1

Motivation for Differential Evolution for Permutative−Based Combinatorial Problems

Godfrey Onwubolu[1] and Donald Davendra[2]

[1] Knowledge Management & Mining, Inc., Richmond Hill, Ontario, Canada
onwubolu_g@dsgm.ca
[2] Tomas Bata Univerzity in Zlin, Faculty of Applied Informatics, Nad Stranemi 4511, Zlin 76001, Czech Republic
davendra@fai.utb.cz

Abstract. It is generally accepted that Differential Evolution (DE) was originally designed to solve problems which are defined in continuous form. Some researchers have however, felt that this is a limiting factor on DE, hence there have been vigorous research work to extend the functionalities of DE to include permutative-based combinatorial problems. This chapter sets the scene for the book by discussing the motivation for presenting the foundational theories for a number of variants of DE for permutative-based combinatorial problems. These DE variants are presented by their initiators or proposers, to the best of our knowledge.

1.1 Introduction

Whether in industry or in research, users generally demand that a practical optimization technique should fulfil three requirements:

1. the method should find the true global minimum, regardless of the initial system parameter values;
2. convergence should be fast; and
3. the program should have a minimum of control parameters so that it will be easy to use.

[2] invented the differential evolution (DE) algorithm in a search for a technique that would meet the above criteria. DE is a method, which is not only astonishingly simple, but also performs extremely well on a wide variety of test problems. It is inherently parallel because it is a population based approach and hence lends itself to computation via a network of computers or processors. The basic strategy employs the difference of two randomly selected parameter vectors as the source of random variations for a third parameter vector.

There are broadly speaking two types of real−world problems that may be solved:

1. those that are characterized by continuous parameters; and
2. those that are characterized by permutative-based combinatorial parameters..

This classification is important in the context of the motivation for this book. The original canonical DE which Storn and Price developed was designed to solve only

G.C. Onwubolu and D. Davendra (Eds.): Differential Evolution, SCI 175, pp. 1–11.
springerlink.com

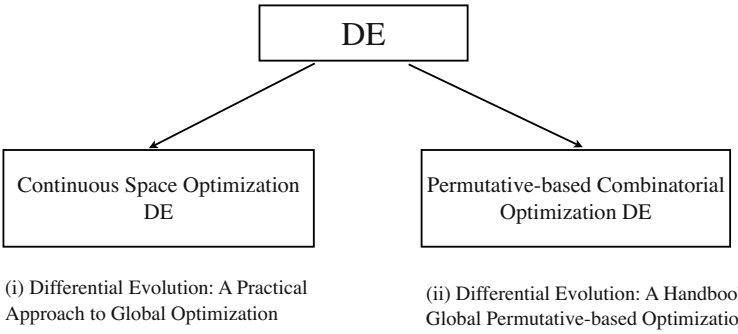

Fig. 1.1. DE framework (i) Existing book (Price et al. 2005); (ii) Present book

problems characterized by continuous parameters. This means that only a subset of real-world problems could be solved by the original canonical DE. For quite some time, this deficiency made DE not to be employed to a vast number of real-world problems which characterized by permutative-based combinatorial parameters. Fig 1.1 shows the framework into which the current book fits, showing that there are two mainstreams of philosophical schools that need to be considered in presenting DE for solving real-world problems. This framework is important as it shows that the current book compliments the first book on DE which only addresses one aspect: continuous parameters.

1.1.1 Continuous Space Optimization DE Problems

A typical continuous space optimization DE problem is the generalized Rosenbrock function given as:

$$\begin{aligned} &f(x) = \sum_{j=0}^{D-2} \left(100. \left(x_{j+1} - x_j^2 \right)^2 + (x_j - 1)^2 \right), \\ &-30 \leq x \leq 30, \quad j = 0,\ 1, ..., D-1,\ D > 1, \\ &f(x^*) = 0,\ x_j^* = 1,\ \varepsilon = 1.0 \times 10^{-6}. \end{aligned} \tag{1.1}$$

The solution of this problem is shown in Fig 1.2.

