*hyp*DE: A Hyper-Heuristic Based on Differential Evolution for Solving Constrained Optimization Problems

José Carlos Villela Tinoco and Carlos A. Coello Coello

Abstract. In this paper, we present a hyper-heuristic, based on Differential Evolution, for solving constrained optimization problems. Differential Evolution has been found to be a very effective and efficient optimization algorithm for continuous search spaces, which motivated us to adopt it as our search engine for dealing with constrained optimization problems. In our proposed hyper-heuristic, we adopt twelve differential evolution models for our low-level heuristic. We also adopt four selection mechanisms for choosing the low-level heuristic. The proposed approach is validated using a well-known benchmark for constrained evolutionary optimization. Results are compared with respect to those obtained by a state-of-the-art constrained differential evolution algorithm (CDE) and another hyper-heuristic that adopts a random descent selection mechanism. Our results indicate that our proposed approach is a viable alternative for dealing with constrained optimization problems.

1 Introduction

Heuristics have been a very effective tool for solving a wide variety of real-world problems having a very large and little known search space. In its origins, research on heuristics spent a great deal of efforts in designing generic heuristics that were meant to be superior to the others in all classes of problems. This effort radically switched after the publication of the *No Free Lunch Theorems* for search in the 1990s [16]. This work provided a mathematical proof of the impossibility to design a heuristic that can be better than all the others in all classes of problems. This led to a different type of research in which the focus switched to analyzing the strenghts

José Carlos Villela Tinoco · Carlos A. Coello Coello
CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Computación, Av. IPN No. 2508
Col. San Pedro Zacatenco, México, D.F. 07360, MÉXICO
e-mail: ccoello@cs.cinvestav.mx

O. Schütze et al. (Eds.): EVOLVE – A Bridge between Probability, AISC 175, pp. 267–282.
springerlink.com © Springer-Verlag Berlin Heidelberg 2012

and limitations of heuristics in particular classes of problems, aiming to identify the cases in which a certain type of heuristic may be better than others. These studies naturally led to the idea of combining the efforts of different heuristics into a single scheme. The motivation here would be to compensate the weaknesses of one heuristic with the strenghts of another one in a certain type of problem. From the different research proposals in this direction, one of the most promising has been that of the *hyper-heuristics*, in which the idea is to design an approach that uses a control mechanism for selecting from among several possible low-level heuristics. The main motivation of hyper-heuristics is to release the user from the burden of selecting a particular heuristic for the problem at hand (something that tends to be cumbersome). Although hyper-heuristics have been mainly used in combinatorial optimization problems [5] we adopt this same framework here for constrained continuous optimization.

In this paper, we propose a new hyper-heuristic based on Differential Evolution (DE) variants which is aimed to solve constrained optimization problems in which the decision variables are real numbers. The main contribution of this work is a new selection mechanism designed to coordinate the different Differential Evolution models incorporated into the proposed hyper-heuristic.

The remainder of this paper is organized as follows. In Section 2, we provide a short description of the Differential Evolution algorithm. In Section 3, we briefly describe the origins of hyper-heuristics and their core idea. The previous related work is discussed in Section 4. Our proposed approach is provided in Section 5. Our experimental results are presented in Section 6. Finally, Section 7 presents our main conclusions and some possible paths for future research.

2 Differential Evolution

Differential Evolution (DE) was proposed in 1995 by Kenneth Price and Rainer Storn in 1995 as a new heuristic for optimization of nonlinear and non-differentiable functions [15]. In DE, the decision variables are assumed to be real numbers, and new solutions are generated by combining a parent with other individuals. The main DE algorithm can be defined based on the following concepts:

(i) **The population:**

$$P_{x,g} = (\mathbf{x}_{i,g}) \;\; i = 0,1,\ldots,NP-1, \; g = 0,1,\ldots,G_{\max} \tag{1}$$

$$\mathbf{x}_{i,g} = [x_0, x_1, \ldots, x_{D-1}]^T$$

where NP denotes the maximum number of vectors that make up the population, g is the generation counter, $G_{\max}$ is the maximum number of generations and D is the number of decision variables of the problem.

(ii) **Mutation operator:**

$$\mathbf{v}_{i,G+1} = \mathbf{x}_{r_1,G} + F\left(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}\right) \ , r_1 \neq r_2 \neq r_3 \neq i \tag{2}$$

where r_1, r_2 and $r_3 \in [1, NP]$ are randomly selected vectors. $F > 0$ is a real value that controls the amplification of the difference vector and $\mathbf{x}_{r_1,G}$ is the base vector.

(iii) **Crossover operator:**

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{if } U(0,1) \leq Cr \text{ or } j = j_{\text{rand}} \\ x_{ji,G} & \text{otherwise} \end{cases} \tag{3}$$

where $Cr \in [0,1]$ is the crossover constant which has to be determined by the user and j_{rand} is a randomly chosen index $\in 1, 2, \ldots, D$.

