Design of hybrid metaheuristics for real-world continuous optimization problems

Mokhtar Essaid

IRIMAS UNIVERSITÉ HAUTE-ALSACE
Institut de Recherche en Informatique, Mathématiques, Automatique et Signal

ÉCOLE DOCTORALE 269

PhD thesis presented by :

Mokhtar ESSAID

November 4th 2019

Field: Computer science

Design of hybrid metaheuristics for real-world continuous optimization problems

Thesis supervised by: Pr. Lhassane Idoumghar, University of Haute Alsace (Supervisor)
Dr. Julien Lepagnot, University of Haute Alsace (Co-advisor)
Dr. Mathieu Brévilliers, University of Haute Alsace (Co-advisor)

Referees: Ammar Oulamara, Professor, University of Lorraine, France
Cyril Fonlupt, Professor, University of Littoral Cote d'OPAL, France

---

Examiners: Clarisse Dhaenens, Professor, University of Lille, France.
Hongying FEI, Professor, University of Shanghai, China.
Daniel Fodorean, Professor, University of Cluj-Napoca, Romania.
Edward Keedwell, Professor, University of Exeter, England.

Acknowledgments

I would like to express my gratitude to my supervisors Prof. Lhassane Idoumghar, Dr. Julien Lepagnot and Dr. Mathieu Brévilliers for motivation and support throughout my research program. I feel honored to be able to work under their guidance. Finally, I would like to thank my family: my father, my mother, my brother and sisters for supporting me spiritually throughout this study and supporting me in my whole life in general. I also owe to my grandmothers, who rest in peace. I attribute all my success in life to the moral, intellectual and physical education I received from them. This one is for you both!

And last but not least, I would like to thank my friends: Yacine, Mohamed, Tayeb, Amir, Hafnaoui, Abdelhak, Kamel and Amine, who have been always with me and helped me in my worst days during these three years.

Publications

1- Papers published/accepted for publication in journals

- M. Essaid, L. Idoumghar, J. Lepagnot and M. Brévilliers. GPU parallelization strategies for metaheuristics: a survey. International Journal of Parallel, Emergent and Distributed Systems, 1-26. Online Jan. 2018.

- M. Essaid, L. Idoumghar, J. Lepagnot and M. Brévilliers, D. Fodorean, A dimension-based adaptive differential evolution for optimization problems. Journal of Soft Computing. Publisher: Springer, impact factor: 2.367 (accepted for publication).

2- Communications published/submitted at international conferences with program committee and Proceedings:

- M. Essaid, L. Idoumghar, J. Lepagnot and M. Brévilliers, D. Fodorean, An eigenvector-enhanced parallel adaptive differential evolution for electric mo-

tor design. International Conference on Tools with Artificial Intelligence ICTAI, November 4-6, 2019, Portland, Oregon. (submitted).

- M. Essaid, M. Brévilliers, J. Lepagnot L. Idoumghar and D. Fodorean, Hybrid parameter adaptation strategy for differential evolution to solve real-world problems IEEE International Congress on Evolutionary Computation 2019, Wellington, New Zealand. Rank A

- M. Essaid, L. Idoumghar, M. Brévilliers, J. Lepagnot, D. Fodorean, A hybrid differential evolution algorithm for real world problems. In: 2018 IEEE Congress on Evolutionary Computation (CEC). IEEE, pp. 2341-2347, Rio de janeiro, Brasil, July 2018 Rank A.

- M. Essaid, L. Idoumghar, M. Brévilliers, J. Lepagnot, D. Fodorean, A Hybrid Optimization Algorithm for Electric Motor Design. In: International Conference on Computational Science. Springer, Cham. p. 501-517. Wuxi, China, 2018. Rank A

- M. Essaid, L. Idoumghar, J. Lepagnot and M. Brévilliers, D. Fodorean A Parallel Adaptive Differential Evolution Algorithm for Electric Motor Design. 7th International Conference on Metaheuristics and Nature Inspired Computing (META18) Marrakech, Morocco, October 2018.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Engineers and decision makers are daily confronted with problems that can be involved in several technical fields such as image processing, chemistry, biology and mechanics. Often, these problems can be modeled as optimization problems: an objective function is defined to be maximized or minimized with the respect of a set of decision variables. According to [9], optimization problems can be classified as: combinatorial, continuous, mono-objective, multi-objective, static, dynamic, with or without constraints.

Indeed, several algorithms have been proposed to solve optimization problems in the last decades such as exact optimization methods [94], where optimal solutions are guaranteed to be found. Several exact methods have been proposed as the dynamic programming, branch and bound algorithm and constraint programming. Unfortunately, due to the high computational time, these methods are only effective when small instances of optimization problems are handled [94]. For

this reason, metaheuristics have been introduced to provide near optimal solutions within a reasonable time. Indeed, they can be defined as approximated solution methods which insure interaction between local and global procedure improvement to escape local optima. Thanks to their simple implementation, metaheuristics can be easily adopted to a wide range of optimization problems. They can be classified into two categories.

- Population-based metaheuristics, that start the search with an initial set of solutions. Then, the set is evolved using an ensemble of search operators. As examples of population-based metaheuristics, we can cite Genetic algorithms (GA) [32], particle swarm optimization (PSO) [43], differential evolution (DE) [78].

- Single-solution-based metaheuristics which start with only one solution. Then, the solution is evolved by introducing the concept of neighborhood, that is the algorithm chooses one of the actual solution neighbors based on a predefined selection criterion. Numerous proposals have been introduced within this category such as tabu search (TS) [31] and simulated annealing (SA) [45].

According to the no free lunch theorem, no metaheuristic is able to solve all the existent problems to the optimality. However, researchers are paying their attention to propose new optimization architectures in order to provide relatively resilient algorithms. Following the current state-of-the-art metaheuristics, it can be seen that designing new metaheuristics relies on three axes:

1- Hybridization of two or more metaheuristics: hybridization can be an effective choice, because it allows to benefit from the advantages of sevral algorithms [6]. Generally speaking, hybridization combines the algorithmic components of two or more metaheuristics in order to balance exploration and exploitation capabilities of the combined metaheuristics. Several hybridization models have been proposed as:

- Collaborative hybrids, where two or more metaheuristics are performed in multi-stage, sequentially or in parallel. These algorithmic scenarios usually starts with a metaheuristic favoring exploration. Then, the solutions are provided to an exploitative metaheuristic to improve them using local search procedures.

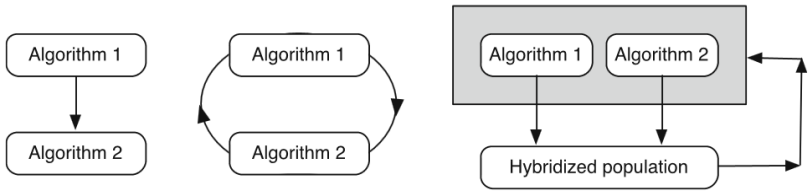


Figure 1.1: Collaborative framework of hybrid algorithm, depicting multi-stage, sequential, and parallel structures [85]

- Integrative hybridization, where a search operator is integrated in a metaheuristic to enhance its capability.

2- Adaptation/tuning of metaheuristic parameters: the parameter values of a given algorithm can have a great influence on the final results [10]. Sometimes, fixed values tend to be ineffective due to the complex landscapes of optimization

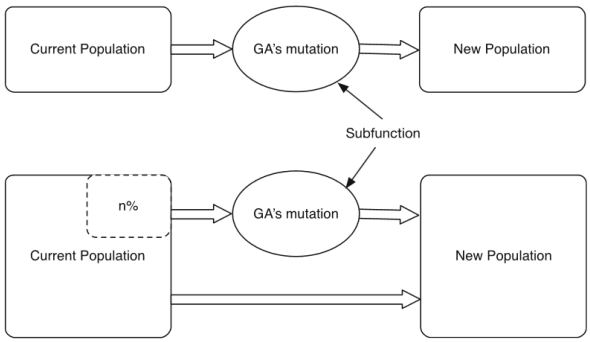


Figure 1.2: Integrative structure of a hybrid algorithm[85]

problems. Several proposals have been introduced to tune optimization algorithms such as iRace software [5], or to auto-adapt them for each iteration based on their successful history [10]. We are interested in new self-adaptation strategies where a given algorithm try to find by itself parameter values that are as much as possible appropriate to the considered problem.

3- The integration of learning approaches: the integration of machine learning techniques into existing algorithms has become a very important area of research. Indeed, several algorithms have exploited these techniques on different contexts, such as improving the initial solutions quality, incorporating historical search experience, classifying the current solutions and the intelligent choice between the search operators [101]. In addition, many problems are very expensive to solve (in terms of computational time) because of the complexity of the objective function. However, it is possible to reduce the computational time by using surrogate models instead of systematically using the objective function.

Following these considerations may increase the computational time. More-

over, the complexity and the high dimensionality of optimization problems can be a serious burden when evaluating solutions. In the attempt to remediate these issues, new hardware/software technologies have been proposed to reduce the computational time:

- Distributed algorithms, where independent tasks are synchronously/asynchronously performed within different machines or different interconnected processors (cluster). Indeed, distributed algorithms can be implemented using different technologies. For instance, Message passing interface (MPI), which is an interface that insures communication between nodes (machines or processors) and manages the computation specified for each node.

- Massively parallel algorithms, where the algorithms are fully/partially implemented using graphics processing units (GPU). The tasks of parallel algorithms can be executed within kernels, which represent parallel computation routines. Afterwards, the kernels are launched in order by CPU. Thanks to the cheap cost and the simple interfaces of GPUs, it has become relatively straight-forward to implement parallel metaheuristics. GPUs frameworks are single instruction/multiple data architectures, which is appropriate in the context of metaheuristics. They can leverage efficiently large instances of optimization problems (parallel evaluation of solutions). Besides, search operators of algorithms can be easily handled in parallel thanks to their general independent nature.

This thesis mainly focuses on proposing new algorithms that rely on the three axes mentioned above. Besides, a parallel counterpart is provided when a serious

computational time is noticed. Our proposals are applied on a proposed engineering problem. In order to validate the results of the proposed algorithms, different real-world applications from the literature are optimized

Chapter 2 is devoted to survey several state-of-the-art of optimization algorithms in the context of continuous optimization. First, well-known metaheuristics are covered, such as cuckoo search and differential evolution. Secondly, different hybridization and parallelization models are presented. Finally, a set of self-adaptive differential evolution proposals are discussed to show the advantage of parameter adaptation strategies.

Chapter 3 defines the test suites and the engineering problem used in the experimentation. Moreover, a technical definition of the engineering problem is given to show its advantage as a real-world application.

Chapter 4 represents the contribution of this thesis by by proposing five algorithms. Each algorithm is explained in detail revealing its performance compared to recent state-of-the-art algorithms, where statistical tests are performed to show the significance superiority of our proposals compared to several powerful optimization algorithms, which are surveyed in chapter 3. Finally, a comparison of some proposals is conducted.

Chapter 5 concludes the thesis showing how the proposals are relevant to its main axes. Besides, an ensemble of potential perspectives are given to show the possible extensions of this thesis.

Chapter 2

Related Work

Metaheuristics are solution methods that ensure an interaction between local improvement procedure (local search) and high level strategies (global search). This interact on allows to escape from the local optima and achieves a certain balance between exploration and exploitation. These approaches transit from one solution/one population to another by applying a set of search operators and a predefined selection criterion.

2.1 State-of-the-art of metaheuristics

Generally speaking, metaheuristics are not able to guarantee the optimality of the final solutions compared to the exact methods. However, an appropriate adjustment of their algorithmic components can achieve satisfying results for a wide range of problems.

In the following subsections, several metaheuristics are explained.

2.1.1 Differential evolution

Differential evolution (DE) is a population-based metaheuristic which starts the search process with random initial solutions (population). Then, solutions are evolved using three search operators: mutation, crossover, selection. These search operators are applied until a termination criterion is met. DE phases can be organized as follows:

1. Initialization: mutation parameter F, crossover parameter CR, the population size PS, the problem dimension D, and the number of generations G_{max} are initialized. The initial individuals are randomly initialized.

2. Mutation: for each solution x_i^G in the parent population, a mutant vector v_i^{G+1} is computed as follows:

$$v_i^{G+1} = x_{r1}^G + F.(x_{r2}^G - x_{r3}^G) \tag{2.1}$$

   where $r1$, $r2$ and $r3$ are distinct randomly generated integers within the range [1, PS] and different from the index i.

3. Binomial crossover: for each individual x_i^G, a trial vector u_i^{G+1} is generated as follows:

$$u_i^{G+1} = \begin{cases} v_{i,j}^{G+1} & \text{if } j = \sigma_j \quad \text{or } R_j < CR \\ x_{i,j}^G & otherwise \end{cases} \tag{2.2}$$

   where σ_j is a random integer generated within the range [1,D], and R_j is randomly generated number within the range [0,1].

4. Evaluation: The trial vector is evaluated and replaces the parent individual if it has a lower fitness (minimization).

5. G is incremented and phases 2 to 5 are repeated while G is less than G_{max}.

It should be stated that different mutation and crossover strategies have been proposed to improve DE performance.

1- Mutation: several mutation strategies have been proposed for DE, and the most popular among them are listed below:

DE/rand/2 [70]:

$$v_i^{G+1} = x_{r1}^G + F.(x_{r2}^G\text{-}x_{r3}^G\ ) + F.(x_{r4}^G\text{-}x_{r5}^G\ )$$

DE/best/2 [77]:

$$v_i^{G+1} = x_{best}^G + F.(x_{r1}^G\text{-}x_{r2}^G\ ) + F.(x_{r3}^G\text{-}x_{r4}^G\ )$$

DE/current-to-best/1 [77]:

$$v_i^{G+1} = x_i^G + F.(x_{best}^G\text{-}x_i^G\ ) + F.(x_{r1}^G\text{-}x_{r2}^G\ )$$

DE/current-to-best/2 [70]:

$$v_i^{G+1} = x_{best}^G + F.(x_{best}^G\text{-}x_i^G\ ) + F.(x_{r1}^G\text{-}x_{r2}^G\text{+}x_{r3}^G\text{-}x_{r4}^G\ )$$

where x_{best}^G is the best individual in the population at generation G and r_1,r_2,r_3 and r_4 are randomly generated numbers within the discrete range [1,PS]

2- Besides to the binomial crossover, several crossover operators have been proposed such as:

- Exponential crossover [105]: First, a randomly generated integer n is generated among [1, D]. n represents the starting point where the crossover or exchange of components with the donor vector starts. Another randomly integer L from the numbers in [1, D] is also generated. L represents the number of components the donor vector contributes to the target vector. The exponential crossover is depicted as follows:

$$u_i^{G+1} = \begin{cases} v_{i,j}^{G+1} & \text{if } j = n, j = n+1, \quad j = n+2, ..., j = n+L-1 \\ x_{i,j}^{G} & otherwise \end{cases} \tag{2.3}$$

- The arithmetic recombination [69]: the trial vector can be produced as a combination of the target vector and a donor vector as follows:

$$u_i^{G+1} = x_i^G + k_{i,j}(v_i^G - x_i^G) \tag{2.4}$$

where $k_{i,j}$ is a scalar combination coefficient generated between [0,1].

The algorithmic structure of DE is depicted in Algorithm 1

2.1.2 Cuckoo search

Cuckoo Search (CS) is a population based metaheuristic designed to solve continuous optimization problems. CS is inspired by the brood parasitism behaviour of cuckoo birds [96], where the three following rules are used:

- Each cuckoo lays one egg at a time, and leaves its egg in randomly chosen nest.

Algorithm 1 Pseudo-code of DE.

```
Generate initial population pop of PS individuals
while stopping criterion is not met do
   popold ← pop
   for Each individual i in the population do
      Generate a mutated vector V_i /*mutation step*/
      Generate a trial vector U_i using V_i and popold_i /*crossover step*/
      if f(popold_i) > f(U_i) then
         pop_i ← U_i
      end if
   end for
end while
```

- The best nests with high quality of eggs will be kept to the next generations.
- The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p \in [0, 1]$. If the host bird discovers the strange egg, it throws the egg away or it abandons the nest and builds completely new one.

Based on these three rules, the basic steps of CS are summarized in Algorithm 2. In CS, a balanced combination of local random walk and a global random walk is obtained through a switching parameter Pa. The local random walk is written as:

$$x_i^{t+1} = x_i^t + s \oplus H * (Pa - \epsilon) \oplus (x_j^t - x_k^t) \tag{2.5}$$

where x_j^t, x_k^t are solutions randomly selected and H is heaviside function. ϵ and s are random numbers generated from a uniform distribution. The global random

Algorithm 2 Pseudocode of the Cuckoo Search (CS).

```
Generate initial population of n host nests x_i (i=1,2, ..., n)
while (t < MaxGeneration) or (stopping criterion is not met) do
    Get a cuckoo randomly by Lévy flights
    Evaluate its fitness F_i
    Choose a nest among n (say,j) randomly
    if F_i > F_j then
        Replace j by the new solution
    end if
    A fraction Pa of worse nests are abandoned and new ones are built
    Keep the best solutions
    Rank the solutions and find the current best solution
end while
```

walk is handled using Levy flights as follows:

$$x_i^{t+1} = x_i^t + \alpha \oplus levy(\lambda) \tag{2.6}$$

where

$$\alpha = \alpha_0 \oplus (x_j^t - x_i^t) \tag{2.7}$$

$$levy(\lambda) = \frac{u}{|v|^{\frac{1}{\alpha}}} \tag{2.8}$$

α_0 is a step size scaling factor and α is levy Flight exponent. Finally, u and v are two predefined numbers with zero means and associated variance.

CS has shown to be competitive ahead well-known metaheuristics such GA and PSO [96]. Besides, CS has been exploited to optimize several combinatorial

problems such as in [47, 61, 65]

2.1.3 CMA-ES

The Covariance Matrix Adaptation Evolution Strategy (CMAES) is a an evolutionary algorithm where a new population is produced by sampling from a probability distribution constructed during the search process [37]. CMAES is explained in Algorithm 3. In CMAES, solutions are generated using a multivariate normal

Algorithm 3 Pseudocode of CMAES.

```
λ ← number of samples per iteration
μ ← number of recombination points
Initialize state variables m, σ, C = I, p_σ = 0, p_c = 0
while Stopping criterion is not met do
    for i=1 to λ do
        x_i^{t+1} ← sample i^{th} solution according to (2.9)
        f_i ← evaluate i^{th} solution
    end for
    Sort the new solutions and find the first μ solutions
    m_{t+1} ← update the mean value according to (2.10)
    p_c^{t+1} ← update anisotropic evolution path according to (2.12)
    C_{t+1} ← update the covariance matrix according to (2.13)
    p_σ^{t+1} ← update isotropic evolution path according to (2.14)
    σ^{t+1} ← update the step size using isotropic path length according to (2.15)
end while
```

distribution N with mean m and a covariance C. A new solution x^{t+1} is generated as follows:

$$x^{t+1} = m^t + \sigma^t N(0, C^t) \tag{2.9}$$

$$m^t = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^t \tag{2.10}$$

$$w_i = log(\mu + 1/2) - log(i), \sum_{i=1}^{\mu} w_i = 1 \tag{2.11}$$

where m^t is the weighted mean of the μ best solutions, $x_{i:\lambda}^t$ is the t^{th} ranked individual, λ is the number of samples, σ^t is the step size parameter. Besides, a covariance matrix C^t is adapted using an evolution path p_c^{t+1}. It is generated with the following equation:

$$p_c t + 1 = (1 - c_c)p_c^t + \sqrt{c_c(2 - c_c)}\frac{\sqrt{\mu}}{\sigma t}(m^{t+1} - m^t) \tag{2.12}$$

$$C^{t+1} = (1 - c_{cov})C^t + c_{cov}p_c^{t+1}(p_c^{t+1})^T \tag{2.13}$$

where c_c and $c_{cov} \in [0,\ 1]$ are learning rates for p_c^{t+1} and C^{t+1} respectively. The step size parameter is updated using the evolution path p_σ^{t+1} as follows:

$$p_\sigma^{t+1} = (1 - c_\sigma)p_\sigma^t + \sqrt{c_\sigma(2 - c_\sigma)}\sqrt{\mu}B^t m^{t+1} \tag{2.14}$$

where c_σ is a learning rate controller, and B^t is the normalized eigenvectors of C^t. Then, σ^{t+1} is updated as follows:

$$\sigma^{t+1} = \sigma^t exp(\frac{||p_\sigma^{t+1}|| - T_n}{d_\sigma T_n}) \tag{2.15}$$

$$T_n = \sqrt{n}(1 - \frac{1}{4n} + \frac{1}{21n^2}) \tag{2.16}$$

where n represents the problem dimension and $d_\sigma > 1$ is a damping parameter.

2.1.4 Hybrid metaheuristics for continuous optimization

Hybrid metaheuristics are considered as one of the major contributions in the field of optimization. According to [6], the main objective of hybridization is to exploit the complementary aspect of metaheuristics. However, choosing the best synergy is the key to attain the best results, which is not a straight-forward task. Indeed, finding the best combination of algorithmic components requires optimization expertises. Moreover, since the hybridization itself is an optimization algorithm, it can not be guaranteed to solve all the optimization problems. In other words, one hybridization design can be successful for one problem and can work poorly for another. Even though, several hybridization models have shown to be effective in tackling a wide range of applications. It should be stated that there are hundreds of similar hybridizations proposed in the literature. However, in this subsection, we only cover some works in the hope to give the reader a general idea about the designs proposed so far. [6] Generally speaking, population-based hybridization models are frequently used. Population-based hybridization methods apply an explorative population based on a global search procedure to identify the promising regions of the search space. Then, a local search procedure is performed to quickly obtain better solutions. However, there are other prominent examples which will be illustrated in this sub-section.

A real-world application called digital filter design problem has been optimized by a hybrid algorithm based on DE and CMA-ES [89]. On the one hand, DE performs the global exploration and optimizes the parameters of exponential functions that define the bounded search space. On the other hand, CMA-ES is used as a local search engine. Its initial search point and boundary constraint estimates are provided by DE. Additionally, periodic feedback from CMA-ES is provided to the DE. The goal of the algorithm is to optimize the finite impulse response (FIR) filter coefficients which minimizes the error between the actual and the ideal filter frequency response.

In the same context, a different hybridization has been introduced between an adaptive version of DE and CMAES [63]. The hybridization called LSHADE-SPACMA, where a modified version of CMAES undergoes the crossover operation to improve the exploration capability of DE. The performance of LSHADE-SPACMA has been investigated on CEC 2017 test suite comparing the hybridization to CMAES and LSHADE.

A hybrid CMAES/CS algorithm has been proposed in [103]. It has been stated that the self adaptive mutation distribution and the cumulative path of CMAES can ultimately speed up the convergence rate of CS. Comparative experiments using the CEC 2008 test suite and one engineering problem of CEC 2011 has been performed. The results reveal the advantage of the hybridization compared to CS and CMAES.

Another hybridization of CMAES and CS called S-CSCMAES has been introduced in [73]. The hybridization relies on integrating the recombination operator

of CMAES into CS procedure. By using the recombinationo of CMAES (m^s), the weighted means are computed. In order to produce the new solutions, the best solutions gained from CS algorithm and m^s are combined to produce the new solutions X^s with as follows:

$$X^s = X^s_{CS} + m^s \tag{2.17}$$

In order to prove the efficiency of S-CSCMAES, a set of constrained and Unconstrained test functions are optimized revealing an encouraging performance compared to CS, CMAES, PSO, firefly algorithm.

Other hybridizations models have been introduced using other promising algorithms. For instance, in 2013, a hybridization between PSO and artificial bee colony (ABC) has been introduced in [44]. This hybridization is based on a recombination procedure, where the global best solutions of PSO and ABC are recombined. Then, the new solution is used to evolve solutions of both algorithms. The proposal has been compared using CEC 2005 test suite and an energy demand estimation problem revealing better performance compared to ABC and PSO.

A different hybridization has been proposed in [64], where the algorithmic combination consists in incorporating SaDE [71], which is a modified version of DE and a local search procedure (LS). During the search process, SaDE and the local search procedure are iteratively performed. Besides, the current best solution improved by LS is integrated into SaDE to guide its search. LS phase contains several local search methods, where each method is performed based on a probability

computed as follows:

$$P_{LS_M} = \frac{I_{LS_M}}{\sum_{m \in LS} I_{LS_m}} \tag{2.18}$$

where I_{LS_M} represents the number of improvement obtained by LS_M . This hybridization has been tested on the CEC 2015 test suite of large scale global optimization showing a competetive performance compared to DE-CC-CG [98], SACC [92].

In [25], GA, PSO and symbiotic organisms search (SOS) [13] have been incorporated. In the main loop of this hybridization, one iteration of GA (selection, mutation and crossover) is firstly performed. The output population is given in order to perform one iteration of PSO. However, PSO only updates the best experience of each solution (best local solution). In the last phase, one iteration of SOS (mutualism, commonsalism, parastism) is applied using the best experience updated by PSO. The hybrid algorithm has been tested on several unimodal and multimodal benchmark functions. Furtheromore, it has been tested on clustering problems, where it was confirmed that the proposed algorithm shows a better accuracy and error rate.

Another hybridization of GA and cross entropy method (CE) [72] has been proposed in [55]. The novel algorithm divides the population into two sub-populations. Then, one iteration of each algorithm is performed on its associated population. Finally, an elitism phase takes place. In this phase, the best solution found so far is checked whether it is present in the current population or not. If not, it will be inserted replacing the worst solution. The proposed method is tested on 24 continuous benchmark functions, with four different dimension configurations,

where GACE revealed better results compared to several modified versions of GA.

