Enabling the Extended Compact Genetic Algorithm for Real-Parameter Optimization by Using Adaptive Discretization

Ying-ping Chen* ypchen@nclab.tw
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan

Chao-Hong Chen chchen@nclab.tw
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan

Abstract

An adaptive discretization method, called *split-on-demand* (SoD), enables estimation of distribution algorithms (EDAs) for discrete variables to solve continuous optimization problems. SoD randomly splits a continuous interval if the number of search points within the interval exceeds a threshold, which is decreased at every iteration. After the split operation, the nonempty intervals are assigned integer codes, and the search points are discretized accordingly. As an example of using SoD with EDAs, the integration of SoD and the extended compact genetic algorithm (ECGA) is presented and numerically examined. In this integration, we adopt a local search mechanism as an optional component of our back end optimization engine. As a result, the proposed framework can be considered as a memetic algorithm, and SoD can potentially be applied to other memetic algorithms. The numerical experiments consist of two parts: (1) a set of benchmark functions on which ECGA with SoD and ECGA with two well-known discretization methods: the fixed-height histogram (FHH) and the fixed-width histogram (FWH) are compared; (2) a real-world application, the economic dispatch problem, on which ECGA with SoD is compared to other methods. The experimental results indicate that SoD is a better discretization method to work with ECGA. Moreover, ECGA with SoD works quite well on the economic dispatch problem and delivers solutions better than the best known results obtained by other methods in existence.

Keywords

Estimation of distribution algorithm, EDA, ECGA, split-on-demand, SoD, real-parameter optimization, economic dispatch, valve point effect.

1 Introduction

Genetic algorithms are widely applied on many real-world optimization problems. According to the theory of design decomposition (Goldberg, 2002), key components in the GA success include identifying, reproducing, and exchanging solution structures. Recombination, one of the main GA operators, mixes the promising subsolutions, called *building blocks* (BBs), and creates new solutions. Genetic algorithms therefore work very well on those problems that can be decomposed into subproblems. However, problem-independent recombination operators with fixed chromosome representations

*To whom correspondence should be addressed.

often break building blocks and result in ineffective mixing. When traditional genetic algorithms meet complex solution structures composed of groups of related genes, they often fail to appropriately identify and exchange building blocks to create good final solutions (Goldberg et al., 1989).

In order to mix genes effectively, Larrañaga and Lozano (2001) and Pelikan et al. (2002) proposed evolutionary algorithms based on probabilistic models. In such schemes, the offspring population is generated according to the estimated probabilistic model of the selected individuals instead of using regular recombination and mutation operators. The probabilistic model is expected to reflect the problem structure, and better performance can be achieved via exploring and exploiting the relationship among genes. These evolutionary algorithms are called estimation of distribution algorithms (EDAs) or probabilistic model building genetic algorithms (PMBGAs; Larrañaga and Lozano, 2001; Pelikan et al., 2002).

In EDAs, decision variables are often coded with binary codes. It is computationally expensive to find high accuracy solutions in solving continuous problems for EDAs combined with elementary discretization methods (Tsutsui et al., 2001; Pelikan et al., 2003). Moreover, many real-world engineering problems are real-parameter optimization problems, such as structural optimization. In the literature, several attempts to apply EDAs to problems in the continuous domain have been made, including continuous population-based incremental learning with Gaussian distribution (Sebag and Ducoulombier, 1998), a real-coded variant of population-based incremental learning with interval updating (Servet et al., 1997), Bayesian evolutionary algorithms for continuous function optimization (Shin and Zhang, 2001), and the real-coded Bayesian optimization algorithm (Ahn et al., 2004).

However, these approaches require the knowledge of and are clearly specialized for the modified algorithms. In order to provide a good, general interface between problems of continuous variables and algorithms for discrete variables, we propose a framework that enables the EDAs designed for handling bit strings to tackle real-valued optimization problems. Particularly, we develop a new, adaptive discretization encoding scheme that can be easily integrated into EDAs or other algorithms for discrete variables, and we use the extended compact genetic algorithm (ECGA; Harik, 1999) as an example in the present work.

In the next section, we will give a background of this study, including a brief introduction of ECGA and two well-known discretization methods: the fixed-height histogram (FHH) and the fixed-width histogram (FWH). Section 3 proposes split-on-demand (SoD) and describes in detail how SoD encodes individuals of real values into integer codes. In Section 4, we use SoD to enable ECGA to handle real-valued variables. The numerical experiments on benchmark functions are presented in Section 5, including the comparisons with the elementary discretization techniques and other optimization methods in evolutionary computation. The proposed framework is also adopted to tackle the economic dispatch problem in Section 6. Finally, Section 7 concludes this work.