(iv) **Selection operator:**

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G+1} & \text{if} f\left(\mathbf{u}_{i,G+1}\right) < f\left(\mathbf{x}_{i,G}\right) \\ \mathbf{x}_{i,G} & \text{otherwise} \end{cases} \tag{4}$$

To decide whether or not a solution should become a member of generation $G+1$, the vector $\mathbf{u}_{i,G+1}$ is compared to the vector $\mathbf{x}_{i,G}$; if $\mathbf{u}_{i,G+1}$ yields a smaller objective function value than $\mathbf{x}_{i,G}$, then $\mathbf{x}_{i,G+1}$ takes the value $\mathbf{u}_{i,G+1}$; otherwise, the old value is retained.

The main DE variants are named using the following notation: *DE/x/y/z*, where x represents the base vector to disturb, y is the number of pairs of vectors that are to be disturbed and z is the type of recombination to be adopted [11]. Algorithm 17 shows variant of DE called *DE/rand/1/bin*, which is the most popular in the specialized literature. *rand* indicates that the base vector to be disturbed is chosen at random and *bin* means that binomial recombination is adopted.

Algorithm 1 Differential Evolution algorithm in its DE/rand/1/bin variant

```
G ← 0
Initialize P_{x,G}
while Termination criterion not satisfied do
    for i ← {0,...,NP−1} do
        Select r1, r2, r3 ∈ {0,...,NP−1} randomly, where r1 ≠ r2 ≠ r3
        Select j_rand ∈ {0,...,D−1} randomly
        for j ← {1,...,D−1} do
            if U[0,1] < Cr or j = j_rand then
                u_{i,j} ← x_{r3,j,G} + F(x_{r1,j,G} − x_{r2,j,G})
            else
                u_{i,j} ← x_{i,j,G}
        if f(u_i) ≤ f(x_{i,G}) then
            x_{i,G+1} ← u_i
    G ← G+1
```

3 Hyper-Heuristics

The use of heuristics for the solution of high complexity problems has become very popular in recent years, mainly because of their flexibility, efficacy and ease of use. However, this popularity has simultaneously fostered the development of a wide variety of heuristics. Such a diversity of methods makes it difficult to select one for a particular problem. Additionally, there are very few studies that attempt to identify the main advantages or disadvantages of a heuristic with respect to others, in a particular problem (or class of problems).

The term *hyper-heuristic* was originally introduced by Cowling et al. [5] to refer to approaches that operate at a higher level of abstraction than conventional heuristics. Additionally, a hyper-heuristic is capable of identifying which low-level heuristic needs to be used at a certain moment. In other words, hyper-heuristics operate in the **space of available heuristics** while heuristics work directly on the **space of solutions** of the problem [14]. Thus, a generic procedure for a hyper-heuristic is the following [2]:

(i) **Step 1.** Start with a set H of heuristics each of which is applicable to a problem state and transforms it into a new problem state.
(ii) **Step 2.** Let the initial problem state be S_0.
(iii) **Step 3.** From the state S_i of the problem, find the most appropriate heuristic to transform the problem to the next state (S_{i+1}).
(iv) **Step 4.** If the problem has been solved, stop. Otherwise, go to Step 3.

The main aim of hyper-heuristics is to provide a general framework that can offer good quality solutions for a larger number of problems. This suggests that a hyper-heuristic that has been developed for a particular problem could be easily extended to other domains by simply replacing the set of low-level heuristics and the evaluation function [4]. There is, of course, a well-defined interface between the hyper-heuristic and its low-level heuristics in order to achieve this objective. This interface must consider the following aspects:

(i) The interface should be standard, that is, only one interface is required to communicate the hyper-heuristic to the set of heuristics; otherwise, it will require a separate interface for each heuristic.
(ii) The interface should facilitate its portability to other domains. When requiring to solve a new problem, the user only has to supply all the low-level heuristics and the corresponding evaluation function.

4 Previous Related Work

Hyper-heuristics have been mainly used in combinatorial optimization, and their use in continuous optimization problems is still rare (if we consider constrained problems, then their use is even more scarce). The only previous related work

that we found is the paper from Biazzini et al. [1] in which they proposed a distributed hyper-heuristic for solving unconstrained continuous optimization problems. These authors adopted an island model and distributed several low-level heuristics throughout the islands. The authors concluded that their proposed approach produced results that were more consistent than those obtained by any of the low-level heuristics adopted, when considered in an independent manner. Biazzini et al. [1] adopted six DE models, a particle swarm optimizer and a random sampling algorithm. Each island was assigned a population of size NP and implemented the following seven selection mechanisms:

(i) **StatEq**.- assigns a heuristic to each island at the beginning of the run and does not change this assignment anymore.
(ii) **DynEq**.- assigns a random heuristic to each island after each cycle, where one cycle within an island represents the generation of one new solution using a heuristic.
(iii) **Tabu**.- corresponds to an adaptation to the hyper-heuristic proposed in [3] and it runs this algorithm on each of the islands that make up the hyper-heuristic.
(iv) **SDigmo and DDigmo**.- assigns a probability of selecting each heuristic based on the performance of each of the algorithms so that, after that probability is computed, a heuristic can be assigned to each of the nodes (islands).
(v) **Pruner**.- initially uses the entire collection of available algorithms, but as the search proceeds, it removes more and more algorithms from this set and does not consider them anymore.
(vi) **Scanner**.- the algorithms are sorted based on the latest solutions they have found so far and defines a minimum number of consecutive executions for each heuristic in each island.

