{x}_{i_2}$ are two randomly selected individuals. The zero mean of the resulting step sizes ensures that the population will not suffer from genetic drift. It should also be noted that the deviation of this distribution is determined by the magnitude of the difference vectors. Eventually, differentials will become infinitesimal, resulting in very small mutations.

Section 13.2 shows that more than one differential can be used to determine the mutation step size. If n_v is the number of differentials used, and n_s is the population size, then the total number of differential perturbations is given by [429]

$$\binom{n_s}{2n_v} 2n_v! \approx \mathcal{O}(n_s^{2n_v}) \tag{13.1}$$

[1]The central limit theorem states that the probability distribution governing a random variable approaches the Normal distribution as the number of samples of that random variable tends to infinity.

Equation (13.1) expresses the total number of directions that can be explored per generation. To increase the exploration power of DE, the number of directions can be increased by increasing the population size and/or the number of differentials used.

At this point it is important to emphasize that the original DE was developed for searching through continuous-valued landscapes. The sections that follow will show that exploration of the search space is achieved using vector algebra, applied to the individuals of the current population.

13.1.2 Mutation

The DE mutation operator produces a trial vector for each individual of the current population by mutating a target vector with a weighted differential. This trial vector will then be used by the crossover operator to produce offspring. For each parent, $\mathbf{x}_i(t)$, generate the trial vector, $\mathbf{u}_i(t)$, as follows: Select a target vector, $\mathbf{x}_{i_1}(t)$, from the population, such that $i \neq i_1$. Then, randomly select two individuals, $\mathbf{x}_{i_2}$ and $\mathbf{x}_{i_3}$, from the population such that $i \neq i_1 \neq i_2 \neq i_3$ and $i_2, i_3 \sim U(1, n_s)$. Using these individuals, the trial vector is calculated by perturbing the target vector as follows:

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)) \tag{13.2}$$

where $\beta \in (0, \infty)$ is the scale factor, controlling the amplication of the differential variation.

Different approaches can be used to select the target vector and to calculate differentials as discussed in Section 13.2.

13.1.3 Crossover

The DE crossover operator implements a discrete recombination of the trial vector, $\mathbf{u}_i(t)$, and the parent vector, $\mathbf{x}_i(t)$, to produce offspring, $\mathbf{x}_i^{'}(t)$. Crossover is implemented as follows:

$$x_{ij}^{'}(t) = \begin{cases} u_{ij}(t) & \text{if } j \in \mathcal{J} \\ x_{ij}(t) & \text{otherwise} \end{cases} \tag{13.3}$$

where $x_{ij}(t)$ refers to the j-th element of the vector $\mathbf{x}_i(t)$, and $\mathcal{J}$ is the set of element indices that will undergo perturbation (or in other words, the set of crossover points). Different methods can be used to determine the set, $\mathcal{J}$, of which the following two approaches are the most frequently used [811, 813]:

- **Binomial crossover:** The crossover points are randomly selected from the set of possible crossover points, $\{1, 2, \ldots, n_x\}$, where n_x is the problem dimension. Algorithm 13.1 summarizes this process. In this algorithm, p_r is the probability that the considered crossover point will be included. The larger the value of p_r, the more crossover points will be selected compared to a smaller value. This means that more elements of the trial vector will be used to produce the offspring, and less of the parent vector. Because a probabilistic decision is made as to the inclusion of a crossover point, it may happen that no points may be selected, in which case the offspring will simply be the original parent, $\mathbf{x}_i(t)$. This problem becomes more evident for low dimensional search spaces. To enforce that at least one element of the offspring differs from the parent, the set of crossover points, $\mathcal{J}$, is initialized to include a randomly selected point, j^*.

- **Exponential crossover:** From a randomly selected index, the exponential crossover operator selects a sequence of adjacent crossover points, treating the list of potential crossover points as a circular array. The pseudocode in Algorithm 13.2 shows that at least one crossover point is selected, and from this index, selects the next until $U(0,1) \geq p_r$ or $|\mathcal{J}| = n_x$.

Algorithm 13.1 Differential Evolution Binomial Crossover for Selecting Crossover Points

```
$j^* \sim U(1, n_x)$;
$\mathcal{J} \leftarrow \mathcal{J} \cup \{j^*\}$;
for each $j \in \{1, \ldots, n_x\}$ do
    if $U(0,1) < p_r$ and $j \neq j^*$ then
        $\mathcal{J} \leftarrow \mathcal{J} \cup \{j\}$;
    end
end
```

Algorithm 13.2 Differential Evolution Exponential Crossover for Selecting Crossover Points

```
$\mathcal{J} \leftarrow \{\}$;
$j \sim U(0, n_x - 1)$;
repeat
    $\mathcal{J} \leftarrow \mathcal{J} \cup \{j + 1\}$;
    $j = (j + 1) \bmod n_x$;
until $U(0,1) \geq p_r$ or $|\mathcal{J}| = n_x$;
```

13.1.4 Selection

Selection is applied to determine which individuals will take part in the mutation operation to produce a trial vector, and to determine which of the parent or the offspring will survive to the next generation. With reference to the mutation operator, a number of selection methods have been used. Random selection is usually used to select the individuals from which difference vectors are calculated. For most DE implementations the target vector is either randomly selected or the best individual is selected (refer to Section 13.2).

To construct the population for the next generation, deterministic selection is used: the offspring replaces the parent if the fitness of the offspring is better than its parent; otherwise the parent survives to the next generation. This ensures that the average fitness of the population does not deteriorate.

13.1.5 General Differential Evolution Algorithm

Algorithm 13.3 provides a generic implementation of the basic DE strategies. Initialization of the population is done by selecting random values for the elements of each individual from the bounds defined for the problem being solved. That is, for each individual, $\mathbf{x}_i(t)$, $x_{ij}(t) \sim U(x_{min,j}, x_{max,j})$, where $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ define the search boundaries.

Any of the stopping conditions given in Section 8.7 can be used to terminate the algorithm.

Algorithm 13.3 General Differential Evolution Algorithm

```
Set the generation counter, $t = 0$;
Initialize the control parameters, $\beta$ and $p_r$;
Create and initialize the population, $\mathcal{C}(0)$, of $n_s$ individuals;
while stopping condition(s) not true do
    for each individual, $\mathbf{x}_i(t) \in \mathcal{C}(t)$ do
        Evaluate the fitness, $f(\mathbf{x}_i(t))$;
        Create the trial vector, $\mathbf{u}_i(t)$ by applying the mutation operator;
        Create an offspring, $\mathbf{x}_i^{'}(t)$, by applying the crossover operator;
        if $f(\mathbf{x}_i^{'}(t))$ is better than $f(\mathbf{x}_i(t))$ then
            Add $\mathbf{x}_i^{'}(t)$ to $\mathcal{C}(t+1)$;
        end
        else
            Add $\mathbf{x}_i(t)$ to $\mathcal{C}(t+1)$;
        end
    end
end
Return the individual with the best fitness as the solution;
```

13.1.6 Control Parameters

In addition to the population size, n_s, the performance of DE is influenced by two control parameters, the scale factor, β, and the probability of recombination, p_r. The effects of these parameters are discussed below:

- **Population size:** As indicated in equation (13.1), the size of the population has a direct influence on the exploration ability of DE algorithms. The more individuals there are in the population, the more differential vectors are available, and the more directions can be explored. However, it should be kept in mind that the computational complexity per generation increases with the size of the population. Empirical studies provide the guideline that $n_s \approx 10n_x$. The nature of the mutation process does, however, provide a lower bound on the number of individuals as $n_s > 2n_v + 1$, where n_v is the number of differentials used. For n_v differentials, $2n_v$ different individuals are required, 2 for each differential. The additional individual represents the target vector.

