FPGA-Based Compact Differential Evolution for General-Purpose Optimization in Resource-Constrained Devices

Jeng-Shyang Pan, *Senior Member, IEEE*, Pei-Cheng Song, Jyh-Horng Chou, *Fellow, IEEE*, Junzo Watada, and Shu-Chuan Chu

***Abstract*—Resource-constrained devices in open environments face diverse optimization problems, so general-purpose optimization capabilities become important but are currently lacking. Our algorithm framework aims to fill this gap and better understand the issues when implementing general optimization in hardware, especially using field-programmable gate array. Therefore, the challenge is to design an algorithm framework that can handle different optimization problems with fewer hardware resources while maximizing solution performance. Based on the Zynq XC7Z020-2CLG400I device and the compact differential evolution (cDE) algorithm, this article describes the unified software and hardware architecture, the cDE algorithm that incorporates ensemble mutation and crossover strategy as well as uniform mutation operation (cDE-emc-um), providing an efficient and low-resource algorithm framework while maintaining generality. This article also theoretically analyzes the global convergence and low computational complexity of the cDE-emc-um and tests the general optimization capabilities of our algorithm framework through two optimization problems.**

***Index Terms*—Compact differential evolution (cDE), field-programmable gate array (FPGA) implementation, general-purpose optimization, global convergence, resource-constrained devices.**

Manuscript received 17 July 2023; revised 25 November 2023; accepted 30 December 2023. Paper no. TII-23-2634. *(Corresponding author: Shu-Chuan Chu.)*

Jeng-Shyang Pan is with the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China, and also with the Department of Information Management, Chaoyang University of Technology, Taichung 41349, Taiwan (e-mail: jengshyangpan@gmail.com).

Pei-Cheng Song and Shu-Chuan Chu are with the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China (e-mail: spc@sdust.edu.cn; scchu0803@gmail.com).

Jyh-Horng Chou is with the Department of Healthcare Administration and Medical Informatics, Kaohsiung Medical University, Kaohsiung 80708, Taiwan, and also with the Department of Mechanical and Computer-Aided Engineering, Feng Chia University, Taichung 407102, Taiwan (e-mail: choujh@nkust.edu.tw).

Junzo Watada, retired, was with the Graduate School of Information, Production and Systems, Waseda University, Kitakyushu 808-0135, Japan (e-mail: junzo.watada@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2024.3353851.

Digital Object Identifier 10.1109/TII.2024.3353851

I. Introduction

LOW-POWER resource-constrained devices are widely present in industrial environments and play an irreplaceable role [1]. However, they mainly deploy specialized programs for specific tasks, and when faced with more diverse optimization problems in open environments, they are unable to handle them due to resource constraints [2], [3]. Therefore, designing a unified algorithm framework that can handle different optimization problems with limited resources will bring more flexibility and application scenarios for resource-constrained devices.

A. Analysis and Ideas

To design a general-purpose optimization algorithm and framework for resource-constrained devices, it is necessary to ensure the universality of the optimization algorithm, low power consumption and resource usage, and fast processing speed. Optimization methods include mathematics-based, machine-learning-related, and metaheuristic algorithms [4]. Mathematics-based algorithms are accurate but require specific design and task properties [5]. Machine-learning-related algorithms are general but less accurate and more complex, needing special processing for low-power devices [6]. Metaheuristic algorithms are less accurate but general, simple, easy to implement, and suitable for low-power and low-resource devices. This article attempts to use metaheuristic algorithms to handle different optimization problems.

Most of the existing research focuses on hardware optimization of less complex metaheuristic algorithms, using low-power programmable chips like field-programmable gate array (FPGA) for faster computation and lower power consumption [7], [8]. However, most related research only improves algorithm speed by hardware implementation, without applying it to real-world problems or analyzing its theory. This fails to show the value of hardware-based metaheuristic algorithms, which are mostly population-based and resource-intensive. Moreover, the frequent evaluation of these algorithms causes unnecessary interaction between hardware and software, wasting time and hindering real-time performance.

Therefore, this article aims to implement practical problems in the FPGA, avoiding frequent hardware–software calls for objective function evaluation (FE). To do this, a problem model needs to be built in the FPGA and fed with the required

parameters and data. For different optimization problems in resource-constrained devices, we need to consider whether a unified problem model and objective function can represent and solve different optimization problems.

B. Main Work and Contributions

Considering the need of resource-constrained devices to handle different optimization problems under uncertainty, this article designs a unified software and hardware framework and an improved compact differential evolution (cDE) algorithm that incorporate ensemble mutation and crossover strategy as well as uniform mutation operation (cDE-emc-um) and applies them to obstacle avoidance path planning and computing resource allocation. The main contributions of this article are as follows.

1) We design a unified software and hardware architecture in resource-constrained devices for different optimization problems.
2) We propose the cDE-emc-um algorithm with the least computational complexity, improving mutation and integrating it with crossover, and its global probabilistic convergence is theoretically verified.
3) The unified software and hardware architecture and cDE-emc-um algorithm are implemented and verified on the FPGA, and the effects of solving two optimization problems under uncertainty are analyzed.

II. Related Works

This section investigates and analyzes the related research on the cDE algorithm and hardware implementation of compact evolutionary algorithms.

A. Related Analysis of the cDE Algorithm and Its Variants

1) cDE Algorithm and Its Variants: The compact genetic algorithm (cGA) is the first compact evolutionary algorithm, with similar performance to the original genetic algorithm [9]. The cGA uses binary encoding, and later variants include a real-valued compact genetic algorithm (rcGA), with a Gaussian distribution as the probabilistic model. The cDE algorithm follows the rcGA and solves two problems in the DE algorithm, while performing well compared to the cGA and its variants [10]. The cDE algorithm also belongs to a class of estimation of distribution algorithms (EDAs), and there are fewer studies on it.

To the best of our knowledge, Iacca et al. [11] propose a structure of multiple cDE units, each searching the decision space differently. Based on the cDE algorithm, Neri et al. [12] propose disturbed exploitation cDE, which randomly disturbs the virtual population to counteract the search operators and prevent premature convergence. Iacca et al. [13] propose cDE-light, a cDE with less overhead, which modifies crossover and mutation operations without reducing performance. Cruz et al. [14] propose a binary version of cDE and use it to generate binary test sequences for equipment verification. Khalfi et al. [15] propose the compound sinusoidal cDE algorithm, which uses two sinusoidal formulas to adjust the crossover rate and mutation factor in the cDE algorithm.

The cDE algorithm improvement in these studies has two main directions: one is to reduce the resource and computation while keeping the performance unchanged, and the other is to enhance the performance but increase the resource and computation.

2) Related Analysis of the cDE Algorithm: Despite many variants and applications of the compact evolutionary algorithm, its theoretical study is still mainly based on the binary cGA. Droste et al. [16] analyze the expected running time of the cGA on some linear functions and propose a new metric for the EDA. However, they also note that different EDAs need separate analysis due to their structures.

There is less theoretical research on the cDE algorithm than on the cGA. Kononova et al. [17] analyzed the structural bias of some compact optimization algorithms, including cDE, and found that the cDE algorithm has a slight structural bias with the best/1 mutation operator and no structural bias with the rand/2 mutation operator.