2 Background

This section first provides a brief overview of EDAs. EDAs use probabilistic models to represent distributions of candidate solutions. Constructing probabilistic models and sampling from the models replace the usual crossover and mutation operators in genetic algorithms. Next, the feature designs of the ECGA are described, including the probabilistic model and the criterion to evaluate the models. The section ends

by introducing two discretization methods, the FHH and the FWH, which will be integrated with ECGA.

2.1 Estimation of Distribution Algorithms

EDAs (Mühlenbein and Paaß, 1996; Larrañaga and Lozano, 2001; Pelikan et al., 2002) tackle optimization problems by constructing probabilistic models based on promising solutions and by generating offspring solutions from the built models. From the viewpoint of genetic and evolutionary algorithms, EDAs replace the regular genetic operators, such as crossover and mutation, with the construction and utilization of probabilistic models. In EDAs, at each generation, the selection operator selects promising individuals from the current population. A probabilistic model is then constructed according to these selected individuals, and new individuals are produced by sampling from the built model.

An EDA can be algorithmically outlined as

1. Initialize a population at random.
2. Apply the selection operator on the population.
3. Model the selected individuals by constructing a probabilistic model.
4. Stop if the termination criterion is satisfied.
5. Generate a new population by sampling the built model.
6. Return to step 2.

Early EDAs, such as population-based incremental learning (PBIL; Baluja, 1994) and the compact genetic algorithm (cGA; Harik et al., 1999), assume no interaction between variables. Each variable is modeled independently of the others and subsequent studies start by capturing potential pairwise interactions, such as mutual-information-maximizing input clustering (MIMIC; de Bonet et al., 1997), Baluja's dependency tree approach (Baluja and Davies, 1997), and the bivariate marginal distribution algorithm (BMDA; Pelikan and Mühlenbein, 1999). Further studies modeled multivariate interactions, such as the ECGA (Harik, 1999), the Bayesian optimization algorithm (BOA; Pelikan et al., 1999), the estimation of Bayesian network algorithm (EBNA; Etxeberria and Larrañaga, 1999), the factorized distribution algorithm (FDA; Mühlenbein and Mahnig, 1999), and the learning version of FDA (LFDA; Mühlenbein and Höns, 2005). With the reasoning of dependencies among variables by building probabilistic models, these approaches can capture problem structures and achieve good performance.

2.2 Extended Compact Genetic Algorithm

The ECGA was proposed by Harik (1999) based on the idea that probability distributions can be used to model the population in genetic algorithms and the choice of a good probability distribution can be viewed as equivalent to learning linkage between variables. The probabilistic models adopted in ECGA are a class of models known as the marginal product models (MPMs; Harik, 1999). ECGA utilizes MPMs to model partitions of variables. The measurement of distribution quality is quantified according to the minimum description length (MDL) principle (Rissanen, 1989), which can be considered as a realization of Occam's razor. MDL prefers simpler distributions to

more complex ones. The MDL criterion penalizes both inaccuracy and complexity and thereby produces high quality distributions.

ECGA can be algorithmically outlined as

1. Initialize a population of N individuals at random.
2. Apply tournament selection of size S.
3. Model the selected individuals with a greedy MPM search.
4. Stop if the MPM model has converged.
5. Generate a new population by sampling the MPM model.
6. Return to step 2.

The complexity measurement of the MPM model is the sum of (1) model complexity and (2) compressed population complexity:

$$\text{Model complexity} = \log N \sum_I 2^{S[I]}, \tag{1}$$

where N is the population size, and $S[I]$ is the length of the Ith subset of genes.

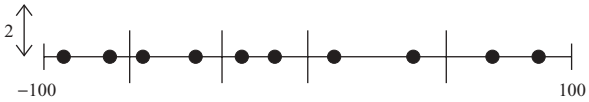
$$\text{Compressed population complexity} = N \sum E(M_I), \tag{2}$$

where $E(M_I)$ is the entropy of the marginal distribution for subset I. According to the MDL principle, the goal for the greedy MPM search is to find an MPM model with the minimal combined complexity.