However, as mentioned above, the work of Biazzini et al. [1] does not include a constraint handling mechanism. This is precisely the issue that we address here: how to design a hyper-heuristic for constrained continuous optimization using DE variants as our low-level heuristics.

5 Our Proposed Approach

As indicated before, we propose here a new hyper-heuristic for constrained optimization (in continuous search spaces), based on the use of different DE variants. Table 1 shows each of the DE variants used as low-level heuristics for our proposed hyper-heuristic.

We adopted a selection mechanism that aims to incorporate some type of knowledge for choosing the low-level heuristics to be applied at any given time. For designing such a selection mechanism, it was necessary to identify some of the characteristics of each of the DE variants. This led us to implement a random descent mechanism [5] to identify the behaviors and characteristics of the different DE variants when applied to unconstrained optimization problems (a constraint-handling

Table 1 Set of low-level heuristics used by our proposed hyper-heuristic

Models with *exp* **recombination**		**Models with** *bin* **recombination**	
h_1	DE/best/1/exp	h_7	DE/best/1/bin
h_2	DE/rand/1/exp	h_8	DE/rand/1/bin
h_3	DE/current-to-best/1/exp	h_9	DE/current-to-best/1/bin
h_4	DE/best/2/exp	h_{10}	DE/best/2/bin
h_5	DE/rand/2/exp	h_{11}	DE/rand/2/bin
h_6	DE/current-to-rand/1/exp	h_{12}	DE/current-to-rand/1/bin

mechanism is incorporated later on). Figure 1 shows the behavior of the DE variants adopted when using two different values of Cr. Here, we can observe that this parameter plays an important role on the performance of each DE variant. In fact, we found out as well that the type of recombination that performed better was related to the value of Cr that was adopted (i.e., for certain values of Cr, either the binary or the exponential recombination performed better). Additionally, we found a correlation between the type of recombination that was more effective and the number of decision variables of the problem. Then, for certain combinations of these elements (Cr value, recombination type, and dimensionality), we found out that a particular DE variant performed better. All of these results were found in our experimental study, but the details are omitted here due to space constraints.

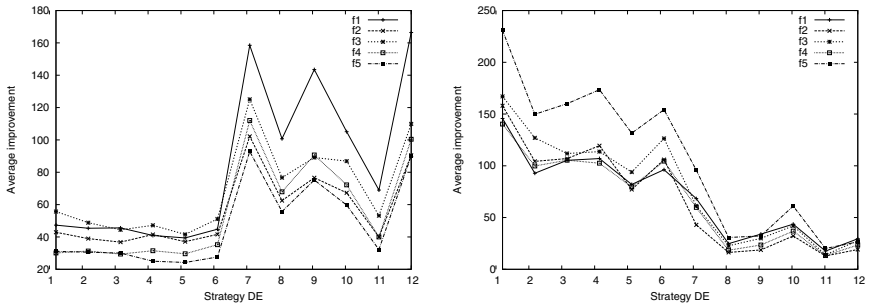

Fig. 1 Total successful steps for each model when using: a) $Cr = 0.2$ and b) $Cr = 0.8$

Based on the experimental results previously indicated, our hyper-heuristic consists of two phases. The first phase is responsible for selecting the type of recombination to be adopted (either *exp* or *bin*). In order to do this, we randomly vary at each generation g, the parameter Cr within the range $[0, 1]$ and, depending on the value of Cr that we adopt, and on the number of decision variables of the problem, we select the type of recombination to be used. Once we have decided what type of

recombination to use, the second phase consists in selecting the specific model to be applied for generating the population $P_{x,g+1}$.

For the second phase, we implemented two selection mechanisms. The first of them is called **Cr random**, and, as its name indicates, it consists of randomly choosing the DE variant to be used. The second mechanism uses a roulette wheel to choose the DE variant to be adopted. The diagram shown in Fig. 2 indicates the process for choosing the low-level heuristic to be applied.

