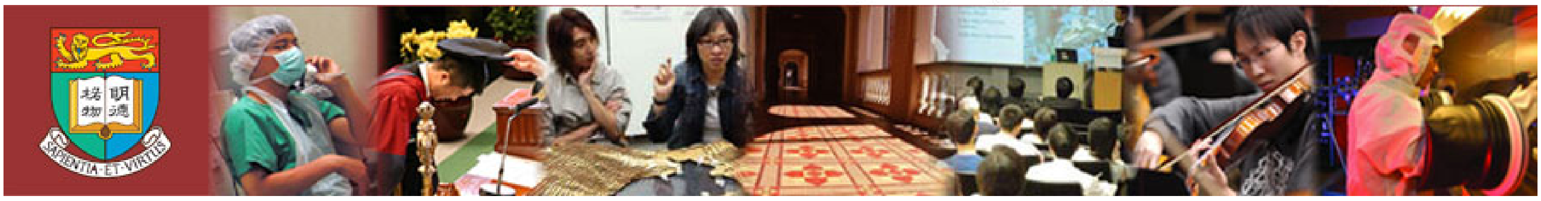

Real-Coded Chemical Reaction Optimization

Albert Y. S. Lam, *Member, IEEE,* Victor O. K. Li, *Fellow, IEEE,* and James J. Q. Yu

***Abstract*—Optimization problems can generally be classified as continuous and discrete, based on the nature of the solution space. A recently developed chemical-reaction-inspired metaheuristic, called chemical reaction optimization (CRO), has been shown to perform well in many optimization problems in the discrete domain. This paper is dedicated to proposing a real-coded version of CRO, namely, RCCRO, to solve continuous optimization problems. We compare the performance of RCCRO with a large number of optimization techniques on a large set of standard continuous benchmark functions. We find that RCCRO outperforms all the others on the average. We also propose an adaptive scheme for RCCRO which can improve the performance effectively. This shows that CRO is suitable for solving problems in the continuous domain.**

***Index Terms*—Chemical reaction optimization, continuous optimization, metaheuristics.**

I. Introduction

We often encounter optimization problems in scientific and technological research and development. Examples include geneticists desire to design the optimal DNA sequences in order to maximize the reliability of molecular computation [1], while physicists may be interested in optimally inferring elastic material properties from the observed deformation data [2]. Economists would like to predict stock market trends precisely (i.e., minimizing the prediction error) [3] and electrical engineers might want to schedule power generation at a power system in order to meet the customers' demand while minimizing the generation cost [4].

The goal of optimization is to determine the best (or optimal) element s out of a set of similar ones, S, where s and S are called a solution and a solution space, respectively. We always have a variable x to carry a particular solution s (i.e., to assign x with the values specified by s). The optimality of x is evaluated by an objective function f and its outcome is its objective function value y, i.e., $y = f(x)$. x is usually a vector of n components, where n specifies the dimensions of f, while y is a scalar. An optimization problem can be subject to certain constraints (assume there are m constraints), $c_1(x), c_2(x), \ldots , c_m(x)$, which confine s to S. In other words, the constraints decide if s is feasible to be included in S (i.e., if s is a feasible solution). Thus, $c_1(x), c_2(x), \ldots , c_m(x)$ define the size and scope of S and f characterizes the "landscape" of S. Our objective can be to either maximize or minimize f. Without loss of generality, we assume minimization throughout this paper. Then we aim to find the minimum solution $s^* \in S$ such that $f(s^*) \leq f(s), \forall s \in S$. Mathematically, a minimization problem can be stated as

$$\min_{x\in R^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in E \\ c_i(x) \leq 0, & i \in I \end{cases} \tag{1}$$

where R, E, and I symbolize the real number set, the index set for equality constraints, and the index set for inequality constraints, respectively.

In general, the solution space of any optimization problem is either a continuous or discrete domain. Continuous optimization assumes that x can take any real numbers within the intervals specified by the problem (i.e., through the constraints) while discrete optimization is restricted to discrete numbers. The continuity of f sometimes provides additional information about the locations of the optimums, e.g., the derivatives f' give us the direction of the optimal solution (i.e., the downhill direction) relative to the current search point. Traditional derivative-based optimization techniques include steepest descent, Newton's method, and others [5]. They are powerful and effective but suffer from a common drawback that they easily get stuck in local optimums. This makes them inapplicable to multimodal, nonlinear, or discrete optimization problems. We are interested in derivative-free techniques which are suitable to be applied to all kinds of optimization problems. Such methods are usually referred to as metaheuristics. Many successful ones are nature-inspired, e.g., simulated annealing (SA) [6], genetic algorithm (GA) [7], ant colony optimization (ACO) [8], particle swarm optimization (PSO) [9], and others. Their basic optimization philosophies do not limit them to either continuous or discrete optimization, but they are always focused on either continuous or discrete types in their original designs. Then researchers tried to extend their ideas to adapt the algorithms to the other type so that they become truly general-purpose.

SA was originally proposed in [6] for combinatorial optimization.[1] Then it was modified to be applicable to problems

Manuscript received September 10, 2010; revised April 2, 2011; accepted June 16, 2011. Date of current version May 24, 2012. This work was supported in part by the Strategic Research Theme of Information Technology of the University of Hong Kong. The work of A. Y. S. Lam was also supported in part by the Croucher Foundation Research Fellowship.

A. Y. S. Lam is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 USA (e-mail: albertlam@ieee.org).

V. O. K. Li is with the Department of Electrical and Electronic Engineering, University of Hong Kong, Pokfulam, Hong Kong, and also with the Department of Computer Engineering, King Saud University, Riyadh 11451, Saudi Arabia (e-mail: vli@eee.hku.hk).

J. J. Q. Yu is with the Department of Electrical and Electronic Engineering, University of Hong Kong, Pokfulam, Hong Kong (e-mail: jamesyjq@hku.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2011.2161091

[1]Combinatorial optimization is a branch of discrete optimization and it mainly focuses on graphs, matroids, and other discrete structures [10].

in the continuous domains in [11]. In the original GAs [12], each solution of a problem is carried by a chromosome, which is a sequence of binary numbers. Thus, an encoding scheme is required to make a solution fit the structure of a chromosome. Generally, the encoding schemes allow solutions in any domains to be mapped to binary strings. Real-coded GAs, whose chromosomes can take real numbers, were designed and it is shown that for real-valued optimization problems, floating-point representations are superior to binary representations since they are more consistent, more precise, and lead to faster execution [13]. Like SA, ACO was originally designed for combinatorial problems and it determines a solution according to a discrete probability distribution [14]. It was extended to continuous problems by generating new solutions with a probability density function instead [15]. PSO was first proposed to be continuous-valued [16]. By considering the velocity on each dimension as a probability of changing a bit, a discrete version of PSO was developed in [17].

A chemical reaction-inspired metaheuristic, called chemical reaction optimization (CRO), was recently proposed in [18]. In a chemical reaction, the initial species (i.e., reactants) in the high-energy unstable states undergo a sequence of collisions, pass through some energy barriers, and become the final products in low-energy stable states. CRO captures this phenomenon of driving high-energy molecules to stable states through various types of elementary reactions. CRO has been demonstrated to be a successful optimization algorithms with many applications [18]–[24] and most of them are combinatorial problems. In other words, the current version of CRO is designed to work in the discrete domain. As other natural-inspired algorithms (e.g., SA, GA, ACO, and PSO) have been shown to work well in both continuous and discrete domains, this paper is dedicated to an extension of CRO to continuous problems. We call this version of CRO real-coded CRO (RCCRO). Moreover, the no-free-lunch theorem states that all metaheuristics which search for extrema are exactly the same in performance when averaged over all possible objective functions [25]. Hence, CRO must possess the same ability in solving optimization problems as the others and we desire to evaluate how RCCRO performs with a board range of continuous problems. We will examine the performance of RCCRO with a large set of continuous benchmark problems.

The rest of this paper is organized as follows. Section II briefly gives the original framework of CRO. In Section III, we explain the modifications to the original CRO to adapt it to the continuous domain and we also propose an adaptive scheme for RCCRO. We describe the benchmark problems and compare the simulation results of the basic RCCRO scheme with those of some existing continuous metaheuristics and include a comparison among various versions of RCCRO in Section IV. We conclude this paper and suggest potential future work in Section V.

II. Framework of Chemical Reaction Optimization

A. Molecules

The manipulated agents are molecules. Each molecule contains a profile of several properties of the molecule, including: 1) the molecular structure ω; 2) (current) potential energy (PE); 3) (current) kinetic energy (KE); and 4) some optional attributes. The optional attributes can be used to construct other versions of CRO for particular problems provided that the implementations meet the characteristics of the elementary reactions (explained in Section II-B). Users can remove any of these or add some more to suit the designs of their own operators. The meanings of the necessary attributes in the profile are given as follows.

1) *Molecular Structure:* ω actually represents the solution currently held by a molecule. There is no specific requirement on the configuration of ω. Depending on the problem, it can be in the form of a number, a vector, a matrix, or even a graph.

2) *Current PE:* PE is the objective function value of the current molecular structure ω, i.e., $PE_\omega = f(\omega)$.

3) *Current KE:* KE can be thought of as the current tolerance for the molecule to hold a worse molecular structure with higher PE than the existing one.

B. Elementary Reactions

There are four types of elementary reactions, consisting of the: 1) on-wall ineffective collision; 2) decomposition; 3) intermolecular ineffective collision; and 4) synthesis. All of them are triggered by collisions and a successful completion of an elementary reaction (subject to the energy limitation) results in an internal change of a molecule (i.e., updated attributes in the profile). In other words, we explore the solution space through a sequence of elementary reactions. They have different characteristics and the details are shown in the following sections.

1) *On-Wall Ineffective Collision:* An on-wall ineffective collision takes place when a molecule hits a wall of the container and then bounces back. This reaction is not vigorous and there is only a small change to its molecular structure ω and PE. This can be thought as the molecule tries to transform ω to ω' in the neighborhood of ω, that is

$$\omega' = N(\omega)$$

where $N(\cdot)$ is the neighborhood search operator which returns a member from the neighborhood of the operand. Since this reaction involves an interaction with an external substance (i.e., a wall of the container), a certain portion of its KE will be extracted and stored in the central energy buffer (*buffer*) when the transformation is complete. The size of KE loss depends on a random number $a \in [KELossRate, 1]$, where *KELossRate* is a parameter of CRO. Its KE is updated with

$$KE_{\omega'} = (PE_\omega - PE_{\omega'} + KE_\omega) \times a.$$

It is possible for a molecule with lower PE to transform into one with higher PE, corresponding to a worse solution, provided it has enough KE to begin with. This transformation may be desirable to allow the algorithm to escape from a local minimum. After experiencing collision, the molecule has less KE. In this way, its tolerance of getting a worst solution is lower and its ability of escaping from local minima diminishes.