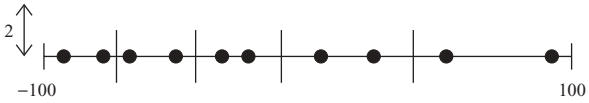
Instead of applying regular crossover and mutation operators, ECGA creates the new population by sampling the MPM model obtained in Step 3. This creates offspring without disrupting linkage groups of decision variables. In the original framework, ECGA can handle only binary variables. In order to make ECGA capable of tackling the real-parameter optimization problems, a certain technique is required to play as an interface between the method for optimization and the problem to solve. In the present work, we propose an adaptive discretization method, SoD, which will be described in Section 3. In the next section, we will briefly introduce two well-known discretization methods, the FHH and the FWH.

2.3 Discretization Methods

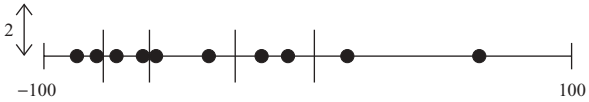
Discretization can be used to transform problems from the continuous domain into the discrete domain and to provide an interface between problems and solvers of different domains. After discretization, the candidate solutions in the continuous domain are encoded as discrete codes, such as binary strings or integer vectors. The optimization techniques designed for discrete variables can then operate on these binary strings or integer vectors. In this section, two elementary discretization methods are described (Bosman and Thierens, 2000; Cantú-Paz, 2001; Tsutsui et al., 2001).



(a) Population # 1: five-bin FHH (two individuals in each bin).



(b) Population # 2: five-bin FHH (two individuals in each bin).



(c) Population # 3: five-bin FHH (two individuals in each bin).

Figure 1: Three populations discretized by the five-bin FHH.

2.3.1 Fixed-Height Histogram (FHH)

One way to discretize an interval is to place equal numbers of individuals in the bins. Real-valued individuals within one bin are all encoded as the discrete code assigned to that bin. One of the intuitive ways to discretize an interval is to make all the bins contain the same number of individuals. Such a method is called the fixed-height histogram, where the word *height* refers to the number of individuals contained in each bin. Figure 1 shows an example to discretize three different populations of size 10 with a five-bin FHH. Because of the fixed height, each bin contains exactly two individuals.

2.3.2 Fixed-Width Histogram (FWH)

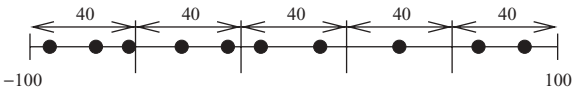
Another discretization technique fixes the *width* of the bins. Figure 2 shows the discretization of three populations of size 10 into five bins using this technique, called fixed-width histogram. For an interval $[\ell, u)$, a k-bin FWH divides the interval into k bins of equal width $(u - \ell)/k$. The range of the ith bin can be given as

$$\left[\ell + \frac{i(u-\ell)}{k}, \ell + \frac{(i+1)(u-\ell)}{k}\right),$$

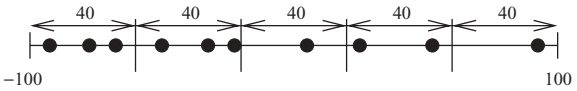
where $i \in \{0, \ldots, k-1\}$. Figure 2 shows that the discretization configuration is identical for the three different populations, and the width of each bin is fixed as 40.

3 Split-on-Demand

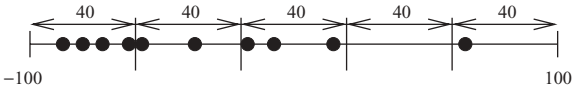
An adaptive discretization method, called split-on-demand, encodes real-number decision variables into discrete numerical codes (Chen et al., 2006). The main idea of SoD is to encode with more integer codes those regions that we are more interested in and want to know more about. For this purpose, SoD splits the continuous intervals in which there are equal to or more than a specified number of individuals at random. Thus, when SoD works with EDAs, more accurate probabilistic models regarding these promising



(a) Population # 1: 5-bin FWH (the width is 40 for each bin).



(b) Population # 2: 5-bin FWH (the width is 40 for each bin).



(c) Population # 3: 5-bin FWH (the width is 40 for each bin).

Figure 2: Three populations discretized by the five-bin FWH.

regions may be built, while less computational resources may be spent on modeling the regions containing fewer individuals. A *split rate* γ, where $0 < \gamma < 1$, is employed to determine whether or not a real-number interval should be split. If the population size is N, an interval containing equal to or more than $N \times \gamma$ individuals should be split. By adjusting γ, we can control the precision of the probabilistic model (of discrete variables) as well as avoid having unnecessarily long bit strings for discretization.