The cDE algorithm uses the same crossover and mutation operations as the classic differential evolution (DE) algorithm, so it can also benefit from its theoretical analysis. Hu et al. [18], [19] studied the convergence of the DE algorithm and proved that it does not have global probabilistic convergence.

Based on the related theoretical analysis of the DE algorithm, there is a lack of literature on the convergence and time complexity of the cDE algorithm. We can further explore the theory of the cDE algorithm.

B. Hardware Implementation of Compact Evolutionary Algorithms

There are fewer studies on running compact evolutionary algorithms on hardware than on running evolutionary algorithms. Most of them use the cGA. The authors of [20], [21], and [22] have used quantum computers, Compute Unified Device Architecture platform, and GPUs for hardware implementation. Some papers have also designed and run cGAs on hardware and found that they are very fast [23], [24]. However, the speed of the cGA depends on how complex the fitness function is. The FPGA is programmable and low-power, so it is good for embedded systems. Therefore, this article chooses the FPGA to run the cDE algorithm on hardware.

However, only Jewajinda [25] has studied how to run the cDE algorithm on hardware, designing a Gaussian random number generator for hardware and using it to make the cDE algorithm without changing the cDE algorithm itself, showing that the cDE algorithm on the FPGA was faster than on software.

The cDE algorithm itself has less computation and good performance, but lacks theoretical analysis and further improvement. We will try to modify the cDE to achieve faster speed and less resource usage, to be used as a general-purpose optimization algorithm in resource-constrained devices.

TABLE I
HARDWARE RESOURCE OCCUPATION OF erf AND erf^{-1}

Name	erf		erf^{-1}		Equation (1)		Available
	Total	%	Total	%	Total	%	
LUT	19 398	36%	14 633	27%	39 854	**74%**	53 200
FF	10 949	10%	9703	9%	25 116	23%	106 400
DSP	119	54%	89	40%	222	**101%**	220

The bold data corresponds to the text description in the main body for a quicker understanding of the table data.

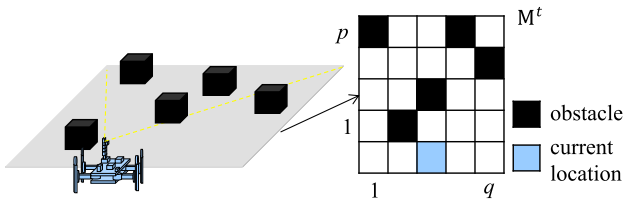

Fig. 1. Obstacle recognition and data acquisition.

III. PRELIMINARIES AND PROBLEM STATEMENT

This section explores the difficulties of the hardware implementation of the cDE algorithm and introduces two different optimization problems for testing.

A. Hardware Implementation Complexity of cDE

The cDE algorithm uses a generic error function (erf) to sample by the inverse cumulative distribution function of truncated normal distribution as

$$y = \sqrt{2}\delta\,\mathrm{erf}^{-1}\left[-\mathrm{erf}\left(\frac{\mu+1}{\sqrt{2}\delta}\right) - x\,\mathrm{erf}\left(\frac{\mu-1}{\sqrt{2}\delta}\right) + x\,\mathrm{erf}\left(\frac{\mu+1}{\sqrt{2}\delta}\right)\right] + \mu \quad (1)$$

where erf^{-1} denotes the inverse erf [26]. The cDE algorithm uses perturbation vector (PV) to represent the distribution probability of the whole population. PV is expressed as PV=$[\mu, \delta]$ with normal distribution. μ is the mean and δ is the standard deviation. However, erf and erf^{-1} are more complex than other operations in the cDE algorithm. Therefore, this article does the hardware implementation first.

Table I shows the resource occupation of erf and erf^{-1}. "Total" is the number of hardware resources used and "%" is the percentage of used resources. The data show that (1) uses 74% of resources. However, the Xilinx Zynq-7020's FPGA chip does not have enough hardware resources left to run the whole cDE algorithm. This makes it hard to do the hardware implementation of the cDE algorithm.

B. Two Different Optimization Problems

Considering the uncertainty of data collection in industrial environments, two simple optimization problems under uncertainty are presented.

1) Obstacle Avoidance Under Perceptual Uncertainty: Fig. 1 shows the microrobot's obstacle avoidance scenario. It captures images in the forward range and creates an obstacle uncertainty map $\mathbf{M}^t$ at time t. The robot then selects optimal drop points based on $\mathbf{M}^t$, avoiding high-uncertainty areas. Suppose that the sensor's observable area in the small jumping robot in Fig. 1 is $\mathbf{U}$. By meshing $\mathbf{U}$, it is divided into $p \times q$ square areas. Due to sensor perception uncertainty, the obstacle map observed at time t is matrix $\mathbf{M}^t = \{m_{ij}, 1 \le i \le p, 1 \le j \le q\}$, where m_{ij} is the obstacle probability in the ith row and jth column region, ranging from 0 to 10. If $m_{ij} = 6$, it is 60%. Then, the obstacle avoidance task goal is to select p drop points across $\mathbf{U}$ and minimize the probability of falling into obstacle regions, as shown in the following equation:

$$D(\mathbf{X}^a) = \sum_{i=1}^{p}\sum_{j=1}^{q}(m_{ij} \times x_{ij}^a)$$
$$\text{s.t.} \sum_{j=1}^{q} x_{ij}^a = 1. \quad (2)$$

x_{ij}^a in (2) shows if the region in the ith row and jth column is chosen as the drop point, and $\mathbf{X}^a = \{x_{ij}^a, 1 \le i \le p, 1 \le j \le q\}$. If $x_{ij}^a = 1$, it is chosen. The constraint means that the robot's next drop point can only jump to the next row of the region within $\mathbf{U}$.

2) Computational Resource Allocation Under Prediction Uncertainty: Microrobots in complex environments often have various sensors. These sensors collect data in real time or periodically. Distributing data from different sensors to different computing units or cores can reduce time consumption. Assume that the robot has l different types of sensors, and the data amount obtained by all sensors at time t is $W^t = [w_1, \ldots, w_i, \ldots, w_l]$. If a robot's processing system has v computing cores, each can allocate resources for tasks. However, tasks and resources change dynamically during operation, and resource allocation by different cores for different tasks is uncertain. Execution time can be predicted based on historical data. Since there is uncertainty in the prediction, then the predicted running time of different tasks on different cores can be $\mathbf{T}^t = \{t_{ij}, 0 \le i \le l, 1 \le j \le v\}$, where t_{ij} is the predicted processing time of data w_i on the jth core. Due to uncertainty, the goal of computing resource allocation is to process all tasks in the shortest time using available cores as follows:

$$Z(\mathbf{X}^b) = \sum_{i=1}^{l}\sum_{j=1}^{v}(t_{ij} \times x_{ij}^b). \quad (3)$$

In (3), x_{ij}^b shows if the ith task is assigned to run on the jth core, and $\mathbf{X}^b = \{x_{ij}^b, 0 \le i \le l, 1 \le j \le v\}$. $x_{ij}^b = 1$ means that the ith task is assigned to the jth core. Based on the above description, the robot can get an allocation scheme by simply passing the estimated time consumption $\mathbf{T}^t$ into the FPGA for processing using the cDE algorithm and then allocate different tasks to different units for processing.