2) *Decomposition:* In a decomposition, a molecule ω hits a wall of the container and then breaks into two or more molecules (assume two, e.g., ω'_1 and ω'_2, in our implementation in this paper). When compared with the on-wall ineffective collision, the decomposition is more vigorous and the molecular structures of the resultant molecules have greater differences from that of the original one. This can be considered as the situation when we finish the local search in the region around ω and decide to explore other regions corresponding to ω'_1 and ω'_2. Due to the conservation of energy (explained in Section II-C), ω may sometimes not have enough energy (both PE and KE) to sustain its transformation into ω'_1 and ω'_2. A certain portion of energy in *buffer* accumulated from antecedent on-wall ineffective collisions can be utilized to support the change.

3) *Intermolecular Ineffective Collision:* An intermolecular ineffective collision occurs when two or more molecules collide with each other and then separate. The number of molecules involved in this collision subsystem remains unchanged after the collision. The more the number of molecules involved, the more energy in the subsystem, and thus, the more flexibility for the molecules to modify their molecular structures. In our implementation in the simulation, we assume only two molecules, e.g., with molecular structures ω_1 and ω_2, involved. Similar to the on-wall ineffective collision, this collision is also not vigorous and the new molecular structures ω'_1 and ω'_2 are produced from their own neighborhoods separately, that is

$$\omega'_1 = N(\omega_1) \quad \text{and} \quad \omega'_2 = N(\omega_2).$$

4) *Synthesis:* A synthesis refers to the situation when two or more molecules (also assume two with ω_1 and ω_2) collide and combine to form one new single molecule. The change is vigorous and the resultant molecular structure ω' is greatly different from ω_1 and ω_2. This implies that we give up the search regions around ω_1 and ω_2 and start the search again in a new territory of S.

C. Energy Handling

Energy handling is one of the unique features of CRO. Energy is allowed to transform from one type to another type but all energy manipulations must follow the conservation of energy, which states that energy can be neither created nor destroyed. Consider the general form of the elementary reaction as follows:

$$\omega_1 + \ldots + \omega_k \rightarrow \omega'_1 + \ldots + \omega'_l \tag{2}$$

where k and l are the numbers of molecules involved before and after a change, respectively. For example, $k = 1$ and $l = 2$ correspond to the implementation of decomposition in Section II-B2. The corresponding energy equation of (2) is

$$\underbrace{(PE_{\omega_1} + \ldots + PE_{\omega_k}) + (KE_{\omega_1} + \ldots + KE_{\omega_k}) + buffer}_{\text{before the change}}$$
$$= \underbrace{(PE_{\omega'_1} + \ldots + PE_{\omega'_l}) + (KE_{\omega'_1} + \ldots + KE_{\omega'_l}) + buffer'}_{\text{after the change}}. \tag{3}$$

With the conservation of energy, the total sum of energy before and after the change indicated by (2) is identical. Uni-molecular reactions involve a single molecule hitting a wall of the container, transferring some of its energy to the central energy buffer, and thus the uni-molecular reactions involve *buffer* and *buffer*'. However, the intermolecular reactions which take place in the interior of the container without hitting its walls do not involve *buffer* and *buffer*'. PE directly corresponds to a solution ω while KE and *buffer* indirectly place restrictions on a transition of a solution. To carry out an elementary reaction, we can first assume that new molecules can be formed by the operators of respective elementary reactions and compute their PE. Then we can determine if the existing molecules with molecular structures $\omega_1, \ldots, \omega_k$ are replaced by the new molecules with $\omega'_1, \ldots, \omega'_l$ by examining the general new solution acceptance rule (for simplicity, we do not consider *buffer* here) as follows:

$$\underbrace{\sum_{\omega} PE_{\omega} + \sum_{\omega} KE_{\omega}}_{\text{existing molecules}} - \underbrace{\sum_{\omega'} PE_{\omega'}}_{\text{new molecules}} \geq 0. \tag{4}$$

Consider a simple example with $k = l = 1$: $\omega \rightarrow \omega'$. Then we have the new solution acceptance rule: $PE_{\omega} + KE_{\omega} - PE_{\omega'} \geq 0$. If $PE_{\omega} \geq PE_{\omega'}$, ω' is always accepted. Otherwise, the acceptance depends on KE_{ω}. We can see that the function of KE is to give tolerance for the molecule to get a new molecular structure with higher PE.

The total energy of the whole system (PE and KE of the molecules and *buffer*) is the same at any instance. This total amount is determined by the initial (incontrollable) PE of the initial set of molecules, the (controllable) initial KE (*initialKE*)[2] assigned to the initial set of molecules, and the initial *buffer*.[3] This total affects the convergence speed and the possibility of getting the global minimum.

D. Algorithm

Imagine that we have a certain number of molecules in a container. They collide, interact, combine, decompose, and finally become stable in the end. The whole process is what we try to mimic with CRO. The flow chart of CRO can be found in [18] and it consists of three stages: initialization, iterations, and the final stage. In initialization, we configure the initial settings for the molecules and the parameters (i.e., *PopSize*, *KELossRate*, *MoleColl*, *buffer*, *InitialKE*, α, and β). We create the initial molecule set with size equal to *PopSize* by randomly generating solutions in the solution space. Their initial PEs are determined by their corresponding objective function values while their initial KEs are set to *InitialKE*. In each iteration, there is one elementary reaction taking place. We first determine whether it is a uni-molecular or intermolecular reactions by comparing a random number $b \in [0, 1]$ against *MoleColl*. If $b >$ *MoleColl* or there is only one molecule left, we will have a uni-molecular reaction. Otherwise, an intermolecular reaction happens. For each uni-molecular reaction,

[2]*initialKE* is a parameter of CRO.

[3]*buffer* is a parameter of CRO and its initial value is usually assigned to zero.

we randomly choose one molecule and check if it satisfies the decomposition criterion: (number of hits − minimum hit number) $> \alpha$, where α can be interpreted as the tolerance of duration for the molecule without obtaining any new local minimum solution. If so, the molecule will experience a decomposition, else it will take an on-wall ineffective collision. For each intermolecular reaction, two (or more) molecules are selected and they are tested against the synthesis criterion: ($KE \leq \beta$), where β can be considered as the minimum KE a molecule should have. If it is satisfied by all the selected molecules, they combine through synthesis. Otherwise, they experience an intermolecular ineffective collision. We can see that molecules with too little KE always get stuck in local minima and synthesis transforms the inactive molecules into an active one. Interested readers may refer to [18] which contains pseudocodes to facilitate the implementation.

III. Real-Coded Implementation

A. Modifications

Only three modifications are required to make CRO suitable for solving continuous optimization problems.

1) *Solution Representation:* Every solution s in a continuous search space is a real number vector, i.e., $s = [s(1), \ldots, s(i), \ldots, s(n)]$, where n is the dimension of the problem. $s(i)$ is usually a floating-point number in the range of $[l(i), u(i)]$, where $l(i)$ and $u(i)$ are the lower and upper bounds of the ith dimension, respectively. Thus, a molecular structure $\omega = [\omega(1), \ldots, \omega(i), \ldots, \omega(n)]$ should be able to carry such a solution representation and each $\omega(i)$ should be implemented with a floating-point type.

2) *Neighborhood Search Operator:* Besides the solution representation, the main concern for RCCRO is how to deal with the continuity of solutions. If we examine the search philosophy of CRO more carefully, the continuity does not influence diversification much but will affect intensification. Both elementary reactions for intensification, i.e., the on-wall and intermolecular ineffective collisions, pick a solution from the neighborhood of the existing one. Therefore, we only need to incorporate the continuity search ability into the neighborhood search operator to develop RCCRO.

Evolutionary programming or similar algorithms for real-valued continuous optimization add perturbations to existing solutions to generate new ones [26]. We adopt a similar approach to modify the neighborhood search operator $N(\cdot)$. Assume that the problem we are solving does not place any constraints on relating the solution variable of one dimension to that of another.[4] We can treat $\omega(i)$ independently, and thus, we have

$$\omega'(i) = N(\omega(i)) = \omega(i) + \delta(i) \tag{5}$$

where $\delta(i)$ is a perturbation for the ith dimension. There are many probability distributions on which $N(\cdot)$ can be based to produce probabilistic perturbations, e.g., Gaussian, Cauchy, Lévy, lognormal, exponential, Student's T [27], and others. In this paper, we employ the Gaussian distribution, which is the most commonly used distribution in the related field, to test CRO's ability to solve continuous optimization problems. Its probability density function is expressed as

$$f_{pdf}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{6}$$

where μ and σ^2 are the mean and variance, respectively. Let $\Delta(i)$ be a random variable to model $\delta(i)$, we have

$$\Delta(i) \sim \mathcal{N}(\mu, \sigma^2). \tag{7}$$

Moreover, a perturbation can be represented by three components, i.e., the starting point, the direction, and the step size. The starting point of δ from an existing solution ω is ω itself. If we do not have any additional information about the location of the global minimum, we normally have no preference for which direction to go for playing safe. Due to the symmetrical property of the Gaussian distribution, setting $\mu = 0$ will result in generating δ from ω in all directions with equal probability. Moreover, σ influences how wide (6) spreads from μ. The larger σ is, the higher the probability a larger δ is generated. Thus, we have σ controlling the step size. Hence, we have

$$\omega'(i) = \omega(i) + \mathcal{N}(0, \sigma^2). \tag{8}$$

The step size affects the performance of the algorithm. Depending on the characteristics of the problem, different values of σ should be adopted. Therefore, σ becomes a parameter of RCCRO. In the rest of this paper, σ or *StepSize* will be used interchangeably. This neighborhood search operator is employed for on-wall and intermolecular ineffective collisions (and the decomposition function in the simulation in Section IV). Readers may refer to the pseudocodes in [18] to build the two ineffective collisions with this Gaussian neighborhood search operator being substituted.

3) *Boundary Constraint Handling:* A real-valued continuous optimization problem is usually bounded. In other words, partial solution s_i can only take a value in the interval $[l_i, u_i]$. However, the Gaussian neighborhood search operator introduced in Section III-A2 may sometimes take a molecule out of the boundaries. There are many ways to handle the boundary constraints and the most popular ones are the random approach, the absorbing approach, and the reflecting approach [28]. In this paper, we adopt two schemes for CRO to handle the boundary constraints. The first one is the reflecting scheme (RS) used in [29] which treats a boundary as a mirror and reflects ω back by the same amount of violation from the boundary. We produce a boundary-constraint-violation-free ω' from ω by

$$\omega'(i) = \begin{cases} 2 \times l(i) - \omega(i), & \text{if} \omega(i) < l(i) \\ 2 \times u(i) - \omega(i), & \text{if} \omega(i) > u(i). \end{cases} \tag{9}$$

RS usually results in solutions away from the boundaries and it may have worse performance when the optimal solutions reside on the boundaries. Therefore, we also implement another scheme called hybrid scheme (HS) which combines

[4]This assumption is valid for all the benchmark problems tested in Section IV.

the absorbing and reflecting approaches [30]. We produce ω' by

$$\omega'(i) = \begin{cases} l(i), & \text{if } (t \leq 0.5) \text{ AND } (\omega(i) < l(i)) \\ u(i), & \text{if } (t \leq 0.5) \text{ AND } (\omega(i) > u(i)) \\ 2 \times l(i) - \omega(i), & \text{if } (t > 0.5) \text{ AND } (\omega(i) < l(i)) \\ 2 \times u(i) - \omega(i), & \text{if } (t > 0.5) \text{ AND } (\omega(i) > u(i)) \end{cases} \tag{10}$$

where t is a random number drawn from [0, 1]. In this case, we will not omit solutions along the boundaries. We will evaluate the performance of CRO with these two schemes in Section IV. There are other sophisticated methods for boundary constraint handling, e.g., periodic mode in [31], boundary search in [32], and others. We leave the dedicated boundary constraint study for RCCRO for future research.