As described, SoD splits a dimension of real numbers into several intervals and assigns each of the intervals an integer code. We can then translate a vector of real numbers into a vector of integers, which can be represented by bits or binary codes more directly. As an example, given a real-parameter optimization problem of two dimensions, one possible code table constructed by SoD is shown in Figure 3. The solution $(-72.3, 24.8)$ is encoded as $(0, 1)$, and the solution $(13.8, -5.3)$ as $(2, 0)$. Figure 4 shows the solution space split by the code table given in Figure 3. Figure 4(a) is the split configuration on dimension 1, Figure 4(b) is the split configuration on dimension 2, and Figure 4(c) is the combined split configuration on (dimension 1, dimension 2), which is the whole solution space. The code table splits the solution space into 12 regions.

In order to construct the table, SoD splits continuous intervals in which there are *too many* search points. Because a selection operator chooses promising individuals at each generation, if there is a host of individuals in certain regions after selection, we may consider those regions important and conclude that the probability of good solutions in those regions is higher. Therefore, we split promising regions to gain higher resolution as well as to achieve better accuracy in assisting the EDA to build high quality models.

In order to determine which interval to split, we employ a split rate γ, where $0 < \gamma < 1$. An interval should be split if it contains equal to or more than $N \times \gamma$ individuals, where N is the population size. The split rate controls the resolution and accuracy of the probabilistic model to be built. If the probabilistic model is required to be more accurate, smaller split rates should be used such that the value range of a variable is split into more intervals. Furthermore, for the same reason, the split rate also stochastically decides the overall code length. The higher the split rate, the shorter the code length, and vice versa.

Dimension 1				Dimension 2			
Interval			Code	Interval			Code
-100	to	-50	0	-100	to	0	0
-50	to	0	1	0	to	50	1
0	to	50	2	50	to	100	2
50	to	100	3				

Figure 3: An example code table constructed by SoD for a real-parameter optimization problem of two dimensions.

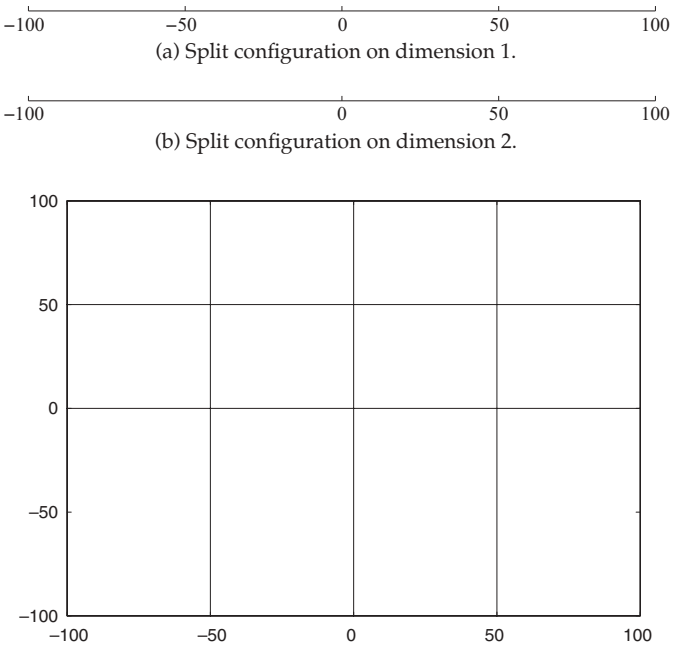


Figure 4: An illustration of the solution space being split according to the code table given in Figure 3.

The procedure of SoD can be described as follows, and the pseudocode of SoD is shown in Figure 5. Procedure `Split-on-Demand` first calls subroutine `Split` on the interval $[lower\_bound, upper\_bound)$, where the $lower\_bound$ and $upper\_bound$ are the bounds of the range of a decision variable. `Split` generates a uniformly distributed random number m within the interval in question and counts the individuals in the two subintervals: $[\ell, m)$ and $[m, u)$. If an interval contains equal to or more than $N \times \gamma$ individuals, it should be further split. `Split` will recursively call itself to split that interval until all the intervals contain fewer than $N \times \gamma$ individuals.

When all the split operations are done, we decrease the split rate by a factor ϵ, where $0 < \epsilon < 1$. The reason to decrease the split rate is to have higher split rates at

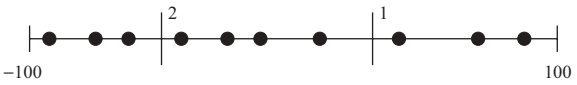
```
1: procedure SPLIT-ON-DEMAND
2:     Split([lower_bound, upper_bound))
3:     γ ← γ × ϵ
4: end procedure
```

```
1: procedure SPLIT([ℓ, u))
2:     m ← random[ℓ, u)
3:     N_ℓ ← number of individuals in [ℓ, m)
4:     N_u ← number of individuals in [m, u)
5:     if N_ℓ ≥ N × γ then
6:         Split([ℓ, m))
7:     else
8:         Add a code for the range [ℓ, m)
9:     end if
10:    if N_u ≥ N × γ then
11:        Split([m, u))
12:    else
13:        Add a code for the range [m, u)
14:    end if
15: end procedure
```

Figure 5: Pseudocode for SoD.