1.1.2 Permutative−Based Combinatorial Optimization DE Problem

A typical permutative-based combinatorial optimization DE problem is the flow shop scheduling five-job-four machine problem whose operation times are shown in Table 1.1 The objective is to find the best sequence to realize the optimal completion time (makespan). As we see, this problem is very different from the continuous space problem because we are interested in sequence such as 1, 2, 3, 4, 5 which is permutative in nature.

The minimization of completion time (makespan) for a flow shop schedule is equivalent to minimizing the objective function $\Im$:

$$\Im = \sum_{j=1}^{n} C_{m,j} \tag{1.2}$$

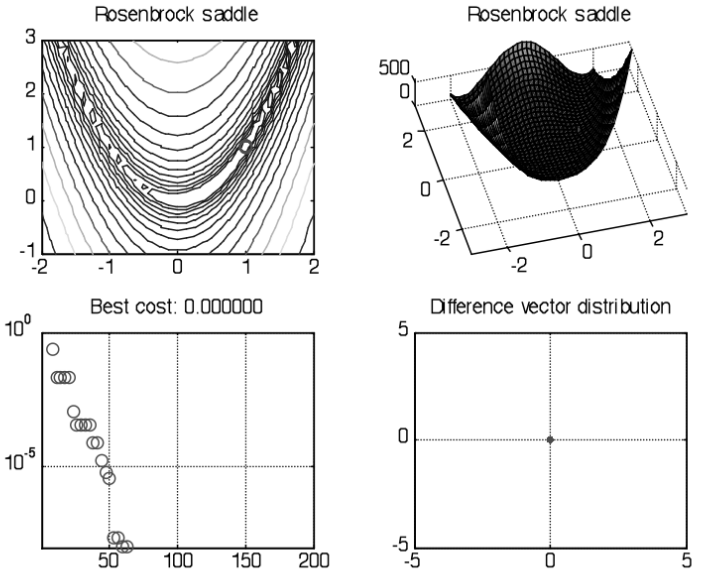

Fig. 1.2. The generalized Rosenbrock function problem

Table 1.1. Example of job times

jobs	j_1	j_2	j_3	j_4	j_5
P_{1,j_k}	6	4	4	5	1
P_{2,j_k}	4	6	2	4	3
P_{3,j_k}	3	3	4	1	3
P_{4,j_k}	4	4	5	3	1

such that:

$$C_{i,j} = \max\left(C_{i-1,j},\, C_{i,j-1}\right) + P_{i,j} \tag{1.3}$$

For this problem the schedule of $\{2,1,4,3,5\}$ gives a fitness (makespan) of 31. The solution includes a permutative schedule as well as the fitness. It is not feasible to solve this kind of problem using the canonical DE that solves only continuous problems as already discussed.

1.1.3 Suitability of Differential Evolution as a Combinatorial Optimizer

An extract from the summary of Section 4.4 of [2] reads:

> "*Although DE has performed well on wide-sense combinatorial problems, its suitability as a combinatorial optimizer is still a topic of considerable debate and a definitive judgment cannot be given at this time*".

Over the years, some researchers have been working in the area of DE permutative−based combinatorial optimization and they have found that DE is quite appropriate for combinatorial optimization and that it is effective and competitive with other approaches in this domain. Some of the DE permutative−based combinatorial optimization approaches that have proved effective include:

1. Forward/Backward Transformation Approach;
2. Relative Position Indexing Approach;
3. Smallest Position Value Approach;
4. Discrete/Binary Approach; and
5. Discrete Set Handling Approach.

These approaches have been applied to combinatorial optimization problems with competitive results when compared to other optimization approaches, and they form the basis for writing this book. The remainder of this book explores available DE approaches for solving permutative-based combinatorial problems. Although there have been discussions regarding DE approaches that rely to varying degrees on repair mechanisms, it is now generally agreed that in order to solve permutative-based combinatorial problems, it is necessary to employ some heuristics as some other evolutionary computation approaches do, rather than insisting on approaches that generate only feasible solutions. Each method proposes an analog of DE′s differential mutation operator to solve permutative-based combinatorial problems.