- **Scaling factor:** The scaling factor, $\beta \in (0, \infty)$, controls the amplification of the differential variations, $(\mathbf{x}_{i_2} - \mathbf{x}_{i_3})$. The smaller the value of β, the smaller the mutation step sizes, and the longer it will be for the algorithm to converge. Larger values for β facilitate exploration, but may cause the algorithm to overshoot good optima. The value of β should be small enough to allow differentials to explore tight valleys, and large enough to maintain diversity. As the population size increases, the scaling factor should decrease. As explained in Section 13.1.1, the more individuals in the population, the smaller the magnitude of the difference vectors, and the closer individuals will be to one another. Therefore, smaller step sizes can be used to explore local areas. More individuals reduce the need for large mutation step sizes. Empirical results suggest that large values for both n_s and β often result in premature convergence [429, 124], and that $\beta = 0.5$ generally provides good performance [813, 164, 19].
- **Recombination probability:** The probability of recombination, p_r, has a direct influence on the diversity of DE. This parameter controls the number of elements of the parent, $\mathbf{x}_i(t)$, that will change. The higher the probability of recombination, the more variation is introduced in the new population, thereby increasing diversity and increasing exploration. Increasing p_r often results in faster convergence, while decreasing p_r increases search robustness [429, 164].

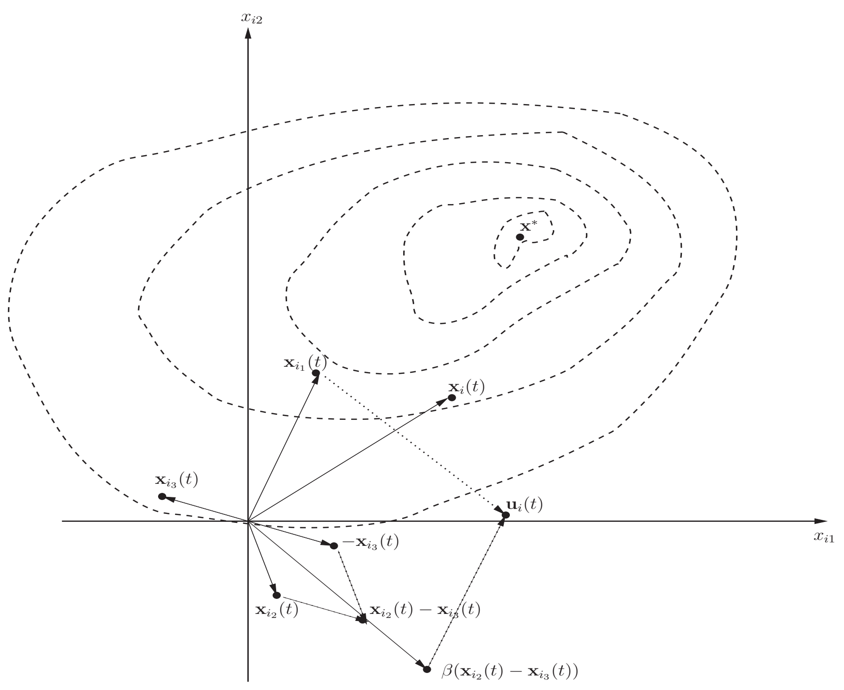
Most implementations of DE strategies keep the control parameters constant. Although empirical results have shown that DE convergence is relatively insensitive to different values of these parameters, performance (in terms of accuracy, robustnes, and speed) can be improved by finding the best values for control parameters for each new problem. Finding optimal parameter values can be a time consuming exercise, and for this reason, self-adaptive DE strategies have been developed. These methods are discussed in Section 13.3.3.

13.1.7 Geometrical Illustration

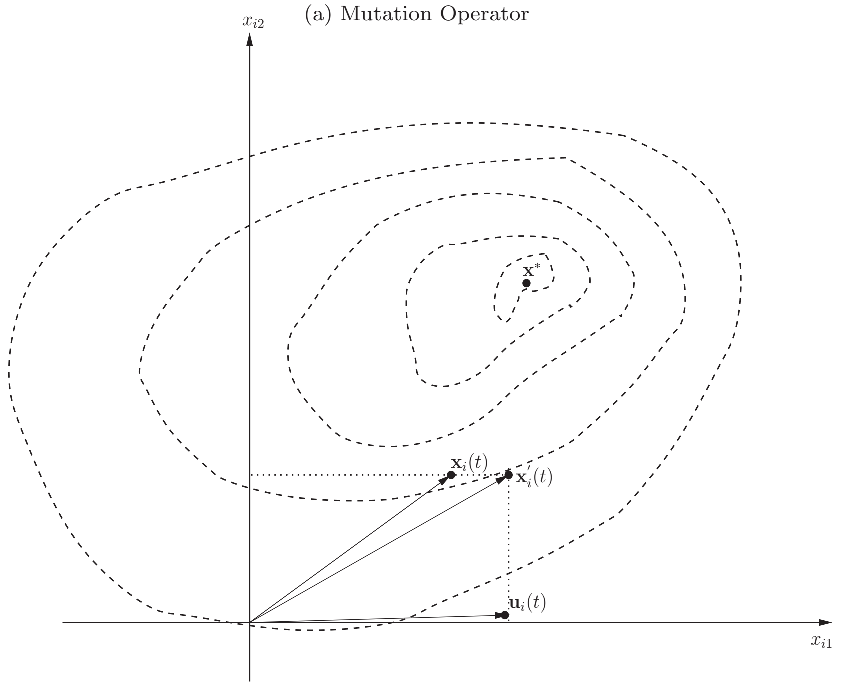
Figure 13.1(a) illustrates the mutation operator of the DE as described in Section 13.1.2. The optimum is indicated by $\mathbf{x}^*$, and it is assumed that $\beta = 1.5$. The crossover operator is illustrated in Figure 13.1(b). For this illustration the offspring consists of the first element of the trial vector, $\mathbf{u}_i(t)$, and the second element of the parent, $\mathbf{x}_i(t)$.