In the attempt to overcome the premature convergence of harmony search algorithm [29] (HS), SA has been proposed to complement the search procedure of HS [2]. Inspired by SA, the proposal HS-SA accepts worse solutions (harmonies in HS) considering a probability parameter called *temperature*. *temperature* value is linearly decreased to shift the set of solutions from exploration to exploitation. HS-SA has been tested on the CEC 2014 test suite and a real-world problem from computer vision field called camera calibration problem. The numerical results demonstrate the advantage of HS-SA over SA and HS.

Bat algorithm (BA) [95] is a population-based metaheuristic inspired by the echolocation process of bats to sense distance. BA may suffer sometimes from premature convergence problem because of its poor local search capability. In the attempt to improve BA performance, some modifications have been introduced in [54]. First, a chaotic initialization is performed to ensure well diversified initial population. Secondly, the time factor parameter used in BA search operators is decreased to gradually shorten the step size of bats (solutions). Finally, the modified BA is hybridized with a local search method called external optimization (EO) [8]. The statistical test results reveal that the hybridization is significantly better than GA, PSO, DE, CMAES and a modified version of DE called L-SHADE [82].

Another hybrid BA has been developed for economic dispatch problem [51], where the objective function is to minimize the operating costs for all committed generators while meeting the supply-demand balance and a set of constraints de-

fined in [50]. In this proposal, a chaotic map procedure to adjust BA parameters is incorporated to prevent premature convergence. Moreover, a random black hole model [102] is introduced in order to accelerate the convergence rate. A comparison with BA, ABC and PSO was conducted, where the proposal has shown a competitive performance. In Table 2.1, The covered hybrid algorithms are summarized.

Table 2.1: Different examples of hybrid algorithms

Ref	The algorithmic components	Benchmark
[89]	CMAES + DE	digital filter design problem
[63]	CMAES + DE	CEC 2017 test suite
[103]	CMAES + CS	CEC 2008 + one problem of CEC 2011
[73]	CMAES + CS	Welded beam design, Tension compression string design.
[44]	PSO + ABC	CEC 2005 test suite
[64]	SaDE + local search strategy	CEC 2015
[25]	GA + PSO + SOS	Clustering problems
[55]	GA + CE	24 benchmark functions from BBOB 2009
[2]	HS + SA	CEC 2014 + camera calibration problem
[54]	BA + EO	CEC 2014
[54]	BA + random black hole model	Economic dispatch problem

2.1.5 Self-adaptive DE

In the last years, It has been noticed that DE performance is sensitive to its parameters. Therefore, researchers turned their attention to propose different strategies in order to find the most suitable parameters for a given problem. DE parameters have been considered by proposing offline/online control parameters strategies. These strategies tend to tune the parameters based on trial-and-error procedures (offline), or adaptively modifying them, where a feedback from the search history is considered [1, 15].

An interesting self-adaptive DE (SaDE) has been proposed in 2006 [11]. DE parameters are adapted by collecting a feedback from the search history. Mutation scale factor F is produced for each solution using a normal distribution with a mean value 0.5 and standard deviation 0.3. The crossover parameter CR is generated using the previous successful CR values. Another DE variant called JADE has been introduced in [99]. In JADE, F and CR are generated by modifying parameters of probability distribution from which they are produced at each generation. JADE performance has been improved by proposing success-history archive [80]. The idea is to store successful values of F and CR in memory archives M_F and M_{CR}. Afterwards, using these archives to generate new F and CR close to the stored parameters. This variant is called successful history-based parameter adaptation for differential evolution (SHADE). The approach has been proposed for CEC 2013 competition, where it was ranked the first among DE variants. In 2014, the population size parameter in SHADE has been studied by [82]. A linear reduction of the population size is proposed to accelerate the convergence rate. The algorithm showed interesting results. It ranked the first in IEEE CEC 2014 competition. This proposal has been again improved in [33], where an eigenvector-based crossover is introduced. The new search operator tends to be effective when highly correlated variables are present.

Meanwhile, ensemble of parameters and mutation strategies (EPSDE) has been proposed in 2011 by [59]. EPSDE provides a pool of discrete values for F and CR, which serves in keeping the successful pair (F/CR) to the next generation. EPSDE has been investigated using IEEE CEC 2005 revealing resilient performance for the

whole benchmark. Nevertheless, the proposal has been reconsidered in [58]. The new extension finds the best parameters by modeling the issue as an optimization problem. Then, it is solved using harmony search algorithm [29].

Finally, adopting a hybridization model and/or a parameter adaptation strategy may involve a high computational time due to the extra computation. In the attempt to overcome this issue, parallel computing may be an interesting alternative to reduce the high computational time. In the following section, several GPU-based parallel metaheuristics are surveyed in order to illustrate the advantage of the parallel computing in reducing the computational time of optimization algorithms.

2.1.6 Self-adaptive CS

Similarly, CS is strognly dependent to its parameters. P_a parameter is investigated in [49], where it has been stated that P_a controls the diversity of the population. This parameter has been adapted using two new parameters setting, which can be described as follows:

$$P_a = 0.05 + 0.15 * rand \tag{2.19}$$

$$P_a = 0.85 + 0.05 * rand \tag{2.20}$$

The algorithm selects one of these equations based on their successful performance in the previous iteration. The proposal has been tested on 16 benchmark functions chosen from literature revealing superior performance compared to the original CS

and several variants of DE.

Another adaptive CS has been introduced, where α step size parameter is modified in each iteration. According to [104], the search step size has to be gradually modified during the search process. In the first iterations, the broadest possible access to information is required. Therfore, a larger step size is needed. In the last iterations, in order to improve CS exploitation, the search should be conducted in a small neighbourhood of individuals and then a small step size is needed. α is gradually decreased as follows:

$$\alpha = \alpha_{min} + \frac{FE_{max} - FE^m}{FE_{max}} * (\alpha_{max} - \alpha_{min}) \tag{2.21}$$

where FE_{max}, FE are the maximum number and the current number of evaluations respectively, α_{min}, α_{max}, α are the max, the current and the min value of step size respectively. The proposal has been tested on several benchmark functions from CEC 2005 test suite revealing competetive performance compared to CS.

In order to improve the exploitation capability of CS, the latter has been combined with a simulated annealing-based strategy to update the detection probability and step size. The new algorithm has been tested on a bus scheduling problem, where it shown optimal scheduling models compared to the original CS.

Despite these interesting strategies, it could be noticed that the parameter adaptation in the context of CS and CMAES are not deeply studied. For instance, the experimental studies of adaptive CS works were conducted only in particular real-world problems. Their performance remains not known when compared to recent state-of-the-art powerful algorithms.

2.2 State-of-the-art of parallel metaheuristics

Unfortunately, most of metaheuristics performance decreases in both terms of time complexity and effectiveness when facing high dimensional problems. In the attempt to overcome this drawback, parallel metaheuristics have been proposed as an alternative. Nowdays, parallel metaheuristics are attracting a growing interest from researchers in order to reduce the execution time and to improve the quality of the solutions found. According to [79], the parallel design of metaheuristics is classified into three classes:

1. Algorithmic level: this level allows launching many algorithms in parallel. The algorithms can run independently with different initial solutions and/or different parameters and choose the best results of the run. In this level, algorithms can be cooperative and exchange solutions in order to improve the results.

2. Iteration level: this level allows a parallelization in each iteration. It concerns the evaluation of solutions and/or the generation of the neighborhood.

3. Solution level: this level allows a deep parallelization of a single solution. For instance, the objective function or constraints for a generated solution can be implemented in parallel. The objective of achieving this level is mainly the speed up of the search.

Thanks to graphics processing units (GPU), solving high dimensional continuous problems has become straightforward. By exploiting GPU, the three levels mentioned above can be efficiently achieved. In this section, we outline a set of papers

addressed to solve continuous optimization problems. We conclude it with Table 2.2 that presents the characteristics of the implementations.

A CUDA implementation of CS has been implemented in [41]. In the parallel CS, separate threads are dedicated to individual nests. Besides, the algorithmic structure of CS requires sorting solutions to abondon the worst ones. To sort solutions, a partial parallel reduction sorting is performed. The proposal has been tested on a set of numerical functions, where a significant decrease of 25 times has been exhibited compared to the sequential version.

In 2016, a parallel implementation of PSO using GPU has been presented in [39]. The implementation has been tested on a set of continuous functions. The implementation has achieved a speedup of 46 times faster than the sequential algorithm. Their GPU proposition consists in seven kernels with a ring topology to form a virtual neighborhood for particles. The first kernel allocates memory on GPU, where each thread uses *Curand* library to generate random numbers. The second kernel initializes positions and velocity of particles. The third kernel generates ((n+32-1)/32) blocks of 32 threads to compute the fitness function (Solution level) where n is the number of individuals in the swarm. It performs a reduction process to sort the fitness values. Then, via these values, *Pbest* is updated. The fourth kernel is responsible of calculating local best *Lbest*. In this kernel, each particle compares his *Pbest* with its neighbors *Pbest* (left and right neighbors). The fifth kernel computes the velocities and positions for the next iteration(solution level/iteration level). A sixth kernel computes the global best position of the whole

swarm. The last kernel liberates the structures used on GPU. The results have shown the positive impact of coalesced memory on the acceleration. However, the speedup is decreasing when the population size is more than 2000 and dimension size is more than 50.

Obtaining a speedup of 17x, Dynamic Cooperative Hybrid MPSO+GA has been designed in [27]. MPSO+GA alternates between Multi-Swarm PSO (MPSO) and genetic algorithm (GA). The proposed design includes the following genetic operations: mutation, crossover, and selection into the update phase of PSO. Besides, particles are divided into sub-swarms forming a ring topology (algorithmic level). GA applies the three operators on the population. GA uses tournament selection to pick groups of individuals. Then, the best individual is selected within each group for crossover (each group is performed in parallel). During mutation phase, genes in individuals are mutated with a predefined probability. Moreover, a heuristic is used to change the algorithm if it does not improve the best solution found during a certain number of iterations. The parallel version of this design is expressed in 8 kernels and they are implemented as follows:

1. Particle Initialization Kernel: using one thread per particle dimension (solution level), this kernel initializes the positions and the velocities vectors (iteration level), i.e. position is randomly initialized between $[-x_{max}, +x_{max}]$ and velocity is initialized to zero.

2. Update Fitness Kernel: in this kernel, multiple threads cooperate to compute the fitness of each particle. First, each thread reads four position values from global memory, computes the values of the corresponding four terms

to perform a partial sum for them. Next, all the partial results are used to perform reduction, in order to compute the fitness value of one particle. Finally, the final fitness of all particles are written back to the global memory fitness buffer.

3. Update Bests Kernel: it uses one thread per particle. First, each thread compares the new and old local fitness for its particle in order to update the best local position. Next, the kernel performs a reduction operation to update the best global position for each swarm.

4. Update Position/Velocity: it uses one thread per dimension. Velocity and position are computed using the equations of PSO. Afterwards, mutation is performed on both position and velocity with a defined probability β. 5-Find Best/Worst Particles: this kernel performs a reduction operation to find the best and the worst particles in each sub-swarm.

5. Swap Particles Kernel: this kernel performs the exchange between sub-swarms. Each thread represents one dimension to be exchanged, and a given number of the best particles in sub-swarm j overwrite the worst particles in sub-swarm $j+1$ mod s (ring topology).

6. Mutation Restoration Kernel: this kernel is proposed to recover the fitness, velocities and positions of *unhealthy* sub-swarms if the mutation performed on the previous iteration has led to very bad results. In this kernel, each thread restores 4 dimensions of each particle after checking if the sub-swarm is *unhealthy*. To do so, it compares the current fitness with the last fitness

that was before mutation. If it is worse, old position and velocity from their saved buffers are restored.

7. Crossover and Mutation Kernel: for each particle, t/4 threads randomly select 4 particles, and perform a parallel reduction in order to find the best fitness. With a probability γ, a single point crossover is performed by randomly choosing a shared crossover point between all the threads. Besides, half of the threads reads the right half of the first parent, and the other half reads the left half of the second parent. Afterwards, one thread combines these chunks. In the same kernel, mutation is applied by all the threads to their chunks of values.

In 2015, a parallel PSO has been designed in [48] using CUDA. This design is implemented in a collection of remote computing services, called Amazon Web Services Cloud (AWS). According to the authors, the PSO computation is proportional to the size of the particle. It means that the larger the particle is, the greater the pressure is. The following phases are parallelized:

- Calculation of the fitness values of particles: one particle represents one block and each thread computes one of its dimensions (solution level/iteration level). Then, partial results are reduced to thread 0 writing the final fitness value in the global memory.

- Update best local position and best global position: since updating positions needs to update each dimension of each particle, each particle is represented

by one block, and each thread updates each dimension of the particle. Finally, the last kernel updates position and velocity of each particle.

The experimentation has been performed by comparing CPU implementation with GPU local implementation, and GPU AWS implementation. Their results show that the GPU AWS based PSO runs 80 times faster than the sequential algorithm, and 64 times faster compared to GPU local implementation.

A parallel DE algorithm has been proposed in [106], which was combined with an elite opposition-based learning strategy (EOBL). The proposed parallel EOBDE includes two parts which are DE and EOBL strategy [86]. First, a population is randomly initialized (iteration level). Then, EOBL is applied. The strategy chooses the best 20% individuals as a set of elite individuals. Then, *opposite* solutions are generated according to a model mentioned in [106]. Finally, the best individuals are inserted in the population. The algorithm handles these operations by implementing three kernels: the first kernel finds maximum and minimum values of each dimension. The second kernel generates opposite solutions of the elite individuals. The last one selects the best individuals to insert them into the next population. To perform this task in parallel, 2 x NP individuals are assigned to each thread block (iteration level). Each individual is compared with other 2 x NP -1 individuals to calculate its rank value in order to find the members of the new population. Afterwards, DE is applied. To perform DE operators, individuals are represented by threads within a separate kernel. This Algorithm has been tested on 10 functions with dimensions 500 and 1000. Besides, it has been compared with four algorithms. The results show that EOBDE achieves the

best results compared to the other algorithms. Besides, EOBDE shows an average speedup of 4.475x compared to the sequential implementation.

A hybridization between DE and Backtracking Search Optimization Algorithm (BSA) and Simulated Annealing (SA) has been proposed in [12]. The algorithm consists mainly in two stages. The first stage consists in five phases. The first, called Selection-I is a backtracking strategy to store the old population of the previous generation (history). This populations is replaced by the current population with a probability of 0.5. Afterwards, the mutation phase takes place, where a hybrid equation is proposed by combining mutation equations of both BSA and DE/target-to-best/1 in which a SA schedule is proposed to decrease the scaling factor. Next, two crossover strategies are randomly used (with probability 0.5) to generate a new trial population T from the current and the mutant population. The first strategy depends on a parameter that controls how many dimensions of the mutant will be incorporated in the trial individual. The second strategy ensures that only one dimension from the mutant individual will be concerned in the new trial individual. Finally, in Selection-II phase, T_i replaces an individual P_i if it is better. The second stage performs a DE/target-to-best/1 iteration on the worst individual. In their GPU based implementation, for the most of the phases (for example, mutation, evaluation) the algorithm assigns to each individual a block (iteration level), and to each dimension a thread (solution level) to compute it (kernel of N blocks of D threads). However, sometimes another data decomposition is needed: for example, a thread is assigned to each individual in order to update the global best solution.

According to [88], detecting objects in images is a well-known problem in computer vision and pattern recognition. This problem can be modeled as a continuous optimization problem, as it is proposed in [88]. A parallel PSO and DE have been proposed to tackle two problems in this field: hippocampus localization in histological images and human body pose estimation in video sequences. The objective of human body pose estimation in video sequences is to estimate accurately the posture of human body in a video stream. In this problem, the input is N views of the body from several angles. Afterwards, the silhouette of the body within each image is extracted. The silhouette is a binary image where all pixels belonging to the body are set to 1. To solve the problem, three steps are followed. First, a pose estimation is generated by the search algorithm using an appropriate parametric model. Then, a 3D rendering of the body is applied for the pose. Finally, a set of N images, corresponding to the projections of the rendered body (silhouettes) on the image planes of the input is computed. For further details about the parametric model used, we refer the reader to [88].

In CUDA-based implementations of PSO and DE, three kernels are implemented. For PSO, the first kernel initializes and updates the velocity and position of all particles. The second kernel evaluates the fitness. Finally, the third kernel updates the best positions. In DE, the first kernel generates the offspring solutions. The second kernel evaluates the fitness of all produced solutions. Then, the third kernel performs the selection of the new population. PSO and DE have the same structure, each thread block is responsible of one particle (iteration level), where each thread updates one dimension of the problem (solution level). The experi-

mentation shows that PSO gives more accurate results than DE when dealing with human body pose estimation. However, DE gives slightly better results in case of hippocampus localization in histological images without mentioning any details about the speedup gained by GPU implementation over CPU.

A new memetic algorithm, called MA-SW-Chains is presented in [46] for GPU architecture. Its main idea is to combine a steady-state genetic algorithm (SSGA) [93] with a local search procedure, called *Solis Wets* search method [76]. The steps that have been parallelized in the algorithm are: evaluation of the fitness function, adaptation of the crossover operator to the GPU, optimization of the local search, random number generation process, and population sorting.

1. Fitness function evaluation: each thread block processes a set of dimensions of each individual (iteration level). Within each block, each thread processes several variables (bucket). Each processed bucket is composed of interleaved elements to make thread warps access consecutive elements (coalesced memory). Afterwards, a reduction operation is performed to compute the partial result for each thread. The final fitness function of an individual is then computed by another reduction operation.

2. Crossover: each thread processes a bucket of pairs of dimensions (solution level). Then, it writes the result in the memory. In fact, the crossover operation is designed such that each thread crosses $BucketSize$ elements of the two parent individuals. It means that each block generates $ThreadsPerBlock * BucketSize$ elements for the new individual. In the proposed design, n crossing operations are performed in parallel to increase the thread parallelism

where n is the number of individuals. Another operation has been parallelized which is the Euclidean distance to select the parents to be crossed.

3. Local search: the operations that have been parallelized are: individual change operations, bias values increment and decrement operations, individual substitution in the population, and fitness evaluation.

4. The generation of random numbers and population sorting.

A parallel GA has been proposed in [75] to solve continuous functions. In their proposition, the selection procedure is implemented using Roulette wheel selection function with a separate kernel, and it is performed by generating random numbers between 0 and the sum of the fitness values of the population. If the fitness of the corresponding individual is greater than the random number, then it becomes a parent chromosome. Afterwards, another kernel performs a uniform distribution crossover with a fixed ratio. Unlike single and double point crossover, where mixing is performed at segment level, uniform distribution crossover creates child chromosome at gene level (solution level). It was stated that it is more suitable for the large populations. Finally, mutation is implemented in a single kernel, where each individual is mutated by a thread (iteration level). The parallel GA has been tested on seven test functions and it shows to be faster with 4.15x than the sequential version.

A parallel bee algorithm (CUBA) has been proposed in [56]. CUBA is a multi-colony bee algorithm that obtains a good efficiency and a high speedup. The algorithm initializes the population and evaluates the fitness of individuals through

parallel threads (iteration level/solution level). To find the best sites, the population is sorted using OddEven Sorting algorithm. In this proposal, bees are grouped into colonies. Each thread is assigned to its colony according to the thread ID. In the standard bee algorithm, more bees are recruited for the best sites. However, in this proposition, the authors aims to balance the loading among the threads. They have proposed to assign *nep* bees to recruit m sites. The algorithm overcomes the overhead due to the communication between the colonies by using shared memory and adapting 2 phase communication strategy. CUBA has been applied on 9 minimization functions, and has achieved a speedup of 13x times compared to the standard sequential bee algorithm.

Table 2.2: Characteristics of GPU-accelerated metaheuristics on continuous problems

Ref	Algorithm	Parallelism level	Acceleration	Benchmark	Quality improvement
[41]	CS	Iteration level	20x	Functions from CEC 2005	No improvement
[106]	DE	Iteration level	4.475x	Functions from CEC 2008 [84], and [38]	Results improved compared to CHC, DE, SOUPDE and GODE
[12]	DE-BSA-SA	Iteration level + Solution level	40x	Functions taken from [14]	No improvement
[88]	DE+PSO	Iteration level + Solution level	Not mentioned	4 test sequences made by the CVSSP, University of Surrey for Human body pose estimation + 15 images of hippocampi by manually segmenting the anatomical structures for Hippocampus localization in histological images	PSO performs better in human body pose estimation in video sequences but DE is better in hippocampus localization in histological images
[46]	MA-SW chains	Iteration level + Solution level	82.17x	CEC 2010 [83] + a benchmark setup mentioned in [46]	No improvement
[39]	PSO	Iteration level + Solution level	46x	Sphere, Rosenbrock, Rastrigin, Griewank, Ackley, De Jong, Easom	Quality improved in Sphere and Griewank function compared to the CPU sequential implementation
[75]	GA	Iteration level	1.18 to 4.15x	Function taken from [40]	No improvement
[27]	PSO	Algorithmic level+Iteration level + Solution level	17x	CEC 2010 [83]	Quality improved compared to Static MPSO+GA and CPU sequential implementation but it doesn't achieve the best results of MPSO-MCS
[56]	CUBA	Iteration level + Solution level	13x	Functions from [68]	No improvement
[48]	PSO	Iteration level + Solution level	80x	Sphere, Rastrigin, Griewank, Rosenbrock	No improvement

2.3 Discussion and conclusion

In this chapter, state-of-the-art of hybrid and parallel metaheuristics were defined. Besides, a brief review of adaptive DE proposals was considered to show the importance of the parameters on the overall performance of DE.

It should be stated that there are hundreds of algorithms proposed in the literature thanks to the variety and explosion of real-world problems. However, we only covered some works to demonstrate the prominent designs of hybrid and parallel metaheuristics. It has been noticed that most of the proposed algorithms are population-based hybridization, where an exploratory metaheuristic is applied to explore the search space. Afterwards, a local search procedure is performed in order to improve the quality of the found solutions. Furthermore, leveraging large instances of optimization problems and accelerating search operators have become relatively easier thanks to the graphics processing units (GPU). Indeed, different designs of parallel metaheuristics have been proposed, which reflects the growing interest of researchers towards this category of algorithms. Following this context, our main focus is to develop efficient optimization algorithms, which are able to produce competitive results compared to the recent state-of-the-art algorithms.

Chapter 3

Applications

In this section, we present a set of problems, which will be solved by our proposals. This set is a well-known test suite designed for testing algorithms performance. Moreover, the real-world problem at hand is defined.

3.1 Test suite for numerical optimization

CEC 2011 [18] test suite is a well-known benchmark of real-world problems, which have been used to test several optimization algorithms. In the following subsections, the test suite is presented.

3.1.1 CEC 2011 test suite

The CEC 2011 test suite represents a set of difficult real-world applications. The benchmark contains 22 functions to be minimized with a limited budget of 150000 evaluations [18]. Table 3.1 shows details about each problem. It is important to

point out that this set is chosen because it represents the most known test suite for real-world applications.

Table 3.1: Problems description

Problem No.	Description	D
F1	Parameter Estimation of FM Sound Waves	6
F2	Lennard-Jones Potential	30
F3	Bi-functional Catalyst Blend Optimal Control	1
F4	Optimal Control of a Non-Linear Stirred Tank Reactor	1
F5	Tersoff Potential Function Min. ProblemSi(B)	30
F6	Tersoff Potential Function Min. ProblemSi(C)	30
F7	Spread Spectrum Radar Polly phase Code Design	20
F8	Transmission Network Expansion Planning	7
F9	Large Scale Transmission Pricing	126
F10	Circular Antenna Array Design	12
F11	Dynamic Economic Dispatch - Instance 1	120
F12	Dynamic Economic Dispatch - Instance 2	240
F13	Static Economic Load Dispatch Instance 1	6
F14	Static Economic Load Dispatch Instance 2	13
F15	Static Economic Load Dispatch Instance 3	15
F16	Static Economic Load Dispatch Instance 4	40
F17	Static Economic Load Dispatch Instance 5	140
F18	Hydrothermal Scheduling - Instance 1	96
F19	Hydrothermal Scheduling - Instance 2	96
F20	Hydrothermal Scheduling - Instance 3	96
F21	Messenger: Spacecraft Trajectory Optimization	26
F22	Cassini 2: Spacecraft Trajectory Optimization	22

3.2 Electric motor design

Our optimization algorithms are employed in order to optimize the design of an electrical machine used as main propulsion unit for an electric vehicle (EV). By investigating the mechanical characteristics of the electrical motorization of todays

cars it is obvious that the trend is to have higher operating speeds on the electric motor [26] . The clear advantage of having higher speed is with respect to the power density of the electric motor, meaning to maximize the ratio between the output power and weight of the machine (and consequently of its volume). This will involve also the decrease of iron losses in the machine, and finally the efficiency of the propulsion motor is improved.

3.2.1 The technical definition of the problem

The most common solution in terms of electric propulsion for the current manufactured light electric cars is the use of a permanent magnet synchronous motor (PMSM) [30], with top speeds beyond 10 000 r/min (like in the case of the Hybrid Toyota Prius, Nissan Leaf, BMW-i3 etc.). Of course, a special care should be paid to the mechanical constraints involved at such high speeds. Instead, with the improvement on the iron material in terms of mechanical stability, and since the higher operating speed permits to reduce the outer rotor diameter, the use of a high speed PMSM becomes an advantage. By increasing the power density of the electric propulsion unit, and consequently by reducing its weight, a more reduced volume will be obtained, which will be a benefit from the cars autonomy point of view. Moreover, a decreased weight will engage a reduced investment on the electric propulsion, which is critical in the automotive industry [26, 30]. That is why, we have considered for our application to evaluate an electric motorization running at high speeds. The main data of the propulsion electric motor are: 20 kW for the output power, 22 000 r/min for the rated speed. The machine will be

supplied from a battery of 380 Vdc, via an inverter. Perhaps the best candidate for such operating conditions is a PMSM with inset magnets, see Fig 3.1, where one can see the main components of the machine, as well as the magnetization of the two magnetic poles (a parallel magnetization was considered in order to increase the chances of obtaining a smoother output torque). Before passing to the optimization of the high speed PMSM, the machine was designed and the analytical approach has been validated numerically by using the finite element analysis, via Flux2D software. From this analysis we will be verifying if the desired performances have been obtained and in what conditions (efficiency, smoother output torque etc.). In Fig 3.2 is shown the magnetic behavior of the studied machine (the saturation level on a very good iron material, Vacoflux50). (After employing the optimization algorithm, the proposed solution will be numerically validated, as trustworthy approach.)