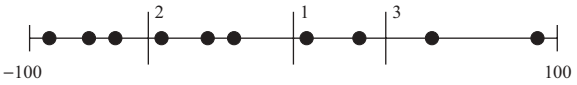
early search stages to maintain the diversity as well as to implement a coarse-grained, global search. As the search process proceeds, we obtain more and more information about the solution space and know where to put more search points to find good solutions. Hence, at late search stages, a lower split rate is needed to build accurate probabilistic models for conducting a fine-grained, local search. The factor ϵ can be set to control the speed of convergence. An appropriate ϵ can help the back end search algorithm to avoid wasting time on useless regions as well as being trapped at local optima, and therefore is key to an efficient search process.

We now give an example of how SoD runs on populations. Assume that the population size is 10, and the initial split rate $\gamma = 0.5$. Figure 6 depicts how the individuals are distributed at different generations. Initially, Figure 6(a) shows that the first position to split, marked by 1, is randomly generated. We then discover that the number of individuals in the left interval is larger than $10 \times \gamma = 5$. Under this condition, SoD calls `Split` to perform a random split on the left interval and gets the second split position, marked by 2. After the second split, the numbers of individuals in the two intervals,



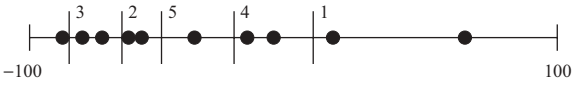
(a) Population #1 and two split positions at generation 1. $\gamma = 0.5$.

$10 \times \gamma = 5$.



(b) Population #2 and three split positions at generation 10. $\gamma = 0.4$.

$10 \times \gamma = 4$.



(c) Population #3 and five split positions at generation 20. $\gamma = 0.3$.

$10 \times \gamma = 3$.

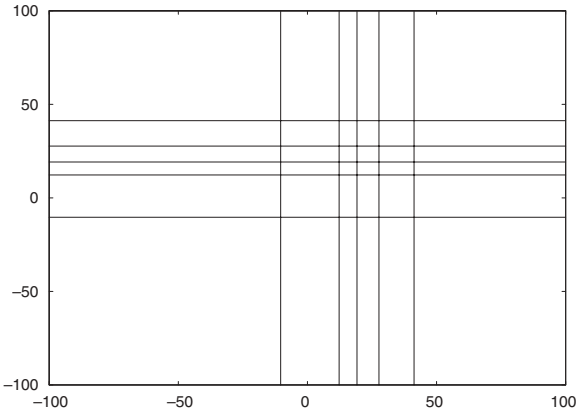
Figure 6: Populations and split positions at different generations.

the left interval and the right interval to the second split position, are both less than $10 \times \gamma = 5$. As a consequence, SoD stops the split operation and decreases the split rate.

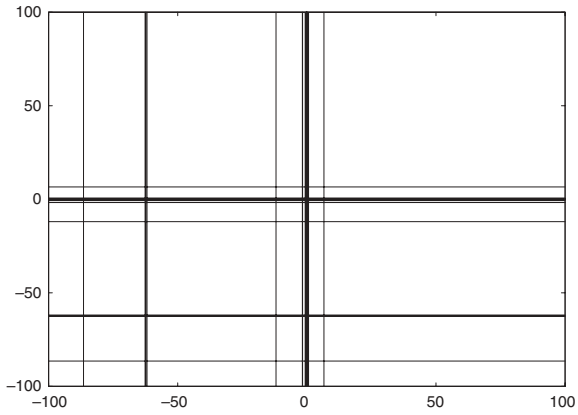
Figure 6(b) shows the population and the splits at generation 10. The split rate γ is now 0.4. SoD performs a random split to cut the whole interval into two intervals. It can be observed that both the left and the right intervals contain equal to or more than $10 \times \gamma = 4$ individuals. As a result, SoD calls `Split` on both the left and the right intervals. For the left interval, SoD randomly splits it into two intervals and finds out that each interval contains three individuals. By conducting the recursive split operation until no more interval has to be split, three splits make the value range four intervals. Moreover, in Figure 6(c) at generation 20, the split rate is 0.3. SoD runs on the population, and the whole interval is split into six regions by five splits.