13.2 DE/x/y/z

A number of variations to the basic DE as discussed in Section 13.1 have been developed. The different DE strategies differ in the way that the target vector is selected, the number of difference vectors used, and the way that crossover points are determined. In order to characterize these variations, a general notation was adopted in the DE literature, namely DE/x/y/z [811, 813]. Using this notation, x refers to the



(a) Mutation Operator



(b) Crossover Operator

Figure 13.1 Differential Evolution Mutation and Crossover Illustrated

method of selecting the target vector, y indicates the number of difference vectors used, and z indicates the crossover method used. The DE strategy discussed in Section 13.1 is referred to as DE/rand/1/bin for binomial crossover, and DE/rand/1/exp for exponential crossover. Other basic DE strategies include [429, 811, 813]:

- **DE/best/1/z:** For this strategy, the target vector is selected as the best individual, $\hat{\mathbf{x}}(t)$, from the current population. In this case, the trial vector is calculated as

  $$\mathbf{u}_i(t) = \hat{\mathbf{x}}(t) + \beta(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)) \tag{13.4}$$

  Any of the crossover methods can be used.

- **DE/x/n_v/z:** For this strategy, more than one difference vector is used. The trial vector is calculated as

  $$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta\sum_{k=1}^{n_v}(\mathbf{x}_{i_2,k}(t) - \mathbf{x}_{i_3,k}(t)) \tag{13.5}$$

  where $\mathbf{x}_{i_2,k}(t) - \mathbf{x}_{i_3,k}(t)$ indicates the k-th difference vector, $\mathbf{x}_{i_1}(t)$ can be selected using any suitable method for selecting the target vector, and any of the crossover methods can be used. With reference to equation (13.1), the larger the value of n_v, the more directions can be explored per generation.

- **DE/rand-to-best/n_v/z:** This strategy combines the *rand* and *best* strategies to calculate the trial vector as follows:

  $$\mathbf{u}_i(t) = \gamma\hat{\mathbf{x}}(t) + (1-\gamma)\mathbf{x}_{i_1}(t) + \beta\sum_{k=1}^{n_v}(\mathbf{x}_{i_2,k}(t) - \mathbf{x}_{i_3,k}(t)) \tag{13.6}$$

  where $\mathbf{x}_{i_1}(t)$ is randomly selected, and $\gamma \in [0, 1]$ controls the greediness of the mutation operator. The closer γ is to 1, the more greedy the search process becomes. In other words, γ close to 1 favors exploitation while a value close to 0 favors exploration. A good strategy will be to use an adaptive γ, with $\gamma(0) = 0$. The value of $\gamma(t)$ increases with each new generation towards the value 1.

  Note that if $\gamma = 0$, the DE/rand/y/z strategies are obtained, while $\gamma = 1$ gives the DE/best/y/z strategies.

- **DE/current-to-best/$1{+}n_v$/z:** With this strategy, the parent is mutated using at least two difference vectors. One difference vector is calculated from the best vector and the parent vector, while the rest of the difference vectors are calculated using randomly selected vectors:

  $$\mathbf{u}_i(t) = \mathbf{x}_i(t) + \beta(\hat{\mathbf{x}}(t) - \mathbf{x}_i(t)) + \beta\sum_{k=1}^{n_v}(\mathbf{x}_{i_1,k}(t) - \mathbf{x}_{i_2,k}(t)) \tag{13.7}$$

Empirical studies have shown that DE/rand/1/bin maintains good diversity, while DE/current-to-best/2/bin shows good convergence characteristics [698]. Due to this observation, Qin and Suganthan [698] developed a DE algorithm that dynamically switch between these two strategies. Each of these strategies is assigned a probability of being applied. If $p_{s,1}$ is the probability that DE/rand/1/bin will be applied, then $p_{s,2} = 1 - p_{s,1}$ is the probability that DE/current-to-best/2/bin will be applied. Then,

$$p_{s,1} = \frac{n_{s,1}(n_{s,2} + n_{f,2})}{n_{s,2}(n_{s,1} + n_{f,1}) + n_{s,1}(n_{s,2} + n_{f,2})} \tag{13.8}$$

where $n_{s,1}$ and $n_{s,2}$ are respectively the number of offspring that survive to the next generation for DE/rand/1/bin, and $n_{f,1}$ and $n_{f,2}$ represent the number of discarded offspring for each strategy. The more offspring that survive for a specific strategy, the higher the probability for selecting that strategy for the next generation.

13.3 Variations to Basic Differential Evolution

The basic DE strategies have been shown to be very efficient and robust [811, 813, 811, 813]. A number of adaptations of the original DE strategies have been developed in order to further improve performance. This section reviews some of these DE variations. Section 13.3.1 describe hybrid DE methods, a population-based DE is described in Section 13.3.2, and self-adaptive DE strategies are discussed in Section 13.3.3.

13.3.1 Hybrid Differential Evolution Strategies

DE has been combined with other EAs, particle swarm optimization (PSO), and gradient-based techniques. This section summarizes some of these hybrid methods.

Gradient-Based Hybrid Differential Evolution

One of the first DE hybrids was developed by Chiou and Wang [124], referred to as the hybrid DE. As indicated in Algorithm 13.4, the hybrid DE introduces two new operations: an acceleration operator to improve convergence speed – without decreasing diversity – and a migration operator to provide the DE with the improved ability to escape local optima.

The acceleration operator uses gradient descent to adjust the best individual toward obtaining a better position if the mutation and crossover operators failed to improve the fitness of the best individual. Let $\hat{\mathbf{x}}(t)$ denote the best individual of the current population, $\mathcal{C}(t)$, before application of the mutation and crossover operators, and let $\hat{\mathbf{x}}(t+1)$ be the best individual for the next population after mutation and crossover have been applied to all individuals. Then, assuming a minimization problem, the acceleration operator computes the vector

$$\mathbf{x}(t) = \begin{cases} \hat{\mathbf{x}}(t+1) & \text{if } f(\hat{\mathbf{x}}(t+1)) < f(\hat{\mathbf{x}}(t)) \\ \hat{\mathbf{x}}(t+1) - \eta(t)\nabla f & \text{otherwise} \end{cases} \tag{13.9}$$

where $\eta(t) \in (0, 1]$ is the learning rate, or step size; ∇f is the gradient of the objective function, f. The new vector, $\mathbf{x}(t)$, replaces the worst individual in the new population, $\mathcal{C}(t)$.

Algorithm 13.4 Hybrid Differential Evolution with Acceleration and Migration

```
Set the generation counter, $t = 0$;
Initialize the control parameters, $\beta$ and $p_r$;
Create and initialize the population, $\mathcal{C}(0)$, of $n_s$ individuals;
while stopping condition(s) not true do
    Apply the migration operator if necessary;
    for each individual, $\mathbf{x}_i(t) \in \mathcal{C}(t)$ do
        Evaluate the fitness, $f(\mathbf{x}_i(t))$;
        Create the trial vector, $\mathbf{u}_i(t)$ by applying the mutation operator;
        Create an offspring, $\mathbf{x}_i^{'}(t)$ by applying the crossover operator;
        if $f(\mathbf{x}_i^{'}(t))$ is better than $f(\mathbf{x}_i(t))$ then
            Add $\mathbf{x}_i^{'}(t)$ to $\mathcal{C}(t+1)$;
        end
        else
            Add $\mathbf{x}_i(t)$ to $\mathcal{C}(t+1)$;
        end
    end
    Apply the acceleration operator if necessary;
end
Return the individual with the best fitness as the solution;
```

The learning rate is initialized to one, i.e. $\eta(0) = 1$. If the gradient descent step failed to create a new vector, $\mathbf{x}(t)$, with better fitness, the learning rate is reduced by a factor. The gradient descent step is then repeated until $\eta(t)\nabla f$ is sufficiently close to zero, or a maximum number of gradient descent steps have been executed.