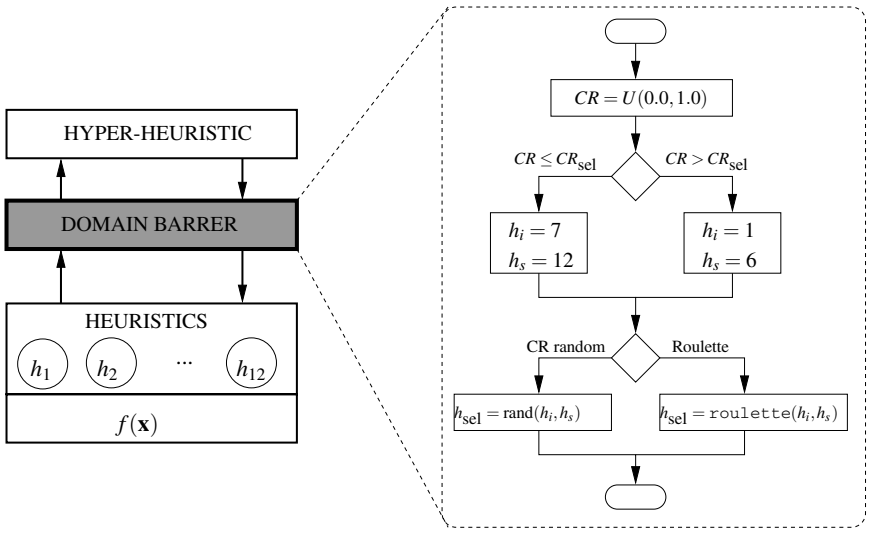

Fig. 2 Diagram that illustrates the selection mechanism of the proposed hyper-heuristic

When using a roulette wheel, a certain probability is assigned to each of the DE variants, based on their performance during the search process, so that the probability of selecting a certain DE variant is proportional to its performance. In order to mitigate the well-known bias problems of the roulette wheel selection mechanism (i.e., the worst individual may be selected several times [5]), we implemented three types of roulette wheel selection:

- **Original roulette (R1)**: this is the original algorithm proposed in [6].
- **Roulette with random init (R2)**: in this case, the initial position of the roulette wheel is randomly selected.
- **Roulette with permutation (R3)**: in this case, we create a permutation of the positions of the roulette and then, we apply the original algorithm.

Algorithm 18 shows the pseudocode of the proposed selection mechanism. Here, *mechanism* refers to the selection mechanism and Cr_{sel} is a parameter indicating the probability of selecting a DE variant either with binomial or with exponential recombination. The parameter Cr_{sel} is calculated based on the equation (5), so that

the percentage of selection of DE variants with either exponential or binomial recombination is determined based on their performance on the problem being solved.

$$Cr_{sel} = \frac{\text{Total success of variants with binomial recombination}}{\text{Total success of all variants}}$$

$$Cr_{sel} = \frac{\sum_{i=7}^{12} success_i}{\sum_{i=1}^{6} success_i + \sum_{i=7}^{12} success_i} \tag{5}$$

Algorithm 2 Selection mechanism proposed

```
Input: mechanism
Output: selected heuristic
switch mechanism do
    case RANDOM
        heuristic_sel ← rand(h_1, h_12)
    case Cr_RANDOM
        Cr ← U(0,1)
        if Cr > Cr_sel then
            /* DE variants with exponential
               recombination                    */
            heuristic_sel ← rand(h_1, h_6)
        else
            /* DE variants with binomial recombination
               */
            heuristic_sel ← rand(h_7, h_12)
    case ROULETTE
        Cr ← U(0,1)
        if Cr > Cr_sel then
            Select strategies with exponential recombination using a
            roulette-wheel
        else
            Select strategies with binomial recombination using a
            roulette-wheel
```

Finally, Algorithm 19 shows the pseudocode of the proposed hyper-heuristic. The control parameters of the proposed algorithm are the following:

- $G_{\max}$: maximum number of generations.
- NP: number of individuals in the population.
- Cr: DE's crossover constant.
- *mechanism:* type of selection mechanism adopted for the low-level heuristics incorporated within the hyper-heuristic.

It is important to note that at the beginning of the search process we do not have information about the performance of each low-level heuristic. Therefore, we require an initial *training stage*, which consists of the implementation of a maximum

number of generations in which we use the random descent selection mechanism proposed in [5] to initialize the expected values (EVs) for each of the DE variants adopted, according to the following equation:

$$EV_i = \frac{success_i}{\frac{1}{12}\sum_j^{12} success_j} \qquad i = 1, 2, \ldots, 12 \tag{6}$$

Algorithm 3 Our proposed hyper-heuristic

```
Input: NP, G_max, Cr, mechanism
G ← 0
Initialize P_{x,G}
if mechanism is Cr_RANDOM or RULETTE then
    /* Training stage                                  */
    G_aux ← 0
    while G_aux < G_proof do
        /* Apply mechanism random descent              */
        selection_mechanism()
        repeat
            apply_heuristic(heuristic_sel)
            G ← G + 1
            G_aux ← G_aux + 1
        until not being able to improve the previous solution;
    Initialize the expected value of each low-level heuristic
while Termination criterion not satisfied do
    selection_mechanism()
    if mechanism DESCENT then
        repeat
            apply_heuristic(heuristic_sel)
            G ← G + 1
        until not improve the previous solution;
    else
        apply_heuristic(heuristic_sel)
        G ← G + 1
```