1.2 Canonical Differential Evolution for Continuous Optimization Problems

The parameters used in DE are $\Im$ = cost or the value of the objective function, D = problem dimension, NP = population size, P = population of X−vectors, G = generation number, $Gmax$ = maximum generation number, X = vector composed of D parameters, V = trial vector composed of D parameters, CR = crossover factor. Others are F = scaling factor ($0 < \mathrm{F} \leq 1.2$), *(U)* = upper bound, *(L)* = lower bound, **u**, and **v** = trial vectors, $x_{best}^{(G)}$ = vector with minimum cost in generation G, $x_i^{(G)}$ = ith vector in generation G, $b_i^{(G)}$ = ith buffer vector in generation G, $x_{r1}^{(G)}, x_{r2}^{(G)}$ = randomly selected vector, L = random integer ($0 < \mathrm{L} < \mathrm{D} - 1$). In the formulation, N = number of cities. Some integers used are *i, j*.

Differential Evolution (DE) is a novel parallel direct search method, which utilizes NP parameter vectors

$$x_i^{(G)}, i = 0, 1, 2, ..., NP - 1 \tag{1.4}$$

as a population for each generation, G. The population size, NP does not change during the minimization process. The initial population is generated randomly assuming a uniform probability distribution for all random decisions if there is no initial intelligent information for the system. The crucial idea behind DE is a new scheme for generating trial parameter vectors. DE generates new parameter vectors by adding the weighted difference vector between two population members to a third member. If the resulting

vector yields a lower objective function value than a predetermined population member, the newly generated vector replaces the vector with which it was compared. The comparison vector can, but need not be part of the generation process mentioned above. In addition the best parameter vector $x_{best}^{(G)}$, is evaluated for every generation G in order to keep track of the progress that is made during the minimization process. Extracting distance and direction information from the population to generate random deviations result in an adaptive scheme with excellent convergence properties [3].

Descriptions for the earlier two most promising variants of DE (later known as DE2 and DE3) are presented in order to clarify how the search technique works, then a complete list of the variants to date are given thereafter. The most comprehensive book that describes DE for continuous optimization problems is [2].

Scheme DE2

Initialization

As with all evolutionary optimization algorithms, DE works with a population of solutions, not with a single solution for the optimization problem. Population P of generation G contains NP solution vectors called individuals of the population and each vector represents potential solution for the optimization problem:

$$P^{(G)} = X_i^{(G)} \quad i = 1,...,NP;\ G = 1,...,G_{\max} \tag{1.5}$$

Additionally, each vector contains D parameters:

$$X_i^{(G)} = x_{j,i}^{(G)} \quad i = 1,...,NP;\ j = 1,...,D \tag{1.6}$$

In order to establish a starting point for optimum seeking, the population must be initialized. Often there is no more knowledge available about the location of a global optimum than the boundaries of the problem variables. In this case, a natural way to initialize the population $P^{(0)}$ (initial population) is to seed it with random values within the given boundary constraints:

$$P^{(0)} = x_{j,i}^{(0)} = x_j^{(L)} + rand_j[0,1] \bullet \left(x_j^{(U)} - x_j^{(L)}\right) \ \forall i \in [1,NP];\ \forall j \in [1,D] \tag{1.7}$$

where $rand_j[0,1]$ represents a uniformly distributed random value that ranges from zero to one. The lower and upper boundary constraints are,$X^{(L)}$ and $X^{(L)}$, respectively:

$$x_j^{(L)} \le x_j \le x_j^{(U)} \ \forall j \in [1,D] \tag{1.8}$$

For this scheme and other schemes, three operators are crucial: mutation, crossover and selection. These are now briefly discussed.