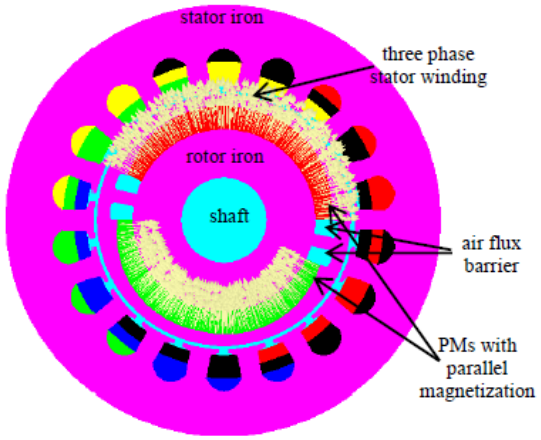


Figure 3.1: Cross-section view of the considered high-speed PMSM.

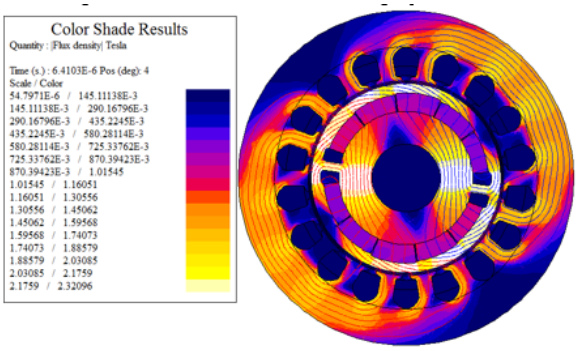


Figure 3.2: Field lines and flux density distribution on the high speed PMSM.

3.2.2 Electric motor design as an optimization problem

The main objective is to improve EVs autonomy through reducing HS-PMSM weight. The problem at hand can be modeled as a multi-objective problem, where the objective functions are represented as follows:

- The first objective function concerns the mass of the electric motor m_{atot}:

$$m_{atot} = m_{cooper} + m_{stat} + m_{rot} + m_{pm} \tag{3.1}$$

  where m_{cooper} is the cooper mass, m_{stat} is the stator iron mass, m_{rot} is the rotor iron mass, and m_{pm} is the magnets mass.

- The second objective function is to maximize the output power density which is defined as follows:

$$P_{out} = P_{in} + \sum losses \tag{3.2}$$

  where P_{out} is the output power density, P_{in} is the input density, and the sum

of losses mainly contains the mechanical, iron and copper loss component.

The two objective functions are aggregated to obtain the following new objective function which will be optimized using the proposed algorithm:

$$minJ(x) = -P_{out}/m_{atot} + penality \tag{3.3}$$

where

$$penality = 10^4 \sum_{i=1}^{7} C_i \tag{3.4}$$

In (3.4), C_i=0 if the constraint i is satisfied, 1 otherwise. The set of constraints are presented in Table 3.2.

It has to be stated that we have two versions of the problem, where the set of constraints and variation limits are different. Table 3.2 presents the set of constraints of version 1 of the problem at hand. The parameters of the problem

Table 3.2: The set of constraints

Parameter	Symbol	Unity	Variation limits of version 1	Variation limits of version 2
Output power	P_{out}	W	[19995; 20005]	[3998; 4005]
Current consumption	I_s	A	[20 ; 56]	[20; 37]
Motor torque	T_m	Nm	[8.5 ; 8.6]	[18.6; 18.63]
Motors efficiency	n	-	[0.9; 0.99]	[0.91; 0.99]
Motors power factor	PF	-	[0.81; 0.99]	[0.94; 0.99]
Rotor inner diameter	Dir	mm	[22; 70]	[20; 40]
Slot filling factor	T	-	[0.1; 0.5]	[0.1; 0.5]

are hand are presented in Table 3.3.

Table 3.3: The geometrical parameters for the weight optimization

Symbol	Description	Variation limits of version 1	Variation limits of version 2
Dis	Inner stator diameter	[50; 80] mm	[99; 150] mm
hjr	Rotor yoke height	[7; 15] mm	[10.5; 20] mm
histm	Tooth isthmus	[0.5; 2] mm	[0.5; 2] mm
hjs	Stator yoke height	[8; 15] mm	[8; 13] mm
wt	Tooth width	[3.5; 8] mm	[5.5; 10] mm
gap0	Air-gap length	[0.5; 1.5] mm	[0.5; 1.5] mm
hmp	PM height	[4; 8] mm	[4; 8] mm
Lm	Machines length	[100; 160] mm	[100; 160.1] mm

Chapter 4

Contribution

This chapter represents the major contributions of this thesis. Three directions of research are followed for proposing new powerful algorithms (hybridization, adaptation of methauristics parmaters, integrating learning techniques). Indeed, this chapter covers several proposals that have been introduced during this thesis. The proposals are classified into two categories: hybrid algorithms and self-adaptive algorithms, which will be explained in the following sub-sections. The proposals are tested using The CEC 2011 test suite along with the optimization of the electric motor

4.1 Hybrid algorithms

It has been stated that hybrid algorithms can be effective in solving a wide a range of algorithms [6, 7, 9]. Following this context, we have proposed several algorithms that combines different metaheuristics or search operators in order to introduce

stable hybrid algorithms.

4.1.1 A Hybrid Optimization Algorithm for Electric Motor Design (HOA)

The first contribution of this thesis is the proposition of a hybrid optimization algorithm, that was mainly designed to address the optimization of the electric motor topology.

The algorithm can be seen as a combination of several algorithms: CS, CMAES, an adaptive version of DE called LSHADE [82] and K-means [57]. Besides, a radial basis function (RBF) surrogate model [36] is incorporated to provide an intelligent choice between search operators.

The proposal starts with a CMAES-enhanced CS initialization technique to produce NP well-distributed points over the search space. The objective is to exploit the capability of CS in exploring the search space as well as ensuring a high quality solutions with a small number of evaluations, which is potentially ensured by CMAES. First, CMAES algorithm is run for a small number α of evaluations. Then, the produced solution is provided along with a randomly generated population to CS procedure, which will be performed for a small number λ of evaluations. The solution produced by CMAES can be seen as a guiding information to speed up the convergence rate of CS. Afterwards, the population *pop* produced by CS is used to train the RBF model. Then, it will be provided to the main loop of our proposal.

The main loop of the algorithm consists in two major procedures: global search

and local search procedure. Each procedure is performed based on a simple yet efficient switching technique. In fact, it is decided with a probability PLV whether the algorithm performs the global search procedure, or the local search procedure is performed. After applying one of the procedures, PLV parameter is linearly decreased using the following equation:

$$PLV = max(0, PLV - \frac{CurrentFes}{MaxFes}) \tag{4.1}$$

Indeed, decreasing PLV parameter will gradually force the algorithm to perform the local search procedure in the last iteration.

4.1.1.1 The global search procedure

The global search procedure consists of a clustering algorithm and a Lévy Flight perturbation. The clustering is used thanks to its high capability of avoiding redundant search points [67], which enhances the exploration of undiscovered regions [28, 35]. The clustering is considered as a multi-parent crossover that exploits the information of the whole population in order to produce a predefined number of centers (new individuals). As a clustering algorithm, one step of K-means algorithm is chosen because it has experimentally shown a superior performance than other algorithms which will be noticed in the results. K-means is exploited in order to generate K central individuals of the current population. Then, the central individuals are shifted to potentially more promising areas using lévy flight perturbation. The lévy flight movement is inspired by the global search of CS [97]

and it is performed according to:

$$StepSize_i = 0.001 * step_i * (z_i - best) i = 1, 2, ..k \tag{4.2}$$

where $step_i$ is generated according to 2.8, $best$ is the best solution so far and z is the set of the central individuals. Then, the new trial solutions are produced as follows:

$$z_i = z_i + StepSize_i \tag{4.3}$$

Afterwards, the best NP individuals of $pop \bigcup z$ are selected for the next iteration. It is important to point out that the global search procedure is repeated for T iterations.

4.1.1.2 The local search procedure

The proposed local search procedure consists of a modified version of SHADE [82] which is an adaptive version of DE that exploites success-history-based parameter adaptation. The performance of SHADE could be enhanced by using the mutation strategy current to-pBest/1/bin [99] to produce mutant vectors, where p represents the fraction of the best solutions in the current population. It has been stated that this mutation strategy is efficient for the generation of promising individuals [99]. Current to-pBest/1/bin is expressed as follows:

$$v_{i,g} = x_{i,g} + F(x_{best,g} - x_{i,g}) + F(x_{r1,g} - x_{r2,g}) \tag{4.4}$$

where $x_{best,g}$ is a randomly selected parent from the best individuals of the current population. Furthermore, an archive A of size A^{init} is proposed in order to maintain the diversity of the population. It should be stated that $x_{r2,g}$ is taken from *pop* $\cup$ A. The parent solutions that are not selected are inserted into this archive. Besides, success-history-based parameter adaptation is a strategy employed to store successful CR, F values that were successful in the past generations. After the generation of a new trial vector u_i, it is compared with its parent. If u_i is better, then the CR and F parameters are stored in the sets S_{CR}, S_F respectively. Finally, the memories M_{CR} and M_F are updated using these successful parameters according to the following equations:

$$M_{CR} = \begin{cases} meanwA(S_{CR}) & \text{if } S_{CR} \neq \emptyset \\ M_{CR} & \text{otherwise} \end{cases} \tag{4.5}$$

$$M_F = \begin{cases} meanwL(S_F) & \text{if } S_F \neq \emptyset \\ M_F & \text{otherwise} \end{cases} \tag{4.6}$$

where $meanwA$ is the weighted mean and $meanwL$ is the Lehmar mean. SHADE uses two different mean equations because they have experimentally shown better results than using one equation to generate M_{CR} and M_F.

In our local search procedure, a surrogate model-based SHADE (S-SHADE) algorithm is proposed, which is summarized in Algorithm 4. In S-SHADE, we insert an additional mutation equation that ultimately favors exploitation and a surrogate based-switching technique to choose the best mutation operator for the

next generation. The additional mutation equation can be also seen as a linear recombination that forces the current solution to move towards the best solution [19]. The mutation equation is is represented as follows:

$$v_{i,g} = x_{i,g} + (x_{i,best} - x_{i,g}) * F \tag{4.7}$$

and $x_{i,best}$ is the i^{th} best solution in the population. To save the budget of evaluations, the RBF surrogate model is used for an approximated evaluation. After applying the two search operators, RBF is used to approximate the fitness of the two mutant populations. The mutant population that contains the best approximated solution is then used in crossover.

4.1.1.3 The algorithmic combination

First, the proposed initialization strategy takes place in order to provide well-distributed solutions. Afterwards, the global search and the local search procedures are performed based on a complex criterion. If less than a parameter max_{fes} , and if PLV parameter is greater than a value randomly generated in [0,1], then, the global search procedure is performed. Otherwise, one generation of S-SHADE is applied as a local search procedure. Besides, in order to gradually shift the algorithm into exploitation, PLV parameter is linearly decreased after each iteration. Finally, linear reduction of the population takes place to remove a fraction of the worst individuals at each iteration. It has to be mentioned that the RBF model is updated using the current population after each quarter of the available budget. The pseudo-code of the full proposal is depicted in Algorithm 5.

Algorithm 4 One generation of our S-SHADE algorithm

Given a population pop of NP individuals
$PopOld \leftarrow pop$
Given S_{CR}, S_F
Compute M_{CR}, and M_F according to 4.5 and 4.6
Generate a mutant population pop_a according to 4.4
Generate a mutant population pop_b according to 4.7
Evaluate approximately pop_a and pop_b using RBF surrogate model
if pop_a contains the best approximated solution **then**
    $pop_{best} \leftarrow pop_a$
else
    $pop_{best} \leftarrow pop_b$
end if
for Each x_i in pop_{best} **do**
    Use binomial crossover to generate the trial vector u_i using pop_{best} and $PopOld$
    Evaluate u_i using the real objective function.
    if $f(u_i) < f(pop_i)$ **then**
        $pop_i \leftarrow u_i$
        $A \leftarrow x_i$, $S_{CR} \leftarrow CR_i$, $S_F \leftarrow F_i$
    end if
end for

Algorithm 5 Pseudocode of the proposed approach

Archive $A \leftarrow \emptyset$
$S_{CR} \leftarrow \emptyset$, $S_F \leftarrow \emptyset$
Set all values in M_{CR}, and M_F to 0.5
Generate a solution using CMA-ES for a number of evaluation α
Perform CS to generate a population *pop* of NP individuals for a number of evaluation λ
Train the RBF surrogate model using each individual in pop and its fitness
while $current_{fes} < Budget$ **do**
  if $current_{fes} < max_{fes}$ and rand $< PLV$ **then**
    for i=1:T **do**
      Generate K central individuals of *pop* using Kmeans
      Generate a step size for each center using Lévy flights
      Generate new individuals according to 4.3
      Evaluate individuals and select the NP best individuals of the population
      $current_{fes} \leftarrow current_{fes} + K$
    end for
    Decrease PLV according to 4.1
  else
    Perform S-SHADE (*pop*, S_{CR}, S_F)
  end if
  Apply a linear reduction of the population
  Update the RBF model using the current population after each quarter of the budget
end while

Table 4.1: Parameter setting of the compared algorithms

Algorithm	Parameters	Electric motor design problem	CEC 2011
Our proposition	α	D*1000	D*1000
	λ	D*1000	D*1000
	NP	200	200
	k	NP / 4	NP / 4
	PLV	0.8	0.8
	Budget	10^6	D*10000
	max_{fes}	2/3 * Budget	1/3 * Budget
	T	100	1
EPSDE	Parameters taken from [90]	=	=
SADE	Parameters taken from [11]	=	=
JADE	Parameters taken from [100]	=	=
SHADE	Parameters taken from [80]	=	=

4.1.1.4 Experimental results

Our proposal has been tested on the CEC 2011 test suite as well as the optimization problem at hand with its two versions. It has been compared with powerful state-of-the-art algorithms as JADE [100], SADE [11], EPSDE [90] and SHADE [82]. First, the parameter setting of our algorithm is shown as well as the parameters of the compared algorithms, which are presented in Table 4.1

4.1.1.5 Comparison on CEC 2011

To study the importance of each component, an experimentation has been conducted to compare the proposed algorithm with other versions of the proposal. In the first version, the initialization method is disabled (variant-1). In the second version, the global search procedure is removed. The algorithm becomes a hybridization between the proposed initialization method and S-SHADE (variant-2).

Moreover, to show the clustering influence, K-means algorithm has been replaced by the FCM clustering algorithm (variant-3). Each column from Table 4.2 shows best and mean of each algorithm for each function. The best fitness found for each function is in bold. Mean results that are significantly better than the ones of the other algorithms, according to the Kruskal-Wallis statistical test at 95% confidence level followed by a Tukey-Kramer post hoc test are also in bold.

The results presented in Table 4.3 reveal a superior performance of our proposition compared to the other algorithms. It can significantly outperform SHADE in 8 functions, SADE in 9 functions, JADE in 7 functions and EPSDE in 12. Moreover, the comparison with the three variants of our proposal reveals that our proposition achieved better performance as well. It could outperform variant-1 and variant-2 in 6 functions, which shows the importance of the proposed initialization method and the global search. Variant-3 shows better performance when compared to variant-1 and variant-2. Even-though, the proposition can outperform it in 2 functions. It is observed that both clustering methods can significantly improve the performance of the proposed algorithm.

4.1.1.6 Comparison on the optimization problem at hand

The proposed algorithm has been run 30 times. The best, the mean, the median, the worst, and the standard deviation of each algorithm are collected. It is stated from Table 4.4 that the proposal achieves the best solution compared to the other algorithms. Besides, a stable performance is obtained, since the proposal could achieve the best solution in each run.

Table 4.2: Comparison of HOA with state-of-the-art algorithms on the CEC 2011 test suite

		SHADE	SADE	JADE	EPSDE	Variant-1	Variant-2	Variant-3	HOA
F1	Best	2.73E-02	6.31E-01	2.57E-10	5.45E+00	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Mean	1.85E+01	**7.59E-01**	2.26E+00	9.10E+00	1.95E+00	1.87E+00	2.08E+00	1.62E+00
F2	Best	-2.44E+01	-1.95E+01	-2.45E+01	-2.19E+01	-2.31E+01	-2.47E+01	**-2.69E+01**	**-2.69E+01**
	Mean	-2.30E+01	-1.70E+01	**-2.34E+01**	-2.02E+01	-2.15E+01	**-2.34E+01**	**-2.35E+01**	**-2.39E+01**
F3	Best	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
	Mean	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
F4	Best	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F5	Best	-3.66E+01	-3.32E+01	-3.64E+01	-3.60E+01	-3.62E+01	-3.54E+01	-3.66E+01	**-3.69E+01**
	Mean	**-3.60E+01**	-3.22E+01	**-3.56E+01**	-3.22E+01	**-3.47E+01**	**-3.45E+01**	**-3.48E+01**	**-3.49E+01**
F6	Best	-2.91E+01	-2.63E+01	**-2.92E+01**	-2.88E+01	-2.90E+01	-2.89E+01	-2.91E+01	-2.91E+01
	Mean	**-2.90E+01**	-2.41E+01	**-2.90E+01**	-2.01E+01	-2.48E+01	-2.43E+01	-2.65E+01	-2.66E+01
F7	Best	9.06E-01	1.24E+00	9.10E-01	1.12E+00	1.17E+00	1.05E+00	8.72E-01	**5.00E-01**
	Mean	1.12E+00	1.37E+00	1.17E+00	1.30E+00	1.32E+00	1.22E+00	1.05E+00	**8.80E-01**
F8	Best	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
	Mean	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
F9	Best	1.14E+03	**7.69E+02**	1.13E+03	3.98E+04	3.67E+03	3.28E+03	3.37E+03	2.15E+03
	Mean	2.22E+03	**1.73E+03**	2.40E+03	9.28E+04	4.45E+03	3.67E+03	3.85E+03	3.37E+03
F10	Best	-2.18E+01	-2.18E+01	**-2.18E+01**	-2.02E+01	-2.15E+01	-2.16E+01	-2.16E+01	**-2.14E+01**
	Mean	**-2.16E+01**	**-2.16E+01**	**-2.14E+01**	-1.74E+01	-1.55E+01	-1.63E+01	-1.69E+01	-1.69E+01
F11	Best	5.15E+04	**5.12E+04**	5.15E+04	5.21E+04	5.17E+04	5.23E+04	5.17E+04	5.16E+04
	Mean	5.32E+04	**5.21E+04**	5.24E+04	5.86E+04	5.34E+04	5.42E+04	5.32E+04	5.32E+04
F12	Best	**1.07E+06**	**1.07E+06**	**1.07E+06**	**1.07E+06**	1.08E+06	**1.07E+06**	**1.07E+06**	**1.07E+06**
	Mean	1.10E+06	1.09E+06	**1.07E+06**	1.09E+06	**1.07E+06**	**1.07E+06**	**1.07E+06**	**1.07E+06**
F13	Best	1.55E+04	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
	Mean	1.55E+04	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
F14	Best	**1.80E+04**	1.81E+04	**1.80E+04**	1.83E+04	**1.80E+04**	**1.80E+04**	**1.80E+04**	**1.80E+04**
	Mean	**1.81E+04**	**1.81E+04**	**1.83E+04**	1.86E+04	1.85E+04	1.87E+04	**1.83E+04**	**1.83E+04**
F15	Best	3.27E+04	3.28E+04	3.27E+04	3.29E+04	**3.26E+04**	3.27E+04	**3.26E+04**	**3.26E+04**
	Mean	**3.27E+04**	3.28E+04	3.29E+04	3.30E+04	**3.27E+04**	**3.27E+04**	**3.27E+04**	**3.27E+04**
F16	Best	1.26E+05	1.26E+05	1.26E+05	1.31E+05	1.28E+05	1.31E+05	1.27E+05	**1.24E+05**
	Mean	1.29E+05	1.28E+05	1.33E+05	1.42E+05	1.33E+05	1.41E+05	1.28E+05	**1.26E+05**
F17	Best	1.88E+08	**1.87E+06**	**1.87E+06**	1.93E+06	1.91E+06	1.92E+06	1.91E+06	1.91E+06
	Mean	1.91E+06	**1.90E+06**	1.91E+06	2.06E+06	1.93E+06	1.93E+06	1.93E+06	1.93E+06
F18	Best	9.37E+05	**9.33E+05**	9.35E+05	3.24E+06	9.35E+05	9.35E+05	9.34E+05	9.34E+05
	Mean	9.40E+05	**9.38E+05**	9.39E+05	6.06E+06	**9.38E+05**	**9.38E+05**	**9.38E+05**	**9.38E+05**
F19	Best	**9.39E+05**	9.41E+05	**9.39E+05**	4.47E+06	**9.39E+05**	9.41E+05	**9.39E+05**	**9.39E+05**
	Mean	9.52E+05	9.46E+05	9.92E+05	7.16E+06	**9.42E+05**	9.44E+05	**9.42E+05**	**9.42E+05**
F20	Best	9.34E+05	9.35E+05	9.36E+05	4.16E+06	9.35E+05	9.34E+05	**9.33E+05**	**9.33E+05**
	Mean	**9.40E+05**	**9.37E+05**	**9.40E+05**	6.21E+06	**9.38E+05**	**9.40E+05**	**9.37E+05**	**9.37E+05**
F21	Best	1.41E+01	1.66E+01	1.31E+01	1.66E+01	1.33E+01	1.27E+01	1.16E+01	**1.15E+01**
	mean	1.75E+01	1.97E+01	1.72E+01	1.97E+01	1.39E+01	1.41E+01	**1.38E+01**	**1.35E+01**
F22	Best	1.31E+01	1.68E+01	1.19E+01	1.68E+01	1.12E+01	1.25E+01	**9.24E+00**	9.26E+00
	Mean	1.99E+01	2.18E+01	1.66E+01	2.18E+01	1.37E+01	1.42E+01	**1.32E+01**	**1.36E+01**

Table 4.3: Comparison of HOA using Kruskal-Wallis test on CEC 2011 test suite

vs	Our proposition	D=30
SHADE	+(better)	2
	-(worse)	8
	=(no sig)	12
SADE	+(better)	5
	-(worse)	9
	=(no sig)	8
JADE	+(better)	2
	-(worse)	7
	=(no sig)	13
EPSDE	+(better)	0
	-(worse)	12
	=(no sig)	10
Variant-1	+(better)	0
	-(worse)	6
	=(no sig)	16
Variant-2	+(better)	0
	-(worse)	6
	=(no sig)	16
Variant-3	+(better)	0
	-(worse)	2
	=(no sig)	20

Table 4.4: Comparison of HOA with state-of-the-art algorithms on the first version of the problem at hand

	Best	Mean	Worst	Std
SADE	-3.271E+03	-2.920E+03	-2.780E+03	81.49
JADE	-3.194E+03	-2.840E+03	-2.420E+03	96.23
EPSDE	-3.200E+03	-2.850E+03	-2.430E+03	102.65
SHADE	-3.097E+03	-2.944E+03	-2.697E+03	44.28
Variant-1	-3.397e+03	-3.114e+03	-2.935e+03	112.10
Variant-2	-3.318e+03	-3.202e+03	-3.130e+03	49.03
Variant-3	-3.380e+03	-3.188e+03	-3.085e+03	58.36
HOA	**-3.397E+03**	**-3.397E+03**	**-3.397E+03**	0

Further details about the best solution found by our proposal are shown in Table 4.5. The proposed algorithm could obtain an important gain of 28% in the mass. Moreover, it could achieve a gain of 17% and 29% decreasing the mechanical loss and the iron loss stator respectively. Similarly, HOA has been applied on the second version of the problem, where it revealed a stable performance compared to the other algorithms. Besides, it has been noticed that variant-3 (HOA with FCM) has shown the same performance. The results are depicted in Table 4.6. The gain and the power density obtained after optimizing the topology of the second version can be expressed in Table 4.7. It can be seen that the optimized weight $m_{atot} = 9.27$ Kg.

Finally, as it has been mentioned above, the proposal has shown a competitive performance on the problems at hand. However, it can be noticed that the proposal

Table 4.5: The best geometrical parameters with the optimized factors

Symbol	Original motor	Optimized motor	Gain %
m_{atot}	8.2513 kg	5.8885 kg	+ 28.63
P_{out}	20000 W	20005 W	+ 0.25e-3
P_{out} / m_{tot}	2.42 kW/kg	3.39 kW/kg	+ 28.653
Iron loss stator	225.73 W	158.9 W	+ 29.60
Mechanical loss	352.69 W	292.15 W	+ 17.16
Efficiency	0.9596	0.9607	+ 1.01
Power factor	0.8187	0.8100	- 1.06
Dis	63 mm	66.7 mm	
hjr	10.5 mm	9.3 mm	
$histm$	1.5 mm	1 mm	
hjs	11.8 mm	9.8 mm	
wt	5 mm	4 mm	
gap	1 mm	0.9 mm	
hmp	6 mm	4 mm	
Lm	135 mm	100 mm	

Table 4.6: Comparison of HOA with state-of-the-art algorithms on the second version of the problem at hand

	Best	Mean	Worst	std
SADE	**-4.3065E+02**	-4.2523E+02	-4.1173E+02	3.5328E+00
JADE	**-4.3065E+02**	-4.2356E+02	-4.0376E+02	5.1813E+00
EPSDE	-4.2842E+02	-4.1968E+02	-4.0693E+02	5.1628E+00
SHADE	**-4.3065E+02**	-4.2370E+02	-4.0569E+02	6.9628E+00
Variant-1	**-4.3065E+02**	-4.2467E+02	-4.2256E+02	4.2569E+00
Variant-2	**-4.3065E+02**	-4.2823E+02	-4.2701E+02	2.1257E+00
Variant-3	**-4.3065E+02**	**-4.3065E+02**	**-4.3065E+02**	0
HOA	**-4.3065E+02**	**-4.3065E+02**	**-4.3065E+02**	0

Table 4.7: The best geometrical parameters for the second version

Symbol	Optimized parameters
m_{atot}	9.27 kg
P_{out} / m_{tot}	430.6538 W/kg
Dis	99mm
hjr	10.5 mm
$histm$	0.5 mm
hjs	8 mm
wt	5 mm
gap	1.2 mm
hmp	7.99 mm
Lm	114.2 mm

combines several complicated algorithms. Despite the low computational time of the proposal, it is not straight-forward to be re-implemented due to the several parameters that have to be tuned before the optimization process. In the next sub-section, a novel hybrid DE algorithm is presented, where the new proposal is ultimately easy to understand and re-implement.