5.1 Handling Constraints

Differential Evolution was designed for solving unconstrained optimization problems. Thus, it is necessary to incorporate to it a constraint-handling scheme for dealing with constrained optimization problems. One of the most popular constraint-handling techniques used with evolutionary algorithms has been Stochastic Ranking (SR) [12]. SR adopts a rank selection mechanism that tries to balance the influence of considering either the objective function value or the degree of violation of the constraints (a parameter called P_f is adopted for this sake). SR has been successfully

incorporated into DE before. For example, [7] showed that when embedding SR into DE, this constraint-handling mechanism can provide information to the mutation operator about the most appropriate direction of movement. This produces, indeed, a speed up in the convergence of the algorithm. Figure 3 shows two hypothetical examples for the search directions to be considered: a) the movement of an infeasible point to the feasible region, and b) the movement of a feasible point to the infeasible region. This illustrates that it is possible to guide the search in such a way that we can generate new solutions either closer or farther away from the feasible region $\mathscr{F}$ (corresponding to cases a) and b) in Figure 3, respectively).

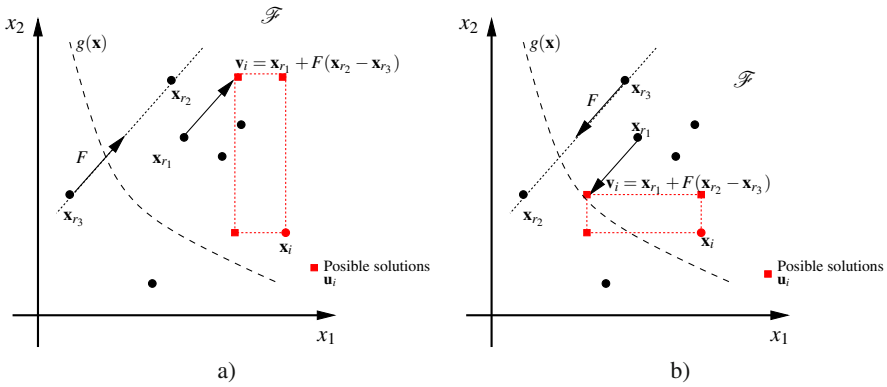

Fig. 3 Example of the two types search direction: a) towards the feasible region $\mathscr{F}$ b) towards the infeasible region

Stochastic Ranking sorts the individuals in the population, and the rank of each individual (i.e., its position in the sorted list) is used to guide the search in a specific direction. In order to select the search direction, we adopt a probabilistic value that regulates the type of movements performed (either to move a solution into the feasible region or towards the infeasible region) and its use aims to explore the boundary between the feasible and the infeasible region. We experimentally found that this probabilistic value provided good results when set with the same value used by the P_f parameter of SR.

Based on the previous discussion, we modified the selection operator of DE using the following criteria:

- If the two solutions being compared are feasible, the one with the best objective function value is chosen.
- Otherwise, we choose the solution with the lowest degree of constraint violation.

Algorithm 20 shows the process carried out to generate, from the current population $P_{x,G}$, the new population $P_{x,G+1}$. The objective function is denoted by f and the degree of constraint violation is denoted by ϕ. Here, ϕ is given by:

$$\phi(\mathbf{x}) = \sum_{i=1}^{m} \max\{0, g_i(\mathbf{x})\}^2 + \sum_{j=1}^{p} |h_j(\mathbf{x})|^2 \tag{7}$$

where $g_i(\mathbf{x})$ correspond to the inequality constraints and $h_j(\mathbf{x})$ denote the equality constraints of the problem.

Algorithm 4 Pseudocode of our proposed DE approach for constrained optimization

```
Input: strategy of DE to use
Result: generation G+1
begin
    Apply Stochastic Ranking to rank the population P_{x,G}
    for i ← {1,...,NP} do
        /* Selecting vectors to disturb                    */
        if U(0,1) < P_f then
            /* To the infeasible region                    */
            r_1 ← rand(1, α)
            r_2 ← rand(α+1, NP)
        else
            /* To the feasible region                      */
            r_1 ← rand(α+1, NP)
            r_2 ← rand(1, α)
        Select vector r_3 according to the DE strategy to be used:
        /* Mutation                                        */
        v_{k.G} = x_{k,G,r3} + F (x_{k,G,r1} − x_{k,G,r2})
        /* Recombination                                   */
        Apply recombination operator according to the DE strategy adopted in
        order to obtain u_i
        /* Selection                                       */
        if φ(u_i) = φ(x_{i,G}) = 0 then
            if f(u_i) < f(x_{i,G}) then
                x_{i,G+1} ← u_i
            else
                x_{i,G+1} ← x_{i,G}
        else
            if φ(u_i) < φ(x_{i,G}) then
                x_{i,G+1} ← u_i
            else
                x_{i,G+1} ← x_{i,G}
```