While use of gradient descent can significantly speed up the search, it has the disadvantage that the DE may get stuck in a local minimum, or prematurely converge. The migration operator addresses this problem by increasing population diversity. This is done by spawning new individuals from the best individual, and replacing the current population with these new individuals. Individuals are spawned as follows:

$$x_{ij}^{'}(t) = \begin{cases} \hat{x}_j(t) + r_{ij}(x_{min,j} - \hat{x}_j) & \text{if } U(0,1) < \frac{\hat{x}_j - x_{min,j}}{x_{max,j} - x_{min,j}} \\ \hat{x}_j(t) + r_{ij}(x_{max,j} - \hat{x}_j) & \text{otherwise} \end{cases} \tag{13.10}$$

where $r_{ij} \sim U(0,1)$. Spawned individual $\mathbf{x}_i^{'}(t)$ becomes $\mathbf{x}_i(t+1)$.

The migration operator is applied only when the diversity of the current population becomes too small; that is, when

$$\left(\sum_{\substack{i=1 \\ \mathbf{x}_i(t) \neq \hat{\mathbf{x}}(t)}}^{n_s} \mathcal{I}_{ij}(t) \right) / (n_x(n_s - 1)) < \epsilon_1 \tag{13.11}$$

with

$$\mathcal{I}_{ij}(t) = \begin{cases} 1 & \text{if } |(x_{ij}(t) - \hat{x}_j(t))/\hat{x}_j(t)| > \epsilon_2 \\ 0 & \text{otherwise} \end{cases} \tag{13.12}$$

where ϵ_1 and ϵ_2 are respectively the tolerance for the population diversity and gene diversity with respect to the best individual, $\hat{\mathbf{x}}(t)$. If $\mathcal{I}_{ij}(t) = 0$, then the value of the j-th element of individual i is close to the value of the j-th element of the best individual.

Magoulas *et al.* [550] combined a stochastic gradient descent (SGD) [549] and DE in a sequential manner to train artificial neural networks (NN). Here, SGD is first used to find a good approximate solution using the process outlined in Algorithm 13.5. A population of DE individuals is then created, with individuals in the neighborhood of the solution returned by the SGD step. As outlined in Algorithm 13.6, the task of DE is to refine the solution obtained from SGD by using then DE to perform a local search.

Algorithm 13.5 Stochastic Gradient Descent for Neural Network Training

```
Initialize the NN weight vector, $\mathbf{w}(0)$;
Initialize the learning rate, $\eta(0)$, and the meta-step size, $\eta_m$;
Set the pattern presentation number, $t = 0$;
repeat
    for each training pattern, $p$ do
        Calculate $\mathcal{E}(\mathbf{w}(t))$;
        Calculate $\nabla\mathcal{E}(\mathbf{w}(t))$;
        Update the weights using
            $$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(t)\nabla\mathcal{E}(\mathbf{w}(t)) \tag{13.13}$$
        Calculate the new step size using
            $$\eta(t+1) = \eta(t) + \eta_m < \nabla\mathcal{E}(\mathbf{w}(t-1)), \nabla\mathcal{E}(\mathbf{w}(t)) > \tag{13.14}$$
        $t = t + 1$;
    end
    Return $\mathbf{w}(t+1)$ as the solution;
until until a termination condition is satisfied;
```

In Algorithms 13.5 and 13.6, $< \bullet, \bullet >$ denotes the inner product between the two given vectors, $\mathcal{E}$ is the NN training objective function (usually the sum-squared error), σ is the standard deviation of mutations to $\mathbf{w}$ used to create DE individuals in the neighborhood of $\mathbf{w}$, and D_T is the training set. The DE algorithm uses the objective function, $\mathcal{E}$, to assess the fitness of individuals.

Algorithm 13.6 Differential Evolution with Stochastic Gradient Descent

```
$\mathbf{w} = SGD(D_T)$;
Set the individual counter, $i = 0$;
Set $\mathcal{C}(0) = \{\}$;
repeat
    $i = i + 1$;
    $\mathbf{x}_i(0) = \mathbf{w} + \mathbf{N}(0, \sigma)$;
    $\mathcal{C}(0) \leftarrow \mathcal{C}(0) + \{\mathbf{x}_i(0)\}$;
until $i = n_s$;
Apply any DE strategy;
Return the best solution from the final population;
```

Evolutionary Algorithm-Based Hybrids

Due to the efficiency of DE, Hrstka and Kucerová [384] used the DE reproduction process as a crossover operator in a simple GA.

Chang and Chang [113] used standard mutation operators to increase DE population diversity by adding noise to the created trial vectors. In [113], uniform noise is added to each component of trial vectors, i.e.

$$u_{ij}(t) = u_{ij}(t) + U(u_{min,j}, u_{max,j}) \tag{13.15}$$

where $u_{min,j}$ and $u_{max,j}$ define the boundaries of the added noise. However, the approach above should be considered carefully, as the expected mean of the noise added is

$$\frac{u_{min,j} + u_{max,j}}{2} \tag{13.16}$$

If this mean is not zero, the population may suffer genetic drift. An alternative is to sample the noise from a Gaussian or Cauchy distribution with zero mean and a small deviation (refer to Section 11.2.1).

Sarimveis and Nikolakopoulos [758] use rank-based selection to decide which individuals will take part to calculate difference vectors. At each generation, after the fitness of all individuals have been calculated, individuals are arranged in descending order, $\mathbf{x}_1(t), \mathbf{x}_2(t), \ldots, \mathbf{x}_{n_s}(t)$ where $\mathbf{x}_{i_1}(t)$ precedes $\mathbf{x}_{i_2}(t)$ if $f(\mathbf{x}_{i_1}(t)) > f(\mathbf{x}_{i_2}(t))$. The crossover operator is then applied as summarized in Algorithm 13.7 assuming minimization. After application of crossover on all the individuals, the resulting population is again ranked in descending order. The mutation operator in Algorithm 13.8 is then applied.

With reference to Algorithm 13.8, $p_{m,i}$ refers to the probability of mutation, with each individual assigned a different probability based on its rank. The lower the rank of an individual, the more unfit the individual is, and the higher the probability that the individual will be mutated. Mutation step sizes are initially large, decreasing over time due to the exponential term used in equations (13.17) and (13.18). The direction of the mutation is randomly decided, using the random variable, r_2.