4.1.2 A hybrid differential evolution algorithm for real world problems (HDE)

It has been mentioned that several mutation strategies have been proposed to improve DE performance. Unfortunately, none of these strategies can be successful for all the optimization problems. In this proposition, we propose to exploit two mutation strategies of DE in one framework: one for exploration and one for

exploitation. The novelty of this algorithm can be summarized as follows:

- A multi-criteria-based selection operator to obtain balance between exploration and exploitation.
- A new self-adaptive strategy based on a pheromone matrix to adapt the DE parameters.
- A restart strategy when early convergence is detected.

4.1.2.1 Mutation strategies

Indeed, multiple mutation strategies could achieve a resilient performance [15], [16]. In the context of our hybridization, DE/current-to-best/1 and DE/rand/2 are used in order to enhance the exploitation and exploration capabilities of DE: DE/current-to-best/1 is a powerful strategy that relies on the best current individual to evolve the population which accelerate the convergence rate. This strategy has shown promising performance in several works [17], [18]. In the context of enhancing exploration, DE/rand/2 evolves the current individual using five random parents from the population. DE/rand/2 behavior would potentially shift the population towards undiscovered search regions [7], [19].

4.1.2.2 Pheromone matrix based self-adaptive strategy

Recent studies have shown the importance of controlling DE parameters on the final results. To this issue, we propose pheromone matrix based self-adaptive (PMS) strategy to enhance the performance of DE. PMS provides a set of discrete

values of F and CR. Each individual selects the best F and CR. After, it updates the pheromone matrix entry that corresponds to the chosen combination based on its performance. Updating the pheromone matrix makes the algorithm more cooperative by sharing the information with the other individuals. If the chosen combination could improve the current individual, it will be rewarded by adding a pheromone quantity to the corresponding matrix entry. This operation is applied as follows:

$$M(i,j) = 0.5 * M(i,j) + 0.5 * (f(i) - f^*(i)) \tag{4.8}$$

Otherwise, it will be penalized as follows:

$$M(i,j) = 0.5 * M(i,j) - 0.5 * (f(i) - f^*(i)) \tag{4.9}$$

where M is the pheromone matrix, i is the index of F parameter, j is the index of CR parameter, $f(i)$ is the fitness of the parent individual and $f^*(i)$ is the fitness of the new generated offspring. The entry update is based on how much the fitness is improved/degraded which offers a kind of heuristic information to the other individuals. It is important to point out that a different pheromone matrix is updated for each mutation strategy, i.e. one for DE/current-to-best/1 and another one for DE/rand/2. Algorithm 6 shows the pseudo-code of DE using PMS.

Algorithm 6 One generation of DE algorithm using PMS

```
1: Input: population pop of NP individuals and pheromone matrix M, popold ← pop
2: Output: pop
3: for each individuals in popold do
4:    Choose the best combination of F and CR from M
5:    Apply the mutation strategy
6:    Apply binomial crossover
7:    if the offspring is better than the parent then
8:       Replace the parent in pop with the offspring
9:       Reward the corresponding entry according to (4.8)
10:   else
11:      Penalize the corresponding entry according to (4.9)
12:   end if
13: end for
```

4.1.2.3 Multi-criteria selection

The balance between exploration and exploitation is essential for successful algorithms [16]. A very elitist based-selection strategy highly favors exploitation and may loss the diversity during the search process. In the contrast, adopting diversified selection strategy would consume a large number of iterations to reach a local optimum. In the attempt to overcome this situation, a multi-criteria selection strategy is proposed, which considers two criteria; the fitness value as an exploitation criterion and the distance to the centroid individual of the current population as a exploration criterion. The selection procedure is applied by performing a non-dominated sorting [22] on a given population using these two conflicting criteria. Non-dominated sorting allows sorting individuals based on several criteria.

Accordingly, the less dominated individuals are selected. In other words, the selection procedure takes only in account the best and the furthest individuals from the centroid point of the population. The proposed multi-criteria selection is depicted in Algorithm 7.

Algorithm 7 Multi-criteria selection strategy

1: Input: population *pop* of NP individuals, fitness
2: Output: pop^* of NP individuals
3: Compute the centroid point of *pop*
4: Compute the distance of each individual to the centroid point
5: Apply a non-dominated sorting on *pop*
6: Choose the N best ranked individuals

4.1.2.4 The proposed restart strategy

DE algorithm can suffer from an early convergence rate [20] and several works have been introduced to tackle this issue [81], [60]. The early convergence causes two issues:

- A stagnation issue due to a number of non-improving generations.
- A close distance between individuals which makes the algorithm more exploitative.

A simple technique is used to detect these two situations. The stagnation issue can be discovered by counting the non-successful generations at improving the best solution. While the close distance between individuals is detected by computing the standard deviation of each dimension. A small standard deviation value means that the whole population is stuck in one region, which potentially means early

convergence. Our proposition stores at each generation an archive A of individuals that represents the previous state of the population before the early convergence. When early convergence is detected, the archive A is used to visit new points in the search space as follows:

$$u_i = x_i + 0.5 * (A_i - x_i) \tag{4.10}$$

where x_i is the current individual, A_i is the previous state of x_i before early convergence. Here, u_i is used in a binomial crossover with x_i to generate a new point as:

$$x_i^j = \begin{cases} u_i^j & \text{if } j = \sigma_j \quad \text{or } R_j < 0.5 \\ x_{i,j}^G & otherwise \end{cases} \tag{4.11}$$

where R_j is a randomly generated number within the range [0,1], i is the current individual index, j is the current dimension and σ_j is a randomly generated integer within the range [1,D]. The proposed restart strategy is performed on the p worst individuals of the population. The main aim here is to shift a fraction of the population towards new search areas. The restart strategy is depicted in Algorithm 8.

4.1.2.5 Combination of the algorithmic components

Each phase in the proposed approach is explained in this sub-section. First, a population *pop* of NP individuals is randomly generated, the archive A is initialized with *pop* and the pheromone matrix is filled with real values randomly

Algorithm 8 The restart strategy

1: Input: population *pop* of NP individuals
2: Output: *pop*
3: Choose p worst individuals from *pop*
4: **for** each individual among the $NP/4$ worst individuals **do**
5: Compute u_i according to (4.10)
6: Perform a binomial crossover according to (4.11)
7: Update the current individual and compute its fitness
8: **end for**

generated within the range [0, 1]. Afterwards, Algorithm 6 is performed on *pop* twice by using the two search operators DE/current-to-best/2 and DE/rand/2 to produce a new population pop^* of $2^*(NP)$ individuals. Then, the multi-criteria selection strategy is applied selecting the NP best individuals to be evolved in the next generations. In the case of a non-successful generation, a stagnation counter $STG_{counter}$ is increased by one. In each generation, the average standard deviation of the population, denoted by $mean_{std}$ is computed as follows:

$$mean_{std} = mean(std(i)) \tag{4.12}$$

where $std(i)$ is standard deviation value of the population in the dimension i. If $mean_{std}$ is less than a parameter ϵ, then the population is in exploitative state and $STG_{counter}$ is increased by one. When $STG_{counter}$ reaches a threshold σ, the restart strategy is performed on the p worst individuals of the population. The archive A is updated only if $mean_{std}$ is greater or equal to ϵ or the current best solution could be improved. All the mentioned steps are presented in Algorithm

9.

Algorithm 9 The proposed approach

```
Generate randomly a population pop of size NP
A ← pop
while the stopping criterion is not satisfied do
   Apply Algorithm 6 on pop using DE/current-to-best/1 to generate pop1
   Apply Algorithm 6 on pop using DE/rand/2 to generate pop2
   pop* ← pop1 U pop2
   Apply Algorithm 7 on pop* and update pop
   Compute the average of standard deviation mean_std as in (4.12)
   if the best individual is not improved or mean_std < ε then
      STG_counter ← STG_counter + 1
   else
      STG_counter ← STG_counter - 1
      A ← pop
   end if
   if STG_counter >= σ then
      Apply Algorithm 8 on pop using the archive A
      STG_counter ← 0
   end if
end while
```

4.1.2.6 Experimental results

This proposal has been tested on the CEC 2011 test suite and the first version of the problem at hand. Indeed, our proposal has a small number of parameters which is illustrated in Table 4.8. Our hybrid algorithm has been compared with powerful state-of-the-art DE variants as well as CS [97] and ABC [42] which have been proved to be efficient when solving engineering problems [66, 87]. Furthermore, the algorithmic components of HDE are investigated by conducting another experimentation with three variants. The variants are expressed as follows:

- HDE1: HDE without the parameter adaptation strategy. F=0.5 and CR=0.9
- HDE2: HDE without the multi-criteria strategy. Instead, the selection procedure of DE is set.
- HDE3: HDE without the restart strategy.

Table 4.8: Parameter setting of the compared algorithms

Algorithm	Parameters
Our proposition	NP=200, $\epsilon = 5$, σ=10, p= NP/4
SHADE	Parameters taken from [42]
SADE	Parameters taken from [11]
JADE	Parameters taken from [100]
EPSDE	Parameters taken from [90]

4.1.2.7 Comparison on CEC 2011

The comparison has been conducted using the CEC 2011 test suite. The obtained results of all the algorithms can be seen in Table 4.9. In this table, the rows show best and mean values of 30 runs of each algorithm for each function. The best fitness found for each function is in bold. Besides, Kruskal-Wallis statistical test at 95% confidence level followed by a Tukey-Kramer post hoc test is performed, where mean results that are significantly better than the ones of the other algorithms are illustrated in bold. The results shown in Table 4.10 reveal a the advantage of our proposition over the other algorithms. It can significantly outperform SHADE and SADE in 8 functions, JADE in 5 functions and EPSDE in 18 functions.

Table 4.9: Comparison of HDE with state-of-the-art algorithms on the CEC 2011 test suite

		SHADE	SADE	JADE	EPSDE	The approach
F1	Best	2.73E-02	6.31E-01	2.57E-10	5.45E+00	**0.00E+00**
	Mean	1.85E+00	2.49E+00	2.26E-01	9.10E+00	**2.72E-23**
F2	Best	-2.44E+01	-1.95E+01	-2.45E+01	-2.19E+01	**-2.84E+01**
	Mean	-2.30E+01	-1.70E+01	-2.34E+01	-2.02E+01	**-2.69E+01**
F3	Best	**1.15E-05**	**1.15E-05**	1.15E-05	**1.15E-05**	**1.15E-05**
	Mean	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
F4	Best	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F5	Best	-3.66E+01	-3.32E+01	-3.64E+01	-3.60E+01	**-3.69E+01**
	Mean	**-3.60E+01**	-3.22E+01	**-3.56E+01**	-3.22E+01	**-3.53E+01**
F6	Best	-2.91E+01	-2.63E+01	**-2.92E+01**	-2.88E+01	**-2.92E+01**
	Mean	**-2.90E+01**	-2.41E+01	**-2.90E+01**	-2.01E+01	**-2.76E+01**
F7	Best	9.06E-01	1.24E+00	9.10E-01	1.12E+00	**5.51E-01**
	Mean	1.12E+00	1.37E+00	1.17E+00	1.30E+00	**7.76E-01**
F8	Best	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
	Mean	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
F9	Best	1.14E+03	**7.69E+02**	1.13E+03	3.98E+04	1.15E+03
	Mean	**2.22E+03**	**1.73E+03**	**2.40E+03**	9.28E+04	**2.32E+03**
F10	Best	-2.18E+01	**-2.18E+01**	**-2.18E+01**	-2.02E+01	**-2.18E+01**
	Mean	**--2.16E+01**	**-2.16E+01**	**-2.14E+01**	-1.74E+01	**-2.13E+01**
F11	Best	5.15E+04	**5.12E+04**	5.15E+04	5.21E+04	**5.12E+04**
	Mean	5.32E+04	**5.21E+04**	**5.24E+04**	5.86E+04	**5.22E+04**
F12	Best	1.07E+06	1.07E+06	**1.07E+06**	**1.07E+06**	**1.07E+06**
	Mean	1.10E+06	1.09E+06	**1.07E+06**	**1.09E+06**	**1.07E+06**
F13	Best	1.55E+04	**1.54E+04**	1.54E+04	1.54E+04	1.54E+04
	Mean	1.55E+04	**1.54E+04**	**1.54E+04**	1.54E+04	**1.54E+04**
F14	Best	**1.80E+04**	1.81E+04	1.80E+04	1.83E+04	**1.80E+04**
	Mean	**1.81E+04**	**1.81E+04**	**1.83E+04**	1.86E+04	**1.81E+04**
F15	Best	**3.27E+04**	3.28E+04	**3.27E+04**	3.29E+04	**3.27E+04**
	Mean	**3.27E+04**	**3.28E+04**	**3.29E+04**	3.30E+04	**3.28E+04**
F16	Best	1.26E+05	1.26E+05	1.26E+05	1.31E+05	**1.26E+05**
	Mean	**1.29E+05**	**1.28E+05**	**1.33E+05**	1.42E+05	**1.28E+05**
F17	Best	1.88E+08	1.87E+06	**1.87E+06**	1.93E+06	1.89E+06
	Mean	**1.91E+06**	**1.90E+06**	**1.91E+06**	2.06E+06	**1.91E+06**
F18	Best	9.37E+05	**9.33E+05**	9.35E+05	3.24E+06	9.36E+05
	Mean	**9.40E+05**	**9.38E+05**	**9.39E+05**	6.06E+06	**9.42E+05**
F19	Best	**9.39E+05**	9.41E+05	**9.39E+05**	4.47E+06	9.43E+05
	Mean	**9.46E+05**	**9.46E+05**	**9.92E+05**	7.16E+06	**1.03E+06**
F20	Best	**9.34E+05**	9.35E+05	9.36E+05	4.16E+06	9.37E+05
	Mean	9.40E+05	**9.37E+05**	9.40E+05	6.21E+06	9.42E+05
F21	Best	1.41E+01	1.66E+01	1.31E+01	1.66E+01	**9.53E+00**
	Mean	1.75E+01	1.97E+01	1.72E+01	1.97E+01	**1.14E+01**
F22	Best	1.31E+01	1.68E+01	1.19E+01	1.68E+01	**8.74E+00**
	Mean	1.99E+01	2.18E+01	1.66E+01	2.18E+01	**1.04E+01**

Table 4.10: Comparison of HDE with state-of-the-art algorithms using the statistical test

VS	Our proposition	22 functions of CEC 2011 test suite
SHADE	+(better)	0
	-(worse)	8
	=(no sign)	14
SADE	+(better)	1
	-(worse)	8
	=(no sign)	13
JADE	+(better)	1
	-(worse)	5
	=(no sign)	16
EPSDE	+(better)	0
	-(worse)	18
	=(no sign)	4

Another experimentation of HDE is conducted with its variants. Table 4.11 demonstrates the results obtained in CEC 2011 test suite, where a statistical test is applied. Similarly, the best means are in bold. After applying the statistical test, it has been found that HDE ultimately outperform HDE1 in 18 functions, which reveals the importance of the adopted parameter adaptation strategy. In the same context, HDE could outperform HDE2 in 11 functions, where the proposed multi-criteria selection strategy has shown to be efficient in F1, F2, F5, F7, F9, F15, F16, F17, F18, F19 and F21. It was also stated that HDE outperforms HDE3 in 13 functions, where it is demonstrated that the global search procedure (the restart strategy) is necessary for an efficient optimization. Table 4.12 summarizes

Table 4.11: Comparison of HDE with its variants

		HDE1	HDE2	HDE3	HDE
F1	Best	2.74E+00	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Mean	5.36E+00	1.27E-05	1.96E-07	**2.72E-23**
F2	Best	-1.67E+01	-2.55E+01	-2.34E+01	**-2.84E+01**
	Mean	-1.19E+01	-2.37E+01	-1.97E+01	**-2.69E+01**
F3	Best	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
	Mean	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
F4	Best	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F5	Best	-3.05E+01	-3.55E+01	-3.12E+01	**-3.69E+01**
	Mean	-3.11E+01	-3.19E+01	-3.05E+01	**-3.53E+01**
F6	Best	-2.25E+01	-2.91E+01	-2.71E+01	**-2.92E+01**
	Mean	-1.87E+01	**-2.72E+01**	-2.32E+01	**-2.76E+01**
F7	Best	1.16E+00	8.29E-01	9.85E-01	**5.51E-01**
	Mean	1.58E+00	1.01E+00	1.15E+00	**7.76E-01**
F8	Best	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
	Mean	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
F9	Best	2.57E+03	1.19E+03	1.62E+03	**1.15E+03**
	Mean	3.75E+03	2.79E+03	1.93E+03	**2.32E+03**
F10	Best	-2.12E+01	**-2.18E+01**	**-2.18E+01**	**-2.18E+01**
	Mean	-1.89E+01	**-2.14E+01**	**-2.16E+01**	**-2.13E+01**
F11	Best	5.23E+04	**5.12E+04**	**5.12E+04**	**5.12E+04**
	Mean	5.46E+04	**5.23E+04**	5.27E+04	**5.22E+04**
F12	Best	**1.07E+06**	**1.07E+06**	**1.07E+06**	**1.07E+06**
	Mean	1.09E+06	**1.07E+06**	**1.07E+06**	**1.07E+06**
F13	Best	1.55E+04	**1.54E+04**	**1.54E+04**	**1.54E+04**
	Mean	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
F14	Best	1.87E+04	**1.80E+04**	**1.80E+04**	**1.80E+04**
	Mean	1.89E+04	**1.81E+04**	**1.81E+04**	**1.81E+04**
F15	Best	3.29E+04	3.27E+04	3.28E+04	**3.27E+04**
	Mean	3.31E+04	3.29E+04	3.29E+04	**3.28E+04**
F16	Best	1.32E+05	1.28E+05	1.27E+05	**1.26E+05**
	Mean	1.36E+05	1.30E+05	1.29E+05	**1.28E+05**
F17	Best	2.12E+06	1.91E+06	1.92E+06	**1.89E+06**
	Mean	2.65E+06	1.92E+06	1.93E+06	**1.91E+06**
F18	Best	9.40E+05	9.39E+05	9.38E+05	**9.36E+05**
	Mean	9.67E+05	9.46E+05	9.45E+05	**9.42E+05**
F19	Best	9.55E+05	9.52E+05	9.51E+05	**9.43E+05**
	Mean	1.06E+06	1.05E+06	1.07E+05	**1.03E+06**
F20	Best	9.45E+05	9.39E+05	9.38E+05	**9.37E+05**
	Mean	9.74E+05	**9.42E+07**	**9.42E+05**	**9.42E+05**
F21	Best	1.28E+01	9.56E+00	9.57+00	**9.53E+00**
	Mean	2.56E+01	1.23E+01	1.25E+01	**1.14E+01**
F22	Best	1.71E+01	8.82E+00	9.52E+00	**8.74E+00**
	Mean	2.28E+01	**1.07E+01**	1.28E+01	**1.04E+01**

the results of the statistical test.

Table 4.12: Comparison of HDE with HDE1, HDE2 and HDE3 using the statistical test

VS	Our proposition	22 functions of CEC 2011 test suite
HDE1	+(better)	0
	-(worse)	18
	=(no sign)	4
HDE2	+(better)	0
	-(worse)	11
	=(no sign)	11
HDE3	+(better)	0
	-(worse)	13
	=(no sign)	9

4.1.2.8 Comparison on the optimization problem at hand

This sub-section represents of our algorithm performance on the problem at hand. The results can be seen in Table 4.13 and 4.14, where unfortunately our hybrid proposal have shown relatively worse results on the problem at hand in terms of mean compared to HOA. However, compared to the state-of-the-art DE algorithms, it could achieve promising results. Moreover, it can obtain the best solution known so far, which has been obtained by the previous algorithm.

Finally, as mentioned above, the contribution of this hybrid DE algorithm consists in combining two mutation strategies, and a multi-criteria selection strategy to potentially achieve balance between exploration and exploitation. Moreover, our proposal includes also a self-adaptive strategy to control F and CR. This strategy has shown promising success in improving the final results. This point

Table 4.13: Comparison of HDE with state-of-the-art algorithms on the first version of the problem at hand

	Best	Mean	Worst	Std
SHADE	-3.097E+03	-2.944E+03	-2.697E+03	4.42E+01
SADE	-3.27E+03	-2.92E+03	-2.78E+03	3.97E+01
JADE	-3.19E+03	-2.84E+03	-2.42E+03	4.03E+01
EPSDE	-3.20E+03	-2.85E+03	-2.43E+03	4.12E+01
Our proposition	**-3.39E+03**	**-3.14E+03**	-2.79E+03	5.17E+01

Table 4.14: Comparison of HDE with state-of-the-art algorithms on the second version of the problem at hand

	Best	Mean	Worst	Std
SHADE	**-4.3065E+02**	-4.2370E+02	-4.0569E+02	6.9628E+00
JADE	**-4.3065E+02**	-4.2356E+02	-4.0376E+02	5.1813E+00
EPSDE	-4.2842E+02	-4.1968E+02	-4.0693E+02	5.1628E+00
SADE	**-4.3065E+02**	-4.2523E+02	-4.1173E+02	3.5328E+00
HDE	**-4.3065E+02**	**-4.2754E+02**	**-4.2597E+02**	2.3471E+00

has motivated us to turn our attention to propose novel self-adaptive strategies for DE, which will be discussed in detail in the next section.

4.2 Self-adaptive algorithms

Due to the rapid developement of optimization problems in terms of complex landscapes and high dimensionality, DE may not always achieve acceptable solutions in a reasonable time [1]. Indeed, it has been shown by several studies that DE

may get trapped in local optima and involves a slow convergence rate[80, 99, 71]. The aformentioned drawbacks can be potentially overcame by proposing flexible search operators and/or introducing novel self-adaptive strategies for DE parameters. These strategies tend to tune the parameters based on trial-and-error procedures (offline), or adaptively modifying them, where a feedback from the search history is considered [1, 15]. Following this context, we are focusing to propose new online self-adaptive strategies. These strategies are integrated in different proposed DE variants, which will be explained in the next sub-sections.

4.2.1 A dimension-based adaptive differential evolution for optimization problems (DADE)

Following the same context, another adaptive differential evolution algorithm is proposed. The proposed parameter adaptation strategy relies on a hybrid self-adaptive technique to adapt F parameter. Furthermore, a novel dimension-based adaptation strategy is proposed to control CR parameter for each dimension. To the best of our knowledge, it is the first technique attempting to control CR parameter for each dimension instead of each individual. In the following subsections, the proposed strategies and the algorithmic combination are explained in detail.

4.2.1.1 A hybrid approach to adapt F parameter

Our hybrid strategy for F parameter is slightly similar to the proposition used in JADE [100] which computes the Lehmer mean of the successful set of F parameters

denoted as S_F at each generation as follows:

$$mean_L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F} \tag{4.13}$$

Then, a location parameter μ_F is computed as follows:

$$\mu_F = (1 - c).\mu_F + c.mean_L \tag{4.14}$$

where c is is a positive constant between 0 and 1 and μ_F is initialized by 0.5. Afterwards, F value parameter for each individual is generated using Cauchy distribution as follows:

$$F_i = randc(\mu_F, 0.1); \tag{4.15}$$

It has been stated that this procedure forces the algorithm to produce permanently large values for F which ultimately decrease the exploitation capability[100]. In order to achieve balance between exploration and exploitation, a switching mechanism randomly chooses between this technique and a simple procedure [10] to produce smaller F values as follows:

$$F = 0.1 + \alpha * 0.9 \tag{4.16}$$

where α is a randomly generated number from uniform distribution in [0,1]. The switching between the strategies is performed by randomly generating a value between [0,1]. If this value is smaller than a parameter λ, then the F values are generated using 4.15. Otherwise, F values are generated using 4.16 . Our param-

eter adaptation strategy for the mutation scale factor is depicted in Algorithm 10.