6 Results

In order to validate the performance of our proposed approach, we adopted several standard test functions from the specialized literature on evolutionary constrained optimization [12]. Our results were compared with respect to those generated by the approach called **CDE** (Constrained Differential Evolution) [10] which is representative of the state-of-the-art on DE-based constrained optimization. Additionally, we also compared results with respect to the hyper-heuristic with a random descent mechanism proposed in [5]. The hardware and software platform adopted for our experiments, as well as the parameters adopted are the following:

(i) PC configuration:
System: Linux Ubuntu 10.04
CPU: Intel Pentium Dual Core Inside T2080 (1.73 GHz)
RAM: 1024 MB
Programming Language: C (gcc 4.4.3 compiler)

(ii) Parameters:
Maximum number of generations: $G_{\max} = 6000$
Population size: $NP = 50$
$F : U(0.3, 0.9)$ each generation
$Cr : U(0, 1)$ each generation
P_f : 0.50 for *g06* and *g11*, and 0.45 for the remaining test problems

From the parameters adopted, it can be seen that all the approaches perform 300,000 objective function evaluations. Our experimental study comprised 100 independent runs per algorithm per problem. In Tables 2, 3 and 4 we show the results obtained by the CDE algorithm, the random descent mechanism and our proposed approach, using the three roulette-wheel selection mechanisms previously indicated (**R1** corresponds to the original roulette-wheel, **R2** corresponds to roulette-wheel with random initial position and **R3** corresponds to the roulette-wheel with permutation). From these results, it can be noticed that not all the approaches required the maximum number of generations to reach the best known result. That is the reason why in Table 5 we show the average number of generations in which the algorithm converges as well as the minimum number of generations that each algorithm required to find the best known solution for each test problem.

6.1 Analysis of Results

From the results presented in Tables 2, 3 and 3, we can see that the selection mechanism based on a roulette-wheel with random initial position (**R2**) obtained good results in most of the test problems. These results also indicate that the algorithm **CDE** obtained very poor results in the test problems *g03, g05, g10* and *g13*. We can also observe that the hyper-heuristic approaches were unable to solve *g11* in a proper way, which is not the case of the CDE algorithm.

Table 2 Statistics with respect to the $f(\mathbf{x})$ obtained by CDE and the random descent mechanism. μ corresponds to the mean values, σ to the standard deviation and M to the best solution found in each case. **BKS** indicates the best known solution for each test problem. We show in **boldface** those cases in which an approach reached the best known solution for that particular test problem.

	BKS	**CDE**			**R. descent**		
		μ	σ	M	μ	σ	M
g01	-15.000	-13.93461	1.23×10^{0}	-14.99999	-11.25425	2.64×10^{0}	-12.000
g02	-0.803619	-0.8033878	6.83×10^{-5}	-0.8033884	-0.784641	3.27×10^{-2}	-0.7930787
g03	-1.000	0.24616	9.50×10^{-2}	-1.000	**-1.000**	5.13×10^{-9}	-1.000
g04	-30665.539	**-30665.539**	0.00×10^{0}	-30665.539	**-30665.539**	0.00×10^{0}	-30665.539
g05	5126.498	5315.60	3.01×10^{2}	5126.498	5195.899	2.20×10^{2}	5126.500
g06	-6961.814	**-6961.814**	0.00×10^{0}	-6961.814	-7229.388	1.41×10^{3}	-7818.326
g07	24.3062091	24.78596	3.12×10^{-1}	24.3062091	24.31361	6.98×10^{-2}	24.30623
g08	-0.095825	-0.09583	1.11×10^{-3}	-0.095826	**-0.095825**	0.00×10^{0}	-0.095825
g09	680.6301	**680.630**	0.00×10^{0}	680.630	**680.6301**	0.00×10^{0}	680.6301
g10	7049.250	7090.50762	3.99×10^{2}	7085.876	7094.163	8.22×10^{1}	7049.396
g11	0.750	**0.750**	0.00×10^{0}	0.750	0.9505711	9.51×10^{-2}	1.000
g12	-1.000	**-1.000**	0.00×10^{0}	-1.000	**-1.000**	0.00×10^{0}	-1.000
g13	0.053950	0.80852	1.87×10^{-1}	0.2476	0.3678503	1.64×10^{-1}	0.4394295

Table 3 Statistics with respect to the $f(\mathbf{x})$ values obtained for a random choice of Cr and for **R1**. μ corresponds to the mean values, σ to the standard deviation and M to the best solution found in each case. **BKS** indicates the best known solution for each test problem. We show in **boldface** those cases in which an approach reached the best known solution for that particular test problem.