Algorithm 13.7 Rank-Based Crossover Operator for Differential Evolution

```
Rank all individuals in decreasing order of fitness;
for $i = 1, \ldots, n_s$ do
    $r \sim U(0, 1)$;
    $\mathbf{x}_i'(t) = \mathbf{x}_i(t) + r(\mathbf{x}_{i+1}(t) - \mathbf{x}_i(t))$;
    if $f(\mathbf{x}_i'(t)) < f(\mathbf{x}_{i+1}(t))$ then
        $\mathbf{x}_i(t) = \mathbf{x}_i'(t)$;
    end
end
```

Algorithm 13.8 Rank-Based Mutation Operator for Differential Evolution

```
Rank all individuals in decreasing order of fitness;
for $i = 1, \ldots, n_s$ do
    $p_{m,i} = \frac{n_s - i + 1}{n_s}$;
    for $j = 1, \ldots, n_x$ do
        $r_1 \sim U(0, 1)$;
        if $(r_1 > p_{m,i})$ then
            $r_2 \sim \{0, 1\}$;
            $r_3 \sim U(0, 1)$;
            if $(r_2 = 0)$ then
                $$x_{ij}'(t) = x_{ij}(t) + (x_{max,j} - x_{ij}(t)) r_3 e^{-2t/n_t} \qquad (13.17)$$
            end
            if $(r_2 = 1)$ then
                $$x_{ij}'(t) = x_{ij}(t) - (x_{ij}(t) - x_{min,j}) r_3 e^{-2t/n_t} \qquad (13.18)$$
            end
        end
    end
    if $f(\mathbf{x}_i'(t)) < f(\mathbf{x}_i(t))$ then
        $\mathbf{x}_i(t) = \mathbf{x}_i'(t)$;
    end
end
```

Particle Swarm Optimization Hybrids

A few studies have combined DE with particle swarm optimization(PSO) (refer to Chapter 16).

Hendtlass [360] proposed that the DE reproduction process be applied to the particles in a PSO swarm at specified intervals. At the specified intervals, the PSO swarm serves as the population for a DE algorithm, and the DE is executed for a number of generations. After execution of the DE, the evolved population is then further optimized using PSO. Kannan *et al.* [437] apply DE to each particle for a number of iterations, and replaces the particle with the best individual obtained from the DE process.

Zhang and Xie [954], and Talbi and Batouche [836] follow a somewhat different approach. Only the personal best positions are changed using

$$y'_{ij}(t+1) = \begin{cases} \hat{y}_{ij}(t) + \delta_j & \text{if } j \in \mathcal{J}_i(t) \\ y_{ij}(t) & \text{otherwise} \end{cases} \tag{13.19}$$

where δ is the general difference vector defined as

$$\delta_j = \frac{y_{1j}(t) - y_{2j}(t)}{2} \tag{13.20}$$

with $\mathbf{y}_1(t)$ and $\mathbf{y}_2(t)$ randomly selected personal best positions; the notations $\mathbf{y}_i(t)$ and $\hat{\mathbf{y}}_i(t)$ are used to indicate a personal best and neighborhood best respectively (refer to Chapter 16). The offspring, $\mathbf{y}'_i(t+1)$, replaces the current personal best, $\mathbf{y}_i(t)$, only if the offspring has a better fitness.

13.3.2 Population-Based Differential Evolution

In order to improve the exploration ability of DE, Ali and Törn [19] proposed to use two population sets. The second population, referred to as the auxiliary population, $\mathcal{C}_a(t)$, serves as an archive of those offspring rejected by the DE selection operator. During the initialization process, n_s pairs of vectors are randomly created. The best of the two vectors is inserted as an individual in the population, $\mathcal{C}(0)$, while the other vector, $\mathbf{x}_i^a(0)$, is inserted in the auxiliary population, $\mathcal{C}_a(0)$. At each generation, for each offspring created, if the fitness of the offspring is not better than the parent, instead of discarding the offspring, $\mathbf{x}'_i(t)$, it is considered for inclusion in the auxiliary population. If $f(\mathbf{x}'_i(t))$ is better than $\mathbf{x}_i^a(t)$, then $\mathbf{x}'_i(t)$ replaces $\mathbf{x}_i^a(t)$. The auxiliary set is periodically used to replace the worst individuals in $\mathcal{C}(t)$ with the best individuals from $\mathcal{C}_a(t)$.

13.3.3 Self-Adaptive Differential Evolution

Although empirical studies have shown that DE convergence is relatively insensitive to control parameter values, performance can be greatly improved if parameter values are optimized. For the DE strategies discussed thus far, values of control parameters are static, and do not change over time. These strategies require an additional search process to find the best values for control parameters for each different problem – a process that is usually time consuming. It is also the case that different values for a control parameter are optimal for different stages of the optimization process. As an alternative, a number of DE strategies have been developed where values for control parameters adapt dynamically. This section reviews these approaches.

Dynamic Parameters

One of the first proposals for dynamically changing the values of the DE control parameters was proposed by Chang and Xu [112], where the probability of recombination is linearly decreased from 1 to 0.7, and the scale factor is linearly increased from 0.3 to 0.5:

$$\begin{aligned} p_r(t) &= p_r(t-1) - (p_r(0) - 0.7)/n_t && (13.21)\\ \beta(t) &= \beta(t-1) - (0.5 - \beta(0))/n_t && (13.22) \end{aligned}$$

where $p_r(0) = 1.0$ and $\beta(0) = 0.3$; n_t is the maximum number of iterations.

Abbass *et al.* [3] proposed an approach where a new value is sampled for the scale factor for each application of the mutation operator. The scale factor is sampled from a Gaussian distribution, $\beta \sim N(0, 1)$. This approach is also used in [698, 735]. In [698], the mean of the distribution was changed to 0.5 and the deviation to 0.3 (i.e. $\beta \sim N(0.5, 0.3)$), due to the empirical results that suggest that $\beta = 0.5$ provides on average good results. Abbass [2] extends this to the probability of recombination, i.e. $p_r \sim N(0, 1)$. Abbass refers incorrectly to the resulting DE strategy as being self-adaptive. For self-adaptive strategies, values of control parameters are evolved over time; this is not the case in [2, 3].

Self-Adaptive Parameters

Self-adaptive strategies usually make use of information about the search space as obtained from the current population (or a memory of previous populations) to self-adjust values of control parameters.

Ali and Törn [19] use the fitness of individuals in the current population to determine a new value for the scale factor. That is,

$$\beta(t) = \begin{cases} \max\left\{\beta_{min}, 1 - \left|\frac{f_{max}(t)}{f_{min}(t)}\right|\right\} & \text{if } \left|\frac{f_{max}(t)}{f_{min}(t)}\right| < 1 \\ \max\left\{\beta_{min}, 1 - \left|\frac{f_{min}(t)}{f_{max}(t)}\right|\right\} & \text{otherwise} \end{cases} \qquad (13.23)$$

which ensures that $\beta(t) \in [\beta_{min}, 1)$, where β_{min} is a lower bound on the scaling factor; $f_{min}(t)$ and $f_{max}(t)$ are respectively the minimum and maximum fitness values for the current population, $\mathcal{C}(t)$. As f_{min} approaches f_{max}, the diversity of the population decreases, and the value of $\beta(t)$ approaches β_{min} – ensuring smaller step sizes when the population starts to converge. On the other hand, the smaller the ratio $\left|\frac{f_{max}(t)}{f_{min}(t)}\right|$ (for minimization problems) or $\left|\frac{f_{min}(t)}{f_{max}(t)}\right|$ (for maximization problems), the more diverse the population and the larger the step sizes will be – favoring exploration.