Algorithm 10 Mutation parameter adaptation strategy

```
Input: successful F parameters set denoted as S_F
Output: New F for each individual
if rand < λ then
    if S_F is not empty then
        Compute the Lehmer mean of S_F using (4.13)
        Compute the location parameter μ_F using (4.14)
        Generate a different value F for each individual using Cauchy distribution
        with μ_F
    else
        Generate a different value F for each individual using Cauchy distribution
        with μ_F = 0.5
    end if
else
    Generate a different value F for each individual using (4.16)
end if
```

4.2.1.2 A reinforcement learning technique to adapt CR parameter

A novel yet simple reinforcement learning technique is proposed to efficiently control CR parameter for each dimension. After applying the crossover between the offspring population and the parents, a data structure called $MatrixCR$ is exploited to record whether a given dimension is taken from the parent or from the offspring. Then, the evaluation phase takes place. If an individual i is improved, a fitness reward $reward(i)$ is computed as follows:

$$reward(i) = 0.5 * reward(i) + 0.5 * (f(i) - f^*(i)) \tag{4.17}$$

Otherwise, it will be penalized as follows:

$$reward(i) = 0.5 * reward(i) - 0.5 * (f(i) - f^*(i)) \tag{4.18}$$

where $f(i)$ and $f^*(i)$ are the fitness of the parent and the new individual respectively. Then, it will be normalized as follows:

$$NormReward(i) = \frac{reward(i) - min(reward)}{max(reward) - min(reward)} \tag{4.19}$$

Afterwards, a parent and offspring scores are computed for each dimension using $NormReward(i)$, which stores the reward of all individuals. The objective is to estimate the contribution of the parents and offspring in improving the fitness. If the parent weight is larger than the offspring one, then the CR value for the given dimension is decreased. This scenario would force sharing the value of the parent for this dimension during the crossover procedure. Otherwise, the CR value is increased. Computing the weights and updating CR parameters are depicted in Algorithm 11.

This proposition is considered as a dimension-based parameter adaptation strategy. To the best of our knowledge, it is the first work that adapts CR parameter for each dimension instead of each individual. Indeed, having a unified CR value for all the dimensions might not be always successful due to the potential weak correlation between dimensions. Moreover, although parameter adaptation strategies does not involve a serious computational burden [70, 10], other strategies in the litterature may need serious computational time [17] as noticed in

Algorithm 11 Crossover parameter adaptation strategy

Input: $MatrixCR$, $NormReward$, dim, $PopSize$, $parent_{weight}$, $offspring_{weight}$
Output: new CR for each dimension
while $i < dim$ **do**
  while $j < PopSize$ **do**
    if $MatrixCR(j, i) = 0$ **then**
      $parent_{weight}(i) += NormReward(j)$
    else
      $offspring_{weight}(i) += NormReward(j)$
    end if
  end while
  if $parent_{weight}(i) = offspring_{weight}(i)$ **then**
    $CR(i) = 0.5$
  else
    if $parent_{weight}(i) < offspring_{weight}(i)$ **then**
      $CR(i) = 0.6$ + rand *(0.3)
    else
      $CR(i) = 0.4$ - rand *(0.3)
    end if
  end if
end while

[58] where DE parameters are adapted using the optimization process of Harmony Search (HS). Therefore, adapting the parameters for each dimension can considerably reduce the computational time involved in the learning process of the algorithm.

4.2.1.3 Combination of the algorithmic components

Initially, our algorithm randomly generates F values (for each individual) and CR values (for each dimension) to be applied. Afterwards, a simple DE algorithm is applied involving DE/current-to-pbest/1 [100] which is expressed as follows:

$$v_i^{G+1} = x_{best}^G + F.(x_{pbest}^G - x_i^G) + F.(x_{r1}^G - x_{r2}^G)$$

where x_{pbest}^G is one of the best $(100p)$ percent of the solutions in the current population, x_{r1}^G and x_{r2}^G are randomly chosen individuals from the current population. It is important to point out that p represent the fraction of the best individuals to be used in the mutation. This mutation strategy has been succesfully applied within several DE variants [80, 82, 100] thanks to its balance capability between exploration and exploitation. However, our proposition picks individuals randomly only from the current population (no archive), which simplifies the algorithm and reduces its memory complexity. Furthermore, the success of any proposed parameter adaptation strategy depends on a mutation strategy that slightly favors exploitation [74]. Following the same context, we have set p=0.02 ($p \in [0.05,0.2]$ in other proposals [82, 100]). Then, a binomial crossover is performed to generate vectors. During this phase, $MatrixCR$ is used to record whether a given dimension in the trial to be generated is taken from the parent or the offspring. This

procedure is applied as an initial step to adapt CR parameter for the next generation. After performing one generation of DE, the proposed parameter adaptaion strategy takes place to generate F and CR for the next generation. Finally, a linear reduction of the population size is performed. It eliminates a fraction of the worst individuals to accelerate the convergence rate during the search process [82]. The new population size is computed as follows:

$$NP_{G+1} = round(\frac{NP^{min} - NP^{init}}{MaxIT}.IT + NP^{init}) \tag{4.20}$$

where NP^{min} is the smallest possible population size, NP^{init} is the starting population size, $MaxIT$ is the maximum number of iterations and IT is the current iteration. The proposition is depicted in Algorithm 12.

4.2.1.4 Experimental results

An evaluation of our proposal is presented on the first version of the problem at hand. Similarly, our algorithm is evaluated as well on the CEC 2011 test suite. The algorithm is compared with several state-of-the-art DE variants. The parameter setting of the compared algorithms are given in Table 4.15.

Table 4.15: Parameter setting of the compared algorithms

Algorithm	Parameters
Our proposition	$PopSize = dim$*7, $\lambda = 0.5$, $p = 0.02$, $c = 0.3$
EPSDE	Parameters taken from [90]
SADE	Parameters taken from [11]
JADE	Parameters taken from [100]
SHADE	Parameters taken from [80]

Algorithm 12 Dimension-based adaptive differential evolution (DADE)

Input: population *pop* of $PopSize$ individuals, dim, $MatrixCR$, $reward$
Output: *pop*
while Budget is not consumed **do**
  $popold \leftarrow pop$
  Apply Algorithm 1 to generate F value for each individual
  Apply Algorithm 2 to generate CR value for each dimension
  for each individuals i in *popold* **do**
    Apply the mutation strategy to generate $mutant_j^i$
    while j $< dim$ **do**
      if rand $< CR(j)$ **then**
        $u_j^{G+1} \leftarrow popold_j^i$
        $MatrixCR(i, j) \leftarrow 1$
      else
        $u_j^{G+1} \leftarrow mutant_j^i$ //crossover procedure
        $MatrixCR(i, j) \leftarrow 0$
      end if
      $j \leftarrow j + 1$
    end while
    if u_j^{G+1} is better than $popold_j^i$ **then**
      $pop_j^i \leftarrow u_j^{G+1}$
      Update the corresponding entry in $reward$ according to (2.14)
    else
      Penalize the corresponding entry in $reward$ according to (2.15)
    end if
    Normalize $reward$ according to (2.16)
    Apply linear reduction of the population size to eliminate a fraction of the worst individuals using (3.1)
  end for
end while

4.2.1.5 Comparison on CEC 2011 benchmark

The results of our proposal as well as the other algorithms are depicted in Table 4.16. This table shows the best and mean values of 25 runs of each algorithm for each problem (CEC 2011 experimentation protocol). The best fitness found for each function is in bold. Mean results that are significantly better than the ones of the other algorithms, according to the Kruskal-Wallis statistical test at 95% confidence level followed by a Tukey-Kramer post hoc test, are also in bold. The results presented in Table 4.17 show that our proposal outperforms the other DE variants. It can significantly outperform SHADE in 8 functions, SADE in 10 functions, JADE in 6 functions and EPSDE in 13 functions.

4.2.1.6 Comparison with variants of the proposition

In order to discuss the influence of each algorithmic component on the performance of our proposition, a comparative study has been conducted. The first variant is without linear size reduction is named DADE1, and the second is the proposal without the proposed control parameter strategy named DADE2. To conduct the comparison, the convergence rate of the aforementioned variants is presented, where an ensemble of functions has been picked to show the convergence rate of each variant. Figure 4.1, 4.2 and 4.4 represents the convergance rate of DADE and its variants, where the horizontal axe represents the fitness value and the vertical one represents the number of iterations. DADE exhibits a quicker convergence rate towards the best solutions in functions 2, 5 and 10 compared to its variants. The results reveal that DADE1 can obtain similar performance compared to our

Table 4.16: Comparison of DADE with state-of-the-art algorithms on the CEC 2011 test suite

		SHADE	SADE	JADE	EPSDE	DADE
F1	Best	2.73E-02	6.31E-01	2.57E-10	5.45E+00	**0.00E+00**
	Mean	1.85E+01	7.59E-01	2.26E-01	9.10E+00	**0.00E+00**
F2	Best	-2.44E+01	-1.95E+01	-2.45E+01	-2.19E+01	**-2.78E+01**
	Mean	-2.30E+01	-1.70E+01	**-2.34E+01**	-2.02E+01	**-2.31E+01**
F3	Best	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
	Mean	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
F4	Best	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F5	Best	-3.66E+01	-3.32E+01	-3.64E+01	-3.60E+01	**-3.70E+01**
	Mean	**-3.60E+01**	-3.22E+01	**-3.56E+01**	-3.22E+01	**-3.36E+01**
F6	Best	-2.91E+01	-2.63E+01	**-2.92E+01**	-2.88E+01	-2.91E+01
	Mean	**-2.90E+01**	-2.41E+01	**-2.90E+01**	-2.01E+01	-2.76E+01
F7	Best	9.06E-01	1.24E+00	9.10E-01	1.12E+00	**6.30E-01**
	Mean	1.12E+00	1.37E+00	1.17E+00	1.30E+00	**1.04E-01**
F8	Best	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
	Mean	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
F9	Best	1.14E+03	**7.69E+02**	1.13E+03	3.98E+04	2.82E+04
	Mean	2.22E+03	**1.73E+03**	2.40E+03	9.28E+04	8.22E+04
F10	Best	-2.18E+01	-2.18E+01	-2.18E+01	-2.02E+01	-2.18E+01
	Mean	**-2.16E+01**	**-2.16E+01**	**-2.14E+01**	-1.74E+01	**-2.16E+01**
F11	Best	5.15E+04	**5.12E+04**	5.15E+04	5.21E+04	5.17E+04
	Mean	5.32E+04	**5.21E+04**	5.24E+04	5.86E+04	5.25E+04
F12	Best	**1.07E+06**	**1.07E+06**	**1.07E+06**	**1.07E+06**	1.08E+06
	Mean	1.10E+06	1.09E+06	**1.07E+06**	1.09E+06	1.22E+06
F13	Best	1.55E+04	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
	Mean	1.55E+04	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
F14	Best	**1.80E+04**	1.81E+04	**1.80E+04**	1.83E+04	**1.80E+04**
	Mean	**1.81E+04**	**1.81E+04**	**1.83E+04**	1.86E+04	**1.81E+04**
F15	Best	**3.27E+04**	3.28E+04	**3.27E+04**	3.29E+04	**3.27E+04**
	Mean	**3.27E+04**	3.28E+04	3.29E+04	3.30E+04	**3.27E+04**
F16	Best	1.26E+05	1.26E+05	1.26E+05	1.31E+05	**1.25E+05**
	Mean	1.29E+05	1.28E+05	**1.33E+05**	1.42E+05	**1.26E+05**
F17	Best	1.88E+08	1.87E+06	**1.87E+06**	1.93E+06	**1.87E+06**
	Mean	1.91E+06	1.90E+06	1.91E+06	2.06E+06	**1.89E+06**
F18	Best	9.37E+05	**9.33E+05**	9.35E+05	3.24E+06	9.38E+05
	Mean	9.40E+05	**9.38E+05**	9.39E+05	6.06E+06	9.41E+05
F19	Best	9.39E+05	9.41E+05	9.39E+05	4.47E+06	**9.38+05**
	Mean	9.52E+05	9.46E+05	9.92E+05	7.16E+06	**9.45+05**
F20	Best	**9.34E+05**	9.35E+05	9.36E+05	4.16E+06	9.37E+05
	Mean	**9.40E+05**	**9.37E+05**	**9.40E+05**	6.21E+06	**9.40E+05**
F21	Best	1.41E+01	1.66E+01	1.31E+01	1.66E+01	**8.66E+00**
	mean	1.75E+01	1.97E+01	1.72E+01	1.97E+01	**1.38E+01**
F22	Best	1.31E+01	1.68E+01	**1.19E+01**	1.68E+01	1.50E+01
	Mean	**1.99E+01**	2.18E+01	**1.66E+01**	2.18E+01	**1.88E+01**

Table 4.17: Comparison of DADE with state-of-the-art algorithms using the statistical test

VS	DADE	22 functions of CEC 2011 test suite
EPSDE	+(better)	0
	-(worse)	13
	=(no sign)	9
SADE	+(better)	3
	-(worse)	10
	=(no sign)	9
JADE	+(better)	2
	-(worse)	6
	=(no sign)	14
SHADE	+(better)	1
	-(worse)	8
	=(no sign)	13

algorithm, which can be concluded from Figure 4.5, 4.6, 4.7, 4.8, 4.9 and 4.10. Moreover, Figure 4.3 shows that DADE2 outperforms DADE for the function 9. It can be noticed as well that DADE1 has a slower convergence rate when compared with the other variants. Indeed, these results show that our proposed control parameter strategy has a great impact on the proposition performance.

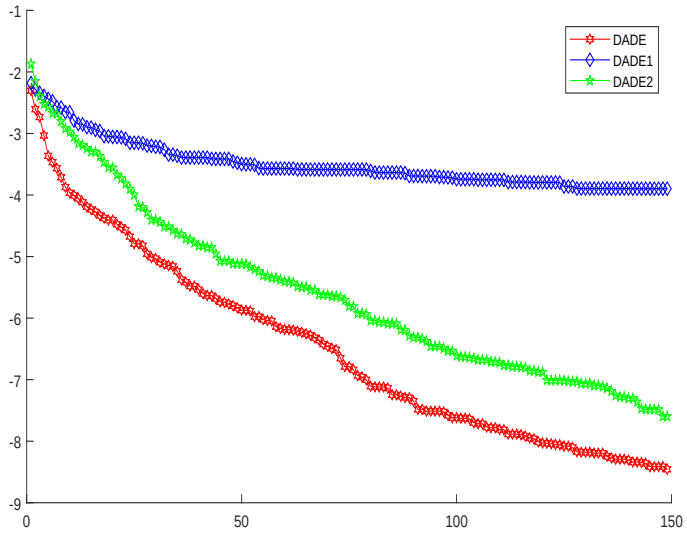


Figure 4.1: Convergence rate of DADE variants in function 2

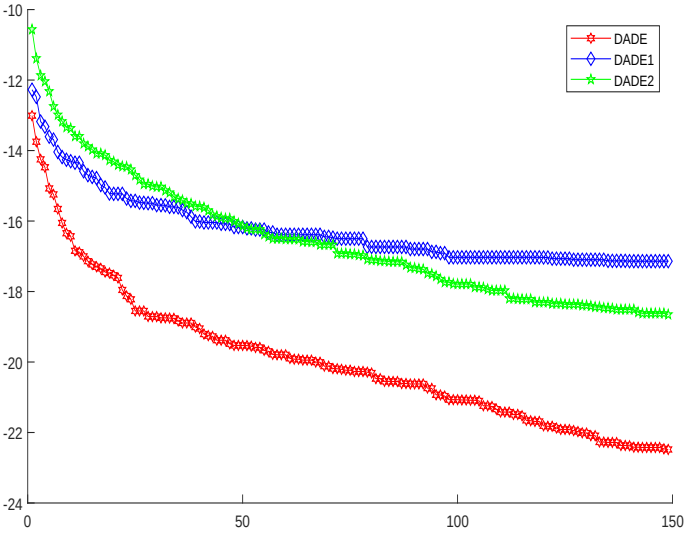


Figure 4.2: Convergence rate of DADE variants in function 5

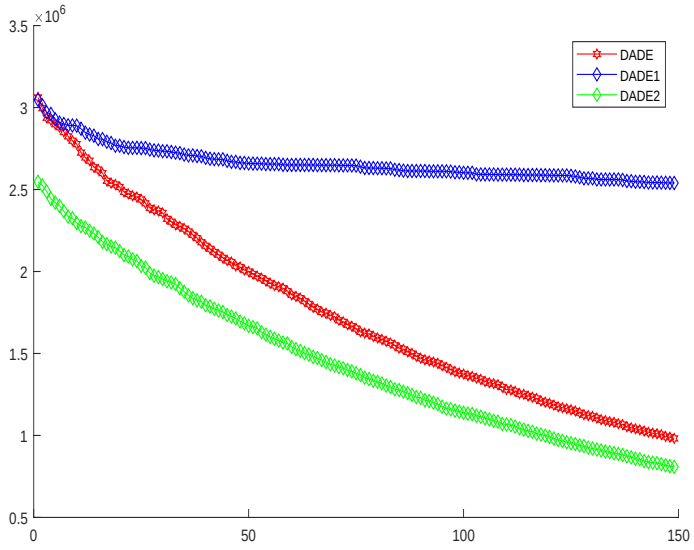


Figure 4.3: Convergence rate of DADE variants in function 9

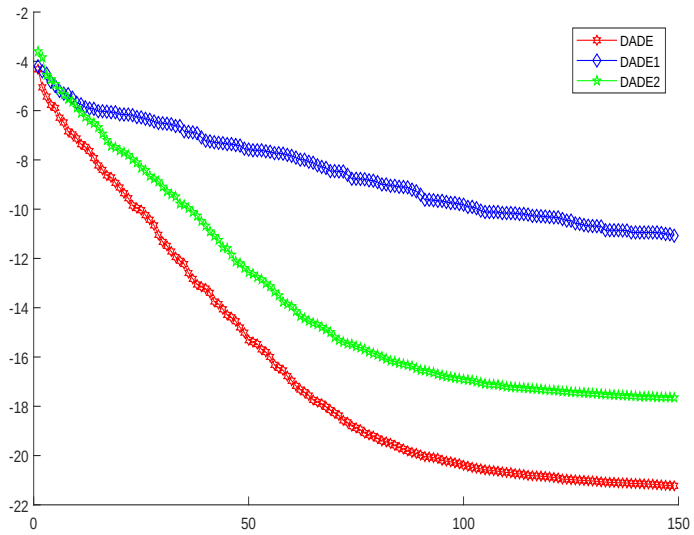


Figure 4.4: Convergence rate of DADE variants in function 10

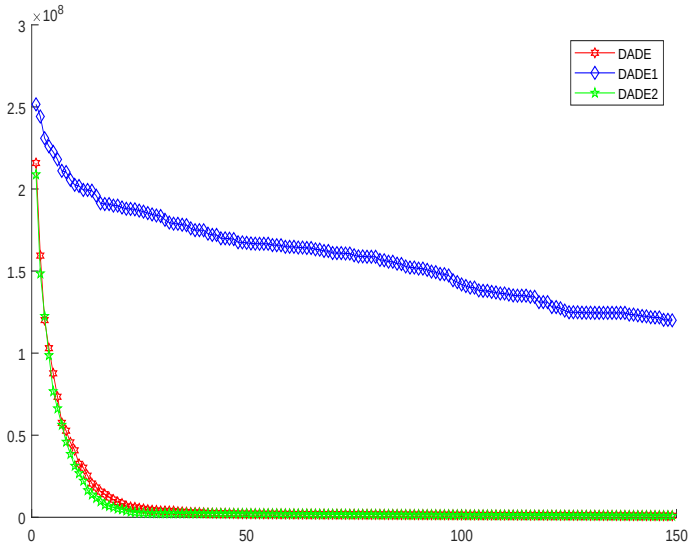


Figure 4.5: Convergence rate of DADE variants in function 11

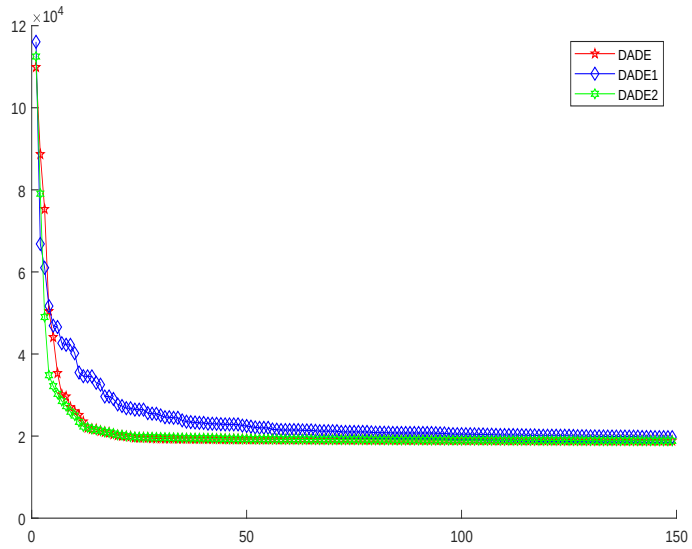


Figure 4.6: Convergence rate of DADE variants in function 14

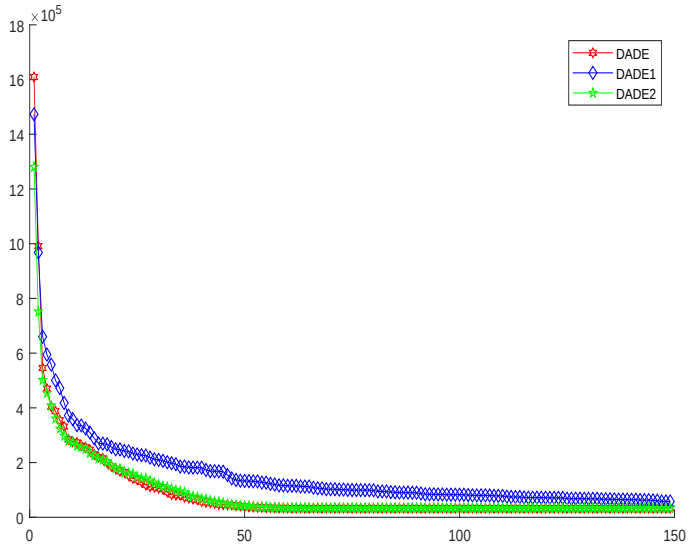


Figure 4.7: Convergence rate of DADE variants in function 15

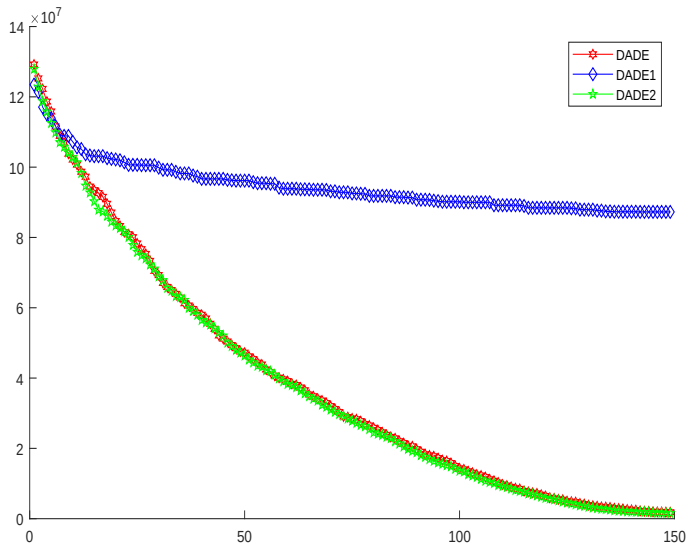


Figure 4.8: Convergence rate of DADE variants in function 20

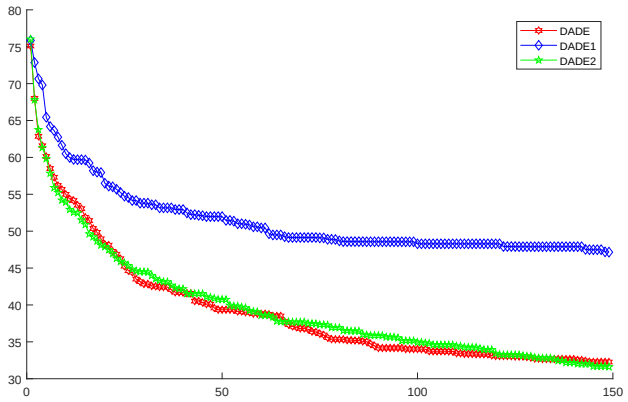


Figure 4.9: Convergence rate of DADE variants in function 21

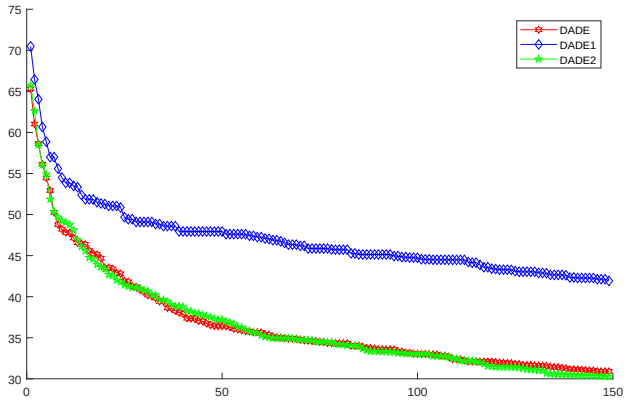


Figure 4.10: Convergence rate of DADE variants in function 22

4.2.1.7 Comparison on the problem at hand

It should be noted that $PopSize$ is set to 400 in our experimentation protocol. The proposal has been run 30 times. The best, mean, worst solutions and the standard deviation of each algorithm have been gathered. It is stated from Table 4.18 that

our proposal could obtain the best solution compared to the other variants on the first version of the problem at hand.

Table 4.18: Comparison of DADE with state-of-the-art algorithms on the first version of the problem at hand

	Best	Mean	Worst	Std
SADE	-3.271E+03	-2.920E+03	-2.780E+03	81.49
JADE	-3.194E+03	-2.840E+03	-2.420E+03	96.23
EPSDE	-3.200E+03	-2.850E+03	-2.430E+03	102.65
SHADE	-3.097E+03	-2.944E+03	-2.697E+03	44.28
DADE	**-3.397E+03**	**-3.275E+03**	**-3.163E+03**	29.49

Moreover, Figure 4.11 shows the advantage of our proposition in terms of convergence rate compared to DADE1 and DADE2. It is noticed that removing the control parameter strategy ultimately decreases the convergence rate, which confirms the results found for CEC 2011 test suite.