IV. METHODOLOGY

A. Overall Framework

This article proposes a hardware–software architecture for different optimization problems under uncertainty on resource-constrained devices, based on the Advanced RISC Machine

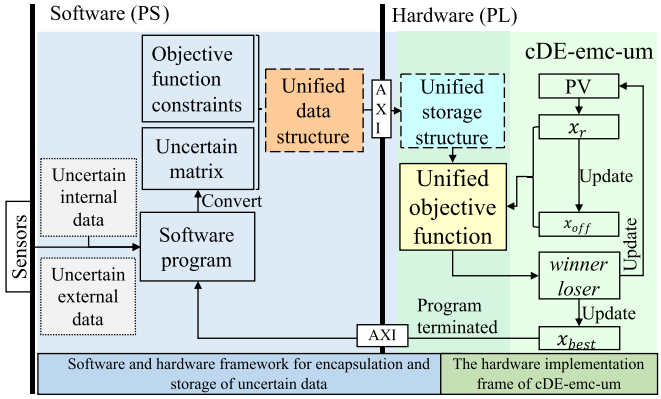

Fig. 2. Hardware–software architecture for handling different optimization problems under uncertainty.

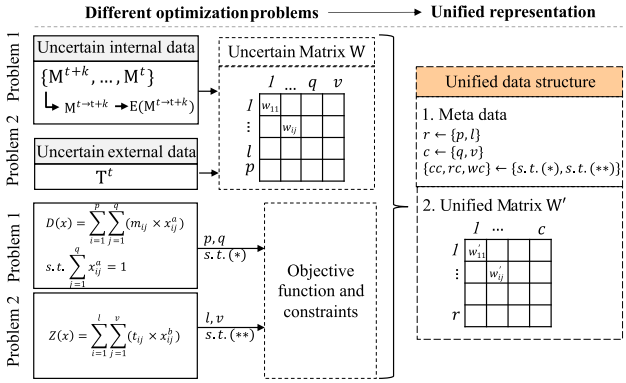

Fig. 3. Encapsulating uncertain data into a unified data structure.

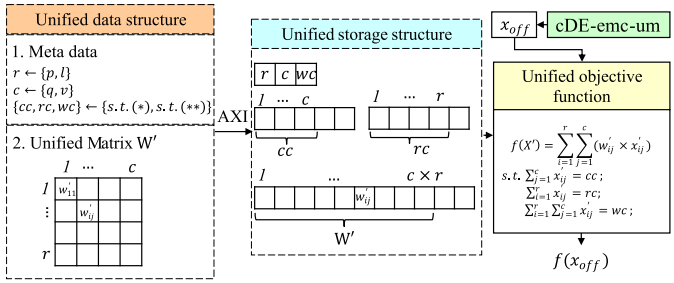

Fig. 4. Transferring and invoking the unified data structure.

(ARM) and FPGA system on chip (SoC) in Fig. 2. To adapt to the resource constraints of Xilinx Zynq-7020, this article proposes the cDE-emc-um algorithm, which incorporates ensemble mutation and crossover strategy as well as uniform mutation operation. This reduces hardware resource usage while ensuring global convergence.

In Fig. 2, the software part is the ARM in SoC, called processor system (PS), which runs different software functions. The hardware part is the FPGA, called programmable logic (PL), which builds hardware circuit logic. The software and hardware architecture encapsulates and preprocesses uncertain data in PS, stores task requirements and data in PL using unified storage structure (UnSS), and uses a configurable unified objective function (UnOF) to solve problems.

The main work includes the following two parts:

1) a unified hardware–software framework for encapsulating and storing uncertain data of different problems;
2) a hardware implementation of cDE-emc-um for general-purpose optimization.

B. Unified Software and Hardware Framework for Different Problems

For the two different problems in Section III-B, Problem 1 is obstacle avoidance and Problem 2 is computational resource allocation. Fig. 3 shows how to encapsulate uncertain data into a unified data structure. Note that Fig. 3 processes the two problems separately, not simultaneously. The steps are as follows.

Step 1: The first part is to represent the uncertain data as an uncertainty matrix W. Problem 1 has observed data from time t to $t+k$ and needs to calculate the expectation $\mathrm{E}(M^{t\to t+k})$. Problem 2 has predicted data, which can be expressed as W.

Step 2: The second part is to construct the objective functions of the two problems in a form that matches the unified data structure, and extract the parameters and constraints.

Step 3: Combine the two parts and encapsulate them into a unified data structure. The unified data structure has two parts. The first part is the metadata, which stores the data parameters, such as the size and constraints of the matrix. The second part is the unified matrix W', which stores W.

The above steps can process two different tasks and encapsulate the uncertain data into a unified format for later use.

Fig. 4 shows how an encapsulated unified data structure is stored in PL, and the metadata and W' in the unified data structure are turned into arrays and passed to the PL through the Advanced eXtensible Interface (AXI) bus. The FPGA has preallocated storage space, which has two parts. The first part is the parameters r, c, cc, rc, and wc, where r and c are the row and column numbers of the input W'. cc, rc, and wc are the constraints on x' in the UnOF. cc and rc denote the constraints on the decision variable matrix in the rows and columns, respectively, while wc denotes the constraints on the sum of the rows and columns. The second part is the stored W'. Then, the cDE-emc-um needs to evaluate the solution x' using the objective function. Based on the UnSS, the UnOF is

$$f(x') = \sum_{i=1}^{r}\sum_{j=1}^{c}\left(w'_{ij} \times x'_{ij}\right)$$
$$\text{s.t.} \sum_{j=1}^{c} x'_{ij} = cc; \sum_{i=1}^{r} x'_{ij} = rc; \sum_{i=1}^{r}\sum_{j=1}^{c} x'_{ij} = wc. \quad (4)$$

The UnOF constructs a uniform template for optimization problems of the class of (2) and (3). The parameters and constraint information in (4) come from the UnSS.

C. cDE-emc-um Algorithm

First, based on the two general conditions for the convergence of the stochastic search algorithm in [27], Lemma 1 is given as follows.

Lemma 1: For the unconstrained optimization problem and its objective function $f()$, the stochastic search algorithm can be defined as $x(t+1) = \text{Update}(x(t), \xi(t))$, where $x(t)$ is the solution at time t, $\xi(t)$ is a new sample from the probabilistic method u_t, and Update is the new solution $x(t+1)$ from $x(t)$ and $\xi(t)$. The stochastic search algorithm converges to the global optimum with probability 1 if the following two conditions are met.

1) *Algorithm condition:*

$$f(x(t+1)) \le \min\{f(x(t)), f(\xi(t))\}.$$

2) *Convergence condition:* For any set A with positive Lebesgue measure

$$\prod_{t=1}^{\infty}(1 - p\{u_t(A)\}) = 0$$

where $u_t(A)$ is the probability of A being generated by u_t.