B. Other Implementation Details

For completeness, the following explains some implementation details of RCCRO in the simulation in Section IV.

1) *Decomposition Operator:* We apply "half-total-change" used to solve channel assignment problem in [18] to our implementation of RCCRO. We add perturbations to $\frac{n}{2}$ variables of the original solution to create new solutions, where n is the dimension of the problem. The following pseudocode illustrates how to produce two new solutions from an existing one, i.e., line 1 of pseudocode "decompose(M, *buffer*)" in [18].

```
gen_2_new_molecules($M$)
Input: a solution $\omega$.
1.  Duplicate $\omega$ to produce $\omega'_1$ and $\omega'_2$
2.  for change = 1 to $\frac{n}{2}$ do
3.      Get $i$ and $j$ randomly in the set $\{1, \ldots, n\}$
4.      Add random perturbations to $\omega'_1(i)$ and $\omega'_2(j)$
5.  end for
6.  Output $\omega'_1$ and $\omega'_2$
```

2) *Synthesis Operator:* Similar to decomposition, we apply "probabilistic select" used in [18] to implement synthesis for the simulation in Section IV. We try to combine two solutions ω_1 and ω_2 into a new one ω'. We assign each component of ω' with the one in the same position of either ω_1 or ω_2 randomly. The following pseudocode demonstrates line 1 of pseudocode "synthesis(M_1, M_2) in [18].

```
combine_2_molecules($M$)
Input: solutions $\omega_1$ and $\omega_2$.
1.  for $i$ = 1 to $n$ do
2.      Get $t$ randomly in [0, 1]
3.      if $t > 0.5$ then
4.          $\omega'(i)$ is set to $\omega_1(i)$
5.      else
6.          $\omega'(i)$ is set to $\omega_2(i)$
7.      end if
8.  Output $\omega'$
```

Recall that synthesis aims at combining two (or more) molecules into one. The effect is similar to the result of a recombination (crossover) operator used in many other evolutionary algorithms. In general, a crossover is a mechanism for different agents (e.g., chromosomes in GA) to share information and to produce new solutions by inheriting their features. In [33], a taxonomy of many effective recombination operators for real parameter optimization is given. [34] also states that hybrid crossover operators can improve the performance. However, the main purpose of synthesis is to accumulate the energy from some energy-deficient molecules (see the definition of the synthesis criterion in Section II-D) so that the resultant molecule has sufficient energy to explore other search regions after a synthesis. In other words, synthesis implements exploration (i.e., diversification). A molecule which lacks energy will get stuck in a local minimum and synthesis is a way to produce a molecule with enough energy by combination. Although a new "offspring" solution is created by using the features from its "parent" molecules, the primary concern is the matter of energy. On the contrary, the focus of a recombination operator in general is on exploitation (i.e., intensification). Moreover, synthesis is triggered less frequently (only when energy-deficient molecules meet) when compared to a crossover used in a general evolutionary algorithms (in which crossover is called many times in each generation). These are the main differences between synthesis in RCCRO and recombination in other evolutionary algorithms.

BLX-α is one of the best crossover operators tested in [33] and it was proved to enhance diversity when α is larger than $(\sqrt{3} - 1)/2$ [35]. In order to test if a recombination operator will help, we also implement BLX-0.5 and compare with our basic scheme "probabilistic select" in Section IV.[5]

3) *Negative Objective Function Value Handling:* In CRO, objective function values are modeled as energy. The former can be negative in some problems[6] while the latter must be non-negative. To ease this contradiction, we add an offset to the objective function to make all possible objective function values non-negative, i.e., $f' = f + \textit{offset} \geq 0$. It is always possible as there must exist a positive number $\xi \geq |\min f|$. *Offset* can be any ξ. As the global minimums of most continuous benchmark problems are known, the easiest way is to set *offset* equal to the global minimum. In our simulation in Section IV, *offset* is applied and equal to the absolute value of the global minimum when the global minimum is known to be negative. In practice, when the global minimum of a problem is not known in advance, one can assign *offset* with a reasonable large positive value. For example, after a few evaluations of the problem (i.e., obtaining several objective function values with some random feasible solutions), we get an objective function value $\overline{f}$ of a feasible solution (e.g., the maximum one of the random solutions). Normally, assigning a value equal to $1000 \times |\overline{f}|$ to *offset* is enough.

In fact, the above manipulation is to make "energy" meaningful as negative energy is not realistic in a chemical reaction. Besides, we can handle negative objective values by the "programming" approach. If we inspect (4) carefully, it can

[5]In [33]'s terminology, "probabilistic select" and BLXα are classified as a discrete crossover operator and a neighborhood-based crossover operator, respectively.

[6]Some benchmark problems used in Section IV can take negative values.

be rearranged into

$$\left(\sum_{\omega} PE_{\omega} - \sum_{\omega'} PE_{\omega'}\right) + \sum_{\omega} KE_{\omega} \geq 0.$$

It means that a change is accepted provided that the difference of PE before and after an elementary reaction together with the KE before the reaction is nonnegative. For the PE terms in the bracket, we only examine the difference of energy. In other words, negativity of PE does not affect (4). When programming RCCRO (or CRO in general), one can allow negative PE (i.e., negative objective function values)[7] and handle the energy difference in (4).

C. Adaptive Scheme

Algorithm parameter adjustment is crucial to the success of an algorithm in solving a problem. It may take a significant effort to find a combination of parameter values for an algorithm with satisfactory performance. The whole process from choosing an algorithm to successfully obtaining a good solution to a problem will become more efficient if the algorithm can adapt to the problem by self-adapting its parameters to fit the problem. Research on adaptation mechanisms has been flourishing since 1967, and [36] is one of the earliest work in this area. One of the most intensely studied parameters is the mutative step size, see σ in (7). Here, we propose the first adaptive scheme for RCCRO by adapting σ to a problem.

In the original proposal of RCCRO (i.e., the basic scheme in Section IV-D), σ is a fixed value during the whole course of RCCRO and it is problem-dependent (i.e., it should be set to different values for different problems to get the best performance). In Section IV, we try to fix σ for each category of the problems. However, if σ is too large for a particular problem, the algorithm may not investigate each region thoroughly (due to ineffective collisions), fail to locate the minimum of the region, and jump to other regions rashly. If σ is too small, the algorithm will become highly inefficient. To adapt to a problem, we try to assign the initial value of σ with a value equal to the range of the solution space of the problem (i.e., $u - l$ in Section III-A1). This ensures that the algorithm will not be too inefficient and too randomized to shuttle around the search space. When the algorithm is running, we try to refine the perturbations so that it searches each region in more details gradually. To do this, in each fixed interval (Δ), we decrease σ by a fixed factor (θ), i.e., $\sigma \leftarrow \sigma \times \theta$. This logarithmic decrease can avoid overly aggressive refinements.

Recall that the purpose of an adaptive scheme is to avoid explicit parameter tuning so that the effort spent on parameter tuning of an algorithm can be reduced. In the above scheme, we try to reduce the effort to tune σ, but it seems that we introduce two more parameters (Δ and θ) while "removing" σ. However, Δ and θ are generally fixed for all problems while σ needs to be tuned for different problems. We actually reduce the exertion of parameter tuning for RCCRO.

[7]Note that KE should still be nonnegative. Since KE is an artificial term of CRO, independent of problems, we still insist on nonnegativity of energy for KE.

IV. Simulation Results

A. Benchmark Functions

In order to have a comprehensive evaluation of RCCRO on problems in the continuous domain, we test the performance of RCCRO with a large set of standard benchmarks used in [37]. These benchmarks are listed in Table I which contains their dimension sizes, their feasible solution space S, and the objective function values of their global minimums f_{min}.

There are 23 benchmark functions in total, classified into three categories according to their characteristics.

1) *Unimodal Functions*: This group consists of functions f_1–f_7 and they are high-dimensional. There is only one global minimum in each of the functions. They are relatively "easy" to solve when compared with those in the next group.
2) *High-Dimensional Multimodal Functions*: This group is composed of functions f_8–f_{13}. They are high-dimensional and contain many local minimums. They are considered as the most difficult problems in the benchmark set.
3) *Low-Dimensional Multimodal Functions*: This group includes functions f_{14}–f_{23}. They have lower dimensions and have fewer local minimums than the previous group.

B. Experimental Setting

All simulations are performed on the same personal computer with Intel Core Quad 2.66 GHz CPU and 4 GB of RAM. RCCRO is implemented with CROToolbox [38] available at [39] in Windows 7 environment.

C. Parameter Tuning

Parameter settings affect an algorithm's performance. Without suitable parameters, the algorithm may result in bad simulation results. Recall that there are eight parameters in RCCRO (i.e. *PopSize*, *StepSize*, *buffer*, *InitialKE*, *MoleColl*, *KELossRate*, α, and β). A complete evaluation on all possible combinations of the parameters is impractical.

Our goal is to assign parameter values to RCCRO with relatively good performance for the 23 benchmark functions. Each function has its own characteristics and a single parameter value combination is hardly suitable for every function. However, configuring a good combination for each function is too purposive and may lead to unfair comparison with the counterparts in the next sections. As the benchmark set is generally divided into three categories (i.e., unimodal, high-dimensional multimodal, and low-dimensional multimodal), we try to determine a parameter value combination generally suitable for each category. As in [37], we select some functions out of each categories as representatives: f_1 and f_2 for Category I, f_{10} and f_{11} for Category II, f_{21}, f_{22}, and f_{23} for Category III. We will use these representatives for parameter tuning.