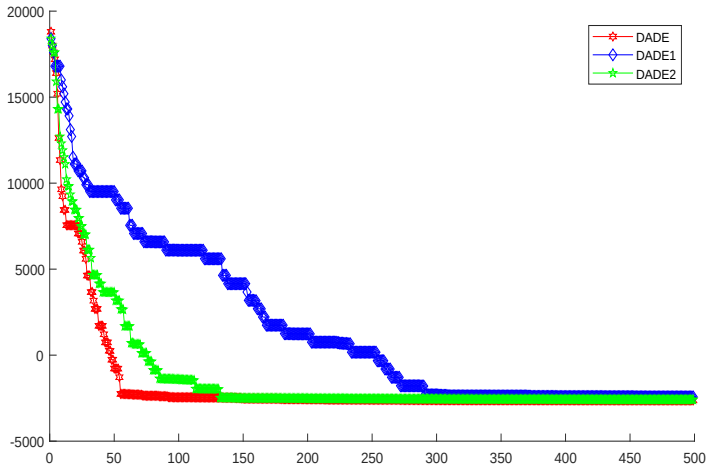


Figure 4.11: Convergence rate of DADE variants in the problem at hand

Table 4.19: Comparison of DADE with state-of-the-art algorithms on the second version of the problem at hand

	Best	Mean	Worst	Std
SADE	**-4.3065E+02**	-4.2523E+02	-4.1173E+02	3.5328E+00
JADE	**-4.3065E+02**	-4.2356E+02	-4.0376E+02	5.1813E+00
EPSDE	-4.2842E+02	-4.1968E+02	-4.0693E+02	5.1628E+00
SHADE	**-4.3065E+02**	-4.2370E+02	-4.0569E+02	6.9628E+00
DADE	**-4.3065E+02**	**-4.2654E+02**	**-4.2397E+02**	2.0171E+00

Similarly, the proposed algorithm could obtain the best solution known so far for the second version of the problem at hand. The results are depicted in 4.19. Besides, DADE could outperform the other variants in terms of mean and worst solutions.

As it can be noticed, DADE relies on a simple reinforcement learning technique to control CR parameter, which has shown to be competitive compared to several state-of-the-art self-adaptive DE algorithms. Indeed, this technique has motivated us to propose other machine learning-based parameter adaptation strategies. Following this context, novel parameter adaptation strategies are proposed in the following sub-sections.

4.2.2 An eigenvector-enhanced parallel adaptive differential evolution for electric motor design (PEADE)

This sub-section is devoted to explain our proposal entitled Parallel eigenvector-enhanced adaptive DE (PEADE). The main contribution of PEADE can be seen as:

- The proposition of a new adaptive differential evolution algorithm.
- The parallelization of the approach using GPU platforms.
- The topology optimization of a recent electric motor.

4.2.2.1 Modified Pheromone matrix-based adaptation strategy

The first component of PEADE is a modified version of PMS, which was proposed in HDE (see Section 1). Our modified strategy relies on a matrix called Pheromone Matrix (PM). PM represents a pool of 100 discret combinations of F and CR, where each combination has a score. It should be stated that PM rows represent F values, and the culumns represent CR values. The combination values varie in the discret range [0.1, 1] as it can be seen in Figure 4.12. The first phase called "learning phase", which consumes 10% of the budget. In this phase, a combination is randomly selected from PM to be applied on the search operators. Afterwards, a score is assigned to the current combination by updating the corresponding entry in PM. Assigning a score depends on the contribution of the given combination on each individual. If it could improve a given individual, it is rewarded according to equation 4.8. It is penalized according to equation 4.9 otherwise. PM values

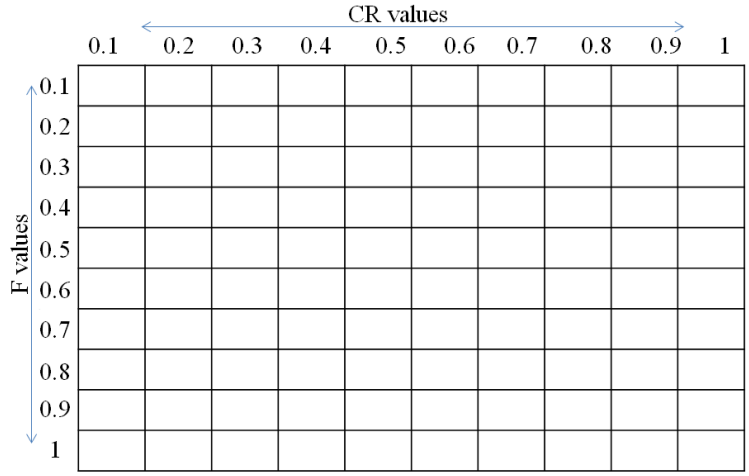


Figure 4.12: The pheromone matrix

are then normalized as follows:

$$PM(i,j) = \frac{PM(i,j) - min(PM)}{max(PM) - min(PM)} \tag{4.21}$$

After the learning phase, the next phase called "deduction phase" takes place. A new combination for the population is computed using the weighted Lehmer mean. The weighted Lehmer mean is applied using the best 10 combinations. Indeed, using the weighted Lehmer mean allows generating new combination closed to the combinations that has highest weight (highest score), which can not achieved applying the arithmetic mean. The new combination is generated as follows:

$$F = \frac{\sum_{k=1}^{n} PM_k(i,j).(i/10)^2}{\sum_{k=1}^{n} PM_k(i,j).(i/10)} \tag{4.22}$$

and

$$CR = \frac{\sum_{k=1}^{n} PM_k(i,j).(j/10)^2}{\sum_{k=1}^{n} PM_k(i,j).(j/10)} \tag{4.23}$$

where i and j are the k^{th} best indices of PM respectively.

4.2.2.2 The proposed mutation framework

A simple mutation framework is introduced in order to gradually enhance the exploitation of the algorithm. The algorithm starts with DE/current-to-pbest/1 which is presented as follows:

$$v_i^{G+1} = x_i^G + F.(x_{pbest}^G - x_i^G) + F.(x_{r1}^G - x_{r2}^G) \tag{4.24}$$

where x_{pbest}^G is one of the $p\%$ best individuals in the current population. DE/current-to-pbest/1 has been introduced in [99], where a balance between exploration and exploitation has been achieved. However, it has been stated that while linear reduction of the population is applied, this strategy performance can be limited when very small population size is present. Accordingly, a high possibility that the difference between x_{pbest}^G and x_i^G is zero may occur. In other words, the parent vector can be one of the x_{pbest} individuals. As a consequence, the mutation would perturb the parent vector using two random individuals from the population. To potentially overcome this scenario, a modified mutation equation called DE/current-to-centroid/1 is introduced. DE/current-to-centroid/1 computes the centroid individual of the $p\%$ best individuals using arithmetic mean. Afterwards, the centroid will be involved in equation (4.24) instead of x_{pbest}^G. This scenario

would allow all the individuals to evolve in the same manner, where they would move toward a promising region that shares the information of all x_{pbest}. Besides, a switching technique to choose between the two strategies is proposed. $SwitchingProbability$ parameter is introduced to select DE/current-to-pbest/1 during the first iterations. Then, gradually, $SwitchingProbability$ is decreased to favor DE/current-to-centroid/1 in the last phases of the search process in order to enhance the exploitation capability of the approach. The general framework is summarized in Algorithm 13.

Algorithm 13 The proposed mutation framework

```
Input: The population, SwitchingProbability
Output: The mutant population
if rand < SwitchingProbability then
    Apply DE/current-to-pbest/1
else
    Apply DE/current-to-centroid/1
end if
```

4.2.2.3 The proposed crossover framework

A recent crossover search operator called eigenvector-based crossover is introduced. Indeed, eigenvector-based crossover can be appropriate when tackling landscapes of highly correlated individuals. According to [21], exploiting information such as mean value, variance and covariance matrix during the crossover phase may have an impact on the final results. It has been shown that the normal binomial crossover does not take such information in consideration [91]. In order to relax DE dependence on the original coordinate system, a covariance matrix is computed to

provide an eigen coordinate system [33, 91]. The new system is then used to apply the crossover. The eigenvector-based crossover can be explained in the following steps:

Step 1: Calculate the covariance matrix C of the current population.

Step 2: Perform eigen decomposition as follows:

$$C = BD^2B^T \tag{4.25}$$

where B and B^T are orthogonal matrices and D is a diagonal matrix composed of Eigen values.

Step 3: Move the parent and the mutant vectors to the new cordinate system as follows:

$$x^{'}_{i,G} = B^T.x_{i,G} \tag{4.26}$$

$$v^{'}_{i,G} = B^T.v_{i,G} \tag{4.27}$$

Step 4: Perform the binomial crossover on $x^{'}_{i,G}$ and $v^{'}_{i,G}$:

$$u^{'}_{i,j,G+1} = \begin{cases} v^{'}_{i,j,G} & \text{if } j = \sigma_j \quad \text{or } R_j < CR \\ x^{'}_{i,j,G} & \text{otherwise} \end{cases} \tag{4.28}$$

Step 5: Transfer $u'_{i,j,G+1}$ to the original coordinate system as follows:

$$u_{i,j,G+1} = B.u'_{i,j,G+1} \tag{4.29}$$

The contribution of the eigenvector-based crossover on DE performance has been deeply investigated in several studies, such as [33, 34]. However, we have noticed that applying this crossover ignoring its powerful exploitation capability can make the algorithm stuck in a local optimum. In our study, the normal binomial crossover can be seen as an exploration phase of PEADE. We propose to hybridize the two search operators in a single framework, where the exploitation is gradually enhanced by decreasing $SwitchingProbability$ parameter that favors the eigenvector-based crossover as time goes by. Our proposed crossover framework can be summarized in Algorithm 14.

Algorithm 14 The proposed crossover framework

```
Input: The population, SwitchingProbability
Output: The mutant population
if rand < SwitchingProbability then
    Apply binomial crossover
else
    Apply eigenvector-based crossover
end if
```

4.2.2.4 Combination of the algorithmic components

This subsection is devoted to present how the explained algorithmic components are combined for our proposal.

First, a population of NP individuals is generated and $SwitchingProbability$

is initialized with 0.9. Afterwards, the proposed parameter adaptation strategy is performed. In the learning phase, a randomly generated pair (F/CR) is applied on the population. The objective of this step is to update PM in order to exploit its results afterwards. Using a chosen combination, the mutation framework is applied, where DE/current-to-pbest/1 is favored to be applied since $SwitchingProbability$ value is large during the first iterations. Then, the crossover framework phase takes place favoring the binomial crossover to be performed at the beginning. The eigenvector-based crossover is favored in the last iterations since it enhances DE exploitation capability on landscapes of highly correlated variables. It should be mentioned that $SwitchingProbability$ is linearly decreased using the following equation:

$$SwitchingProbability = max\left\{0, 0.9 - \frac{CurrentIteration}{Budget}\right\} \tag{4.30}$$

At the end of each iteration of PEADE, a linear reduction of the population size is applied, where a fraction of the worst individuals are removed. This procedure tends to be useful in several proposals such as [4, 33, 82], where it could accelerate the convergence rate. The new population size is computed as follows:

$$NP_{G+1} = round(\frac{NP_{min} - NP_{init}}{Budget}.CurrentIteration + NP_{init}) \tag{4.31}$$

where NP_{min} is the smallest population size, NP_{init} is the initial population size, $Budget$ is the maximum number of iterations and $CurrentIteration$ is the current iteration. The whole approach can be depicted in Algorithm 15.

Algorithm 15 Parallel eigenvector-enhanced adaptive DE (PEADE)

```
Randomly initialize the population pop of NP individuals
SwitchingProbability ← 0.9
while Budget is not consumed do
    Apply pheromone matrix-based adaptation strategy
    Apply the proposed mutation framework using Algorithm 13
    Apply the proposed crossover framework using Algorithm 14
    Apply the selection procedure of DE
    Apply linear reduction of SwitchingProbability using equation 4.30
    Apply linear reduction of the population using equation 4.31
end while
```

4.2.2.5 Empirical study

The obtained results of PEADE for the application at hand as well as a set of 22 real world problems of the CEC 2011 test suite are presented. It can be noticed that PEADE does not contain a big number of parameters compared to recent self-adaptive DE variants. Indeed, there are just four parameters, which are the maximum population size, the minimum population size, the initial value of $SwitchingProbability$ and the fraction of the best solutions in the current population p. PEADE parameters can be summarized in Table 4.20. PEADE per-

Table 4.20: Algorithm parameters

Parameter	Parameter value
maximum population size	$D * 7$
minimum population size	10
$SwitchingProbability$	0.9
p	0.1

formance is investigated by conducting a comparison with recent state-of-the-art adaptive DE such as JADE [99], SHADE [80], L-SHADE [82], SPS-EIG-LSHADE [33] and our hybrid DE previously proposed since it also includes a parameter adaptation strategy and we call it (HDE). Besides, another comparison is performed with several variants of PEADE to show how the algorithmic components influence PEADE performance. PEADE variants are set as follows:

- PEADE 1: the proposal without eigenvectors-based crossover.
- PEADE 2: the proposal with only DE/current-to-pbest/1.
- PEADE 3: the proposal without the proposed parameter adaptation strategy.

Finally, a comparison in terms of acceleration time is conducted with the parallel version.

4.2.2.6 Comparison on CEC 2011

Table 4.21 summarizes the obtained results in terms of mean and best values for each application of CEC 2011 test suite. It is noticed that PEADE outperforms the other algorithms. To validate the results, a Kruskal-Wallis statistical test is performed, where the best means of each function are set in bold as it can be seen in Table 4.21, the rows show best and mean values of 25 runs of each algorithm for each function. Table 4.21 shows the advantage of PEADE over the other adaptive DE approaches. It can significantly outperform SPS-EIG-LSHADE in 9 functions,

Table 4.21: Comparison of PEADE with state-of-the-art algorithms on the CEC 2011 test suite

		EPSDE	SADE	JADE	SHADE	L-SHADE	SPS-EIG-LSHADE	PEADE
F1	Best	5.45E+00	2.57E-10	2.57E-10	2.73E-02	0.00E+00	0.00E+00	0.00E+00
	Mean	9.10E+00	2.26E-01	2.26E-01	1.85E+01	1.62E-04	2.35E-07	**2.87E-18**
F2	Best	-2.19E+01	-2.45E+01	-2.45E+01	-2.44E+01	-2.82E+01	-2.84+01	-2.73+01
	Mean	-2.02E+01	-2.34E+01	-2.34E+01	-2.30E+01	**-2.62E+01**	**-2.61E+01**	-2.29+01
F3	Best	**1.15E-05**	1.15E-05	1.15E-05	1.15E-05	1.15E-05	1.15E-05	1.15E-05
	Mean	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
F4	Best	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F5	Best	-3.60E+01	-3.64E+01	-3.64E+01	-3.66E+01	-3.68E+01	-3.68E+01	-3.68E+01
	Mean	-3.22E+01	**-3.56E+01**	-3.56E+01	**-3.60E+01**	**-3.63E+01**	**-3.62E+01**	-3.19E+01
F6	Best	-2.88E+01	-2.92E+01	-2.92E+01	-2.91E+01	-2.91E+01	-2.91E+01	-2.74E+01
	Mean	-2.01E+01	**-2.90E+01**	**-2.90E+01**	**-2.90E+01**	**-2.90E+01**	**-2.92E+01**	-2.24E+01
F7	Best	1.12E+00	9.10E-01	9.10E-01	9.06E-01	9.63E-01	7.11E-01	7.29E-01
	Mean	1.30E+00	1.17E+00	1.17E+00	1.12E+00	1.22E+00	1.12E+00	**9.91E-1**
F8	Best	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
	Mean	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
F9	Best	3.98E+04	1.13E+03	1.13E+03	1.14E+03	1.12E+03	1.72E+03	1.04E+03
	Mean	9.28E+04	2.40E+03	2.40E+03	2.22E+03	8.17E+03	2.77E+03	**1.78E+03**
F10	Best	-2.02E+01	-2.18E+01	-2.18E+01	-2.18E+01	-2.16E+01	-2.18E+01	-2.18E+01
	Mean	-1.74E+01	**-2.14E+01**	**-2.14E+01**	**-2.16E+01**	**-2.15E+01**	**-2.16E+01**	**-2.16E+01**
F11	Best	5.21E+04	5.15E+04	5.15E+04	5.15E+04	5.11E+04	5.10E+04	4.99E+04
	Mean	5.24E+04	5.24E+04	5.24E+04	5.22E+04	5.20E+04	5.19E+04	**5.08E+04**
F12	Best	**1.07E+06**	**1.07E+06**	1.07E+06	1.07E+06	1.07E+06	1.07E+06	1.07E+06
	Mean	1.09E+06	**1.07E+06**	**1.07E+06**	1.10E+06	**1.07E+06**	**1.07E+06**	**1.07E+06**
F13	Best	1.54E+04	1.54E+04	1.54E+04	1.55E+04	1.54E+04	1.54E+04	1.54E+04
	Mean	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
F14	Best	1.83E+04	1.80E+04	1.80E+04	1.80E+04	1.80E+04	1.80E+04	1.81E+04
	Mean	1.83E+04	1.83E+04	1.83E+04	**1.81E+04**	**1.81E+04**	**1.81E+04**	**1.81E+04**
F15	Best	3.29E+04	3.27E+04	3.27E+04	3.27E+04	3.27E+04	3.27E+04	3.27E+04
	Mean	3.30E+04	3.29E+04	3.29E+04	**3.27E+04**	**3.27E+04**	**3.27E+04**	**3.27E+04**
F16	Best	1.31E+05	1.26E+05	1.26E+05	**1.22E+05**	**1.22E+05**	1.23E+05	1.23E+05
	Mean	1.42E+05	1.33E+05	1.33E+05	1.29E+05	1.26E+05	1.24E+05	**1.23E+05**
F17	Best	1.93E+06	1.87E+06	1.87E+06	1.88E+08	1.83E+06	1.87E+06	**1.81E+06**
	Mean	2.06E+06	1.91E+06	1.91E+06	1.91E+06	1.86E+06	1.85E+06	**1.82E+06**
F18	Best	3.24E+06	9.35E+05	9.35E+05	9.37E+05	**9.29E+05**	9.33E+05	**9.29E+05**
	Mean	6.06E+06	9.39E+05	9.39E+05	9.40E+05	9.33E+05	9.33E+05	**9.31E+05**
F19	Best	4.47E+06	9.39E+05	9.39E+05	9.39E+05	9.38E+05	9.42E+05	**9.37+05**
	Mean	7.16E+06	9.92E+05	9.92E+05	9.52E+05	9.40E+05	9.40E+05	**9.39E+05**
F20	Best	4.16E+06	9.36E+05	9.36E+05	9.34E+05	9.30E+05	9.29E+05	**9.28E+05**
	Mean	6.21E+06	9.40E+05	9.40E+05	9.40E+05	9.32E+05	9.32E+05	**9.31E+05**
F21	Best	1.66E+01	1.31E+01	1.31E+01	1.41E+01	1.44E+01	1.07E+01	**9.45E+00**
	Mean	1.97E+01	1.72E+01	1.72E+01	1.75E+01	1.61E+01	**1.38E+01**	**1.49E+01**
F22	Best	1.19E+01	1.68E+01	1.19E+01	1.31E+01	**8.60E+00**	8.61E+00	1.64E+01
	Mean	2.18E+01	1.66E+01	1.66E+01	1.99E+01	1.42E+01	**1.24E+01**	1.83E+01

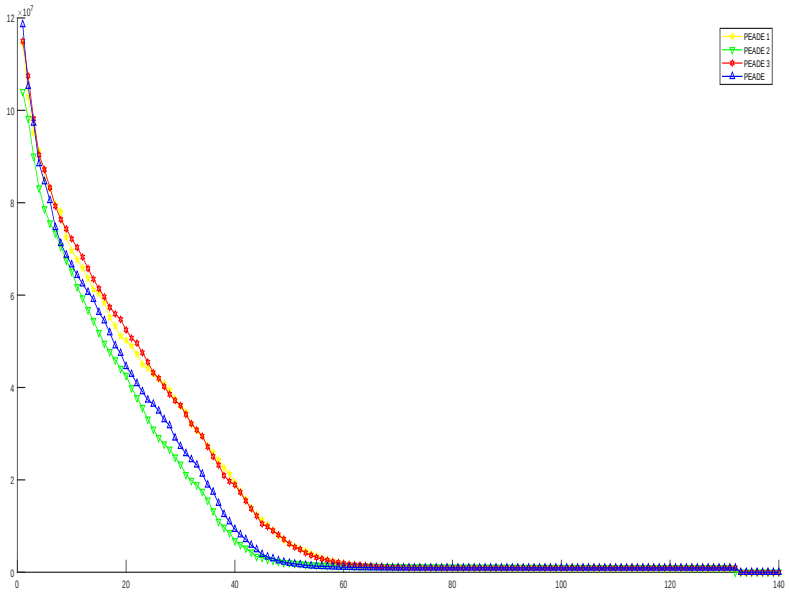
L-SHADE in 10 functions, SHADE and JADE and SADE in 12 functions. Finally, it can outperform EPSDE in 13 functions as it can be seen in Table 4.22.

Table 4.22: Comparison of PEADE using Kruskal-Wallis statistical test

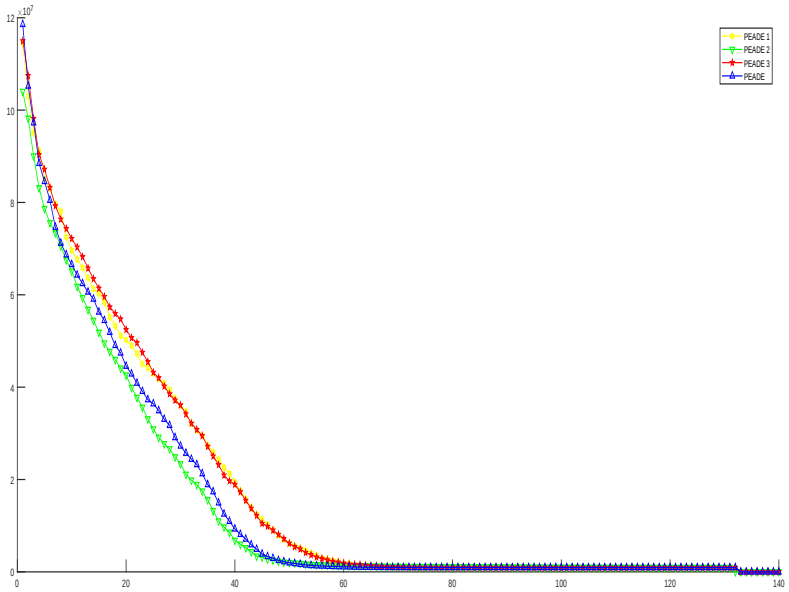
VS	Our proposition	22 functions of CEC 2011 test suite
SPS-EIG-LSHADE	+(better)	4
	-(worse)	9
	=(no sign)	9
L-SHADE	+(better)	3
	-(worse)	10
	=(no sign)	9
SHADE	+(better)	2
	-(worse)	12
	=(no sign)	8
JADE	+(better)	1
	-(worse)	12
	=(no sign)	9
EPSDE	+(better)	0
	-(worse)	13
	=(no sign)	9
SADE	+(better)	2
	-(worse)	12
	=(no sign)	8

The importance of each algorithmic components of PEADE can be noticed by running a different experimentation. This experimentation is performed to compute convergence rate, where functions 17, 18, 19 and 20 are considered.

It is noticed from Figure 4.14 that PEADE 1 and PEADE 3 have a slower convergence rate compared to PEADE and PEADE 2. It confirms the importance of the introduced eigenvectors-based crossover and parameter adaptation strategies. In the other hand, PEADE 2 could slightly obtain a similar performance

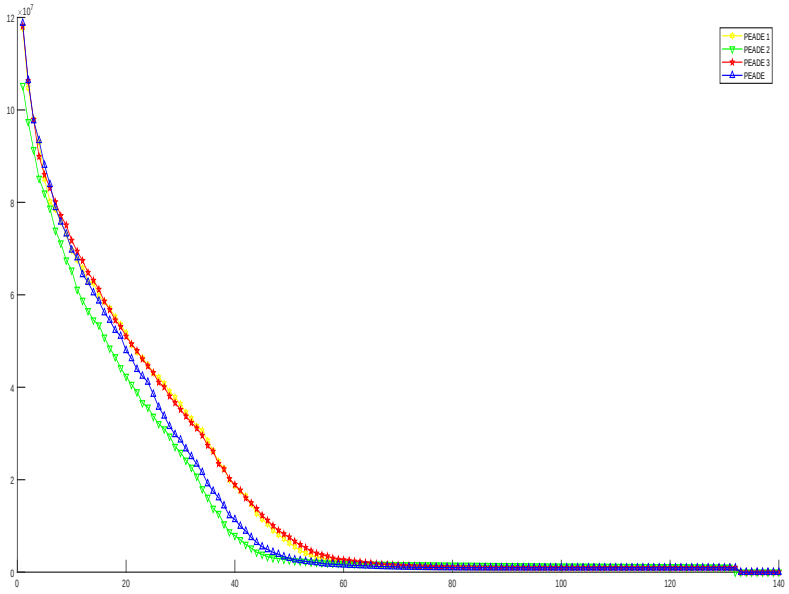


(a) Function 17

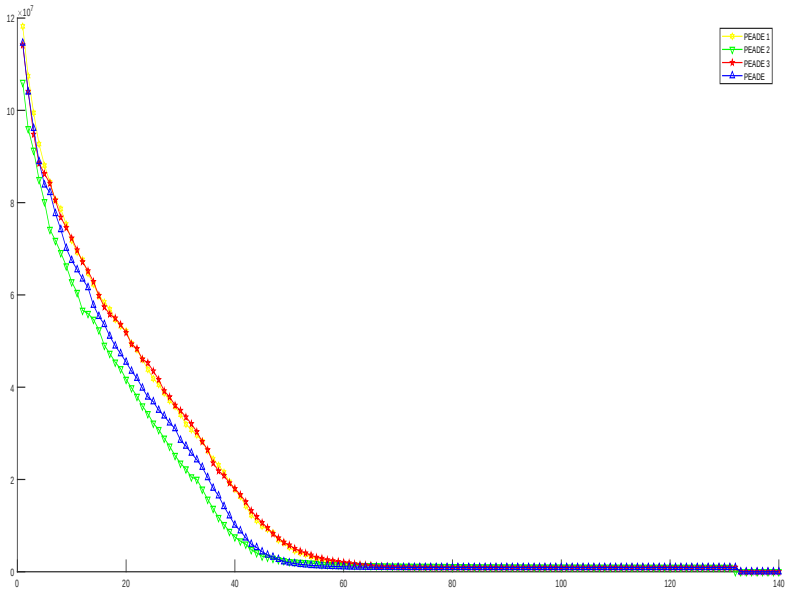


(b) Function 18

Figure 4.13: The average convergence rate of PEADE variants on functions 17, 18 after 25 runs.