One might wonder whether the proposed encoding scheme seems similar to the FHH method. In fact, there is a significant difference between SoD and FHH—the code table construction. In FHH, because the height of histograms, that is, the number of bins employed in the algorithm, is fixed, the code table is deterministically constructed for any given population. However, in SoD, even with the same split rate, for the same population, different code tables may be stochastically constructed. Such a flexibility might make the probabilistic model built by the back end EDA more accurate than what is built according to the individuals encoded by FHH.

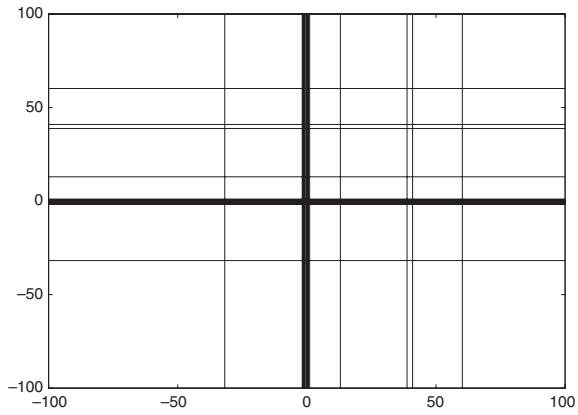
For handling the adaptive discretization during an optimization process, Figure 7 shows an example of how SoD cooperating with ECGA splits the solution space at different generations when a two-dimensional objective function $G_1 = \sum x_i^2$ is minimized, where the bound of every dimension is $[-100, 100]$, and the global optimum is $(0, 0)$.



(a) Generation 1



(b) Generation 50



(c) Generation 100

Figure 7: Split configurations at different generations for $G_1 = \sum x_i^2$.

Figure 7(a) depicts the split configuration on the solution space at generation 1. The split configuration seems random because the population is highly diverse at generation 1. Later, at generation 50, the population begins to converge, and Figure 7(b) shows that SoD splits the solution space around $(0, 0)$ into many regions and leaves other parts of the solution space un-encoded. Finally, in Figure 7(c), it can be observed that SoD focuses on the solution space close to $(0, 0)$ at generation 100. With the population converging to $(0, 0)$, ECGA is able to explore the promising solution space more thoroughly and to find the solutions of higher precision.

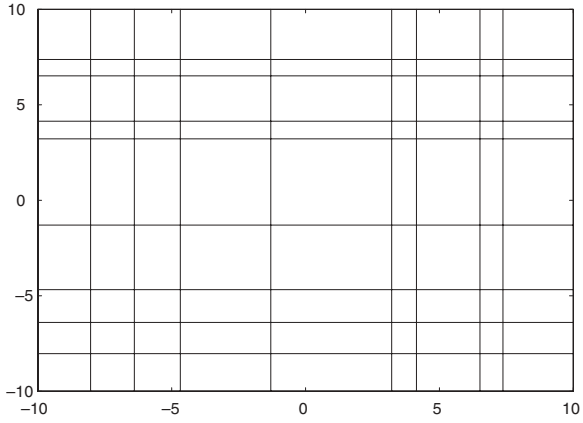
Another example is the two-dimensional objective function $G_2 = \sum 10 - |x_i|$. Figure 8 depicts how SoD splits the solution space, of which the bound of each dimension is $[-10, 10]$, when G_2 is minimized. There are four global minima located at $(-10, -10)$, $(-10, 10)$, $(10, -10)$, and $(10, 10)$. Figure 8(a) is the split configuration on the solution space at generation 1. Because the population is initially random, the split configuration seems random. In Figure 8(b), we observe that at generation 10, because the population begins to converge to the global minima, the split points are close to the four corners where the global minima of G_2 are located. Finally, Figure 8(c) shows that almost all split points are around the region close to $(10, 10)$ because the population converges to one of the four global optima at generation 20.

These two examples demonstrate that the split configuration established by SoD appropriately responds to the status of the population. The split configuration can encode the individuals as precisely as necessary for the backend EDA to build probabilistic models. Hence, SoD is an effective encoding technique to make EDAs able to handle real-parameter optimization problems. In the next section, ECGA, as an example of EDAs, will be employed to show the feasibility of integrating SoD and EDAs.