Mutation

The first variant of DE works as follows: for each vector $x_i^{(G)}, i = 0,1,2,..,NP-1$, a trial vector v is generated according to

$$v_{j,i}^{(G+1)} = x_{j,r1}^{(G)} + F \bullet \left(x_{j,r2}^{(G)} - x_{j,r3}^{(G)}\right) \tag{1.9}$$

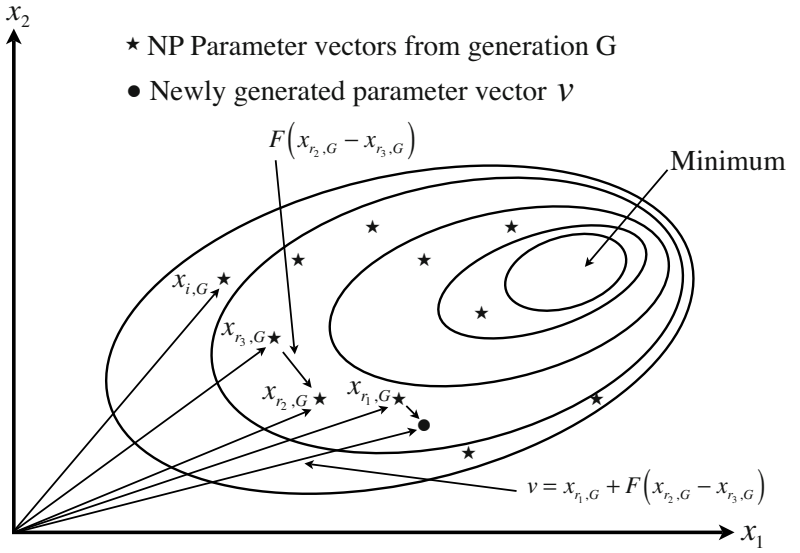

Fig. 1.3. Contour lines and the process for generating v in scheme DE1

where $i \in [1, NP]$; $j \in [1, D]$, $F > 0$, and the integers $r1,\ r2,\ r3 \in [1, NP]$ are generated randomly selected, except: $r1 \neq r2 \neq r3 \neq i$.

Three randomly chosen indexes, *r1*, *r2*, and *r3* refer to three randomly chosen vectors of population. They are mutually different from each other and also different from the running index *i*. New random values for *r1*, *r2*, and *r3* are assigned for each value of index i (for each vector). A new value for the random number *rand*[0, 1] is assigned for each value of index *j* (for each vector parameter). *F* is a real and constant factor, which controls the amplification of the differential variation. A two dimensional example that illustrates the different vectors which play a part in DE2 are shown in Fig 1.3.

Crossover

In order to increase the diversity of the parameter vectors, the vector

$$u = (u_1, u_2, ..., u_D)^T \tag{1.10}$$

$$u_j^{(G)} = \begin{cases} v_j^{(G)} for\ j = \langle n \rangle_D, \langle n+1 \rangle_D, ..., \langle n+L-1 \rangle_D \\ \left(x_i^{(G)}\right)_j \ otherwise \end{cases} \tag{1.11}$$

is formed where the acute brackets $\langle\rangle_D$ denote the modulo function with modulus D. This means that a certain sequence of the vector elements of u are identical to the elements of v, the other elements of u acquire the original values of $x_i^{(G)}$. Choosing a subgroup of parameters for mutation is similar to a process known as crossover in genetic algorithm. This idea is illustrated in Fig 1.4 for D = 7, n = 2 and L = 3. The starting index n in (12)