	BKS	Cr **random**			**R1**		
		μ	σ	M	μ	σ	M
g01	-15.000	-11.82917	2.72×10^{0}	-12.000	-14.81745	2.72×10^{0}	-15.000
g02	-0.803619	-0.8009026	6.16×10^{-3}	-0.8036145	-0.8015735	4.63×10^{-3}	-0.803613
g03	-1.000	**-0.9999999**	3.09×10^{-8}	-0.9999999	**-0.9999997**	7.55×10^{-7}	-0.9999999
g04	-30665.539	**-30665.539**	0.00×10^{0}	-30665.539	**-30665.539**	0.00×10^{0}	-30665.539
g05	5126.498	5171.077	1.42×10^{2}	5126.498	5170.813	1.42×10^{2}	5126.498
g06	-6961.814	**-6961.814**	0.00×10^{0}	-6961.814	**-6961.814**	0.00×10^{0}	-6961.814
g07	24.3062091	24.30722	7.70×10^{-3}	24.306210	24.30708	3.42×10^{-4}	24.30705
g08	-0.095825	**-0.095825**	0.00×10^{0}	-0.095825	**-0.095825**	0.00×10^{0}	-0.095825
g09	680.6301	**680.6301**	0.00×10^{0}	680.6301	**680.6301**	0.00×10^{0}	680.6301
g10	7049.250	7103.163	8.81×10^{1}	7049.63	7104.965	8.83×10^{1}	7049.783
g11	0.750	0.8992142	1.19×10^{-1}	1.000	0.901635	1.18×10^{-1}	1.000
g12	-1.000	**-1.000**	0.00×10^{0}	-1.000	**-1.000**	0.00×10^{0}	-1.000
g13	0.053950	0.3227286	2.22×10^{-1}	0.4388694	0.3071761	1.98×10^{-1}	0.4388515

On the other hand, Table 5 shows that the selection mechanism based on a roulette-wheel with random initial position (**R2**) required a lower number of iterations than the others in ten of the thirteen test problems adopted. This mechanism also had the lowest average number of generations in seven of the test problems adopted. This confirms that this selection mechanism had the best overall performance from all the approaches that were compared in our experimental study.

Table 4 Statistics with respect to the $f(\mathbf{x})$ values obtained for **R2** and **R3**. μ corresponds to the mean values, σ to the standard deviation and M to the best solution found in each case. **BKS** indicates the best known solution for each test problem. We show in **boldface** those cases in which an approach reached the best known solution for that particular test problem.

	BKS	**R2**			**R3**		
		μ	σ	M	μ	σ	M
g01	-15.000	**-15.000**	0.00×10^{0}	-15.000	-14.8200	2.72×10^{0}	-15.000
g02	-0.803619	-0.8014436	4.67×10^{-3}	-0.8014834	-0.8014504	5.91×10^{-3}	-0.8036136
g03	-1.000	**-0.9999997**	7.05×10^{-7}	-0.9999999	**-0.9999996**	1.04×10^{-6}	-0.9999999
g04	-30665.539	**-30665.539**	0.00×10^{0}	-30665.539	**-30665.539**	0.00×10^{0}	-30665.539
g05	5126.498	5171.097	1.42×10^{2}	5126.498	5171.311	1.43×10^{1}	5170.288
g06	-6961.814	**-6961.814**	0.00×10^{0}	-6961.814	**-6961.814**	0.00×10^{0}	-6961.814
g07	24.3062091	**24.30649**	7.54×10^{-4}	24.30627	24.30705	$3.31 \times 10^{-}3$	24.307
g08	-0.095825	**-0.095825**	0.00×10^{0}	-0.095825	**-0.095825**	0.00×10^{0}	-0.095825
g09	680.6301	**680.6301**	0.00×10^{0}	680.6301	**680.6301**	1.00×10^{-5}	680.6301
g10	7049.250	7103.024	8.82×10^{2}	7049.556	7106.478	8.92×10^{1}	7049.664
g11	0.750	0.8997895	1.18×10^{-1}	1.000	0.9016549	1.18×10^{-1}	1.000
g12	-1.000	**-1.000**	0.00×10^{0}	-1.000	**-1.000**	0.00×10^{0}	-1.000
g13	0.053950	0.3132858	2.13×10^{-1}	0.4388565	0.3144292	2.11×10^{-1}	0.4388585

Table 5 Average and minimum number of generations required for each algorithm to reach the best known solution to each of the test problems adopted.