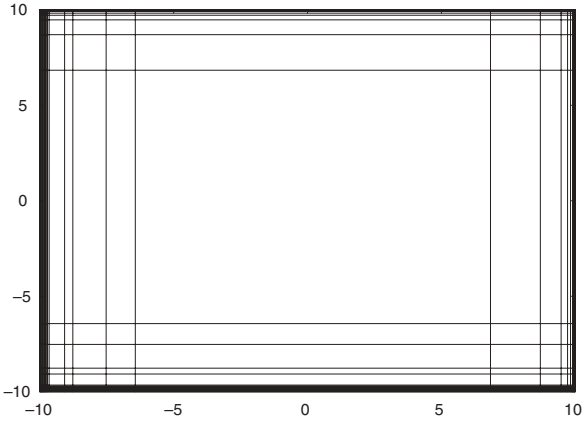
4 Real-Coded ECGA

In the previous section, we proposed the adaptive discretization SoD method, we described the behavior of SoD, and we demonstrated the effect of SoD. In this section, we will show the way to plug SoD into ECGA, as a showcase for the integration of SoD and EDAs. A similar hybridization was proposed by Pelikan et al. (2003) to use a discretization method as a bridge between optimization techniques designed for variables of different domains. In this study, we use SoD as a direct interface between problems and the optimizer. The integration of ECGA and SoD is called the *real-coded* ECGA (rECGA) in the remainder of this paper and can be given as

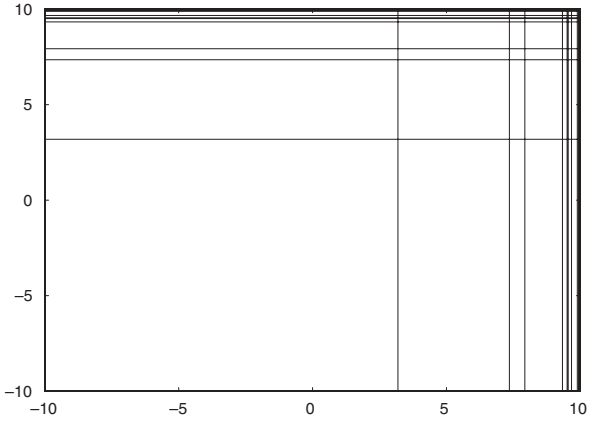
1. Initialize a population of N individuals at random.
2. Apply tournament selection of size S.
3. Use SoD to encode each dimension of the continuous variables.
4. Model the selected, encoded individuals with a greedy MPM search.
5. Stop if the MPM model has converged.
6. Generate a new population by sampling the built MPM model.
7. (Optional local search) for every L generations, run the Nelder-Mead (Nelder and Mead, 1965) method on the best 10% individuals.
8. Return to Step 2.



(a) Generation 1



(b) Generation 10



(c) Generation 20

Figure 8: Split configurations at different generations for $G_2 = \sum 10 - |x_i|$.

The implementation of rECGA used in this paper is available and can be downloaded at http://www.nclab.tw/SM/2009/01.

5 Comparisons on Benchmark Functions

After proposing SoD and rECGA, first we would like to know how SoD performs compared to the well-known discretization methods FHH and FWH. Then, we will compare the performance of rECGA to that of other real-parameter optimization methods in evolutionary computation on a set of benchmark functions proposed in CEC 2005 (Suganthan et al., 2005).

5.1 Comparisons with FHH and FWH

In this section, we use the identical search engine, ECGA, as the platform and plug SoD, FHH, and FWH into ECGA to examine how well they perform under the same conditions. When FHH and FWH are used, the optimization procedure, similar to rECGA, can be described as

1. Initialize a population of N individuals at random.
2. Apply tournament selection of size S.
3. Use FHH/FWH to encode each dimension of the continuous variables.
4. Model the selected, encoded individuals with a greedy MPM search.
5. Stop if the MPM model has converged.
6. Generate a new population by sampling the built MPM model.
7. (Optional local search) for every L generations, run the Nelder-Mead (Nelder and Mead, 1965) method on the best 10% individuals.
8. Return to Step 2.