(a) Function 19



(b) Function 20

Figure 4.14: The average convergence rate of PEADE variants on functions 19, 20 after 25 runs.

compared to PEADE in the first iterations. Even-though, its performance decreases gradually in the last iterations. This behavior reveal the importance of eigenvectors-based crossover for improving the final results.

4.2.2.7 Comparison on the problem at hand

PEADE has been tested on both versions of the problem at hand. PEADE has been run 30 times, where the best, mean and worst of each algorithm are collected. It is observed from Table 4.23 and Table 4.24 that PEADE can achieve the best performance compared to the other algorithms in terms of mean and best. Moreover, it is noticed that PEADE has a stable performance (standard deviation = 0).

Table 4.23: Comparison of PEADE with state-of-the-art algorithms on the first version of the problem at hand

	Best	Mean	Worst	Std
SADE	-3.271E+03	-2.920E+03	-2.780E+03	81.49
JADE	-3.194E+03	-2.840E+03	-2.420E+03	96.23
EPSDE	-3.200E+03	-2.850E+03	-2.430E+03	102.65
SHADE	-3.097E+03	-2.944E+03	-2.697E+03	44.28
L-SHADE	-3.197e+03	-3.044e+03	-2.797e+03	107
SPS-eig-LSHADE	-3.214e+03	-3.107e+03	-2.897e+03	98.42
PEADE	**-3.397E+03**	**-3.397E+03**	**-3.397E+03**	0

Table 4.24: Comparison of PEADE with state-of-the-art algorithms on the second version of the problem at hand

	Best	Mean	Worst	Std
SADE	**-4.3065E+02**	-4.2523E+02	-4.1173E+02	3.5328E+00
JADE	**-4.3065E+02**	-4.2356E+02	-4.0376E+02	5.1813E+00
EPSDE	-4.2842E+02	-4.1968E+02	-4.0693E+02	5.1628E+00
SHADE	**-4.3065E+02**	-4.2370E+02	-4.0569E+02	6.9628E+00
L-SHADE	**-4.3065E+02**	-4.3043E+02	-4.301E+02	2.7463E-01
SPS-eig-LSHADE	**-4.3065E+02**	-4.3050E+02	-4.3010E+02	2.4922E-01
PEADE	**-4.3065E+02**	**-4.3065E+02**	**-4.3065E+02**	0

4.2.2.8 The parallel implementation

The main reason behind proposing a parallel implementation is the high computational time of some components of PEADE. We have noticed that computing covariance matrix, the eigenvectors and the evaluation of individuals with some problems from CEC 2011 test suite are time-consuming. The algorithmic structure of PEADE has not been changed while implementing the parallel version. All the components have been parallelized using separate GPU kernels while CPU launches them in the correct order. On the one hand, seven kernels have been introduced for the parallel version. Six kernels of NP blocks and dim threads have been implemented for the initialization, the mutations, the crossover and the evaluation, whereas one kernel of NP threads has been implemented for the parameter adaptation strategy. On the other hand, three procedures has been implemented

using different CUDA libraries. The procedures are depicted as follows:

- Covariance matrix procedure: it is implemented to compute covariance matrix, where CUDA libraries such as CUBLAS and THRUST are involved.
- Eigenvectors procedure: it is implemented to compute the eigenvectors of the computed covariance matrix, where cuSolver CUDA library has been exploited.
- Linear reduction procedure: It has been stated that erasing the worst individuals from the population can be computationally expensive since it involves an intensive memory access. This procedure has been proposed to compute the new reduced population size. Then, the population is sorted based on the fitness of each individual using thrust CUDA library. Finally, the worst individuals will be placed at the end and they will be just ignored. Besides, this procedure linearly reduce $SwitchingProbability$ value.

4.2.2.9 Comparison with the parallel implementation

The results of parallel version of PEADE are presented in this sub-section. It has been noticed that some problems of the CEC 2011 test suite are appropriate to be parallelized such as functions 12, 17 and 18. It can be stated from Table 4.25 that the sequential PEADE can slightly achieve the same effeciency compared to the parallel version in case NP=400. Neverthless, the computational time decreases when we increase the population size achieving a speed up of 2.75x when $NP = 1000$.

Table 4.25: Comparison in terms of computational time on functions 12, 17 and 18 between the sequential and the parallel implementation of PEADE

		NP=400	NP=600	NP=800	NP=1000
F17	Sequential	35.23s	40.52s	55.57s	59.61s
	Parallel	34.68s	27.67s	24.07s	21.66s
F18	Sequential	55.58s	62.02s	73.54s	75.24s
	Parallel	18.24s	14.39s	14.28s	12.18s
F12	Sequential	184.46s	249.69s	316.80s	386.46s
	Parallel	25.91s	20.87s	17.86s	15.94s

The acceleration achieved in function 18 is investigated. It can be observed that the computational time decreases each time we increase the population size, which is justified thanks to the high occupation of GPU device. The parallel implementation could achieve an approximated speedup from 3x up to 6.17x. Similarly, Table 4.25 shows the acceleration achieved for function 12, where $D = 240$. It has been noticed that the computational time decreases clearly, which is ensured by increasing the population size and the high dimensionality of the given problem (high occupation of GPU device). The parallel implementation could obtain an approximated speedup from 7x up to 24.24x.

4.2.3 Hybrid parameter adaptation strategy for differential evolution to solve real-world problems (HADE)

This sub-section is devoted to investigate the influence of a novel hybrid strategy for controling DE parameters. Besides, a modified mutation strategy and a novel population reduction strategy are introduced.

4.2.3.1 The proposed mutation strategy

Several proposals have been introduced to achieve a satisfying balance between exploration and exploitation. For instance, sub-population-based mutation strategies, where the solutions involved in the mutation can only be from their associated sub-population [53, 59]. A different concept is to use one of the p best solutions in the population, in order to decrease the mutation greediness [82, 81, 100]. In the same context, the mutation strategy introduced in this algorithm attempts to achieve this balance. However, according to [74], parameter adaptation strategies should be often associated with a mutation strategy that slightly favors exploitation. Based on this recommendation, the proposed mutation equation can be described as follows:

$$v_i^{G+1} = x_i^G + F.(x_{pbest}^G - x_i^G) + F.(x_{RandBetter}^G - x_{RandWorse}^G) \tag{4.32}$$

where x_{pbest}^G is randomly chosen among a fraction p of the best solutions in the current population. $x_{RandBetter}^G$ and $x_{RandWorse}^G$ are randomly chosen in the current population such that $RandBetter \neq RandWorse \neq i$ and the fitness of $x_{RandBetter}^G$

is better than the one of $x^G_{RandWorse}$. This scenario would keep a reasonable exploration capability because the population is following multiple good solutions. Meanwhile, exploitation is slightly enhanced thanks to the second part of the equation. In fact, the difference between x_{better} and x_{worse} gives a vector that indicates a direction in which x_i is moved to continue the search towards potentially better solutions.

4.2.3.2 The proposed parameter adaptation strategy

First, it should be mentioned that this mechanism is inspired by the strategy used in L-SHADE algorithm. Our modification consists in incorporating a k-nearest neighbors (KNN) algorithm to judge whether a generated combination F/CR can be promising in the next iteration or not. KNN uses a training set of the recently-used F/CR combinations. They are labeled as *promising* or *not promising* according to the following concepts. An F/CR combination is called $successful$ if it could improve its associated individual. In this phase, an F/CR combination is called *promising* if it is $successful$ or if its associated individual is among the best half of the population. Otherwise, this combination is labeled *not promising*. Our proposed adaptation strategy can then be explained in the following steps:

- Step 1: if less than 10% of the budget has been used so far, random values for F and CR are uniformly generated in the range [0.1,1] for each individual. Otherwise, go to step 2.

- Step 2: $successful$ F/CR combinations from the previous iteration are gath-

ered in S_F and S_{CR} respectively, and a weighted Lehmer mean is computed for F and CR as follows:

$$Weighted_{Lehmar}(S_F) = \frac{\sum_{i=1}^{|S_F|}(W_i * F_i)^2}{\sum_{i=1}^{|S_F|}(W_i * F_i)} \tag{4.33}$$

$$Weighted_{Lehmar}(S_{CR}) = \frac{\sum_{i=1}^{|S_F|}(W_i * CR_i)^2}{\sum_{i=1}^{|S_F|}(W_i * CR_i)} \tag{4.34}$$

where S_F and S_{CR} represent the sets of successful F and CR respectively and

$$W_i = \frac{\Delta f_i}{\sum_{l=1}^{|S_F|} \Delta f_l} \tag{4.35}$$

where Δf_i is the fitness difference between the offspring v_i and the parent x_i.

- Step 3: the computed Lehmer means are inserted in M_F and M_{CR}, which are archives for lehmar mean values from the search process history. Their size is fixed to D (problem dimension) values and their D values are first initialized with 0.5. Each new lehmar mean value replaces the oldest value in M_F and M_{CR}.

- Step 4: for each individual k, random values M_{Fr} and M_{CRr} from M_F and M_{CR} are selected.

- Step 5: generate new F_k and CR_k using cauchy distribution as follows:

$$F_k = Cauchy(M_{Fr}, 0.1) \tag{4.36}$$

$$CR_k = Cauchy(M_{CRr}, 0.1) \tag{4.37}$$

It should be stated that F_k and CR_k are updated to 0.1 if negative values occur or to 1 if values more than 1 occur.

- Step 6: the new F_k and CR_k are classified using KNN classifier with k=50. If they are not promising, return to Step 4.

4.2.3.3 The parabolic reduction scheme

It has been proven in [62] that the linear reduction of the population size [82] implies a quick reduction of the population size at the beginning of the search process. As a result, a bad exposition of the landscape problem may occur. The parabolic reduction scheme has been proposed in [62] as an attempt to overcome this issue. The scheme can be described as follows:

$$PS_{G+1} = round[\frac{PS_{min} - PS_{max}}{nfe_{max} - PS_{max}}.(nfe - PS_{max})^2 + PS_{max}] \tag{4.38}$$

where PS_{G+1} is the new population size, PS_{min} is the minimum population size, PS_{max} is the maximum population size, nfe is the number of function evaluations already used, nfe_{max} is the total budget.

4.2.3.4 The algorithmic combination

This sub-section describes the whole proposal. Firstly, the parameter adaptation strategy is performed at the beginning of each iteration in the main loop of DE. For the first 10% of the budget, step 1 is applied here as an exploration phase,

where random F/CR combinations are produced. For the remaining budget, steps 2 to 6 are performed: our purpose is to produce *promising* F/CR values based on the $successful$ F/CR values from the last iteration, and with the help of a KNN classifier whose training set contains recent past F/CR values. It should be mentioned that the size of the training set is fixed to PS_{max} combinations, and it contains the most recently-used F/CR values. Afterwards, one iteration of DE is performed (i.e. the proposed mutation equation along with the binomial crossover). Thirdly, the new F/CR values (i.e. those produced by the parameter adaptation strategy at the beginning of the iteration) are evaluated and labeled as *promising* or *not promising*. Fourthly, KNN training set is updated by replacing the oldest F/CR combinations with the current set of new ones. Finally, the parabolic population size reduction scheme is applied in order to gradually enhance the exploitation capability of the approach. The whole proposal is depicted in Algorithm 16.

4.2.3.5 Experimental results

This section demonstrates the results of our proposition on second version of the problem at hand. The algorithm is compared with several recent self-adaptive DE variants such as EPSDE [90], JADE [100], SADE [11], L-SHADE [82], Differential Crossover Strategy based on covariance matrix learning with euclidean neighborhood for solving real-world problems (L-ConvSHADE) [3] and our hybrid differential evolution algorithm for real-world problems (HDE). In order to validate the results, 22 real-world problems of the CEC 2011 test suite have been optimized

Algorithm 16 The proposed algorithm

```
Output: Best solution
Generate population pop of PS_max individuals and popold ← pop
while Budget is not consumed do
    Apply the proposed parameter adaptation
    for Each individual k in popold do
        Apply the mutation strategy according to (4.32)
        Apply binomial crossover
        if The offspring is better than the parent then
            Replace the parent in pop with the offspring
        end if
    end for
    for Each individual k in pop do
        if the combination F_k/CR_k was successful then
            S_CR ← [S_CR, CR_k], S_F ← [S_F, F_k]
            Label F_k/CR_k combination as promising
        else
            if The individual is among the best half of the population then
                Label F_k/CR_k combination as promising
            else
                Label F_k/CR_k combination as not promising
            end if
        end if
    end for
    Update KNN training set using the new set of F/CR combinations
    Apply the parabolic reduction using (4.38)
    popold ← pop
end while
```

as well. There are nine parameters in our proposal, which are the maximum population size, the minimum population size, the archive size of M_{CR} and M_F, the initial value of M_{CR} and M_F, standard deviation of the used cauchy distribution, number of evaluation for the training phase, the number of nearest neighbors of KNN algorithm, the size of training set and the fraction of the best solutions to be used in the mutation equation. Further details about the proposal parameters can be found in Table 4.26. The parameters of the other proposals have been kept

Table 4.26: HADE parameters

Parameter	Parameter value
Maximum population size PS_{max}	D*10
Minimum population size PS_{min}	10
Archive size of M_{CR} and M_F	D
The initial value of M_{CR} and M_F	0.5
Standard deviation of the cauchy distribution	0.1
Number of evaluation for the training phase	10% of the budget
The parameter k of KNN	50
Size of training set	D*10
p	0.1

during the configuration as it can be seen in Table 4.27.

4.2.3.6 CEC 2011 test suite

The obtained results by all the proposals are given in Table 4.28. The rows show the mean value, the best value and standard deviation of 25 runs for each problem. These results have been validated by a Kruskal-Wallis statistical test at 95% confidence level followed by a Tukey-Kramer post hoc test. According to the statistical

Table 4.27: The parameters of the DE variants

Algorithm	Parameters
EPSDE	Parameters taken from [90]
SADE	Parameters taken from [11]
JADE	Parameters taken from [100]
SHADE	Parameters taken from [80]
L-SHADE	Parameters taken from [82]
L-ConvSHADE	Parameters taken from [3]

tests, for each function, the mean values of the best algorithms are in bold font in Table 4.28.

The results in Table 4.29 show that our proposal shows a better performance compared to the other adaptive DE variants. It can significantly outperform EPSDE in 8 functions, L-ConvSHADE in 9 functions, L-SHADE in 10 functions, SADE in 13 functions, JADE in 11 functions and SHADE in 12 functions.

4.2.3.7 Comparison on the problem at hand

Our proposal and the other algorithms have been performed 30 times. The best, mean, worst and the standard deviation of each algorithm are collected. It can be stated from Table 4.30 and Table 4.31 that the proposed algorithm achieves the best solution known so far compared to the other algorithms. Besides, we can notice the stable performance (std very small), which reveals the resilience of our proposal.

Table 4.28: Comparison of HADE with state-of-the-art algorithms on the CEC 2011 test suite

		EPSDE	SADE	JADE	SHADE	L-SHADE	L-ConvSHADE	The proposal
F1	Mean	9.10E+00	2.49E+00	2.26E-01	2.73E-02	3.24E-08	1.06E-06	**0.00E+00**
	Best	5.45E+00	6.31E-01	2.57E-10	1.85E+00	**0.00E+00**	**0.00E+00**	**0.00E+00**
F2	Mean	-2.02E+01	-1.70E+01	-2.34E+01	-2.30E+01	**-2.61E+01**	**-2.61E+01**	**-2.58E+01**
	Best	-2.19E+01	-1.95E+01	-2.45E+01	-2.44E+01	-2.82E+01	-2.68E+01	-2.69E+01
F3	Mean	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
	Best	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
F4	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Best	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F5	Mean	-3.22E+01	-3.22E+01	-3.56E+01	**-3.63E+01**	-3.63E+01	-3.61E+01	**-3.64E+01**
	Best	-3.60E+01	-3.32E+01	-3.64E+01	-3.66E+01	**-3.68E+01**	**-3.68E+01**	**-3.68E+01**
F6	Mean	-2.01E+01	-2.41E+01	**-2.90E+01**	**-2.90E+01**	**-2.91+01**	**-2.91+01**	**-2.86E+01**
	Best	-2.88E+01	-2.63E+01	**-2.92E+01**	-2.91E+01	-2.91E+01	-2.91E+01	-2.91E+01
F7	Mean	1.30E+00	1.37E+00	1.17E+00	1.12E+00	1.21E+00	1.12E+00	**8.05E-01**
	Best	1.12E+00	1.24E+00	9.10E-01	9.06E-01	9.63E-01	1.01E+00	7.2E-01
F8	Mean	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
	Best	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
F9	Mean	9.28E+04	**1.73E+03**	**2.40E+03**	**2.22E+03**	8.16E+03	2.76E+04	5.87E+03
	Best	3.98E+04	**7.69E+02**	1.13E+03	1.14E+03	1.12E+03	3.76E+03	1.82E+03
F10	Mean	-1.74E+01	**-2.16E+01**	**-2.14E+01**	**-2.16E+01**	**-2.15E+01**	**-2.15E+01**	**-2.15E+01**
	Best	-2.02E+01	**-2.18E+01**	**-2.18E+01**	**-2.18E+01**	-2.16E+01	**-2.18E+01**	**-2.18E+01**
F11	Mean	5.86E+04	5.21E+04	5.24E+04	5.22E+04	5.20E+04	**5.18E+04**	**5.18E+04**
	Best	5.21E+04	5.12E+04	5.15E+04	5.15E+04	5.11E+04	**5.08E+04**	5.10E+04
F12	Mean	1.09E+06	1.09E+06	**1.07E+06**	1.10E+06	**1.07E+06**	**1.07E+06**	**1.07E+06**
	Best	1.07E+06	1.07E+06	1.07E+06	1.07E+06	**1.07E+06**	**1.06E+06**	1.07E+06
F13	Mean	1.54E+04	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
	Best	1.54E+04	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
F14	Mean	**1.83E+04**	**1.81E+04**	**1.83E+04**	**1.81E+04**	**1.80E+04**	**1.80E+04**	**1.81E+04**
	Best	1.83E+04	1.81E+04	**1.80E+04**	**1.80E+04**	**1.80E+04**	**1.80E+04**	**1.80E+04**
F15	Mean	3.30E+04	3.28E+04	3.29E+04	3.27E+04	3.27E+04	3.27E+04	**3.26E+04**
	Best	3.29E+04	3.28E+04	3.27E+04	3.27E+04	3.27E+04	3.27E+04	**3.26E+04**
F16	Mean	1.42E+05	1.28E+05	1.33E+05	1.29E+05	1.23E+05	1.23E+05	**1.23E+05**
	Best	1.31E+05	1.26E+05	1.26E+05	**1.22E+05**	**1.22E+05**	1.23E+05	1.23E+05
F17	Mean	2.06E+06	1.90E+06	1.91E+06	1.91E+06	1.85E+06	1.84E+06	**1.83E+06**
	Best	1.93E+06	1.87E+06	1.87E+06	1.88E+08	1.83E+06	1.82E+06	**1.80E+06**
F18	Mean	6.06E+06	9.38E+05	9.39E+05	9.40E+05	9.33E+05	9.33E+05	**9.32E+05**
	Best	3.24E+06	9.33E+05	9.35E+05	9.37E+05	**9.29E+05**	9.31E+05	9.30E+05
F19	Mean	7.16E+06	9.46E+05	9.92E+05	9.52E+05	9.40E+05	9.39E+05	**9.39E+05**
	Best	4.47E+06	9.41E+05	9.39E+05	9.39E+05	9.38E+05	**9.37E+05**	9.38E+05
F20	Mean	6.21E+06	9.37E+05	9.40E+05	9.40E+05	9.32E+05	9.31E+05	**9.30E+05**
	Best	4.16E+06	9.35E+05	9.36E+05	9.34E+05	9.30E+05	**9.29E+05**	**9.29E+05**
F21	Mean	1.97E+01	1.97E+01	1.72E+01	1.75E+01	**1.54E+01**	**1.52E+01**	**1.60E+01**
	Best	1.66E+01	1.66E+01	1.31E+01	1.41E+01	1.44E+01	1.17E+01	1.30E+01
F22	Mean	2.18E+01	2.18E+01	1.66E+01	1.99E+01	**1.12E+01**	**1.13E+01**	1.44E+01
	Best	1.68E+01	1.68E+01	1.19E+01	1.31E+01	**8.60E+00**	**8.60E+00**	1.05E+01

Table 4.29: The aggregate results of HADE using the statistical test

VS	Our proposition	22 functions of CEC 2011 test suite
EPSDE	+(better)	0
	-(worse)	8
	=(no sign)	14
SADE	+(better)	1
	-(worse)	13
	=(no sign)	8
JADE	+(better)	1
	-(worse)	11
	=(no sign)	10
SHADE	+(better)	1
	-(worse)	12
	=(no sign)	9
L-SHADE	+(better)	1
	-(worse)	10
	=(no sign)	11
L-ConvSHADE	+(better)	1
	-(worse)	9
	=(no sign)	12

Table 4.30: Comparison of HADE with state-of-the-art algorithms on the first version of the problem at hand

	Best	Mean	Worst	Std
SADE	-3.271E+03	-2.920E+03	-2.780E+03	81.49
JADE	-3.194E+03	-2.840E+03	-2.420E+03	96.23
EPSDE	-3.200E+03	-2.850E+03	-2.430E+03	102.65
SHADE	-3.097E+03	-2.944E+03	-2.697E+03	44.28
L-SHADE	-3.197e+03	-3.044e+03	-2.797e+03	107
L-ConvSHADE	-3.347e+03	-3.216e+03	-3.158e+03	74.13
HADE	**-3.397E+03**	**-3.397E+03**	**-3.397E+03**	0

Table 4.31: Comparison of HADE with state-of-the-art algorithms on the second version of the problem at hand

	Best	Mean	Worst	Std
EPSDE	-4.2842E+02	-4.1968E+02	-4.0693E+02	5.1628E+00
SADE	**-4.3065E+02**	-4.2270E+02	-4.0869E+02	4.7628E+00
JADE	**-4.3065E+02**	-4.2527E+02	-4.1701E+02	1.2785E+00
SHADE	**-4.3065E+02**	-4.2370E+02	-4.0569E+02	6.9628E+00
L-SHADE	**-4.3065E+02**	-4.3043E+02	-4.301E+02	2.7463E-01
L-ConvSHADE	**-4.3065E+02**	-4.2970E+02	-4.270E+02	1.2745E+00
Our proposition	**-4.3065E+02**	**-4.3065E+02**	**-4.3065E+02**	1.0387E-05

4.3 Overall Comparison of the algorithms

In this section, the proposed algorithmic components are shown in Table 4.32 to summarize the metaheuristics and the contribution of each proposal.

An overall comparison of the introduced proposals is presented. Since the context of this thesis is to optimize real-world applications, it is better to present a comprehensive comparison using the CEC 2011 test suite. All the proposed algorithms are concerned in the comparison, where finally, a statistical test is performed in order to prove their advantage. As it was used in the experimentation phase of each proposal, the comparative table 4.33 is introduced to present the results of each algorithm on each problem. The mean results of each algorithm are in bold when it can significantly outperform the other algorithms.