We tune the parameters in an ad hoc manner. We test the performance with certain parameter value combinations and and each simulation run terminates when a certain number of function evaluations (FEs) have been reached. The FE limits of RCCRO for different functions are listed in Table II. For each category, we first determine the "initial" combination

TABLE I
23 BENCHMARK FUNCTIONS

Category	Test Function	Name	n	S	f_{min}
I	$f_1(x) = \sum_{i=1}^{n} x_i^2$	Sphere model	30	$[-100, 100]^n$	0
	$f_2(x) = \sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$	Schwefel's problem 2.22	30	$[-10, 10]^n$	0
	$f_3(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$	Schwefel's problem 1.2	30	$[-100, 100]^n$	0
	$f_4(x) = \max_i\{\lvert x_i\rvert, 1 \le i \le n\}$	Schwefel's problem 2.21	30	$[-100, 100]^n$	0
	$f_5(x) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	Generalized Rosenbrock's function	30	$[-30, 30]^n$	0
	$f_6(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5\rfloor)^2$	Step function	30	$[-100, 100]^n$	0
	$f_7(x) = \sum_{i=1}^{n} ix_i^4 + \text{random}[0, 1)$	Quartic function with noise	30	$[-1.28, 1.28]^n$	0
II	$f_8(x) = -\sum_{i=1}^{n} \left(x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)\right)$	Generalized Schwefel's problem 2.26	30	$[-500, 500]^n$	−12569.5
	$f_9(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	Generalized Rastrigin's function	30	$[-5.12, 5.12]^n$	0
	$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e$	Ackley's function	30	$[-32, 32]^n$	0
	$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	Generalized Griewank function	30	$[-600, 600]^n$	0
	$f_{12}(x) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} + \sum_{i=1}^{30} u(x_i, 10, 100, 4)$; $y_i = 1 + \frac{1}{4}(x_i + 1)$; $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$	Generalized penalized functions	30	$[-50, 50]^n$	0
	$f_{13}(x) = 0.1\left\{\sin^2(\pi 3x_1) + \sum_{i=1}^{29} (x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + \sin^2(2\pi x_{30})]\right\} + \sum_{i=1}^{30} u(x_i, 5, 100, 4)$		30	$[-50, 50]^n$	0
III	$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}\right]^{-1}$	Shekel's Foxholes function	2	$[-65.536, 65.536]^n$	1
	$f_{15}(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	Kowalik's function	4	$[-5, 5]^n$	0.0003075
	$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	Six-hump camel-back function	2	$[-5, 5]^n$	−1.0316285
	$f_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	Branin function	2	$[-5, 10] \times [0, 15]$	0.398
	$f_{18}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	Goldstein-Price function	2	$[-2, 2]^n$	3
	$f_{19}(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{4} a_{ij}(x_j - p_{ij})^2\right]$	Hartman's family	3	$[0, 1]^n$	−3.86
	$f_{20}(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right]$		6	$[0, 1]^n$	−3.32
	$f_{21}(x) = -\sum_{i=1}^{5} [(x - a_i)(x - a_i)^T + c_i]^{-1}$	Shekel's family	4	$[0, 10]^n$	−10
	$f_{22}(x) = -\sum_{i=1}^{7} [(x - a_i)(x - a_i)^T + c_i]^{-1}$		4	$[0, 10]^n$	−10
	$f_{23}(x) = -\sum_{i=1}^{10} [(x - a_i)(x - a_i)^T + c_i]^{-1}$		4	$[0, 10]^n$	−10

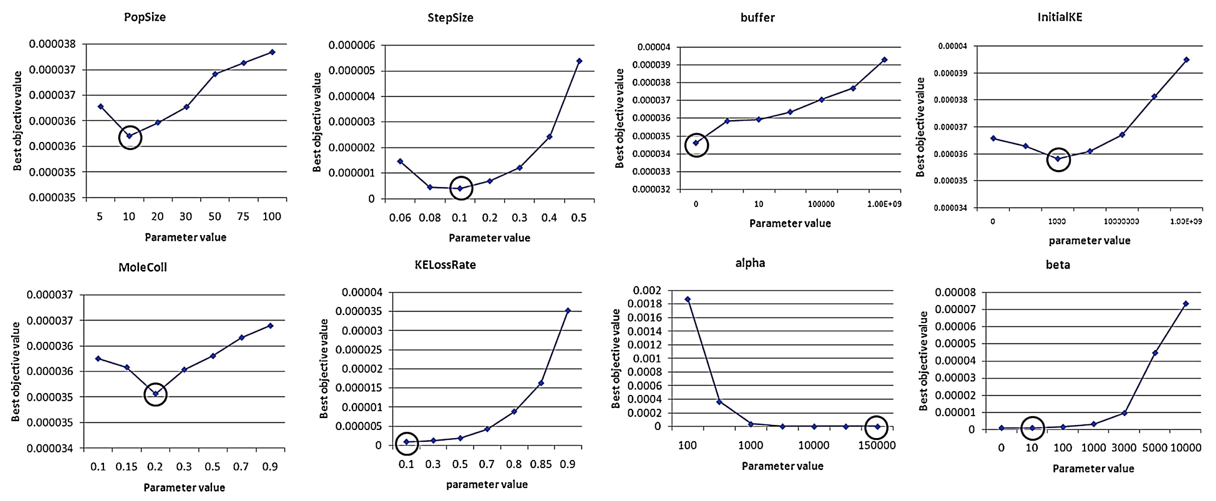

Fig. 1. Parameter tuning for f_1. (a) *PopSize*. (b) *StepSize*. (c) *buffer*. (d) *InitialKE*. (e) *MoleColl*. (f) *KELossRate*. (g) α. (h) β.

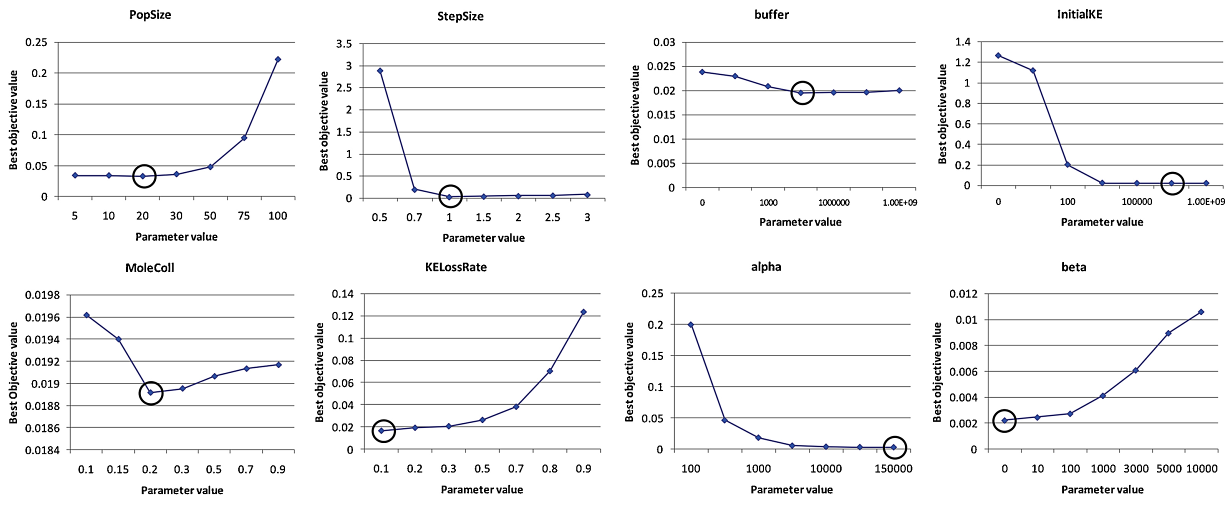

Fig. 2. Parameter tuning for f_{10}. (a) *PopSize*. (b) *StepSize*. (c) *buffer*. (d) *InitialKE*. (e) *MoleColl*. (f) *KELossRate*. (g) α. (h) β.

for tuning (shown in Table III) by running RCCRO on the representative functions with random parameter values for a few times. Then, based on the "initial" parameter values, we adjust the parameters, one at a time, in the order given in Table III. For each parameter, we select some values for testing and we perform 100 runs for each of the chosen values. We compare the averages of the sets of 100 runs among the chosen values and select the one with the smallest averaged objective values. We run the tests for all the representative functions. To enhance readability, we only provide the averages of the parameter values for f_1, f_{10}, and f_{21}, one for each function category. They are shown in Figs. 1–3 and the best values are circled. After confirming the appropriate value for a particular parameter, we replace the corresponding one in the initial combination with the new value and then we proceed to the next parameters with the amended combination. For example, we tune the parameters with f_{21} (see Fig. 3). The initial combination for Category III is $[25, 2, 10^8, 10^6, 0.2, 0.2, 1000, 0]$ (see Table III). With other parameters fixed, we vary the values of *PopSize* and the best value is found to be 100, and then we update the parameter combination to $[\underline{100}, 2, 10^8, 10^6, 0.2, 0.2, 1000, 0]$. Next we utilize the new combination to tune *StepSize*. After tuning, the next combination becomes $[100, \underline{0.5}, 10^8, 10^6, 0.2, 0.2, 1000, 0]$. The process continues until we have updated β. Note that the above procedure does not guarantee that the tuned parameter combination maximizes the performance for solving a particular function unless we try all possible combinations. We aim to determine a good combination so that RCCRO can sustain satisfactory performance.

After the tuning process, we obtain a parameter value combination for each representative function, listed in Table III. Then we need to decide a final combination for each function category. Sometimes there are some discrepancies between the tuned combinations for the representative functions in a category. For example, we get *StepSize* equal to 0.1 and 0.01 for f_1 and f_2 in Category I, respectively. We choose the one for the smaller function index, and in this example, we pick 0.1. In this way, we fix a final combination for each category. Moreover, since the ranges of the solution spaces for f_8 and f_{11} are exceptionally larger than the rest in the same category, we keep *StepSize* equal to 300 and 15 for f_8

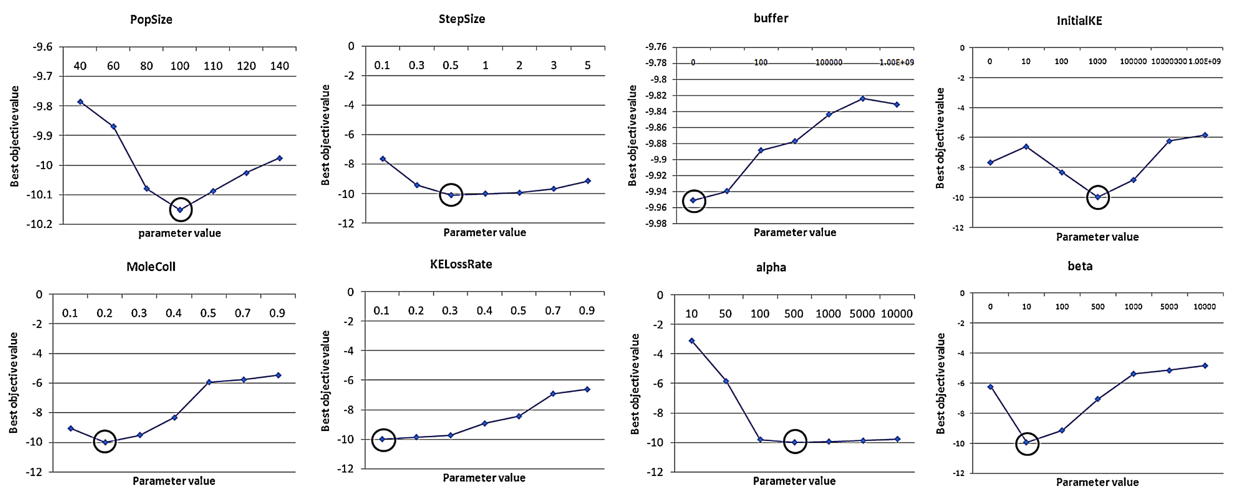

Fig. 3. Parameter tuning for f_{21}. (a) *PopSize*. (b) *StepSize*. (c) *buffer*. (d) *InitialKE*. (e) *MoleColl*. (f) *KELossRate*. (g) α. (h) β.