Qin and Suganthan [698] propose that the probability of recombination be self-adapted as follows:

$$p_r(t) \sim N(\mu_{p_r}(t), 0.1) \tag{13.24}$$

where $\mu_{p_r}(0) = 0.5$, and $\mu_{p_r}(t)$ is calculated as the average over successful values of $p_r(t)$. A $p_r(t)$ value can be considered as being successful if the fitness of the best individual improved under that value of $p_r(t)$. It is not clear if one probability is used in [698] for the entire population, or if each individual has its own probability, $p_{r,i}(t)$. This approach to self-adaptation can, however, be applied for both scenarios.

For the self-adaptive Pareto DE, Abbass [2] adapts the probability of recombination dynamically as

$$p_{r,i}(t) = p_{r,i_1}(t) + N(0,1)[p_{r,i_2}(t) - p_{r,i_3}(t)] \tag{13.25}$$

where $i_1 \neq i_2 \neq i_3 \neq i \sim U(1, \ldots, n_s)$, while sampling the scale factor from $N(0,1)$. Note that equation (13.25) implies that each individual has its own, learned probability of recombination.

Omran *et al.* [641] propose a self-adaptive DE strategy that makes use of the approach in equation (13.25) to dynamically adapt the scale factor. That is, for each individual,

$$\beta_i(t) = \beta_{i_4}(t) + N(0, 0.5)[\beta_{i_5}(t) - \beta_{i_6}(t)] \tag{13.26}$$

where $i_4 \neq i_5 \neq i_6 \neq i \sim U(1, \ldots, n_s)$. The mutation operator as given in equation (13.2) changes to

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta_i(t)[\mathbf{x}_{i_2}(t) + \mathbf{x}_{i_3}(t)] \tag{13.27}$$

The crossover probability can be sampled from a Gaussian distribution as discussed above, or adapted according to equation (13.25).

13.4 Differential Evolution for Discrete-Valued Problems

Differential evolution has been developed for optimizing continuous-valued parameters. However, a simple discretization procedure can be used to convert the floating-point solution vectors into discrete-valued vectors. Such a procedure has been used by a number of researchers in order to apply DE to integer and mixed-integer programming [258, 390, 499, 531, 764, 817]. The approach is quite simple: each floating-point value of a solution vector is simply rounded to the nearest integer. For a discrete-valued parameter where an ordering exists among the values of the parameter, Lampinen and Zelinka [499] and Feoktistov and Janaqi [258] take the index number in the ordered sequence as the discretized value.

Pampará *et al.* [653] proposed an approach to apply DE to binary-valued search spaces: The angle modulated DE (AMDE) [653] uses the standard DE to evolve a generating function to produce bitstring solutions. This chapter proposes an alternative, the binary DE (binDE) which treats each floating-point element of solution vectors as a probability of producing either a bit 0 or a bit 1. These approaches are respectively discussed in Sections 13.4.1 and 13.4.2.

13.4.1 Angle Modulated Differential Evolution

Pampará *et al.* [653] proposed a DE algorithm to evolve solutions to binary-valued optimization problems, without having to change the operation of the original DE. This is achieved by using a homomorphous mapping [487] to abstract a problem (defined in binary-valued space) into a simpler problem (defined in continuous-valued space), and then to solve the problem in the abstracted space. The solution obtained in the abstracted space is then transformed back into the original space in order to solve the problem. The angle modulated DE (AMDE) makes use of angle modulation (AM), a technique derived from the telecommunications industry [697], to implement such a homomorphous mapping between binary-valued and continuous-valued space.

The objective is to evolve, in the abstracted space, a bitstring generating function, which will be used in the original space to produce bit-vector solutions. The generating function as used in AM is

$$g(x) = \sin(2\pi(x-a) \times b \times \cos(2\pi(x-a) \times c)) + d \tag{13.28}$$

where x is a single element from a set of evenly separated intervals determined by the required number of bits that need to be generated (i.e. the dimension of the original, binary-valued space).

The coefficients in equation (13.28) determine the shape of the generating function: a represents the horizontal shift of the generating function, b represents the maximum frequency of the sin function, c represents the frequency of the cos function, and d represents the vertical shift of the generating function. Figure 13.2 illustrates the function for $a = 0, b = 1, c = 1$, and $d = 0$, with $x \in [-2, 2]$. The AMDE evolves values for the four coefficients, a, b, c, and d. Solving a binary-valued problem thus reverts to solving a 4-dimensional problem in a continuous-valued space. After each iteration of the AMDE, the fitness of each individual in the population is determined by substituting the evolved values for the coefficients (as represented by the individual) into equation (13.28). The resulting function is sampled at evenly spaced intervals and a bit value is recorded for each interval. If the output of the function in equation (13.28) is positive, a bit-value of 1 is recorded; otherwise, a bit-value of 0 is recorded. The resulting bit string is then evaluated by the fitness function defined in the original binary-valued space in order to determine the quality of the solution.

The AMDE is summarized in Algorithm 13.9.

Pampará *et al.* [653] show that the AMDE is very efficient and provides accurate solutions to binary-valued problems. Furthermore, the AMDE has the advantage that

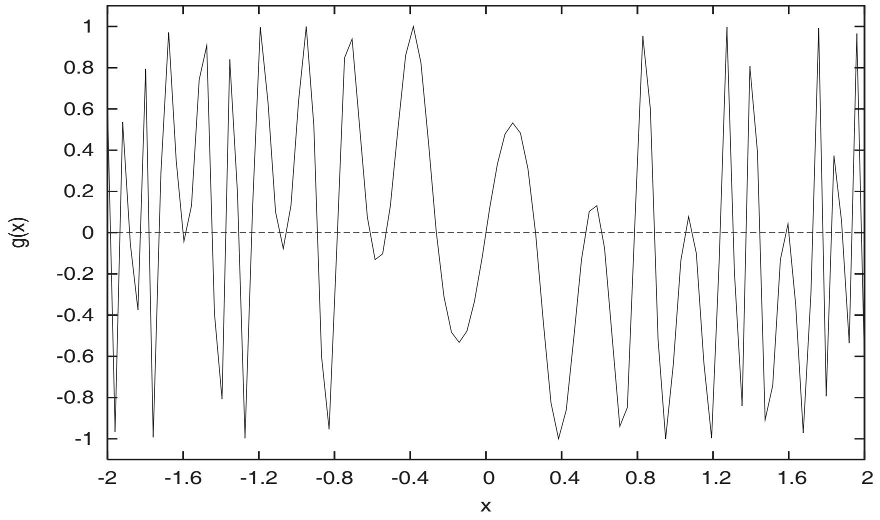


Figure 13.2 Angle Modulation Illustrated

Algorithm 13.9 Angle Modulated Differential Evolution

```
Generate a population of 4-dimensional individuals;
repeat
    Apply any DE strategy for one iteration;
    for each individual do
        Substitute evolved values for coefficients $a, b, c$ and $d$ into equation (13.28);
        Produce $n_x$ bit-values to form a bit-vector solution;
        Calculate the fitness of the bit-vector solution in the original bit-valued space;
    end
until a convergence criterion is satisfied;
```

an n_x-dimensional binary-valued problem is transformed into a smaller 4-dimensional continuous-valued problem.