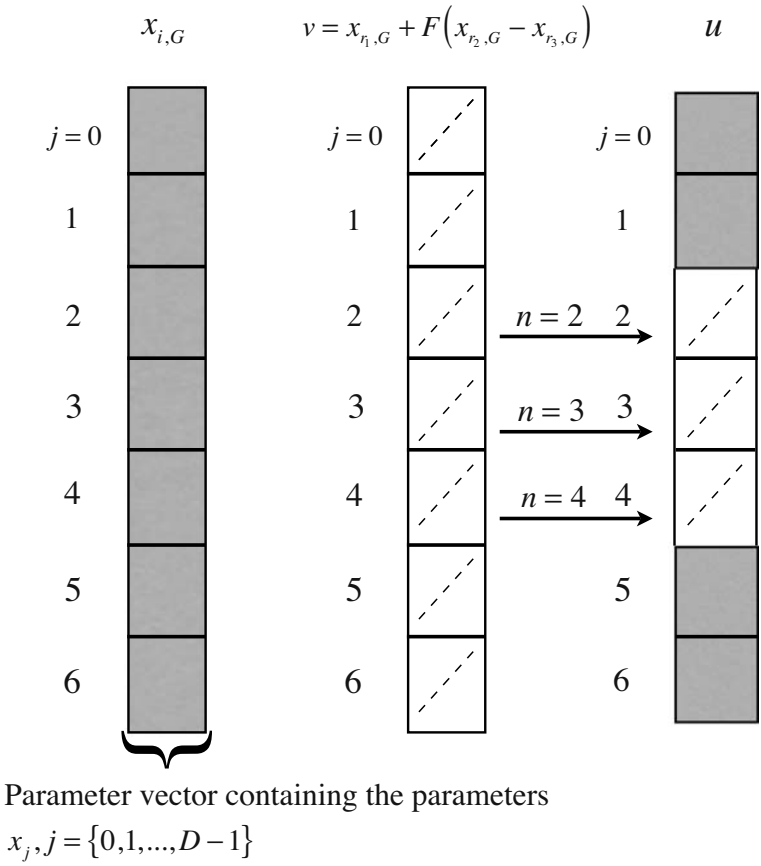

Fig. 1.4. Crossover process for D = 7, n = 2 and L = 3

is a randomly chosen integer from the interval [0, D-1]. The integer L is drawn from the interval [0, D-1] with the probability $\Pr(L = v) = (CR)^v$. $CR \in [0, 1]$ is the crossover probability and constitutes a control variable for the DE2-scheme. The random decisions for both n and L are made anew for each trial vector v.

Crossover

In order to decide whether the new vector u shall become a population member of generation *G+1*, it will be compared to $x_i^{(G)}$. If vector u yields a smaller objective function value than $x_i^{(G)}$, $x_i^{(G+1)}$ is set to u, otherwise the old value $x_i^{(G)}$ is retained.

Scheme DE3

Basically, scheme DE3 works the same way as DE2 but generates the vector v according to

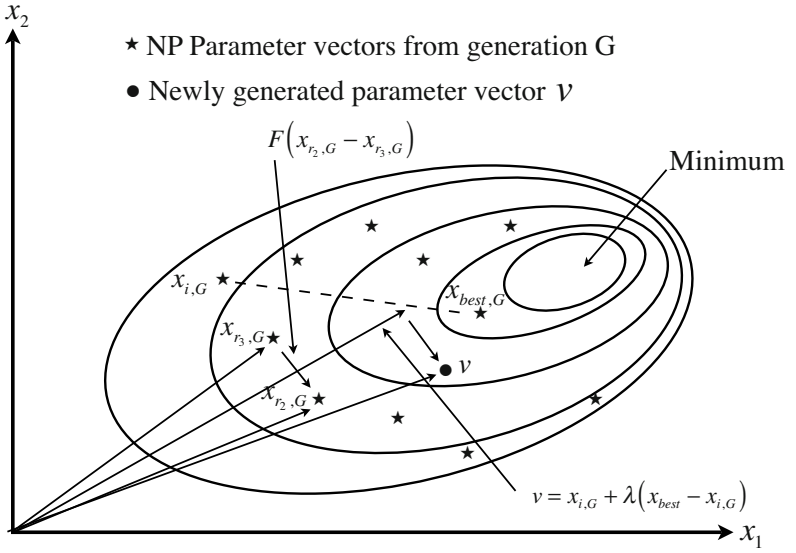

Fig. 1.5. Contour lines and the process for generating v in scheme DE3.

$$v = x_i^{(G)} + \lambda \bullet \left(x_{best}^{(G)} - x_i^{(G)}\right) + F \bullet \left(x_{r2}^{(G)} - x_{r3}^{(G)}\right) \tag{1.12}$$

introducing an additional control variable λ. The idea behind λ is to provide a means to enhance the greediness of the scheme by incorporating the current best vector $x_{best}^{(G)}$. This feature can be useful for non−critical objective functions. Fig 1.5 illustrates the vector−generation process defined by Equation 1.12. The construction of u from v and as well as the decision process are identical to DE2.