TABLE II
NUMBER OF FEs FOR FUNCTIONS f_1–f_{23}

Function	RCCRO	GA	FEP	CEP	FES	CES	PSO	GSO	RCBBO	DE	CMAES	G3PCX
f_1	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000
f_2	150 000	150 000	200 000	200 000	200 000	200 000	150 000	150 000	200 000	150 000	150 000	150 000
f_3	250 000	250 000	500 000	500 000	500 000	500 000	250 000	250 000	500 000	250 000	250 000	250 000
f_4	150 000	150 000	500 000	500 000	500 000	500 000	150 000	150 000	500 000	150 000	150 000	150 000
f_5	150 000	150 000	2 000 000	2 000 000	2 000 000	2 000 000	150 000	150 000	500 000	150 000	150 000	150 000
f_6	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000
f_7	150 000	150 000	300 000	300 000	300 000	300 000	150 000	150 000	300 000	150 000	150 000	150 000
f_8	150 000	150 000	900 000	900 000	900 000	900 000	150 000	150 000	300 000	150 000	150 000	150 000
f_9	250 000	250 000	500 000	500 000	500 000	500 000	250 000	250 000	300 000	250 000	250 000	250 000
f_{10}	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000
f_{11}	150 000	150 000	200 000	200 000	200 000	200 000	150 000	150 000	300 000	150 000	150 000	150 000
f_{12}	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000
f_{13}	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000	150 000
f_{14}	7500	7500	10 000	10 000	10 000	10 000	7500	7500	10 000	7500	7500	7500
f_{15}	250 000	250 000	400 000	400 000	400 000	400 000	250 000	250 000	100 000	250 000	250 000	250 000
f_{16}	1250	1250	10 000	10 000	10 000	10 000	1250	1250	10 000	1250	1250	1250
f_{17}	5000	5000	10 000	10 000	10 000	10 000	5000	5000	10 000	5000	5000	5000
f_{18}	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000
f_{19}	4000	4000	10 000	10 000	10 000	10 000	4000	4000	10 000	4000	4000	4000
f_{20}	7500	7500	20 000	20 000	20 000	20 000	7500	7500	20 000	7500	7500	7500
f_{21}	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000
f_{22}	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000
f_{23}	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000	10 000

and f_{11}, respectively, but *StepSize* for the other functions in Category II are still set to 1.

Note that the above procedures for parameter tuning does not guarantee the tuned parameter combinations will maximize RCCRO's performance in solving the test functions. Here, we try to give a preliminary empirical study which may serve as a guide to suggest parameter values for RCCRO when it is applied to other problems similar to the test functions. To have a better analysis on the choice of parameter values for RCCRO, we can apply the relevant estimation and value calibration method [40], [41] but we leave this detailed study for future research.

D. Comparisons

In Section III, we have two approaches to handle the boundary constraints, i.e., (9) and (10), and two approaches for synthesis, i.e., "probabilistic select" and "BLX-0.5." Moreover, we try to make RCCRO adaptive. Thus, we have several versions of RCCRO for comparison.

1) *RCCRO1*: We utilize the neighborhood search operator (8), boundary constraint handling (9), decomposition operator "half-total-change," and synthesis operator "probabilistic select." We also call this version the "basic scheme."
2) *RCCRO2*: This version is identical to the basic scheme, except that the boundary constraint handling is replaced by (10).
3) *RCCRO3*: This version is identical to the basic scheme, except that the synthesis operator is replaced by BLX-0.5.
4) *RCCRO4*: This is the adaptive scheme discussed in Section III-C with Δ and θ set to 100 FE and 0.99, respectively.

We will first compare the basic scheme with other representative evolutionary algorithms. Then we will study if

TABLE III
PARAMETER TUNING FOR THE REPRESENTATIVE FUNCTIONS IN EACH CATEGORY

		Category I				Category II				Category III				
Order	Parameter	Initial	f_1	f_2	Final	Initial	f_{10}	f_{11}	Final	Initial	f_{21}	f_{22}	f_{23}	Final
1	*PopSize*	10	10	10	10	10	20	20	20	25	100	100	100	100
2	*StepSize*	0.1	0.1	0.01	0.1	1.5	1	15	1[a]	2	0.5	0.5	0.5	0.5
3	*buffer*	1E+6	0	0	0	1E+8	1E+5	1000	1E+5	1E+8	0	0	0	0
4	*InitialKE*	1000	1000	1000	1000	1E+6	1E+07	1000	1E+7	1E+6	1000	1000	1000	1000
5	*MoleColl*	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
6	*KELossRate*	0.9	0.1	0.1	0.1	0	0.1	0.1	0.1	0.2	0.1	0.1	0.2	0.1
7	α	1.5E+5	1.5E+5	1.5E+5	1.5E+5	2000	1.5E+5	1.5E+5	1.5E+5	1000	500	500	500	500
8	β	0	10	10	10	0	0	10	10	0	10	10	10	10

[a] 300 for f_8 and 15 for f_{11}.

TABLE IV
AVERAGE COMPUTATION TIME

Function	Time (s)
f_1	0.063
f_2	0.068
f_3	0.095
f_4	0.078
f_5	0.123
f_6	0.165
f_7	0.088
f_8	0.533
f_9	0.443
f_{10}	0.335
f_{11}	0.553
f_{12}	0.295
f_{13}	0.308
f_{14}	0.008
f_{15}	0.152
f_{16}	0.005
f_{17}	0.005
f_{18}	0.007
f_{19}	0.009
f_{20}	0.013
f_{21}	0.006
f_{22}	0.006
f_{23}	0.007

the modifications to the basic scheme help improve the performance.

1) *Basic Scheme and Other Algorithms:* We perform simulations on the benchmark functions with RCCRO1 and compare the results with GA, fast evolutionary programming (FEP) [37], classical evolutionary programming (CEP) [37], fast evolutionary strategy (FES) [42], conventional evolutionary strategy (CES) [42], PSO, group search optimizer (GSO) [43], real-coded biogeography-based optimization (RCBBO) [44], differential evolution (DE) [45], [46], covariance matrix adaptation evolution strategy (CMAES) [47], and generalized generation gap model with generic parent-centric recombination operator (G3PCX) [48]. Evolutionary programming [26] and evolutionary strategies [49] are two branches of evolutionary algorithms. CEP and CES are their canonical implementations while FEP and FES are their improved variants, respectively. RCBBO is an improved version of biogeography-based optimization [50] on solving problems in the continuous domain. DE is usually considered an improved version of GA and it generates offsprings by perturbing the candidate solutions with the scaled differences of some random chosen ones. DE has been applied to solve many real-world problems [46] and a state-of-the-art survey can be found at [51]. CMAES is a variant of ES, implemented with an adaption of the covariance matrix to model a second-order approximation of the objective function. G3PCX incorporates an elite-preserving and scalable model and the parent-centric recombination operator with GA. CMAES and G3PCX are very competitive with the classical optimization schemes and they are considered as benchmark algorithms for real-valued optimization.

Besides RCCRO1, DE, CMAES, and G3PCX, all data are adopted from published results where those of FEP and CEP are from [37], those of FES and CES are from [42], those of GA, PSO, and GSO are from [43], and those of RCBBO are from [44]. We implement DE, CMAES, and G3PCX with the source codes available at [46], [52], and [53], respectively. Their parameter values are selected according to the recommendations of their respective authors: for DE, population size $NP = 7 \times n$, weighting factor $F = 0.5$, and crossover constant $CR = 0.1$ [45], [46]; for CMAES, initial point $x^{(0)}$ set to a random point in the search region, initial step size $\sigma^{(0)}$ set to one third of the search region, population size $\lambda = 4 + \lfloor 3\ln(n) \rfloor$ and parent number $\mu = \lfloor \lambda/2 \rfloor$ [47]; and for G3PCX, population size $N = 100$, variances for PCX $\sigma_\zeta = \sigma_\eta = 0.1$ [48]. The FE limits of all algorithms for all functions are also shown in Table II. Note that RCCRO is evaluated with the least FE in each function while the results for some other algorithms are obtained with more FE (because the data are acquired from the literature. For each function, we run RCCRO 100 times and obtain the averaged computed minimum value (Mean) and standard deviation (StdDev). We compare the performance among the algorithms according to categories of the benchmark functions. For completeness, we also give the averaged computation times in Table IV. The results are discussed according to the function categories as follows.

a) *Unimodal functions*: Table V gives the results for the unimodal functions (Category I). As in [43], we first rank the algorithms from the lowest Mean to the highest. Then we average the ranks over the seven functions and obtain the average rank. Finally, we order the average rank and get the overall rank. According to the overall rank, RCCRO1 outperforms the rest of the algorithms. In general, RCCRO1 is efficient in solving high-dimensional unimodal functions.

b) *High-dimensional multimodal functions*: Table VI gives the results for high-dimensional multimodal functions (Category II). RCCRO1 gives poorer results on solving f_{12}. However, it performs best on f_{13} and ranks second

TABLE V
SIMULATION RESULTS FOR f_1–f_7

		RCCRO1	GA	FEP	CEP	FES	CES	PSO	GSO	RCBBO	DE	CMAES	G3PCX
f_1	Mean	6.427E−07	3.171E+00	5.700E−04	2.200E−04	2.500E−04	3.400E−05	3.693E−37	1.948E−08	1.390E−03	6.576E−06	6.093E−29	6.404E−79
	StdDev	2.099E−07	1.662E+00	1.300E−04	5.900E−04	6.800E−04	8.600E−06	2.460E−36	1.163E−08	5.500E−04	1.132E−06	1.554E−29	1.248E−78
	Rank	5	12	10	8	9	7	2	4	11	6	3	1
f_2	Mean	2.196E−03	5.771E−01	8.100E−03	2.600E−03	6.000E−02	2.100E−02	2.917E−24	3.704E−05	7.990E−02	2.894E−04	3.480E−14	2.803E+01
	StdDev	4.341E−04	1.306E−01	7.700E−04	1.700E−04	9.600E−03	2.200E−03	1.136E−23	8.619E−05	1.440E−02	2.518E−05	4.034E−15	1.012E+01
	Rank	5	11	7	6	9	8	1	3	10	4	2	12
f_3	Mean	2.966E−07	9.750E+03	1.600E−02	5.000E−02	1.400E−03	1.300E−04	1.198E−03	5.783E+00	2.270E+01	1.212E+04	1.511E−26	1.064E−76
	StdDev	1.146E−07	2.595E+03	1.400E−02	6.600E−02	5.300E−04	8.500E−05	2.111E−03	3.681E+00	1.030E+01	1.554E+03	3.644E−27	1.532E−76
	Rank	3	11	7	8	6	4	5	9	10	12	2	1
f_4	Mean	9.318E−03	7.961E+00	3.000E−01	2.000E+00	5.500E−03	3.500E−01	4.123E−01	1.078E−01	3.090E−02	5.790E+00	3.994E−15	4.543E+01
	StdDev	3.657E−03	1.506E+00	5.000E−01	1.200E+00	6.500E−04	4.200E−01	2.500E−01	3.998E−02	7.270E−03	4.559E−01	5.311E−16	8.092E+00
	Rank	3	11	6	9	2	7	8	5	4	10	1	12
f_5	Mean	2.706E+01	3.386E+02	5.060E+00	6.170E+00	3.328E+01	6.690E+00	3.736E+01	4.984E+01	5.540E+01	9.338E+01	5.581E−01	3.091E+00
	StdDev	3.427E+01	3.615E+02	5.870E+00	1.361E+01	4.313E+01	1.445E+01	3.214E+01	3.018E+01	3.520E+01	1.734E+01	1.390E+00	1.639E+01
	Rank	6	12	3	4	7	5	8	9	10	11	1	2
f_6	Mean	0.000E+00	3.697E+00	0.000E+00	5.778E+02	0.000E+00	4.112E+02	1.460E−01	1.600E−02	0.000E+00	0.000E+00	7.000E−02	9.462E+01
	StdDev	0.000E+00	1.952E+00	0.000E+00	1.126E+03	0.000E+00	6.954E+02	4.182E−01	1.333E−01	0.000E+00	0.000E+00	2.932E−01	5.969E+01
	Rank	1	9	1	12	1	11	8	6	1	1	7	10
f_7	Mean	5.405E−03	1.045E−01	7.600E−03	1.800E−02	1.200E−02	3.000E−02	9.902E−03	7.377E−02	1.750E−02	3.967E−02	2.209E−01	9.797E−01
	StdDev	2.985E−03	3.622E−02	2.600E−03	6.400E−03	5.800E−03	1.500E−02	3.538E−02	9.256E−02	6.430E−03	7.832E−03	8.653E−02	4.627E−01
	Rank	1	10	2	6	4	7	3	9	5	8	11	12
Average rank		3.429	10.857	5.143	7.571	5.429	7.000	5.000	6.429	7.286	7.429	3.857	7.143
Overall rank		**1**	**12**	**4**	**11**	**5**	**7**	**3**	**6**	**9**	**10**	**2**	**8**