1) Improvements in Computational Complexity and Global Convergence: To design a general optimization algorithm for resource-constrained devices using the cDE algorithm, the algorithm's global search capability must be ensured to adapt to different problems, and the algorithm's hardware resource usage must be reduced while ensuring optimization performance.

Based on the idea of enhancing the global search capability of the DE algorithm [18], we improved the mutation operation of the cDE algorithm as shown in the following equation:

$$x^{\text{PV}}_{\text{off},i} = x^{\text{PV}}_{r,i} + F\left(x^{\text{PV}}_{\text{best},i} - x^{\text{PV}}_{r,i}\right) + \text{rand}_{(-1,1)} \tag{5}$$

where $x^{\text{PV}}_{\text{best},i}$ is the value of $x_{\text{best},i}$ mapped to -1 to 1, and $x^{\text{PV}}_{\text{off},i}$ and $x^{\text{PV}}_{r,i}$ are the values of the ith dimension from PV, ranging from -1 to 1, which need to be mapped to the search space Φ of the actual objective function. $\text{rand}_{(-1,1)}$ is a random number from -1 to 1 generated by a uniform distribution. Therefore, if $x^{\text{PV}}_{\text{off},i}$ is outside the range of values from -1 to 1, it needs to be constrained. The update formula of scale factor F is $F = F + \frac{0.2t}{G}$, where t is the current iteration number and G is the total iteration number set by the algorithm. Since x_{best} is known, then using (1) generates $x^{\text{PV}}_{r,i}$ in (5), which can then be simplified to reduce unnecessary calculations, resulting in

$$\begin{aligned} x^{\text{PV}}_{\text{off},i} &= F \times (x^{\text{PV}}_{\text{best},i} - \mu) + \mu \\ &\quad - \sqrt{2} \times (F-1) \times \delta \,\text{erf}^{-1}\left[-z_i \,\text{erf}\left(\frac{-1+\mu}{\sqrt{2}\delta}\right)\right. \\ &\quad \left. +(z_i - 1)\,\text{erf}\left(\frac{1+\mu}{\sqrt{2}\delta}\right)\right] + \text{rand}_{(-1,1)} \end{aligned} \tag{6}$$

where z_i is a uniformly distributed random number between 0 and 1. Compared with (5), (6) is simplified by calling the erf function twice and the erf^{-1} function once, avoiding redundant computations.

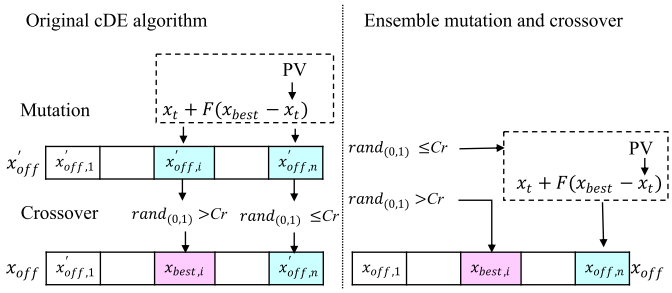

Fig. 5. Integration of mutation and crossover operations.

Algorithm 1: cDE-emc-um.

Input: G, F, Cr, N, N_p
Output: x_{best}
Initialize PV, $\{\mu_i = 0, \delta_i = 10, i \in [1, 2, \ldots, N]\}$.
Generate x^{PV}_{best} by means of PV.
Map x^{PV}_{best} to Φ get x_{best}.
$F_{\min}$=$f(x_{\text{best}})$, $t = 1$.
while $t < G$ or (stop criterion) **do**
  for $i = (1 \to N)$ **do** ▷ Equation (7).
    if $\text{rand}_{(0,1)} > Cr$ **then**
      Generate $x^{PV}_{r,i}$ using PV by (1), $x^{PV}_{\text{off},i}$ by (6).
    else
      $x^{PV}_{\text{off},i} = x^{PV}_{\text{best},i}$.
    end if
  end for
  Map x^{PV}_{off} to Φ get x_{off}, compared with x_{best} to get winner and loser, map winner and loser to Φ.
  for $i = (1 \to N)$ **do** ▷Update PV.
    $\mu_i^{t+1} = \mu_i^t + \frac{\text{winner}_i - \text{loser}_i}{N_p}$.
    $\delta_i^{t+1} = \sqrt{(\delta_i^t)^2 + (\mu_i^t)^2 - (\mu_i^{t+1})^2 + \frac{\text{winner}_i^2 - \text{loser}_i^2}{N_p}}$.
  end for
  Compare winner with x_{best} and update x_{best}, $t = t + 1$.
end while
return x_{best}

Referring to the original cDE algorithm, $x'_{\text{off},i}$ is mutated from $x_{t,i}$, but if $\text{rand}_{(0,1)} > Cr$ in dimension i, it is replaced by the value of $x_{\text{best},i}$ in the crossover operation, as shown in Fig. 5. Therefore, the mutation operation on dimension i is not valid. Therefore, this article integrates crossover and mutation operations, as shown in Fig. 5. If $\text{rand}_{(0,1)} > Cr$, the corresponding dimension value of x_{best} is used directly; otherwise, it is computed. This reduces the calculation of PV-generated solution and mutation as follows:

$$x^{\text{PV}}_{\text{off},i} = \begin{cases} \text{Equation}(6), & \text{if } \text{rand}_{(0,1)} > Cr \\ x^{\text{PV}}_{\text{best},i}, & \text{otherwise.} \end{cases} \tag{7}$$

Based on the above, Algorithm 1 is called pe-cDE/emc-um-best/1/bin or cDE-emc-um for short, where N is the dimension of Φ and N_p is the number of virtual populations.

2) Convergence of the cDE-Emc-Um Algorithm: Based on Algorithm 1 as well as Lemma 1, the convergence of the proposed cDE-emc-um algorithm is proved in this section.

Theorem 1: The cDE-emc-um generates the solution sequence $x(t), t = 1, 2, \ldots$. If the cDE-emc-um gets stuck in a local optimum, it can escape and search globally, which is

$$\prod_{t=1}^{\infty}(1 - p\{u_t(A)\}) = 0.$$

Proof: Let $\mathrm{Emc}()$ denote the operation shown in (7). Let $\mathrm{PV}()$ and $\mathrm{Comp}()$ denote the PV generation solution and competitive selection operations, respectively.

If cDE-emc-um falls into local optimum L^* at moment t_k, according to (5), it is known that

$$p\{\mathrm{Emc}(x(t)) = x(t_k) \mid x(t_k) \in \mathrm{L}^*\} < 1.$$

Then, for any moment t,

$$\begin{aligned} p\{(x(t+1)) = x(t) \mid x(t) \in \mathrm{L}^*\} &= p\{\mathrm{PV}(x(t)) = x(t)\} \cdot p\{\mathrm{Emc}(x(t)) = x(t)\} \\ &\quad \cdot p\{\mathrm{Comp}(x(t)) = x(t)\} \quad \forall x(t) \in \mathrm{L}^* \\ &< 1. \end{aligned}$$

Therefore

$$p\{(x(t+1)) \neq x(t) \mid x(t) \in \mathrm{L}^*\} \geq \zeta(t) > 0$$

where $\zeta(t)$ denotes a small positive value, which may change as t.