DE Strategies

[1] have suggested ten different working strategies of DE and some guidelines in applying these strategies for any given problem (see Table1.2). Different strategies can be adopted in the DE algorithm depending upon the type of problem for which it is applied. The strategies can vary based on the vector to be perturbed, number of difference vectors considered for perturbation, and finally the type of crossover used.

The general convention used above is as follows: DE/x/y/z. DE stands for differential evolution algorithm, x represents a string denoting the vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z is the type of crossover being used. Other notations are exp: exponential; bin: binomial). Thus, the working algorithm outline by [2] is the seventh strategy of DE, that is, DE/rand/1/bin. Hence the perturbation can be either in the best vector of the previous generation or in any randomly chosen vector. Similarly for perturbation, either single or two vector differences can be used. For perturbation with a single vector difference, out of the three distinct randomly chosen vectors, the weighted vector differential of any two vectors

Table 1.2. DE Strategies

Strategy	Formulation
Strategy 1: DE/best/1/exp:	$v = x_{best}^{(G)} + F \bullet \left(x_{r2}^{(G)} - x_{r3}^{(G)}\right)$
Strategy 2: DE/rand/1/exp:	$v = x_{r1}^{(G)} + F \bullet \left(x_{r2}^{(G)} - x_{r3}^{(G)}\right)$
Strategy 3: DE/rand-to-best/1/exp	$v = x_{i}^{(G)} + \lambda \bullet \left(x_{best}^{(G)} - x_{i}^{(G)}\right) + F \bullet \left(x_{r1}^{(G)} - x_{r2}^{(G)}\right)$
Strategy 4: DE/best/2/exp:	$v = x_{best}^{(G)} + F \bullet \left(x_{r1}^{(G)} + x_{r2}^{(G)} - x_{r3}^{(G)} - x_{r4}^{(G)}\right)$
Strategy 5: DE/rand/2/exp:	$v = x_{r5}^{(G)} + F \bullet \left(x_{r1}^{(G)} + x_{r2}^{(G)} - x_{r3}^{(G)} - x_{r4}^{(G)}\right)$
Strategy 6: DE/best/1/bin:	$v = x_{best}^{(G)} + F \bullet \left(x_{r2}^{(G)} - x_{r3}^{(G)}\right)$
Strategy 7: DE/rand/1/bin:	$v = x_{r1}^{(G)} + F \bullet \left(x_{r2}^{(G)} - x_{r3}^{(G)}\right)$
Strategy 8: DE/rand-to-best/1/bin:	$v = x_{i}^{(G)} + \lambda \bullet \left(x_{best}^{(G)} - x_{i}^{(G)}\right) + F \bullet \left(x_{r1}^{(G)} - x_{r2}^{(G)}\right)$
Strategy 9: DE/best/2/bin	$v = x_{best}^{(G)} + F \bullet \left(x_{r1}^{(G)} + x_{r2}^{(G)} - x_{r3}^{(G)} - x_{r4}^{(G)}\right)$
Strategy 10: DE/rand/2/bin:	$v = x_{r5}^{(G)} + F \bullet \left(x_{r1}^{(G)} + x_{r2}^{(G)} - x_{r3}^{(G)} - x_{r4}^{(G)}\right)$

is added to the third one. Similarly for perturbation with two vector differences, five distinct vectors other than the target vector are chosen randomly from the current population. Out of these, the weighted vector difference of each pair of any four vectors is added to the fifth one for perturbation.