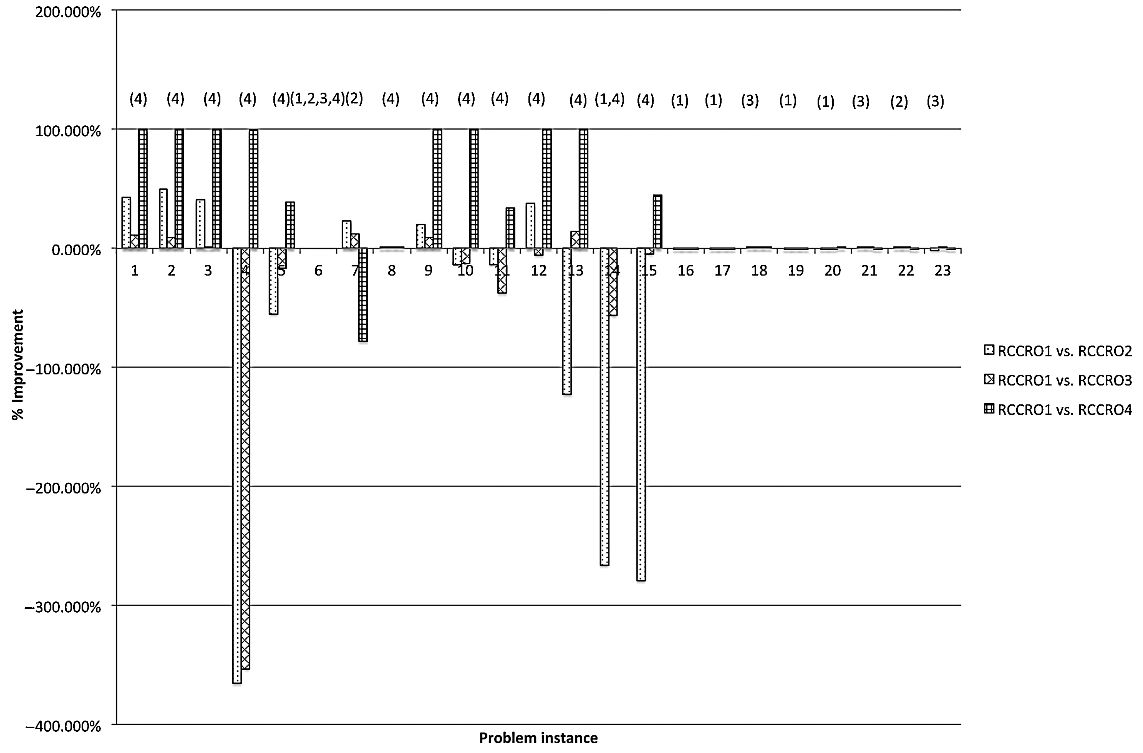

Fig. 4. % Improvement of RCCRO2, RCCRO3, and RCCRO4 over RCCRO1.

on f_8, f_9. As a whole, RCCRO1 ranks second in this category.

c) *Low-dimensional multimodal functions*: Table VII shows the comparisons for the low-dimensional multimodal functions (Category III). RCCRO1 performs particular well on f_{16}, the Hartman's family (f_{19} and f_{20}), and the Shekel's family (f_{21}–f_{23}). The reason may be due to the fact that we utilize these functions to tune the parameter values for the whole function category. As a whole, RCCRO1 is superior to other algorithms according to the overall rank given in Table VII.

2) *Various Versions of RCCRO:* We compare RCCRO2, RCCRO3, and RCCRO4 with the basic scheme. Our purpose is to study if some sophisticated operators designed for other evolutionary algorithms and the adaptive scheme work with RCCRO. We compare the results in terms of percentage improvement (%Improvement) defined as

$$\%Improvement = \frac{Result_{RCCRO1} - Result_{\xi}}{|Result_{RCCRO1}|} \tag{11}$$

where ξ is RCCRO2, RCCRO3, or RCCRO4. If ξ has better performance than the basic scheme, %Improvement will be

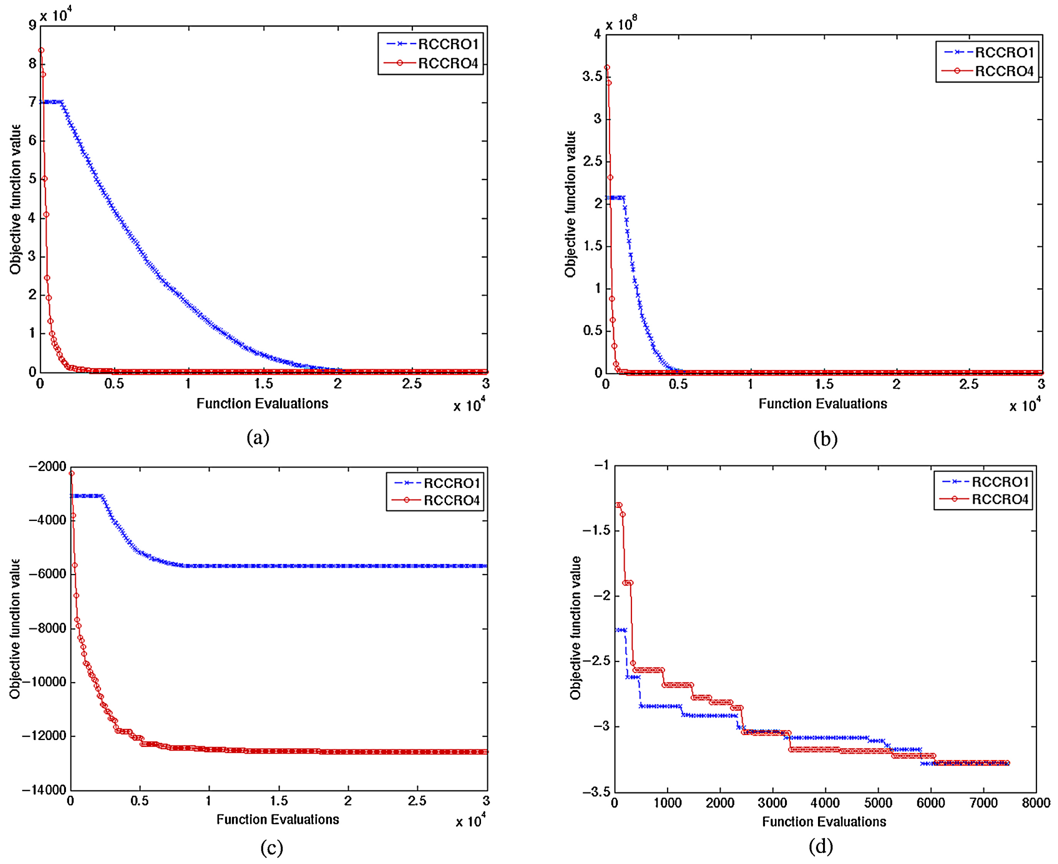

Fig. 5. Convergence curves of RCCRO1 and RCCRO4 for (a) f_1, (b) f_5, (c) f_8, and (d) f_20.

positive. Otherwise, it is negative. The results are given in Fig. 4. The numbers in brackets shown above the bars indicate which versions give the best performance on the problem instances. We have several observations as follows.

a) There is probably no general scheme which can work best on all the functions. As shown in Fig. 4, each scheme can outperform the others on certain functions: RCCRO1 performs best on f_{16}, f_{17}, f_{19}, and f_{20}; RCCRO2 on f_7 and f_{22}; RCCRO3 on f_{18}, f_{21}, and f_{23}; and RCCRO4 on f_1–f_5, f_8–f_{13}, and f_{15}.
b) Changing operators does not have a significant effect on some of the functions, i.e., f_6, f_8, f_{16}–f_{23}. It may be due to the effect of CRO's fundamental characteristics (i.e., the energy handling mechanisms) and the unaltered operators (e.g., the "half-total-change" operator used in decomposition) dominate the new operators for these functions.
c) Without considering the adaptive scheme, besides the exceptional case on f_4, HS (10) is preferable on unimodal functions (Category I) while RS (9) works better on multimodal functions (Categories II and III). For HS, there is a higher probability of searching the boundary regions and thus it searches through the whole solution space more thoroughly. In the cases under Category I, the functions are simpler and algorithms converge to global minimums faster. Thus, HS with more thorough search may give better results. For RS, there is a higher probability of searching in the interior of the solution space. For those cases whose global minimums are not on the boundaries or the problems are more complicated so that algorithms tend to getting stuck in local minimums (e.g., functions under Categories II and III), focusing on the interior may give better performance.
d) In general, RCCRO1 and RCCRO3 have similar performance. In other words, RCCRO is invariant to the choice of "recombination" operators for synthesis. Recall that synthesis focuses on accumulating energy from multiple molecules so that the resultant molecule has sufficient energy to "explore" other regions in the solution space. The operator used in synthesis is for creating the structure of the new solution for diversification, but a recombination operator designed for general evolutionary algorithms is used to combine features from other solutions for exploitation. Thus, our results are consistent with the discussion in Section III-B2.
e) Fig. 5 shows the convergence curves for some of the functions from a particular run of RCCRO1 and RCCRO4. The adaptive scheme generally converges faster than the basic scheme.
f) RCCRO4 works best on the average. This shows that our adaptive scheme is useful to solve the benchmark problems.