Similarly, we can obtain

$$p\{(x(t+1))=A\} \geq p\{(x(t+1))=A \mid x(t) \in \mathrm{L}^*\} \geq \zeta(t) > 0.$$

Then

$$\prod_{t=1}^{\infty}(1 - p\{u_t(A)\}) \leq \prod_{t=1}^{\infty}(1 - \zeta(t)).$$

According to [18], if series $\sum_{i=1}^{+\infty}\zeta(i)$ diverges, then $\prod_{i=1}^{+\infty}(1-\zeta(i)) = 0$.

Since $\zeta(t)$ is divergent in (5), we can obtain

$$\prod_{t=1}^{\infty}(1 - p\{u_t(A)\}) = 0.$$

Theorem 1 shows that cDE-emc-um can escape the local optimum and converge *globally with probability 1*.

This article compares the convergence, memory consumption, and erf and erf^{-1} function calls of the proposed cDE-emc-um algorithm and the original cDE algorithm in Table II.

3) Simplify the Hardware Implementation Complexity: As mentioned in Section III-A, the erf and erf^{-1} functions in the cDE algorithm have high hardware implementation complexity. Although Algorithm 1 reduces unnecessary computations, to further lower the hardware complexity, the erf function can be approximated by (8)–(11), and the erf^{-1} function can be approximated by (12)

$$\mathrm{erf}(x) \approx 1 - \frac{1}{(1 + a_1x + a_2x^2 + a_3x^3 + a_4x^4)^4} \tag{8}$$

TABLE II
COMPARISON OF THE CDE-EMC-UM AND CDE ALGORITHMS

Algorithm	cDE	cDE-emc-um
Global Convergence	×	√
erf calls	9	**2**
erf^{-1} calls	3	**1**
Memory size	$8 \times N + 7$	$4 \times N + 7$

The bold data is employed to showcase the advantages of the proposed method or the disadvantages of the compared methods.

$$\mathrm{erf}(x) \approx 1 - \left(a_1t + a_2t^2 + a_3t^3\right)e^{-x^2} \tag{9}$$

$$\mathrm{erf}(x) \approx 1 - \frac{1}{(1 + a_1x + a_2x^2 + \cdots + a_6x^6)^{16}} \tag{10}$$

$$\mathrm{erf}(x) \approx 1 - \left(a_1t + a_2t^2 + \cdots + a_5t^5\right)e^{-x^2} \tag{11}$$

$$\mathrm{erf}^{-1}(z) = \frac{\sqrt{\pi}}{2}\left(z + \frac{\pi}{12}z^3 + \frac{7\pi^2}{480}z^5 + \frac{127\pi^3}{40320}z^7 + \frac{4369\pi^4}{5806080}z^9 + \frac{34807\pi^5}{182476800}z^{11} + \cdots\right). \tag{12}$$

The maximum error of (8) is 5×10^{-4}, where $a_1 = 0.278393$, $a_2 = 0.230389$, $a_3 = 0.000972$, and $a_4 = 0.078108$. The maximum error of (9) is 2.5×10^{-5}, where $t = 1/(1 + px)$, $p = 0.47047$, $a_1 = 0.3480242$, $a_2 = -0.0958798$, and $a_3 = 0.7478556$. The maximum error of (10) is 3×10^{-7}, where $a_1 = 0.0705230784$, $a_2 = 0.0422820123$, $a_3 = 0.0092705272$, $a_4 = 0.0001520143$, $a_5 = 0.0002765672$, and $a_6 = 0.0000430638$. The maximum error of (11) is 1.5×10^{-7}, where $t = 1/(1 + px)$, $p = 0.3275911$, $a_1 = 0.254829592$, a_2 = -0.284496736, $a_3 = 1.421413741$, $a_4 = -1.453152027$, and $a_5 = 1.061405429$. For (8)–(11), the range of x is $x \geq 0$. If x is negative, we can use the property that the erf function is an odd function, $\mathrm{erf}(-x) = -\mathrm{erf}(x)$.

Let the four approximations be denoted as erf_{a1}, erf_{a2}, erf_{a3}, and erf_{a4}; the above approximation to erf^{-1} is denoted by erf^{-1}_{a2}. Since random numbers are required for uniform distribution, the hardware implementation of the cDE-emc-um algorithm uses the linear feedback shift register (LFSR) as the random number generator, which is $x^{32} + x^{22} + x^2 + x + 1$.

V. EXPERIMENTS AND EVALUATION

A. Experimental Settings

This article proposes the cDE-emc-um algorithm, suitable for resource-constrained devices, which reduces computational complexity while maintaining global search performance. In addition, this article improves the original cDE algorithm based on the ideas in [18], called cDE-um (pe-cDE/um-best/1/bin), which also has the ability to escape local optima. It will be compared with the cDE-emc-um and cDE algorithms.

The proposed algorithm has both software and hardware implementations, so its performance is tested on a personal computer (PC), ARM chip (software), and FPGA chip (hardware) for different optimization problems. Considering the limited hardware resources of FPGA platforms and the use of the same

TABLE III
EIGHT BENCHMARK FUNCTIONS (IMPLEMENTED IN FPGA)

Functions	Equation $f(x)$	Search Domain
F1	$\sum_{i=1}^{D} x_i^2$	$[-100, 100]$
F2	$x_1^2 + 10^6 \sum_{i=2}^{D} x_i^2$	$[-100, 100]$
F3	$10^6 x_1^2 + \sum_{i=2}^{D} x_i^2$	$[-100, 100]$
F4	$\sum_{i=1}^{D-1} \left(100\left(x_i^2 - x_{i+1}\right)^2 + \left(x_i - 1\right)^2\right)$	$[-30, 30]$
F5	$\sqrt{\left\lvert\left(\sum_{i=1}^{D} x_i^2\right)^2 - \left(\sum_{i=1}^{D} x_i\right)^2\right\rvert} + (0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i)/D + 0.5$	$[-100, 100]$
F6	$\sum_{i=1}^{D} \left(x_i^2 - 10\cos\left(2\pi x_i\right)\right) + 10 \cdot D$	$[-5.12, 5.12]$
F7	$\sum_{i=1}^{D} \lvert x_i \sin\left(x_i\right) + 0.1 x_i\rvert$	$[-10, 10]$
F8	$\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-100, 100]$

TABLE IV
RESOURCE CONSUMPTION OF DIFFERENT APPROXIMATION FUNCTIONS

Combinations	DSP	FF	LUT	Latency	Interval	Error
$\mathrm{erf} + \mathrm{erf}^{-1}$	222	25 116	39 854	–	–	0
$\mathrm{erf}_{a1} + \mathrm{erf}^{-1}$	**139**	**16 228**	**24 740**	138-455	139-456	0.00257696
$\mathrm{erf}_{a2} + \mathrm{erf}^{-1}$	176	18 023	28 037	138-455	139-456	0.328255
$\mathrm{erf}_{a3} + \mathrm{erf}^{-1}$	150	16 611	25 523	158-475	159-476	0.32784
$\mathrm{erf}_{a4} + \mathrm{erf}^{-1}$	198	18 823	29 367	148-465	149-466	0.327849
$\mathrm{erf}_{a1} + \mathrm{erf}_{a2}^{-1}$	**61**	**6884**	**11 030**	191-191	191-191	0.00257695
$\mathrm{erf}_{a2} + \mathrm{erf}_{a2}^{-1}$	87	8444	13 808	191-191	191-191	0.328255
$\mathrm{erf}_{a3} + \mathrm{erf}_{a2}^{-1}$	61	6968	11 228	211-211	211-211	0.32784
$\mathrm{erf}_{a4} + \mathrm{erf}_{a2}^{-1}$	109	9116	15 093	201-201	201-201	0.327849

The bold data is employed to showcase the advantages of the proposed method or the disadvantages of the compared methods.

benchmarks on different platforms, eight simple benchmark functions in Table III are selected in this article, including four unimodal functions (F1:Sphere, F2:BentCigar, F3:Discus, F4:Rosenbrock) and four multimodal functions (F5:HGBat, F6:Rastrigin, F7:Alpine1, and F8:Griewank).