13.4.2 Binary Differential Evolution

The binary DE (binDE) borrows concepts from the binary particle swarm optimizer (binPSO), developed by Kennedy and Eberhart [450] (also refer to Section 16.5.7). As with DE, particle swarm optimization (PSO; refer to Chapter 16) uses vector algebra to calculate new search positions, and was therefore developed for continuous-valued problems. In PSO, a velocity vector represents the mutation step sizes as stochastically weighted difference vectors (i.e. the social and cognitive components). The binPSO does not interpret the velocity as a step size vector. Rather, each component of the velocity vector is used to compute the probability that the corresponding component of the solution vector is bit 0 or bit 1.

In a similar way, the binDE uses the floating-point DE individuals to determine a probability for each component. These probabilities are then used to generate a bitstring solution from the floating-point vector. This bitstring is used by the fitness function to determine its quality. The resulting fitness is then associated with the floating-point representation of the individual.

Let $\mathbf{x}_i(t)$ represent a DE individual, with each $x_{ij}(t)$ ($j = 1, \ldots, n_x$, where n_x is the dimension of the binary-valued problem) floating-point number. Then, the corresponding bitstring solution, $\mathbf{y}_i(t)$, is calcualted using

$$y_{ij} = \begin{cases} 0 & \text{if } f(x_{ij}(t)) \geq 0.5 \\ 1 & \text{if } f(x_{ij}(t)) < 0.5 \end{cases} \tag{13.29}$$

where f is the sigmoid function,

$$f(x) = \frac{1}{1 + e^{-x}} \tag{13.30}$$

The fitness of the individual $\mathbf{x}_i(t)$ is then simply the fitness obtained using the binary representation, $\mathbf{y}_i(t)$.

The binDE algorithm is summarized in Algorithm 13.10.

Algorithm 13.10 Binary Differential Evolution Algorithm

```
Initialize a population and set control parameter values;
$t = 0$;
while stopping condition(s) not true do
    $t = t + 1$;
    Select parent $\mathbf{x}_i(t)$;
    Select individuals for reproduction;
    Produce one offspring, $\mathbf{x}^{'}(t)$;
    $\mathbf{y}_i(t)$ = generated bitstring from $\mathbf{x}_i(t)$;
    $\mathbf{y}_i^{'}(t)$ = generated bitstring from $\mathbf{x}_i^{'}(t)$;
    if $f(\mathbf{y}_i^{'}(t))$ is better than $f(\mathbf{x}_i^{'}(t))$ then
        Replace parent, $\mathbf{x}_i(t)$, with offspring, $\mathbf{x}_i^{'}(t)$;
    end
    else
        Retain parent, $\mathbf{x}_i(t)$;
    end
end
```

13.5 Advanced Topics

The discussions in the previous sections considered application of DE to unconstrained, single-objective optimization problems, where the fitness landscape remains static. This section provides a compact overview of adaptations to the DE such that different types of optimization problems as summarized in Appendix A can be solved using DE.

13.5.1 Constraint Handling Approaches

With reference to Section A.6, the following methods have been used to apply DE to solve constrained optimization problems as defined in Definition A.5:

- Penalty methods (refer to Section A.6.2), where the objective function is adapted by adding a function to penalize solutions that violate constraints [113, 394, 499, 810, 884].
- Converting the constrained problem to an unconstrained problem by embedding constraints in an augmented Lagrangian (refer to Section A.6.2) [125, 390, 528, 758]. Lin *et al.* [529] combines both the penalty and the augmented Lagrangian functions to convert a constrained problem to an unconstrained one.
- In order to preserve the feasibility of initial solutions, Chang and Wu [114] used feasible directions to determine step sizes and search directions.
- By changing the selection operator of DE, infeasible solutions can be rejected, and the repair of infeasible solutions facilitated. In order to achieve this, the selection operator accepts an offspring, $\mathbf{x}_i^{'}$, under the following conditions [34, 56, 498]:
  - if $\mathbf{x}_i^{'}$ satisfies all the constraints, and $f(\mathbf{x}_i^{'}) \leq f(\mathbf{x}_i)$, then $\mathbf{x}_i^{'}$ replaces the parent, $\mathbf{x}_i$ (assuming minimization);
  - if $\mathbf{x}_i^{'}$ is feasible and $\mathbf{x}_i$ is infeasible, then $\mathbf{x}_i^{'}$ replaces $\mathbf{x}_i$;
  - if both $\mathbf{x}_i^{'}$ and $\mathbf{x}_i$ are infeasible, then if the number of constraints violated by $\mathbf{x}_i^{'}$ is less than or equal to the number of constraints violated by $\mathbf{x}_i$, then $\mathbf{x}_i^{'}$ replaces $\mathbf{x}_i$.

  In the case that both the parent and the offspring represent infeasible solutions, there is no selection pressure towards better parts of the fitness landscape; rather, towards solutions with the smallest number of violated constraints.

Boundary constraints are easily enforced by clamping offspring to remain within the given boundaries [34, 164, 498, 499]:

$$x_{ij}^{'}(t) = \begin{cases} x_{min,j} + U(0,1)(x_{max,j} - x_{min,j}) & \text{if } x_{ij}^{'}(t) < x_{min,j} \text{ or } x_{ij}^{'} > x_{max,j} \\ x_{ij}^{'}(t) & \text{otherwise} \end{cases} \tag{13.31}$$

This restarts the offspring to a random position within the boundaries of the search space.

13.5.2 Multi-Objective Optimization

As defined in Definition A.10, multi-objective optimization requires multiple, conflicting objectives to be simultaneously optimized. A number of adaptations have been made to DE in order to solve multiple objectives, most of which make use of the concept of dominance as defined in Definition A.11.

Multi-objective DE approaches include:

- Converting the problem into a minimax problem [390, 925].
- Weight aggregation methods [35].
- Population-based methods, such as the vector evaluated DE (VEDE) [659], based on the vector evaluated GA (VEGA) [761] (also refer to Section 9.6.3). If K objectives have to be optimized, K sub-populations are used, where each sub-population optimizes one of the objectives. These sub-populations are organized in a ring topology (as illustrated in Figure 16.4(b)). At each iteration, before application of the DE reproduction operators, the best individual, $\mathcal{C}_k.\hat{\mathbf{x}}(t)$, of population $\mathcal{C}_k$ migrates to population $\mathcal{C}_{k+1}$ (that of $\mathcal{C}_{k+1}$ migrates to $\mathcal{C}_0$), and is used in population $\mathcal{C}_{k+1}$ to produce the trial vectors for that population.
- Pareto-based methods, which change the DE operators to include the dominance concept.

  Mutation: Abbass *et al.* [2, 3] applied mutation only on non-dominated solutions within the current generation. Xue *et al.* [928] computed the differential as the difference between a randomly selected individual, $\mathbf{x}_{i_1}$, and a randomly selected vector, $\mathbf{x}_{i_2}$, that dominates $\mathbf{x}_{i_1}$; that is, $\mathbf{x}_{i_1} \preceq \mathbf{x}_{i_2}$. If $\mathbf{x}_{i_1}$ is not dominated by any other individual of the current generation, the differential is set to zero.