Finally, a Friedman test is performed, where the result of each proposal is computed. Table 4.34 summarizes the score of each algorithm proposed in this

Table 4.32: Metaheuristics and the contribution of each algorithm

	HOA	HDE	DADE	PEADE	HADE
Metaheuristics	DE, CMEAS and CS	DE	DE	DE	DE
Mutations	-	-	-	DE/current-to-centroid/1	-
Crossover	-	-	-	A framework to switch between normal binomial crossover and eigevector-based crossover	-
Selection	-	Multi-criteria selection	-	-	-
Other search operators	a global search procedure based on clustering and lévy flight	-	-	-	-
Parameter adaptation strategies	-	Pheromone matrix-based strategy	Hybrid strategy for F and a reinforcement learning strategy for CR	Modified pheromone matrix-based strategy	Hybrid strategy between the technique of LSHADE and KNN

Table 4.33: Comparison of our proposals on the CEC 2011 test suite

		HOA	PEADE	HDE	DADE	HADE
F1	Mean	1.62E+00	2.87E-18	2.72E-23	**0.00E+00**	**0.00E+00**
	Best	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F2	Mean	-2.39E+01	-2.29+01	**-2.69E+01**	-2.31E+01	-2.58E+01
	Best	-2.69E+01	-2.73E+01	**-2.84E+01**	-2.78E+01	-2.69E+01
F3	Mean	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
	Best	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**	**1.15E-05**
F4	Mean	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Best	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F5	Mean	-3.49E+01	-3.19E+01	-3.53E+01	-3.36E+01	**-3.64E+01**
	Best	-3.68E+01	-3.68E+01	-3.69E+01	**-3.70E+01**	-3.68E+01
F6	Mean	-2.66E+01	-2.24E+01	-2.76E+01	-2.76E+01	**-2.86E+01**
	Best	-2.91E+01	-2.74E+01	**-2.92E+01**	-2.91E+01	-2.91E+01
F7	Mean	**8.80E-01**S	**9.91E-1**	**7.76E-01**	**1.04E-01**	**8.05E-01**
	Best	**5.00E-01**	7.29E-01	5.51E-01	6.30E-01	7.2E-01
F8	Mean	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
	Best	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**	**2.20E+02**
F9	Mean	3.37E+03	**1.78E+03**	2.32E+03	8.22E+04	5.87E+03
	Best	2.15E+03	**1.04E+03**	1.15E+03	2.82E+04	1.82E+03
F10	Mean	-1.69E+01	**-2.15E+01**	**-2.13E+01**	**-2.16E+01**	**-2.15E+01**
	Best	-2.14E+01	**-2.18E+01**	**-2.18E+01**	**-2.18E+01**	**-2.18E+01**
F11	Mean	5.32E+04	**5.08E+04**	5.22E+04	5.25E+04	5.18E+04
	Best	5.16E+04	**4.99E+04**	5.12E+04	5.17E+04	5.10E+04
F12	Mean	**1.07E+06**	**1.07E+06**	**1.07E+06**	1.22E+06	**1.07E+06**
	Best	**1.07E+06**	**1.07E+06**	**1.07E+06**	1.08E+06	**1.07E+06**
F13	Mean	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
	Best	**11.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**	**1.54E+04**
F14	Mean	1.83E+04	**1.81E+04**	**1.81E+04**	**1.81E+04**	**1.81E+04**
	Best	**1.80E+04**	1.81E+04	**1.80E+04**	**1.80E+04**	**1.80E+04**
F15	Mean	3.27E+04	3.27E+04	3.28E+04	3.27E+04	**3.26E+04**
	Best	3.26E+04	3.27E+04	3.27E+04	3.27E+04	**3.26E+04**
F16	Mean	1.26E+05	**1.23E+05**	1.28E+05	1.25E+05	**1.23E+05**
	Best	1.24E+05	1.23E+05	1.26E+05	**1.26E+05**	**1.23E+05**
F17	Mean	1.93E+06	1.89E+06	1.91E+06	1.89E+06	**1.83E+06**
	Best	1.91E+06	1.87E+06	1.82E+06	1.87E+06	**1.80E+06**
F18	Mean	9.38E+05	9.37+05	9.42E+05	9.41E+05	**9.32E+05**
	Best	9.34E+05	**9.29E+05**	9.36E+05	9.38E+05	9.30E+05
F19	Mean	9.42E+05	**9.39E+05**	1.03E+06	9.45+05	**9.39E+05**
	Best	9.39E+05	**9.31E+05**	9.43E+05	9.38+05	9.38E+05
F20	Mean	9.37E+05	**9.31E+05**	9.42E+05	9.40E+05	**9.30E+05**
	Best	9.33E+05	**9.28E+05**	9.37E+05	9.37E+05	9.29E+05
F21	Mean	1.35E+01	**9.45E+00**	1.14E+01	1.38E+01	1.60E+01
	Best	1.15E+01	1.49E+01	9.53E+00	**8.66E+00**	1.30E+01
F22	Mean	1.36E+01	1.83E+01	**1.04E+01**	1.88E+01	1.44E+01
	Best	9.26E+00	1.64E+01	**8.74E+00**	1.50E+01	1.05E+01

thesis. After performing the Friedman test, a critical value CV=16.35 is obtained. According to Table 4.34, it can be noticed that HOA is significantly outperformed

Table 4.34: Pairwise score of our proposals

VS	PEADE	HDE	DADE	HADE
HOA	13.50	9.00	3.00	25.50
PEADE	-	4.50	16.50	12.00
HDE	-	-	12.00	16.50
DADE	-	-	-	28.50

by only HADE. PEADE and HDE are outperformed by DADE and HADE respectively. Finally, HADE outperforms DADE, which is the last algorithm proposed in this thesis. These results reveal that HADE has not been outperformed by the other algorithms, which can reveal the advantage of this algorithm.

4.4 Conclusion

In this chapter, several hybrid and self-adpative algorithms have been introduced. It can be stated that our proposals mainly rely on DE algorithm, which is shown to be successful when an appropriate adjustment is applied on its canonical structure. Furthermore, several strategies have been considered in order to balance between exploration and exploitation phases. These strategies can be represented as:

- Integrating intelligent choice techniques between search operators, as it can be stated in HOA and HDE.

- Integrating new algorithms and/or new search operators such as HOA and PEADE, where a new search operator called "eigenvector-based crossover" has been introduced to enhance the exploitation phase of the proposal.

- Controling the parameters of DE algorithm: indeed, several strategies have proposed in this thesis to adapt DE parameters. The proposed strategies rely on efficient machine learning techniques, as it can be noticed in PEADE, DADE and HADE.

Our effort has focused on optimizing real-world problems, where recent topologies of a given electric motor have been optimized. Furthermore, our proposals performance have been validated on the CEC 2011 test suite, which represents the most recent benchmark that contains real-world applications. It can be noticed from the experimental results that the introduced algorithms have been compared with several recent state-of-the-art algorithms revealing satisfying results.

Chapter 5

Conclusion and perspectives

Nowadays, numerous real-world problems are considered as optimization problems due to the existence of one or many objective functions to be optimized. These problems represent an actual challenge to propose new efficient algorithms providing high quality solutions. In this context, metaheuristics have become well-known optimization algorithms thanks to their relatively simple structure and their low computational time. Indeed, they have shown to be a promising alternative to common mathematical methods.

Furthermore, metaheuristics have been recently improved by integrating several considerations, such as the combination of algorithms, parameter tuning/adaptation, proposing new search operators etc. This thesis is mainly focused on proposing new optimization algorithms by considering the aforementioned issues. Besides, it can be sometimes noticed that a given proposal is computationally time-consuming. In order to overcome this issue, a parallel version of the concerned algorithm is

implemented when a serious computational burden is stated.

The first part of this thesis has covered on the one hand several classic algorithms and hybridization designs between metaheuristics. Besides, several self-adaptive DE and CS proposals have been addressed to investigate the influence of the parameters on their results. On the other hand, several GPU-based parallel algorithms have been covered to show the advantage of parallelization on reducing the computational time of optimization algorithms.

The effort is continued in the second part, where recent topologies of an electric motor have been addressed. In this chapter, the problem at hand has been modeled as an optimization problem and a corresponding objective function has been defined. It should be stated that the two topologies are distinguished by different constraints. Moreover, the CEC 2011 test suite has been used to validate the results of our algorithms.

The third part represents the major chapter of this thesis. Following the considerations mentioned above, we have proposed five optimization algorithms. The contribution of these algorithms can be stated as follows:

- Efficient hybridization designs between metaheuristics and/or search operators.
- The adoption of recent search operators within proposed frameworks in order to provide better results.
- The proposition of novel parameter adaptation strategies.
- The optimization of a recent engineering problem.

Our proposals have been described in details, and a comprehensive comparison has been conducted with recent state-of-the-art optimization algorithms. Our proposals are mainly designed to optimize real-world applications, where the problem at hand is optimized as well as the well-known CEC 2011 test suite. Finally, an overall comparison between the proposals has been performed.

This thesis has covered several improved optimization algorithms to solve a diverse set of real-world optimization problems. However, several research directions can be considered in the near future. Our perspectives can be organized as follows:

- Automatic generation of optimization algorithms: it can be noticed that metaheuristics rely on a static structure. Indeed, predefined search operators are set and then applied for a given problem. Our aim is to exploit several techniques, such as machine learning and genetic programming methods to provide dynamic algorithms. In fact, dynamic optimization algorithms would provide flexible search operators thanks to their potential capability to exploit useful information from the problem at hand. This information would help afterwards in generating appropriate search operators for the problem at hand.

- Machine learning-based parameter adaptation strategies: in fact, we have focused on proposing several adaptation strategies for DE parameters. It can be noticed that the proposed strategies exploit several simple machine learning techniques, which could improve to the final results. This issue has motivated us to boost our effort towards this direction, where pure machine

learning-based techniques can be proposed. Besides, our aim is to investigate these strategies on other metaheuristics such as PSO and CS, where few studies have proven the critical influence of their parameters on the final performance.

- Solving other real-world problems: real-world applications have emerged in numerous fields, such as mechanical engineering, transportation, chemistry, biology, security etc. Our aim is to test our proposals in order to investigate their scalability and resilience when different problems are handled. Moreover, parallel versions of the proposals could be proposed using graphics processing units (GPU) where a serious computational time is noticed.

Bibliography

[1] Rawaa Dawoud Al-Dabbagh, Ferrante Neri, Norisma Idris, and Mohd Sapiyan Baba. Algorithmic design issues in adaptive differential evolution schemes: review and taxonomy. *Swarm and Evolutionary Computation*, 2018.

[2] Assif Assad and Kusum Deep. A hybrid harmony search and simulated annealing algorithm for continuous optimization. *Information Sciences*, 450:246–266, 2018.

[3] N. H. Awad, M. Z. Ali, P. N. Suganthan, R. G. Reynolds, and A. M. Shatnawi. A novel differential crossover strategy based on covariance matrix learning with euclidean neighborhood for solving real-world problems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 380–386, June 2017.

[4] Noor H Awad, Mostafa Z Ali, Ponnuthurai N Suganthan, and Robert G Reynolds. An ensemble sinusoidal parameter adaptation incorporated with

l-shade for solving cec2014 benchmark problems. In *CEC*, pages 2958–2965, 2016.

[5] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

[6] Christian Blum, Jakob Puchinger, Günther R Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[7] Christian Blum, Andrea Roli, and Michael Sampels. *Hybrid metaheuristics: an emerging approach to optimization*, volume 114. Springer, 2008.

[8] Stefan Boettcher and Allon G Percus. Extremal optimization: an evolutionary local-search algorithm. In *Computational Modeling and Problem Solving in the Networked World*, pages 61–77. Springer, 2003.

[9] Ilhem BoussaïD, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information sciences*, 237:82–117, 2013.

[10] Janez Brest, Sao Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE transactions on evolutionary computation*, 10(6):646–657, 2006.

[11] Janez Brest, Viljem Zumer, and Mirjam Sepesy Maucec. Self-adaptive differential evolution algorithm in constrained real-parameter optimization. In

Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, pages 215–222. IEEE, 2006.

[12] Mathieu Brévilliers, Omar Abdelkafi, Julien Lepagnot, and Lhassane Idoumghar. Fast hybrid BSA-DE-SA algorithm on GPU. In *Swarm Intelligence Based Optimization: Second International Conference, ICSIBO 2016, Mulhouse, France, June 13-14, 2016, Revised Selected Papers*, volume 10103, page 75. Springer, 2017.

[13] Min-Yuan Cheng and Doddy Prayogo. Symbiotic organisms search: a new metaheuristic optimization algorithm. *Computers & Structures*, 139:98–112, 2014.

[14] Pinar Civicioglu. Backtracking search optimization algorithm for numerical optimization problems. *Applied Mathematics and Computation*, 219(15):8121 – 8144, 2013.

[15] Carlos Cotta, Marc Sevaux, and Kenneth Sörensen. *Adaptive and multilevel metaheuristics*, volume 136. Springer, 2008.

[16] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, 45(3):35, 2013.

[17] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N Suganthan. Recent advances in differential evolution–an updated survey. *Swarm and Evolutionary Computation*, 27:1–30, 2016.

[18] Swagatam Das and Ponnuthurai N Suganthan. Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems. Technical report, Jadavpur University, Nanyang Technological University, Kolkata, 2010.

[19] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2010.

[20] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2011.

[21] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: a survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2011.

[22] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[23] M. Essaid, L. Idoumghar, J. Lepagnot, M. Brvilliers, and D. Fodorean. A hybrid differential evolution algorithm for real world problems. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7, July 2018.

[24] Mokhtar Essaid, Lhassane Idoumghar, Julien Lepagnot, Mathieu Brévilliers, and Daniel Foderean. A hybrid differential evolution algorithm for real world

problems. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7. IEEE, 2018.

[25] Behnam Farnad, Ahmad Jafarian, and Dumitru Baleanu. A new hybrid algorithm for continuous optimization problem. *Applied Mathematical Modelling*, 55:652–673, 2018.

[26] Daniel Fodorean, Lhassane Idoumghar, Mathieu Brévilliers, Paul Minciunescu, and Cristi Irimia. Hybrid differential evolution algorithm employed for the optimum design of a High-Speed PMSM used for EV Propulsion. *IEEE Transactions on Industrial Electronics*, 64(12):9824–9833, 2017.

[27] Wayne Franz and Parimala Thulasiraman. A dynamic cooperative hybrid MPSO + GA on hybrid CPU+ GPU fused multicore. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–8. IEEE, 2016.

[28] Weifeng Gao, Gary G Yen, and Sanyang Liu. A cluster-based differential evolution with self-adaptive strategy for multimodal optimization. *IEEE transactions on cybernetics*, 44(8):1314–1327, 2013.

[29] Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. A new heuristic optimization algorithm: harmony search. *simulation*, 76(2):60–68, 2001.

[30] Stefan Giurgea, Daniel Fodorean, Giansalvo Cirrincione, Abdellatif Miraoui, and Maurizio Cirrincione. Multimodel optimization based on the response

surface of the reduced FEM simulation model with application to a pmsm. *IEEE Transactions on Magnetics*, 44(9):2153–2157, 2008.

[31] Fred Glover. Tabu search methods in artificial intelligence and operations research. *ORSA Artificial Intelligence*, 1(2):6, 1987.

[32] David E Goldberg. Genetic algorithms in search. *Optimization, and MachineLearning*, 1989.

[33] Shu-Mei Guo, Jason Sheng-Hong Tsai, Chin-Chang Yang, and Pang-Han Hsu. A self-optimization approach for l-shade incorporated with eigenvector-based crossover and successful-parent-selecting framework on cec 2015 benchmark set. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1003–1010. IEEE, 2015.

[34] Shu-Mei Guo and Chin-Chang Yang. Enhancing differential evolution utilizing eigenvector-based crossover operator. *IEEE Transactions on Evolutionary Computation*, 19(1):31–49, 2015.

[35] Udit Halder, Swagatam Das, and Dipankar Maity. A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments. *IEEE transactions on cybernetics*, 43(3):881–897, 2013.

[36] Zhong-Hua Han and Ke-Shi Zhang. Surrogate-based optimization. In *Real-world applications of genetic algorithms*. IntechOpen, 2012.

[37] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

[38] F Herrera, M Lozano, and D Molina. Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. 2010.

[39] Md Maruf Hussain, Hiroshi Hattori, and Noriyuki Fujimoto. A CUDA implementation of the standard particle swarm optimization. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2016 18th International Symposium on*, pages 219–226. IEEE, 2016.

[40] Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

[41] Raka Jovanovic, Milan Tuba, and Ivona Brajevic. Parallelization of the cuckoo search using cuda architecture. In *Proc. 7th Int. Conf. Appl. Math. Simulat. Model.(ASM)*, pages 137–142, 2013.

[42] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.

[43] J Kennedy and R Eberhart. Particle swarm optimization (pso). In *Proc. IEEE International Conference on Neural Networks, Perth, Australia*, pages 1942–1948, 1995.

[44] Mustafa Servet KıRan and Mesut GüNdüZ. A recombination-based hybridization of particle swarm optimization and artificial bee colony algorithm for continuous optimization problems. *Applied Soft Computing*, 13(4):2188–2203, 2013.

[45] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[46] Miguel Lastra, Daniel Molina, and José M Benítez. A high performance memetic algorithm for extremely high-dimensional problems. *Information Sciences*, 293:35–58, 2015.

[47] Abdesslem Layeb. A novel quantum inspired cuckoo search for knapsack problems. *International Journal of bio-inspired Computation*, 3(5):297–305, 2011.

[48] Jianming Li, Wei Wang, and Xiangpei Hu. Parallel particle swarm optimization algorithm based on CUDA in the AWS cloud. In *Frontier of Computer Science and Technology (FCST), 2015 Ninth International Conference on*, pages 8–12. IEEE, 2015.

[49] Xiangtao Li and Minghao Yin. Modified cuckoo search algorithm with self adaptive parameter method. *Information Sciences*, 298:80–97, 2015.

[50] Xueping Li, Liangxing Fang, Zhigang Lu, Jiangfeng Zhang, and Hao Zhao. A line flow granular computing approach for economic dispatch with line constraints. *IEEE Transactions on Power Systems*, 32(6):4832–4842, 2017.

[51] Huijun Liang, Yungang Liu, Yanjun Shen, Fengzhong Li, and Yongchao Man. A hybrid bat algorithm for economic dispatch with random wind power. *IEEE Transactions on Power Systems*, 33(5):5052–5061, 2018.

[52] JJ Liang, BY Qu, PN Suganthan, and Q Chen. Problem definitions and evaluation criteria for the cec 2015 competition on learning-based real-parameter single objective optimization. Technical report, 2014.

[53] Jingliang Liao, Yiqiao Cai, Yonghong Chen, Tian Wang, and Hui Tian. Improving differential evolution with ring topology-based mutation operators. In *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 103–109. IEEE, 2014.

[54] Qi Liu, Lei Wu, Wensheng Xiao, Fengde Wang, and Linchuan Zhang. A novel hybrid bat algorithm for solving continuous optimization problems. *Applied Soft Computing*, 73:67–82, 2018.

[55] Pedro Lopez-Garcia, Enrique Onieva, Eneko Osaba, Antonio D Masegosa, and Asier Perallos. Gace: A meta-heuristic based in the hybridization of genetic algorithms and cross entropy methods for continuous optimization. *Expert Systems with Applications*, 55:508–519, 2016.

[56] Guo-Heng Luo, Sheng-Kai Huang, Yue-Shan Chang, and Shyan-Ming Yuan. A parallel bees algorithm implementation on GPU. *Journal of Systems Architecture*, 60(3):271–279, 2014.

[57] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[58] Rammohan Mallipeddi. Harmony search based parameter ensemble adaptation for differential evolution. *Journal of Applied Mathematics*, 2013, 2013.

[59] Rammohan Mallipeddi, Ponnuthurai N Suganthan, Quan-Ke Pan, and Mehmet Fatih Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied soft computing*, 11(2):1679–1696, 2011.

[60] Rammohan Mallipeddi, Ponnuthurai N Suganthan, Quan-Ke Pan, and Mehmet Fatih Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied soft computing*, 11(2):1679–1696, 2011.

[61] MK Marichelvam, Thirumoorthy Prabaharan, and Xin-She Yang. Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Applied Soft Computing*, 19:93–101, 2014.

[62] Zhenyu Meng, Jeng-Shyang Pan, and Kuo-Kun Tseng. Pade: An enhanced differential evolution algorithm with novel control parameter adaptation schemes for numerical optimization. *Knowledge-Based Systems*, 2019.

[63] A. W. Mohamed, A. A. Hadi, A. M. Fattouh, and K. M. Jambi. Lshade with semi-parameter adaptation hybrid with cma-es for solving cec 2017 benchmark problems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 145–152, June 2017.

[64] Daniel Molina and Francisco Herrera. Iterative hybridization of de with local search for the cec'2015 special session on large scale global optimization. In *2015 IEEE congress on evolutionary computation (CEC)*, pages 1974–1978. IEEE, 2015.

[65] Xinxin Ouyang, Yongquan Zhou, Qifang Luo, and Huan Chen. A novel discrete cuckoo search algorithm for spherical traveling salesman problem. *Applied mathematics & information sciences*, 7(2):777, 2013.

[66] Quan-Ke Pan, Ling Wang, Kun Mao, Jin-Hui Zhao, and Min Zhang. An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process. *IEEE Transactions on Automation Science and Engineering*, 10(2):307–322, 2013.

[67] Ihsan Pence, Melike Siseci Cesmeli, Fatih Ahmet Senel, and Bayram Cetisli. A new unconstrained global optimization method based on clustering and parabolic approximation. *Expert Systems with Applications*, 55:493–507, 2016.

[68] Duc Truong Pham and Michele Castellani. The bees algorithm: Modelling foraging behaviour to solve continuous optimization problems. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 223(12):2919–2938, 2009.

[69] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[70] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417, 2009.

[71] A Kai Qin and Ponnuthurai N Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1785–1791. IEEE, 2005.

[72] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

[73] Sana Saeed, Hong Choon Ong, and Saratha Sathasivam. Self-adaptive single objective hybrid algorithm for unconstrained and constrained test functions: An application of optimization algorithm. *Arabian Journal for Science and Engineering*, pages 1–17, 2018.

[74] Carlos Segura, Carlos A Coello Coello, Eduardo Segredo, and Coromoto León. On the adaptation of the mutation scale factor in differential evolution. *Optimization Letters*, 9(1):189–198, 2015.

[75] Rashmi Sharan Sinha, Satvir Singh, Sarabjeet Singh, and Vijay Kumar Banga. Speedup genetic algorithm using C-CUDA. In *2015 Fifth International Conference on Communication Systems and Network Technologies*, pages 1355–1359, April 2015.

[76] Francisco J. Solis and Roger J.-B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[77] Rainer Storn. On the usage of differential evolution for function optimization. In *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American*, pages 519–523. IEEE, 1996.

[78] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[79] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009, 624 pages.

[80] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 71–78. IEEE, 2013.

[81] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 71–78. IEEE, 2013.

[82] Ryoji Tanabe and Alex S Fukunaga. Improving the search performance of shade using linear population size reduction. In *2014 IEEE congress on evolutionary computation (CEC)*, pages 1658–1665. IEEE, 2014.

[83] Ke Tang, XD Li, PN Suganthan, ZY Yang, and Thomas Weise. Benchmark functions for the CEC 2010 special session and competition on large-scale global optimization (2010). *Nature Inspired Computation and Applications Laboratory, USTC, China*.

[84] Ke Tang, Xın Yáo, Ponnuthurai Nagaratnam Suganthan, Cara MacNish, Ying-Ping Chen, Chih-Ming Chen, and Zhenyu Yang. Benchmark functions for the CEC 2008 special session and competition on large scale global optimization. *Nature Inspired Computation and Applications Laboratory, USTC, China*, pages 153–177, 2007.

[85] TO Ting, Xin-She Yang, Shi Cheng, and Kaizhu Huang. Hybrid metaheuristic algorithms: past, present, and future. In *Recent advances in swarm intelligence and evolutionary computation*, pages 71–83. Springer, 2015.

[86] Hamid R Tizhoosh. Opposition-based learning: A new scheme for machine intelligence. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference*

on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), volume 1, pages 695–701, Nov 2005.

[87] Milan Tuba, Milos Subotic, and Nadezda Stanarevic. Modified cuckoo search algorithm for unconstrained optimization problems. In *Proceedings of the 5th European conference on European computing conference*, pages 263–268. World Scientific and Engineering Academy and Society (WSEAS), 2011.

[88] Roberto Ugolotti, Youssef SG Nashed, Pablo Mesejo, ŠPela Ivekovič, Luca Mussi, and Stefano Cagnoni. Particle swarm optimization and differential evolution for model-based object detection. *Applied Soft Computing*, 13(6):3092–3105, 2013.

[89] Krzysztof Walczak. Hybrid differential evolution with covariance matrix adaptation for digital filter design. In *2011 IEEE Symposium on Differential Evolution (SDE)*, pages 1–7. IEEE, 2011.

[90] Yong Wang, Zixing Cai, and Qingfu Zhang. Differential evolution with composite trial vector generation strategies and control parameters. *IEEE Transactions on Evolutionary Computation*, 15(1):55–66, 2011.

[91] Yong Wang, Han-Xiong Li, Tingwen Huang, and Long Li. Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. *Applied Soft Computing*, 18:232–247, 2014.

[92] Fei Wei, Yuping Wang, and Yuanliang Huo. Smoothing and auxiliary functions based cooperative coevolution for global optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 2736–2741. IEEE, 2013.

[93] Darrell Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[94] Gerhard J Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial optimizationeureka, you shrink!*, pages 185–207. Springer, 2003.

[95] Xin-She Yang. A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pages 65–74. Springer, 2010.

[96] Xin-She Yang and Suash Deb. Cuckoo search via lévy flights. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 210–214. IEEE, 2009.

[97] Xin-She Yang and Suash Deb. Cuckoo search via Lévy flights. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 210–214. IEEE, 2009.

[98] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.

[99] Jingqiao Zhang and Arthur C Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009.

[100] Jingqiao Zhang and Arthur C Sanderson. JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009.

[101] Jun Zhang, Zhi-hui Zhan, Ying Lin, Ni Chen, Yue-jiao Gong, Jing-hui Zhong, Henry SH Chung, Yun Li, and Yu-hui Shi. Evolutionary computation meets machine learning: A survey. *IEEE Computational Intelligence Magazine*, 6(4):68–75, 2011.

[102] Junqi Zhang, Kun Liu, Ying Tan, and Xingui He. Random black hole particle swarm optimization and its application. In *2008 International Conference on Neural Networks and Signal Processing*, pages 359–365. IEEE, 2008.

[103] Xin Zhang, Xiang-tao Li, and Ming-hao Yin. Hybrid cuckoo search algorithm with covariance matrix adaption evolution strategy for global optimisation problem. *International Journal of Bio-Inspired Computation*, 13(2):102–110, 2019.

[104] Yongwei Zhang, Lei Wang, and Qidi Wu. Modified adaptive cuckoo search (macs) algorithm and formal description for global optimisation. *International Journal of Computer Applicationsin Technology*, 44(2):73, 2012.

[105] Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Swagatam Das. Self-adaptive differential evolution with multi-trajectory search for large-scale optimization. *Soft Computing*, 15(11):2175–2185, 2011.

[106] Xinyu Zhou, Zhijian Wu, and Hui Wang. Elite opposition-based differential evolution for solving large-scale optimization problems and its implementation on GPU. In *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 727–732, Dec 2012.