E. Discussion

In Section IV-D1, the algorithms chosen to compare with RCCRO1 may not be their best variants in solving the bench-

TABLE VI
SIMULATION RESULTS FOR f_8–f_{13}

		RCCRO1	GA	FEP	CEP	FES	CES	PSO	GSO	RCBBO	DE	CMAES	G3PCX
f_8	Mean	−1.257E+04	−1.257E+04	−1.255E+04	−7.917E+03	−1.256E+04	−7.550E+03	−9.660E+03	−1.257E+04	−1.257E+04	−1.257E+04	−9.873E+07	−2.577E+03
	StdDev	2.317E-02	2.109E+00	5.260E+01	6.345E+02	3.253E+01	6.314E+02	4.638E+02	2.214E−02	2.200E−05	2.333E−05	8.547E+08	4.126E+02
	Rank	2	2	8	10	7	11	9	2	2	2	1	12
f_9	Mean	9.077E−04	6.509E−01	4.600E−02	8.900E+01	1.600E−01	7.082E+01	2.079E+01	1.018E+00	2.620E−02	7.261E−05	4.950E+01	1.740E+02
	StdDev	2.876E−04	3.594E−01	1.200E−02	2.310E+01	3.300E−-01	2.149E+01	5.940E+00	9.509E-01	9.760E-03	3.376E-05	1.229E+01	3.199E+01
	Rank	2	6	4	11	5	10	8	7	3	1	9	12
f_{10}	Mean	1.944E−03	8.678E−01	1.800E−02	9.200E+00	1.200E−02	9.070E+00	1.340E−03	2.655E−05	2.510E−02	7.136E−04	4.607E+00	1.352E+01
	StdDev	4.190E−04	2.805E−01	2.100E−02	2.800E+00	1.800E−03	2.840E+00	4.239E−02	3.082E−05	5.510E−03	6.194E−05	8.725E+00	4.815E+00
	Rank	4	8	6	11	5	10	3	1	7	2	9	12
f_{11}	Mean	1.117E−02	1.004E+00	1.600E−02	8.600E−02	3.700E−02	3.800E−01	2.323E−01	3.079E−02	4.820E−01	9.054E−05	7.395E−04	1.127E−02
	StdDev	1.622E−02	6.755E−02	2.200E−02	1.200E−01	5.000E−02	7.700E−01	4.434E−01	3.087E−02	8.490E−02	3.402E−05	2.389E−03	1.310E−02
	Rank	3	12	5	8	7	10	9	6	11	1	2	4
f_{12}	Mean	2.074E−02	4.372E−02	9.200E−06	1.760E+00	2.800E−02	1.180E+00	3.950E−02	2.765E−11	3.280E−05	1.886E−07	5.167E−03	4.593E+00
	StdDev	5.485E−02	5.058E−02	6.140E−05	2.400E+00	8.100E−11	1.870E+00	9.142E−02	9.167E−11	3.330E−05	4.266E−08	7.338E−03	5.984E+00
	Rank	6	9	3	11	7	10	8	1	4	2	5	12
f_{13}	Mean	7.048E−07	1.681E−01	1.600E−04	1.400E+00	4.700E−05	1.390E+00	5.052E−02	4.695E−05	3.720E−04	9.519E−07	1.639E−03	2.349E+01
	StdDev	5.901E−07	7.068E−02	7.300E−05	3.700E+00	1.500E−05	3.330E+00	5.691E−01	7.001E−04	4.630E−04	2.021E−07	4.196E−03	2.072E+01
	Rank	1	9	5	11	4	10	8	3	6	2	7	12
Average rank		3.000	7.667	5.167	10.333	5.833	10.167	7.500	3.333	5.500	1.667	5.500	10.667
Overall rank		**2**	**9**	**4**	**11**	**7**	**10**	**8**	**3**	**5**	**1**	**5**	**12**

TABLE VII
SIMULATION RESULTS FOR f_{14}–f_{23}

		RCCRO1	GA	FEP	CEP	FES	CES	PSO	GSO	RCBBO	DE	CMAES	G3PCX
f_{14}	Mean	9.980E−01	9.989E−01	1.220E+00	1.660E+00	1.200E+00	2.160E+00	1.024E+00	9.980E−01	9.980E−01	1.576E+00	1.246E+01	1.231E+01
	StdDev	1.197E−07	4.433E−03	5.600E−01	1.190E+00	6.300E−01	1.820E+00	1.450E−01	0.000E+00	2.740E−05	2.140E+00	5.529E+00	5.882E+00
	Rank	2	4	7	9	6	10	5	1	3	8	12	11
f_{15}	Mean	5.555E−04	7.088E−03	5.000E−04	4.700E−-04	9.700E−04	1.200E−03	3.807E−04	3.771E−04	7.860E−04	5.372E−04	6.554E−04	5.332E−04
	StdDev	8.944E−05	7.855E−03	3.200E−04	3.000E−04	4.200E−04	1.600E−05	2.509E−04	2.597E−04	1.800E−04	1.221E−04	3.730E−04	3.784E−04
	Rank	7	12	4	3	10	11	2	1	9	6	8	5
f_{16}	Mean	−1.032E+00	−1.030E+00	−1.030E+00	−1.030E+00	−1.032E+00	−1.032E+00	−1.016E+00	−1.032E+00	−1.031E+00	−1.019E+00	−1.015E+00	−4.928E−01
	StdDev	4.843E−04	3.143E−03	4.900E−04	4.900E−04	6.000E−07	6.000E−07	1.279E−02	0.000E+00	9.010E−04	1.869E−02	1.148E−01	3.367E−01
	Rank	1	8	6	6	3	3	10	2	5	9	11	12
f_{17}	Mean	3.979E−01	4.040E−01	3.980E−01	3.980E−01	3.980E−01	3.980E−01	4.040E−01	3.979E−01	3.984E−01	3.995E−01	3.979E−01	5.560E+01
	StdDev	8.525E−07	1.039E−02	1.500E−-07	1.500E−07	6.000E−08	6.000E−08	6.881E−02	0.000E+00	6.770E−04	4.281E−03	1.047E−15	1.071E−13
	Rank	2	10	4	4	4	4	10	3	8	9	1	12
f_{18}	Mean	3.001E+00	7.503E+00	3.020E+00	3.000E+00	3.000E+00	3.000E+00	3.005E+00	3.000E+00	3.010E+00	3.479E+00	5.700E+00	8.670E+00
	StdDev	1.171E−03	1.040E+01	1.100E−01	0.000E+00	0.000E+00	0.000E+00	1.212E−03	0.000E+00	1.120E−02	3.319E+00	1.051E+01	1.290E+01
	Rank	5	11	8	1	1	1	6	1	7	9	10	12
f_{19}	Mean	−3.863E+00	−3.862E+00	−3.860E+00	−3.860E+00	−3.860E+00	−3.860E+00	−3.858E+00	−3.863E+00	−3.862E+00	−3.862E+00	−3.725E+00	−3.598E+00
	StdDev	1.464E−03	6.284E−04	1.400E−05	1.400E−02	4.000E−03	1.400E−05	3.213E−03	3.843E−06	3.650E−04	1.672E−03	5.744E−01	1.869E−01
	Rank	1	5	6	6	6	6	10	2	3	4	11	12
f_{20}	Mean	−3.319E+00	−3.263E+00	−3.270E+00	−3.280E+00	−3.230E+00	−3.240E+00	−3.185E+00	−3.270E+00	−3.317E+00	−3.316E+00	−3.290E+00	−.1980E+00
	StdDev	2.115E−03	6.040E−02	5.900E−02	5.800E−02	1.200E−01	5.700E−02	6.105E−02	5.965E−02	2.360E−02	6.674E−03	5.305E−02	4.327E−01
	Rank	1	8	6	5	10	9	11	7	2	3	4	12
f_{21}	Mean	−1.011E+01	−5.165E+00	−5.520E+00	−6.860E+00	−5.540E+00	−6.960E+00	−7.544E+00	−6.090E+00	−5.513E+00	−8.739E+00	−6.683E+00	−7.476E−01
	StdDev	3.505E−02	2.925E+00	1.590E+00	2.670E+00	1.820E+00	3.100E+00	3.030E+00	3.456E+00	3.350E+00	1.571E+00	3.719E+00	3.170E−01
	Rank	1	11	9	5	8	4	3	7	10	2	6	12
f_{22}	Mean	−1.035E+01	−5.443E+00	−5.520E+00	−8.270E+00	−6.760E+00	−8.310E+00	−8.355E+00	−6.555E+00	−6.800E+00	−9.199E+00	−6.574E+00	−9.468E−01
	StdDev	4.838E−02	3.278E+00	2.120E+00	2.950E+00	3.010E+00	3.100E+00	2.018E+00	3.244E+00	3.520E+00	1.217E+00	3.641E+00	3.761E−01
	Rank	1	11	10	5	7	4	3	9	6	2	8	12
f_{23}	Mean	−1.048E+01	−4.911E+00	−6.570E+00	−9.100E+00	−7.630E+00	−8.500E+00	−8.944E+00	−7.402E+00	−7.285E+00	−9.229E+00	−7.576E+00	−1.130E+00
	StdDev	3.885E−02	3.487E+00	3.140E+00	2.920E+00	3.270E+00	1.250E+00	1.630E+00	3.213E+00	3.380E+00	1.325E+00	3.741E+00	3.678E-01
	Rank	1	11	10	3	6	5	4	8	9	2	7	12
Average rank		2.200	9.100	7.000	4.700	6.100	5.700	6.400	4.100	6.200	5.400	7.800	11.200
Overall rank		**1**	**11**	**9**	**3**	**6**	**5**	**8**	**2**	**7**	**4**	**10**	**12**

mark problems and their parameter settings may not be tailored to the benchmarks. Thus, we do not intend to claim that RCCRO is the best algorithm in solving the benchmarks. In this paper, we try to show that CRO has the ability to work well in the continuous domain, as a complement to the original CRO working in the discrete domain [18]. As a whole, we try to demonstrate that the CRO framework can be applied to a large range of problems. For those interested in the state-of-the-art algorithms in solving real-parameter optimization, they may refer to black-box optimization benchmarking [54] and the competition on constrained real-parameter optimization [55]. Moreover, a recently proposed method which makes use of an ensemble of several constraint handling techniques has shown to be promising for real-parameter optimization [56]. However, pursuit of the best algorithm outperforming all existing ones and integrating CRO with other algorithms are beyond the scope of this paper. In fact, we have shown that RCCRO has a practical application; it has been applied to train artificial neural networks and it has outstanding performance when compared to many other evolutionary algorithm strategies [24].

V. CONCLUSION

In a chemical reaction, molecules start from high-energy states and terminate at low-energy states via a sequence of collisions and molecular changes. CRO captures this idea to develop a metaheuristic for optimization problems. The CRO framework is not restricted to or favor continuous or discrete optimization. As CRO has been demonstrated to work well in solving discrete problems, we decided to propose a real-coded version of CRO, i.e., RCCRO, for optimization problems in the continuous domain. In this paper, Gaussian distribution is adopted to produce perturbations to search the continuous neighborhoods. This simple modification allowed CRO to work in the continuous domain. By considering other recombination and boundary constraint handling operators, we have developed several versions of the algorithms based on the basic RCCRO scheme. We also proposed an adaptive scheme for RCCRO. We applied RCCRO to a widely studied set of continuous benchmark problems and compared the performance of RCCRO with a large set of representative algorithms. Simulation results revealed that CRO works well in all three categories of the benchmark functions. We also compared the various proposed versions of RCCRO and found that the adaptive RCCRO is the most effective.