  Selection: A simple change to the selection operator is to replace the parent, $\mathbf{x}_i$, with the offspring $\mathbf{x}_i^{'}$, only if $\mathbf{x}_i^{'} \preceq \mathbf{x}_i$ [3, 2, 659]. Alternatively, ideas from non-dominated sorting genetic algorithms [197] can be used, where non-dominated sorting and ranking is applied to parents and offspring [545, 928]. The next population is then selected with preference to those individuals with a higher rank.

13.5.3 Dynamic Environments

Not much research has been done in applying DE to dynamically changing landscapes (refer to Section A.9). Chiou and Wang [125] applied the DE with acceleration and migration (refer to Algorithm 13.4) to dynamic environments, due to the improved exploration as provided by the migration phase. Magoulas *et al.* [550] applied the SGDDE (refer to Algorithm 13.6) to slowly changing fitness landscapes.

Mendes and Mohais [577] develop a DE algorithm, referred to as DynDE, to locate and maintain multiple solutions in dynamically changing landscapes. Firstly, it is important to note the following assumptions:

1. It is assumed that the number of peaks, $n_{\mathcal{X}}$, to be found are known, and that these peaks are evenly distributed through the search space.
2. Changes in the fitness landscape are small and gradual.

DynDE uses multiple populations, with each population maintaining one of the peaks. To ensure that each peak represents a different solution, an exclusion strategy is followed: At each iteration, the best individuals of each pair of sub-populations are compared. If these global best positions are too close to one another, the sub-population with the worst global best solution is re-initialized. DynDE re-initializes the one sub-population when

$$\mathcal{E}(\mathcal{C}_{k_1}.\hat{\mathbf{x}}(t), \mathcal{C}_{k_2}.\hat{\mathbf{x}}(t)) < \frac{X}{2n_{\mathcal{X}}^{1/n_x}} \tag{13.32}$$

where $\mathcal{E}(\mathcal{C}_{k_1}.\hat{\mathbf{x}}(t), \mathcal{C}_{k_2}.\hat{\mathbf{x}}(t))$ is the Euclidean distance between the best individuals of sub-populations $\mathcal{C}_{k_1}$ and $\mathcal{C}_{k_2}$, X represents the extent of the search space, $n_{\mathcal{X}}$ is the number of peaks, and n_x is the search space dimension. It is this condition that requires assumption 1, which suffers from obvious problems. For example, peaks are not necessarily evenly distributed. It may also be the case that two peaks exist with a distance less than $\frac{X}{2n_{\mathcal{X}}^{1/n_x}}$ from one another. Also, it is rarely the case that the number of peaks is known.

After a change is detected, a strategy is followed to increase diversity. This is done by assigning a different behavior to some of the individuals of the affected sub-population. The following diversity increasing strategies have been proposed [577]:

- Re-initialize the sub-populations: While this strategy does maximize diversity, it also leads to a severe loss of knowledge obtained about the search space.
- Use quantum individuals: Some of the individuals are re-initialized to random points inside a ball centered at the global best individual, $\hat{\mathbf{x}}(t)$, as outlined in Algorithm 13.11. In this algorithm, R_{max} is the maximum radius from $\hat{\mathbf{x}}(t)$.
- Use Brownian individuals: Some positions are initialized to random positions around $\hat{\mathbf{x}}(t)$, where the random step sizes from $\hat{\mathbf{x}}(t)$ are sampled from a Gaussian distribution. That is,
$$\mathbf{x}_i(t) = \hat{\mathbf{x}}(t) + \mathbf{N}(0, \sigma) \tag{13.33}$$
- Introduce some form of entropy: Some individuals are simply added noise, sampled from a Gaussian distribution. That is,
$$\mathbf{x}_i(t) = \mathbf{x}_i(t) + \mathbf{N}(0, \sigma) \tag{13.34}$$

Algorithm 13.11 Initialization of Quantum Individuals

```
for each individual, $\mathbf{x}_i(t)$, to be re-initialized do
    Generate a random vector, $\mathbf{r}_i \sim \mathbf{N}(0,1)$;
    Compute the distance of $\mathbf{r}_i$ from the origin, i.e.
        $\mathcal{E}(\mathbf{r}_i, \mathbf{0}) = \sqrt{\sum_{j=1}^{n_x} r_{ij}}$    (13.35)
    Find the radius, $R \sim U(0, R_{max})$;
    $\mathbf{x}_i(t) = \hat{\mathbf{x}}(t) + R\mathbf{r}_i/\mathcal{E}(\mathbf{r}_i, \mathbf{0})$;
end
```

13.6 Applications

Differential evolution has mostly been applied to optimize functions defined over continuous-valued landscapes [695, 811, 813, 876]. Considering an unconstrained optimization problem, such as listed in Section A.5.3, each individual, $\mathbf{x}_i$, will be represented by an n_x-dimensional vector where each $x_{ij} \in \mathbb{R}$. For the initial population, each individual is initialized using

$$x_{ij} \sim U(x_{min,j}, x_{max,j}) \tag{13.36}$$

The fitness function is simply the function to be optimized.

DE has also been applied to train neural networks (NN) (refer to Table 13.1 for references). In this case an individual represents a complete NN. Each element of an individual is one of the weights or biases of the NN, and the fitness function is, for example, the sum-squared error (SSE).

Table 13.1 summarizes a number of real-world applications of DE. Please note that this is not meant to be a complete list.

Table 13.1 Applications of Differential Evolution

Application Class	**Reference**
Clustering	[640, 667]
Controllers	[112, 124, 164, 165, 394, 429, 438, 599]
Filter design	[113, 810, 812, 883]
Image analysis	[441, 521, 522, 640, 926]
Integer-Programming	[390, 499, 500, 528, 530, 817]
Model selection	[331, 354, 749]
NN training	[1, 122, 550, 551, 598]
Scheduling	[528, 531, 699, 748]
System design	[36, 493, 496, 848, 839, 885]

13.7 Assignments

1. Show how DE can be used to train a FFNN.
2. Discuss the influence of different values for the population diversity tolerance, ϵ_1, and the gene diversity tolerance, ϵ_2, as used in equations (13.11) and (13.12) for the hybrid DE.
3. Discuss the merits of the following two statements:
   (a) If the probability of recombination is very low, then DE exhibits a high probability of stagnation.
   (b) For a small population size, it is sensible to have a high probability of recombination.

4. For the DE/rand-to-best/y/z strategies, suggest an approach to balance exploration and exploitation.
5. Discuss the consequences of too large and too small values of the standard deviation, σ, used in Algorithm 13.6.
6. Explain in detail why the method for adding noise to trial vectors as given in equation (13.15) may result in genetic drift.
7. With reference to the DynDE algorithm in Section 13.5.3, explain the effect of very small and very large values of the standard deviation, σ.
8. Researchers in DE have suggested that the recombination probability should be sampled from a Gaussian distribution, $N(0, 1)$, while others have suggested that $N(0.5, 0.15)$ should be used. Compare these two suggestions and provide a recommendation as to which approach is best.
9. Investigate the performance of a DE strategy if the scale factor is sampled from a Cauchy distribution.