The experimental equipment includes three environments. The FPGA chip used in the hardware implementation is the XC7Z020-2CLG400I model chip of ZYNQ 7000 series. The XC7Z020-2CLG400I is divided into the PS part and the PL part. The PS section includes two ARM Cortex-A9 processor (800 MHz). The hardware resources of PL (100 MHz) include 53 200 lookup tables (LUTs), 106 400 FlipFlops (FF), 140 Block RAMS (BRAM), and 220 digital signal processing (DSP) units. The processor of a PC is configured as AMD Ryzen 7 5800H with Radeon Graphics@3.20 GHz. This article is based on the Zynq-7020 chip using Vivado Design Suite for hardware and software design. First, the proposed cDE-emc-um algorithm, UnOF, and UnSS need to be implemented inside the FPGA, and this step is implemented using Vivado High-Level Synthesis. The code to reproduce all of the experiments is available at https://github.com/Spacewe-outlook/cDE-emc-um.

B. Hardware Implementation Complexity of erf and erf^{-1} Approximation Functions

Based on Section IV-C3, this section implements the approximate functions of erf and erf^{-1} in the FPGA and compares the resource occupation with the original function in Table IV.

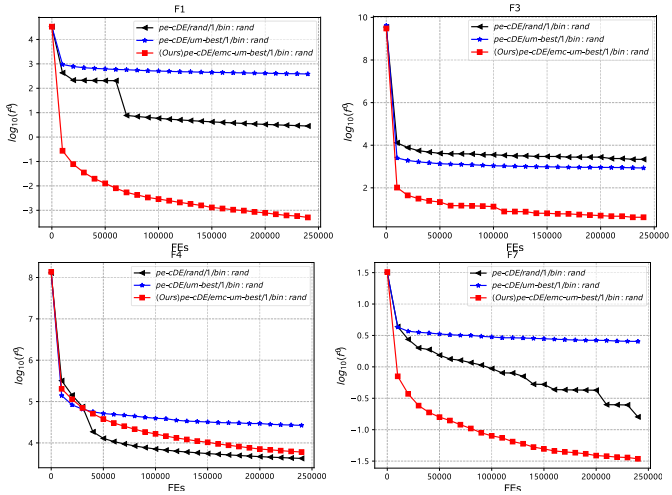

Fig. 6. Convergence of the three algorithms on F1, F3, F4, and F7.

"Latency" in Table IV is the clock period from input to output data. "Interval" is the clock period between consecutive input data. Table IV shows that the original erf and erf^{-1} functions use 39 854 LUTs, but the PL part has only 53 200 LUTs. In addition, the first combination in Table IV is the actual value, and we used it as a benchmark to test the errors of the other eight combinations of approximate functions with the actual value, the test code is based on (1), where $\mu = 0$ and $\delta = 10$, and it was tested and averaged 1 000 000 times, and the results are shown in the Error column in Table IV. It can be seen that there are different combinations of the error in (1) with the actual value; still, the first combination is more competitive . Combining (8)–(11), the resources required for erf_{a1} and erf_{a2}^{-1} are less, so we will use (8) and (12) to replace the erf and erf^{-1} functions for hardware implementation, respectively.

C. Performance Evaluation of the cDE-Emc-Um Algorithm

1) Convergence and Stability of cDE-Emc-Um: This article compares the convergence and stability of the three algorithms. They have the same parameters $F = 0.7$ and $Cr = 0.9$. Each algorithm runs 51 times on each benchmark function with 250 000 FEs on PC and records data. The algorithms use C++ programming on a PC. Fig. 6 shows the convergence of the algorithms on F1, F3, F4, and F7, which are four representative benchmark functions. ": rand" after the algorithm legend in Fig. 6 means using rand() function in C++ to generate random numbers. The vertical axis is the logarithm of the algorithm's result at the base of 10.

Fig. 6 shows that the cDE-emc-um algorithm converges faster than the other two algorithms, but the proposed cDE-emc-um algorithm is less effective on F4. The cDE-um algorithm only changes the mutation operation from the original cDE algorithm, which improves the global search ability but slows down the convergence speed. Fig. 7 compares the stability of the final results of 51 runs on F1 and F8. The cDE-emc-um algorithm has more stable results in general, but less stable results on F8, even

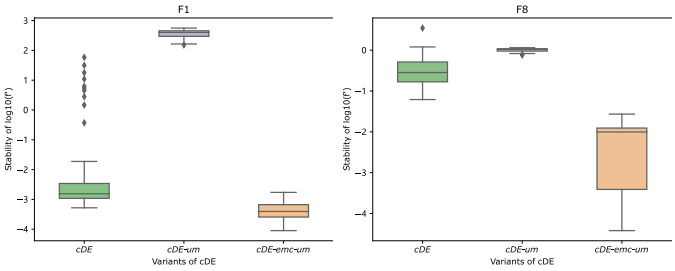

Fig. 7. Stability of the three algorithms on F1 and F8.

TABLE V
RESULTS AND TIME ON ARM (800 MHZ) FOR THE THREE ALGORITHMS (FES = 25 000)

Function	Results			Time (ms)		
	cDE	cDE-um	cDE-emc-um	cDE	cDE-um	cDE-emc-um
F1	1.908E+01(+)	7.315E+02(+)	**4.763E-02**	1549.51	1071.902	698.6471
F2	4.169E+06(+)	5.219E+08(+)	**1.978E+04**	1548.804	1072.725	698.1176
F3	1.139E+04(+)	1.974E+03(+)	**3.987E+01**	1546.902	1070.412	697.4902
F4	2.292E+05(+)	8.172E+04(+)	**7.691E+04**	1569.882	1096.49	722.6275
F5	2.455E+01(+)	7.987E+02(+)	**1.285E-01**	1590.569	1114.275	741.1176
F6	4.309E+01(+)	4.250E+01(+)	**2.981E-02**	1612.471	1149.078	754.4118
F7	2.941E+00(+)	3.333E-01(+)	**1.890E-05**	1616.235	1147.941	768.5686
F8	5.266E-01(+)	1.158E+00(+)	**4.794E-02**	1856.294	1390.647	1003.529

The bold data is employed to showcase the advantages of the proposed method or the disadvantages of the compared methods.