	CDE		**R. Desc.**		***Cr* random**		**R1**		**R2**		**R3**	
	μ	min	μ	min	μ	min	μ	min	μ	min	μ	min
g01	614.88	543	725.93	714	727.84	563	722.37	763	**549.32**	**541**	720.14	701
g02	**5843.77**	5954	5963.33	5934	5960.73	5979	5939.29	5954	5950.26	**5907**	5975.90	5995
g03	**4134.24**	3152	5719.55	4217	5693.44	4218	5650.55	4174	5682.42	**3128**	5598.37	3505
g04	675.33	476	872.34	758	887.19	738	867.98	749	**510.26**	**449**	868.86	719
g05	**3456.54**	1652	4395.93	1786	4293.48	3059	4448.92	1411	3855.19	**1346**	4316.96	1613
g06	1162.60	**154**	1100.83	435	**931.24**	423	1092.50	445	1070.48	291	965.39	496
g07	5945.12	**4130**	5973.51	5726	**5762.79**	5834	5766.33	5730	5903.23	4976	6783.88	5757
g08	165.75	98	214.59	100	163.16	97	161.65	97	**161.61**	**78**	162.18	102
g09	2621.86	1068	3878.70	1481	3520.84	1581	3518.51	1374	**2393.24**	**1028**	3417.16	1384
g10	5954.63	5921	5920.25	5975	5921.15	5985	5943.33	**4895**	**5875.74**	4987	5907.12	4989
g11	536.14	367	457.28	302	401.55	302	417.25	302	**166.14**	**230**	391.43	466
g12	201.72	163	158.95	134	163.48	119	161.76	129	**159.32**	**119**	172.54	132
g13	**4119.88**	4003	4577.57	3876	5184.46	2529	5007.46	3906	5208.23	**2258**	5136.14	3111

7 Conclusions and Future Work

We have proposed here a new hyper-heuristic for solving constrained optimization problems. The proposed approach uses as its low-level heuristics a set of twelve differential evolution variants. Additionally, the selection mechanism of differential evolution was modified in order to make it able to handle constraints (stochastic ranking was adopted for this sake).

The results obtained by our proposed approach are very promising, since they are better than those produced by a state-of-the-art DE-based evolutionary optimization approach (CDE). This indicates that the mechanism adopted by our hyper-heuristic is working in a proper way.

As part of our future work, we aim to improve the selection mechanism of our hyper-heuristic. In order to achieve that, it is required to perform a more in-depth study of the different DE variants adopted here, so that we can understand in a better way how they work when dealing with constrained optimization problems. We are also interested in adding to our hyper-heuristic other low-level heuristics such as particle swarm optimization [9] and evolution strategies [13], since we believe that such approaches perform search movements that could complement those produced by differential evolution. Evidently, the goal of adding more heuristics would be to improve the performance of our hyper-heuristic.

Acknowledgements. The second author acknowledges support from CONACyT project 103570.

References

1. Biazzini, M., Bánhelyi, B., Montresor, A., Jelasity, M.: Distributed hyper-heuristics for real parameter optimization. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 1339–1346. ACM, Montréal Québec (2009)
2. Burke, E., Hart, E., Kendall, G., Newall, J.: Hyper-Heuristics: An Emerging Direction In Modern Search Technology, handbook of metaheuristics edn., ch. 16, pp. 457–474. Springer, New York (2003)
3. Burke, E., Kendall, G., Soubeiga, E.: A tabu-search hyper-heuristic for timetabling and rostering. Journal of Heuristics 9(6), 451–470 (2004)
4. Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent Developments. SCI, vol. 136, pp. 3–29. Springer, Berlin (2008)
5. Cowling, P.I., Kendall, G., Soubeiga, E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)
6. De Jong, K.A.: An analysis of the behaviour of a class of genetic adaptive systems. Ph.D. thesis, University of Michigan (1975)
7. Fan, Z., Liu, J., Sorensen, T., Wang, P.: Improved differential evolution based on stochastic ranking for robust layout synthesis of mems components. IEEE Transactions On Industrial Electronics 56(4), 937–948 (2008)
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Company, Reading (1989)
9. Kennedy, J., Eberhart, R.C.: Swarm Intelligence. Morgan Kaufmann Publishers, San Francisco (2001)
10. Lampinen, J.: Constraint handling approach for the differential evolution algorithm. In: Proceedings of the Congress on Evolutionary Computation 2002 (CEC 2002), vol. 2, pp. 1468–1473. IEEE Service Center, Piscataway (2002)
11. Price, K.: An introduction to differential evolution. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 79–106. McGraw-Hill (1999)
12. Runarsson, T.P., Yao, X.: Stochastic ranking for constrained evolutionary optimization. Transactions On Evolutionary Computation 4(3), 284–294 (2000)
13. Schwefel, H.P.: Evolution and Optimum Seeking. John Wiley & Sons, New York (1995)

14. Storer, R., Wu, S., Vaccari, R.: Problem and heuristic search space strategies for job shop scheduling. ORSA Journal on Computing 7, 453–467 (1995)
15. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Tech. Rep. TR-95-012, International Computer Science Institute (1995)
16. Wolpert, D., MacReady, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997)