In exponential crossover, the crossover is performed on the D (the dimension or number of variables to be optimized) variables in one loop until it is within the CR bound. For discrete optimization problems, the first time a randomly picked number between *0* and *1* goes beyond the CR value, no crossover is performed and the remaining D variables are left intact. In binomial crossover, the crossover is performed on each the D variables whenever a randomly picked number between *0* and *1* is within the CR value. Hence, the exponential and binomial crossovers yield similar results.

1.3 Differential Evolution for Permutative−Based Combinatorial Optimization Problems

The canonical DE cannot be applied to discrete or permutative problems without modification. The internal crossover and mutation mechanism invariably change any applied value to a real number. This in itself will lead to infeasible solutions. The objective then becomes one of transformation, either that of the population or that of the internal crossover/mutation mechanism of DE. A number of researchers have decided not to modify in any way the operation of DE strategies, but to manipulate the population in such a way as to enable DE to operate unhindered. Since the solution for a population is permutative, suitable conversion routines are required in order to change the solution from integer to real and then back to integer after crossover.

Application areas where DE for permutative-based combinatorial optimization problems can be applied include but not limited to the following:

Table 1.3. Building blocks and the enhanced versions of DE

Building Blocks	Enhanced DE versions	Chapter
Forward/Backward Transformation Approach	Enhanced DE (EDE)	3
Relative Position Indexing Approach	HPS	4
Smallest Position Value Approach	-	5
Discrete/Binary Approach	-	6
Discrete Set Handling Approach	DE DSH	7

1. Scheduling: Flow Shop, Job Shop, etc.
2. Knapsack
3. Linear Assignment Problem (LAP)
4. Quadratic Assignment Problem (QAP)
5. Traveling Salesman Problem (TSP)
6. Vehicle Routine Problem (VRP)
7. Dynamic pick-and-place model of placement sequence and magazine assignment in robots

In this book, some methods for realizing DE for permutative-based combinatorial optimization problems that are presented in succeeding chapters are as follows:

1. Forward/Backward Transformation Approach: [chapter 3];
2. Relative Position Indexing Approach: [chapter 4];
3. Smallest Position Value Approach: [chapter 5];
4. Discrete/Binary Approach: [chapter 6]; and
5. Discrete Set Handling Approach: [chapter 7].

While the above−listed foundations have been presented in the book, it should be mentioned that a number of enhancement routines have been realized which are based on these fundamental building blocks. For example, the enhanced DE (EDE) is based on fundamentals of the forward/backward transformation approach presented in chapter 3. This philosophy threads throughout the book and should be borne in mind when reading the chapters. Consequently, we have the building blocks and the enhanced versions of DE listed in Table 1.3.

1.4 Conclusions

This chapter has discussed and differentiated both the continuous space DE formulation and the permutative-based combinatorial DE formulation and shown that these formulations compliment each other and none of them is complete on its own. Therefore we have shown that this book complements that of [2] and vice versa. Taken together therefore, both books will be needed by practitioners and students interested in DE in order to have the full potentials of DE at their disposal. In other words, DE as an area of optimization is incomplete unless it can deal with real-life problems in the areas of continuous space as well as permutative-based combinatorial domain.

At least five DE permutative-based combinatorial optimization approaches that have proved effective have been presented in this book and they have been used to solve real-life problems. The results obtained are found to be quite competitive for each approach presented in chapters 3 to 7. Some of these approaches have become the building blocks for realizing higher-order or enhanced version of DE permutative-based combinatorial optimization approaches. Some of these enhanced versions have presented in some of the chapters. Their results are better in terms of quality than the basic versions; and some cases computation times have been drastically reduced when these enhanced version are used for solving the same real-life problems which the basic versions are used for.

References

1. Price, K., Storn, R.: Differential evolution homepage (2001) (Cited September 10, 2008), http://www.ICSI.Berkeley.edu/~storn/code.html
2. Price, K., Storn, R., Lampinen, J.: Differential Evolution. Springer, Heidelberg (2005)
3. Storn, R., Price, K.: Differential evolution − a simple and efficient adaptive scheme for global optimization over continuous spaces, Technical Report TR-95-012, ICSI March 1999 (1995) (available via ftp), ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.ps.Z