TABLE VI
HARDWARE RESOURCE USAGE OF CDE AND CDE-EMC-UM ALGORITHMS

Resource	cDE		cDE-emc-um		Available
	Total	Utilization	Total	Utilization	
LUT	57 964	**108.95%**	20 680	38.87%	53 200
FF	34 969	32.86%	16 050	15.08%	106 400
BRAM	13	9.28%	3	2.14%	140
DSP	332	**150.9%**	166	75.45%	220

The bold data is employed to showcase the advantages of the proposed method or the disadvantages of the compared methods.

though it performs better due to its global search capability and its ability to jump out of the local optimum. The other algorithms are more stable because the other algorithms tend to fall into the local optimum.

2) Performance Evaluation on ARM: This article runs three algorithms on ARM 51 times on each of the eight benchmark functions with 25,000 FEs each time and shows in Table V. The Wilcoxon rank sum test with a significance level of 5% is applied to compare the significance of the differences between cDE, cDE-um, and cDE-emc-um. "+," "−," and "≈" indicate that cDE-emc-um is significantly better than, worse than, and not significantly different from the other algorithms, respectively.

In Table V, the proposed cDE-emc-um algorithm has a significant advantage in both the final results and execution time for all eight functions.

3) Performance and Resource Consumption of Hardware Implementation: The original cDE algorithm has too many operations and uses more FPGA chip resources than available, as shown in Table VI. Therefore, this article only compares the cDE-emc-um algorithm on ARM and FPGA.

Table VI shows the resources for the hardware implementation of the cDE-emc-um algorithm, which includes the two test functions (F4, F6) on the FPGA. Compared with the cDE, the cDE-emc-um algorithm reduces the resource consumption, which helps to implement it on resource-constrained devices.

TABLE VII
RESULTS AND TIME OF CDE-EMC-UM ALGORITHM AT DIFFERENT FES ON ARM (800 MHZ) AND FPGA (100 MHZ)

Functions	25 000 FEs		5000 FEs		1000 FEs	
	Result	Time (ms)	Result	Time (ms)	Result	Time (ms)
F4 (ARM)	7.69E+04	722.6275	3.72E+05	135.64	1.48E+06	28.0588
F4 (FPGA)	**6.09E+04**	489.9969	**3.03E+05**	97.9803	**1.34E+06**	19.5686
F6 (ARM)	2.98E-02	754.4118	1.19E+00	143	2.96E+01	29.8039
F6 (FPGA)	**1.62E-02**	561.7434	**1.03E+00**	112.3333	**2.93E+01**	22.4509

The bold data is employed to showcase the advantages of the proposed method or the disadvantages of the compared methods.

TABLE VIII
RESOURCE OCCUPANCY OF SYSTEM HARDWARE IMPLEMENTATION

Resource	Total	Available	Utilization
LUT	38 594	53 200	72.55%
FF	28 373	106 400	26.66%
BRAM	11	140	7.86%
DSP	174	220	79.09%

This article tests the cDE-emc-um algorithm on ARM and FPGA. The tests compare different FEs (FEs = 25 000, 5000, 1000). The algorithm runs 51 times on each benchmark function on both ARM and FPGA and records the final results and execution times, as shown in Table VII.

Table VII shows that the cDE-emc-um algorithm performs better on FPGAs than on ARMs, but the difference is small. The cDE-emc-um algorithm on FPGAs does not use hardware optimization techniques, such as pipelining or loop unfolding, so it does not fully exploit the FPGAs' advantages. The cDE-emc-um algorithm on FPGAs takes less time than on ARMs, but the difference is not significant.

D. cDE-Emc-Um for Different Optimization Problems

To verify the general optimization ability of the proposed algorithm and framework on resource-constrained devices, this article implements the proposed UnSS, UnOF, and cDE-emc-um on the FPGA to handle two different optimization problems.

1) Hardware Resource Consumption: According to Fig. 2, Table VIII shows the resource usage of hardware implementation of UnSS, UnOF, and cDE-emc-um in the PL part. It uses more resources than Table VI, which implements the two benchmark functions in hardware, because the architecture in Fig. 4 needs preopened storage space for the UnSS and the UnOF. However, it can still fit in the available resources.

2) General-Purpose Optimization Ability for Two Different Problems: Based on the design of UnSS and UnOF in Fig. 2, using cDE-emc-um as a solver, for different types of optimization problems, only need to be converted into a unified data structure in PS for direct solution. The test results of two different optimization problems are shown in Table IX. Table IX shows the average results and times of 51 runs of the proposed architecture and algorithm.

It should be noted that the cDE-emc-um algorithm on ARM uses LFSR to generate random numbers with the same seeds as the algorithm on FPGA for each test . Therefore, the results of the

TABLE IX
PERFORMANCE OF THE PROPOSED ALGORITHM AND ARCHITECTURE ON TWO DIFFERENT OPTIMIZATION PROBLEMS

FEs		Problem 1		Problem 2	
		ARM	Our	ARM	Our
25,000	Result	3904.55	3897.54	76.5598	76.42234
	Time (ms)	34 952.12	12 851.8	685.34	**252.00**
5000	Result	3954.47	3945.75	77.53863	77.36765
	Time (ms)	7009.957	2570.197	137.45	**50.40**
1000	Result	3956.72	3939.71	77.58274	77.24921
	Time (ms)	1394.785	513.8771	27.35	**10.08**

The bold data is employed to showcase the advantages of the proposed method or the disadvantages of the compared methods.

two methods in Table IX are similar, and the difference in calculation accuracy causes them. The algorithm reinitializes the random number seed and reruns when evaluating different iteration times. Therefore, the difference of random number seeds causes the result of 5000 iterations to be worse than 1000 iterations.

Table IX also verifies that the proposed architecture and cDE-emc-um algorithm perform the same in hardware and software, but take less time. 1000 FEs need 10 ms, while ARM takes two to three times longer. This helps resource-constrained devices to process different optimization tasks with low power consumption and fast speed.

VI. CONCLUSION

In order to make resource-constrained devices capable of handling different optimization problems, this article designed a unified software and hardware processing framework for different optimization problems, using a cDE-emc-um algorithm as a general solver with low complexity and global convergence. This article offered new insights for developing general solvers on resource-constrained devices, addressed the challenges of compact evolutionary algorithms in such environments, and demonstrated the feasibility of achieving general optimization capabilities on resource-constrained devices.

However, this work is limited by the fact that the optimization problems used for verification do not represent all cases, and the proposed framework only applies to certain simple problems. Further research is needed to consider problem types and data uncertainty. In addition, the resource-constrained devices used have limited resources, and better algorithms can be designed for devices with stronger performance.