In the future, we plan to apply RCCRO to more continuous problems. Besides, we can have a detailed study on the parameter values of RCCRO using the value calibration method. Instead of the adaptive approach described in this paper, we will try to develop a self-adaptive scheme which associates the parameter adaptation mechanism to each molecule. Moreover, we may adopt tailor-made operators in the implementation of RCCRO to solve specific problems. In addition, we may include RCCRO in an ensemble-based algorithm for real-parameter optimization.

References

[1] S.-Y. Shin, I.-H. Lee, D. Kim, and B.-T. Zhang, "Multiobjective evolutionary optimization of DNA sequences for reliable DNA computing," *IEEE Trans. Evol. Comput.*, vol. 9, no. 2, pp. 143–158, Apr. 2005.

[2] Y. Zhang, L. O. Hall, D. B. Goldgof, and S. Sarkar, "A constrained genetic approach for computing material property of elastic objects," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 341–357, May 2006.

[3] L. Yu, H. Chen, S. Wang, and K. K. Lai, "Evolving least squares support vector machines for stock market trend mining," *IEEE Trans. Evol. Comput.*, vol. 13, no. 1, pp. 87–102, Feb. 2009.

[4] M. Alrashidi and M. El-Hawary, "A survey of particle swarm optimization applications in electric power systems," *IEEE Trans. Evol. Comput.*, vol. 13, no. 4, pp. 913–918, Aug. 2009.

[5] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.

[6] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[8] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.

[9] J. Kennedy and R. Eberhart, *Swarm Intelligence*. San Francisco, CA: Morgan Kaufmann, 2001.

[10] Wikipedia. (2009, Sep.). *Discrete Optimization* [Online]. Available: http://en.wikipedia.org/wiki/Discrete_optimization

[11] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the 'simulated annealing' algorithm," *ACM Trans. Math. Softw.*, vol. 13, no. 3, pp. 262–280, Sep. 1987.

[12] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA: MIT Press, 1992.

[13] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1992.

[14] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[15] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *Eur. J. Oper. Res.*, vol. 185, no. 3, pp. 1155–1173, Mar. 2008.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, Nov.–Dec. 1995, pp. 1942–1948.

[17] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 1997, pp. 4104–4109.

[18] A. Y. S. Lam and V. O. K. Li, "Chemical-reaction-inspired metaheuristic for optimization," *IEEE Trans. Evol. Comput.*, vol. 14, no. 3, pp. 381–399, Jun. 2010.

[19] J. Xu, A. Y. S. Lam, and V. O. K. Li, "Chemical reaction optimization for the grid scheduling problem," in *Proc. IEEE ICC*, May 2010, pp. 1–5.

[20] J. Xu, A. Y. S. Lam, and V. O. K. Li, "Chemical reaction optimization for task scheduling in grid computing," *IEEE Trans. Parallel Distrib. Syst.*, 2011, to be published.

[21] A. Y. S. Lam and V. O. K. Li, "Chemical reaction optimization for cognitive radio spectrum allocation," in *Proc. IEEE GLOBECOM*, Dec. 2010, pp. 1–5.

[22] A. Y. S. Lam, J. Xu, and V. O. K. Li, "Chemical reaction optimization for population transition in peer-to-peer live streaming," in *Proc. IEEE CEC*, Jul. 2010, pp. 1–8.

[23] J. Xu, A. Y. S. Lam, and V. O. K. Li, "Parallel chemical reaction optimization for quadratic assignment problem," in *Proc. Int. Conf. GEM*, 2010, pp. 125–131.

[24] J. J. Q. Yu, A. Y. S. Lam, and V. O. K. Li, "Evolutionary artificial neural network based on chemical reaction optimization," in *Proc. IEEE CEC*, Jun. 2011, pp. 2083–2090.

[25] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[26] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

[27] K. Krishnamoorthy, *Handbook of Statistical Distributions with Applications*. Boca Raton, FL: Chapman and Hall/CRC, 2006.

[28] W. Chu, X. Gao, and S. Sorooshian, "Handling boundary constraints for particle swarm optimization in high-dimensional search space," *Inform. Sci.*, vol. 181, no. 20, pp. 4569–4581, Oct. 2011.

[29] J. Ronkkonen, S. Kukkonen, and K. V. Price, "Real-parameter optimization with differential evolution," in *Proc. IEEE CEC*, Sep. 2005, pp. 506–513.

[30] J. Brest, "Constrained real-parameter optimization with ϵ-self-adaptive differential evolution," in *Constraint-Handling in Evolutionary Optimization* (Series Studies in Computational Intelligence), vol. 198, E. Mezura-Montes, Ed. Berlin/Heidelberg, Germany: Springer, 2009, pp. 73–93.

[31] W.-J. Zhang, X.-F. Xie, and D.-C. Bi, "Handling boundary constraints for numerical optimization by particle swarm flying in periodic search space," in *Proc. IEEE CEC*, Jun. 2004, pp. 2307–2311.

[32] G. Leguizamón and C. A. C. Coello, "Boundary search for constrained numberical optimization problems with an algorithm inspired by the ant colony metaphor," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 350–368, Apr. 2009.

[33] F. Herrera, M. Lozano, and A. M. Sánchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study," *Int. J. Intell. Syst.*, vol. 18, no. 3, pp. 309–338, 2003.

[34] F. Herrera, M. Lozano, and A. M. Sánchez, "Hybrid crossover operators for real-coded genetic algorithms: An experimental study," *Soft Comput.*, vol. 9, pp. 280–298, Apr. 2005.

[35] T. Nomura and K. Shimohara, "An analysis of two-parent recombinations for real-valued chromosomes in an infinite population," *Evol. Computat.*, vol. 9, no. 3, pp. 283–308, 2001.

[36] J. Reed, R. Toombs, and N. A. Barricelli, "Simulation of biological evolution and machine learning. I. Selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing," *J. Theoretic. Biol.*, vol. 17, no. 3, pp. 319–342, Dec. 1967.

[37] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Aug. 1999.

[38] A. Y. S. Lam, V. O. K. Li, and J. J. Q. Yu, "CROToolbox: A toolbox for chemical reaction optimization," submitted for publication.

[39] *Chemical Reaction Optimization*. University of Hong Kong, Pokfulam, Hong Kong [Online]. Available: http://cro.eee.hku.hk

[40] V. Nannen and A. Eiben, "A method for parameter calibration and relevance estimation in evolutionary algorithms," in *Proc. 8th Annu. Conf. GECCO*, 2006, pp. 183–190.

[41] V. Nannen and A. E. Eiben, "Relevance estimation and value calibration of evolutionary algorithm parameters," in *Proc. 20th Int. Joint Conf. Artif. Intell.*, 2007, pp. 975–980.

[42] X. Yao and Y. Liu, "Fast evolution strategies," in *Proc. 6th Int. Conf. Evol. Program. VI*, 1997, pp. 151–162.

[43] S. He, Q. H. Wu, and J. R. Saunders, "Group search optimizer: An optimization algorithm inspired by animal searching behavior," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 973–990, Aug. 2009.

[44] W. Gong, Z. Cai, C. X. Ling, and H. Li, "A real-coded biogeography-based optimization with mutation," *Appl. Math. Comput.*, vol. 216, no. 9, pp. 2749–2758, Jul. 2010.

[45] R. Storn and K. Price, "Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimiz.*, vol. 11, no. 4, pp. 341–359, Dec. 1997.

[46] K. V. Price, R. M. Storn, and J. L. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Berlin, Germany: Springer, 2005.

[47] N. Hansen and A. Ostermeier, "Completely derandomized self-adaption in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, Jun. 2001.

[48] K. Deb, A. Anand, and D. Joshi, "A computationally efficient evolutionary algorithm for real-parameter optimization," *Evol. Comput.*, vol. 10, no. 4, pp. 371–395, Dec. 2002.

[49] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies: A comprehensive introduction," *Nat. Comput.*, vol. 1, no. 1, pp. 3–52, Mar. 2002.

[50] D. Simon, "Biogeography-based optimization," *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 702–713, Nov. 2008.

[51] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, Feb. 2011.

[52] *CMA-ES Source Code*. Laboratoire de Recherche en Informatique, Orsay, France [Online]. Available: http://www.lri.fr/∼hansen/cmaes_inmatlab.html

[53] *G3PCX Source Code*. Kanpur Genetic Algorithms Laboratory, Kanpur, India [Online]. Available: http://www.iitk.ac.in/kangal/codes.shtml

[54] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Posik, "Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009," in *Proc. 12th Annu. Conf. GECCO*, 2010, pp. 1689–1696.

[55] R. Mallipeddi and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2010 competition on constrained real-parameter optimization," Nanyang Technol. Univ., Singapore, Tech. Rep., 2010.

[56] R. Mallipeddi and P. N. Suganthan, "Ensemble of constraint handling techniques," *IEEE Trans. Evol. Comput.*, vol. 14, no. 4, pp. 561–579, Feb. 2011.

Albert Y. S. Lam (S'03–M'10) received the B.E. (first class honors) degree in information engineering and the Ph.D. degree in electrical and electronic engineering from the University of Hong Kong, Pokfulam, Hong Kong, in 2005 and 2009, respectively.

He is currently a Post-Doctoral Scholar with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA. He is a Croucher Research Fellow. His current research interests include optimization theory and algorithms, evolutionary computation, information theory, smart grid, wireless and mobile networking, and Internet protocols and applications.

Dr. Lam also serves on the IEEE Computational Intelligence Society (CIS) Social Media Subcommittee as the Chair, the CIS GOLD Subcommittee as the Vice Chair, CIS Webinars Subcommitee as a member, and an ad hoc committee as a member.

Victor O. K. Li (S'80–M'81–SM'86–F'92) received the S.B., S.M., E.E., and S.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1977, 1979, 1980, and 1981, respectively.

He is the Associate Dean of engineering and Chair Professor of information engineering with the University of Hong Kong (HKU), Pokfulam, Hong Kong, a Visiting Professor with King Saud University, Riyadh, Saudi Arabia, and a Guest Chair Professor of wireless communication and networking with Tsinghua University, Beijing, China. He also served as the Managing Director of Versitech Ltd., Pokfulam, the technology transfer and commercial arm of HKU, and on the boards of SUNeVision Holdings Ltd. and China.com Ltd. Previously, he was a Professor of electrical engineering with the University of Southern California (USC), Los Angeles, and the Director of the USC Communication Sciences Institute. Sought by government, industry, and academic organizations, he has lectured and consulted extensively around the world.

Dr. Li has received numerous awards, including the PRC Ministry of Education Changjiang Chair Professorship at Tsinghua University, the U.K. Royal Academy of Engineering Senior Visiting Fellowship in Communications Award, the Croucher Foundation Senior Research Fellowship, and the Order of the Bronze Bauhinia Star, Government of the Hong Kong Special Administrative Region, China. He is a Registered Professional Engineer and a fellow of the IAE and the HKIE.

James J. Q. Yu is currently pursuing the B.E. degree from the Department of Electrical and Electronic Engineering, University of Hong Kong, Pokfulam, Hong Kong.

His current research interests include evolutionary algorithms, wireless networking, and sensor networks.