REFERENCES

[1] M. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, "Efficient acceleration of deep learning inference on resource-constrained edge devices: A review," *Proc. IEEE*, vol. 111, no. 1, pp. 42–91, Jan. 2023.
[2] C. Chen et al., "Deep learning on computational-resource-limited platforms: A survey," *Mobile Inf. Syst.*, vol. 2020, 2020, Art. no. 8454327.
[3] F. Simonetti, A. D'Innocenzo, and C. Cecati, "Neural network model predictive control for CHB converters with FPGA implementation," *IEEE Trans. Ind. Informat.*, vol. 19, no. 9, pp. 9691–9702, Sep. 2023.
[4] D. Zhu, B. Zhou, and S. X. Yang, "A novel algorithm of multi-AUVs task assignment and path planning based on biologically inspired neural network map," *IEEE Trans. Intell. Veh.*, vol. 6, no. 2, pp. 333–342, Jun. 2021.
[5] F. Bu and D. E. Chang, "Feedback gradient descent: Efficient and stable optimization with orthogonality for DNN," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 6106–6114.
[6] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, 2021.
[7] A. L. X. Da Costa, C. A. D. Silva, M. F. Torquato, and M. A. C. Fernandes, "Parallel implementation of particle swarm optimization on FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 11, pp. 1875–1879, Nov. 2019.
[8] E. Alqudah and A. Jarrah, "Parallel implementation of genetic algorithm on FPGA using Vivado high level synthesis," *Int. J. Bio-Inspired Comput.*, vol. 15, no. 2, pp. 90–99, 2020.
[9] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 287–297, Nov. 1999.
[10] E. Mininno, F. Neri, F. Cupertino, and D. Naso, "Compact differential evolution," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 32–54, Feb. 2011.
[11] G. Iacca, E. Mininno, and F. Neri, "Composed compact differential evolution," *Evol. Intell.*, vol. 4, no. 1, pp. 17–29, 2011.
[12] F. Neri, G. Iacca, and E. Mininno, "Disturbed exploitation compact differential evolution for limited memory optimization problems," *Inf. Sci.*, vol. 181, no. 12, pp. 2469–2487, 2011.
[13] G. Iacca, F. Caraffini, and F. Neri, "Compact differential evolution light: High performance despite limited memory requirement and modest computational overhead," *J. Comput. Sci. Technol.*, vol. 27, no. 5, 2012, Art. no. 1056.
[14] A. M. Cruz, R. B. Fernández, H. M. Lozano, M. A. R. Salinas, and L. A. V. Vargas, "Automated functional test generation for digital systems through a compact binary differential evolution algorithm," *J. Electron. Testing*, vol. 31, no. 4, pp. 361–380, 2015.
[15] S. Khalfi, A. Draa, and G. Iacca, "A compact compound sinusoidal differential evolution algorithm for solving optimisation problems in memory-constrained environments," *Expert Syst. Appl.*, vol. 186, 2021, Art. no. 115705.
[16] S. Droste, "A rigorous analysis of the compact genetic algorithm for linear functions," *Nat. Comput.*, vol. 5, no. 3, pp. 257–283, 2006.
[17] A. V. Kononova, F. Caraffini, H. Wang, and T. Bäck, "Can compact optimisation algorithms be structurally biased?," in *Proc. Int. Conf. Parallel Probl. Solving Nature*, 2020, pp. 229–242.
[18] Z. Hu, S. Xiong, Q. Su, and X. Zhang, "Sufficient conditions for global convergence of differential evolution algorithm," *J. Appl. Math.*, vol. 2013, 2013, Art. no. 193196.
[19] Z. Hu, S. Xiong, Q. Su, and Z. Fang, "Finite Markov chain analysis of classical differential evolution algorithm," *J. Comput. Appl. Math.*, vol. 268, pp. 121–134, 2014.
[20] S. Yingchareonthawornchai, C. Aporntewan, and P. Chongstitvatana, "An implementation of compact genetic algorithm on a quantum computer," in *Proc. IEEE 9th Int. Conf. Comput. Sci. Softw. Eng.*, 2012, pp. 131–135.
[21] V. Sumati, "Parallel compact genetic algorithm on CUDA-C platform," *Int. J. Comput. Appl.*, vol. 84, no. 5, pp. 13–16, 2013.
[22] A. Ferigo and G. Iacca, "A GPU-enabled compact genetic algorithm for very large-scale optimization problems," *Mathematics*, vol. 8, no. 5, 2020, Art. no. 758.
[23] M. A. Moreno-Armendáriz, N. Cruz-Cortés, and A. León-Javier, "A novel hardware implementation of the compact genetic algorithm," in *Proc. IEEE Int. Conf. Reconfigurable Comput. FPGAs*, 2010, pp. 156–161.
[24] T. C. Oliveira and V. P. Júnior, "An implementation of compact genetic algorithm on FPGA for extrinsic evolvable hardware," in *Proc. IEEE 4th Southern Conf. Program. Log.*, 2008, pp. 187–190.
[25] Y. Jewajinda, "FPGA-based compact differential evolution," in *Proc. IEEE Int. Comput. Sci. Eng. Conf.*, 2015, pp. 1–6.
[26] P.-C. Song, J.-S. Pan, and S.-C. Chu, "A parallel compact cuckoo search algorithm for three-dimensional path planning," *Appl. Soft. Comput.*, vol. 94, 2020, Art. no. 106443.
[27] F. J. Solis and R. J.-B. Wets, "Minimization by random search techniques," *Math. Oper. Res.*, vol. 6, no. 1, pp. 19–30, 1981.

Jeng-Shyang Pan (Senior Member, IEEE) received the M.S. degree in communication engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1988, and the Ph.D. degree in electrical engineering from the University of Edinburgh, Edinburgh, U.K., in 1996.

He is currently a Professor with the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China. His current research interests include soft computing, information security, and signal processing.

Pei-Cheng Song was born in 1997. He received the B.S. degree in Internet of Things (IoT) technology in 2019 from Shandong University of Science and Technology, Qingdao, China, where he is currently working toward the Ph.D. degree with the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China.

His current research interests include swarm intelligence and artificial intelligence.

Junzo Watada was born in 1945. He received the B.Sc. and M.Sc. degrees in electrical engineering from Osaka City University, Osaka, Japan, 1970 and 1972, respectively, and the Ph.D. degree in knowledge management and information science from Osaka Prefecture University, Sakai, Japan, in 1983 .

He, retired, was a Professor with the Graduate School of Information, Production, and Systems, Waseda University, Tokyo, Japan, until March 2016. His research interests include big data analytics, soft computing, deep learning.

Jyh-Horng Chou (Fellow, IEEE) received the B.S. and M.S. degrees in engineering science from National Cheng Kung University, Tainan, Taiwan, in 1981 and 1983, respectively, and the Ph.D. degree in mechatronic engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 1988.

He is currently with the Department of Healthcare Administration and Medical Informatics, Kaohsiung Medical University, Kaohsiung, Taiwan. His research interests include intelligent systems and control, computational intelligence, robust control, and robust optimization.

Shu-Chuan Chu received the Ph.D. degree in evolutionary algorithms and artificial intelligence from the School of Computer Science, Engineering and Mathematics, Flinders University, Adelaide, SA, Australia, in 2004.

In December 2009, she joined Flinders University, after nine years with Cheng Shiu University, Kaohsiung, Taiwan. She has been a Research Fellow with the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China, since 2019. Her research interests include swarm intelligence, intelligent computing, and data mining.