The following eight test functions were used to serve as a testbed for observing the relative performance of the three discretization methods:

1. Sphere function

$$F_1(x) = \sum_{i=1}^{D} x_i^2$$

2. Rosenbrock's function

$$F_2(x) = \sum_{i=1}^{D-1} \left(100\left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2\right)$$

3. Ackley's function

$$F_3(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e$$

4. Griewanks's function

$$F_4(x) = \sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

5. Weierstrass function

$$F_5(x) = \sum_{i=1}^{D}\left(\sum_{k=0}^{kmax}(a^k\cos(2\pi b^k(x_i+0.5)))\right) - D\sum_{k=0}^{kmax}(a^k\cos(2\pi b^k\cdot 0.5)),$$

where $a = 0.5$, $b = 3$, $kmax = 20$

6. Rastrigin's function

$$F_6(x) = \sum_{i=1}^{D}\left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$$

7. Noncontinuous Rastrigin's function

$$F_7(x) = \sum_{i=1}^{D}\left(y_i^2 - 10\cos(2\pi y_i) + 10\right),$$

$$\text{where } y_i = \begin{cases} x_i, & \text{if } |x_i| < \frac{1}{2} \\ \dfrac{\text{round}(2x_i)}{2}, & \text{if } |x_i| >= \frac{1}{2} \end{cases}$$

8. Schwefel's function

$$F_8(x) = 418.9829 \times D - \sum_{i=1}^{D} x_i \sin\left(|x_i|^{\frac{1}{2}}\right)$$

The global optimum positions, x^*, the global optimum objective values, $f(x^*)$, and the search intervals, $[x_{min}, x_{max}]$, of each test function are listed in Table 1. The number of dimension, D, in all the experiments is 10.

The parameters for rECGA (i.e., ECGA+SoD) we used in this series of experiments are given in the following: population size = 250, crossover probability = 1.0, tournament size = 8, $L = 5$, the maximum function evaluations = 30,000, and 50 independent runs for each test function. The maximum function evaluations include the function evaluations made by ECGA and the Nelder-Mead method. Because the code length is constant for FHH and FWH while it varies for SoD, in order to achieve fair comparisons, we conducted two sets of experiments to examine the effect of different code lengths

Table 1: Global optima and search intervals of the test functions.

F	x^*	$f(x^*)$	Search Intervals
F_1	$[0, 0, \ldots, 0]$	0	[–100,100]
F_2	$[0, 0, \ldots, 0]$	0	[–2.048,2.048]
F_3	$[0, 0, \ldots, 0]$	0	[–32.768,32.768]
F_4	$[0, 0, \ldots, 0]$	0	[–600,600]
F_5	$[0, 0, \ldots, 0]$	0	[–0.5,0.5]
F_6	$[0, 0, \ldots, 0]$	0	[–5.12,5.12]
F_7	$[0, 0, \ldots, 0]$	0	[–5.12,5.12]
F_8	$[0, 0, \ldots, 0]$	0	[–500,500]

on the three discretization methods. In each experiment set, we executed ECGA with FHH or FWH for one code length slightly shorter as well as for one slightly longer than the mean SoD code length.

In the first set of experiments, for SoD, $\gamma = 0.7$, and $\epsilon = 0.99$. Under this condition, the mean SoD code length for each dimension is 13.23. We compare the results of rECGA to that of ECGA+FHH and ECGA+FWH with bin numbers 10 and 15. The mean objective values, variances, and t-test results of each function are shown in Table 2. In the other set of experiments, for SoD, $\gamma = 0.45$, and $\epsilon = 0.988$. In this case, the mean SoD code length for each dimension becomes 22.81. Accordingly, we compare the results of rECGA to that of ECGA+FHH and ECGA+FWH with bin numbers 20 and 25. The results are given in Table 3.

According to the experimental results presented in Tables 2 and 3, we first observe that SoD outperforms FHH or FWH on the eight test functions by comparing the obtained mean objective values. We can also see that SoD in general provides smaller variances of the solutions than do FHH and FWH. The t-values listed in the tables further indicate that the experimental results are statistically significant. Except for FHH-25 on F_7 (marked by * in Table 3), all t-values are greater than the critical value, 1.68, for one tail significance $\alpha = 0.05$ and degree of freedom around 50. As a consequence, we can conclude that SoD can better discretize the continuous search intervals than FHH and FWH can, at least when SoD cooperates with ECGA.

For this series of numerical experiments, because a local search method, the Nelder-Mead method, is utilized for the purpose of solution quality improvement, the integrations, including ECGA+SoD, ECGA+FHH, and ECGA+FWH, can be considered as hybridizations of global and local searchers and therefore as memetic algorithms. Readers interested in the design and development of memetic algorithms can refer to the details in the related publications (Bambha et al., 2004; Ong et al., 2006; Li and Jiang, 2000; Hart, 1994) in the literature.

5.2 Comparisons with Other Evolutionary Algorithms

In this section, we will use rECGA to solve a set of benchmark functions proposed in CEC 2005 (Suganthan et al., 2005). The benchmark also defines the evaluation criteria as well as providing the performance results for comparison. Hence, rECGA will be compared to the existing algorithms included in the special session at CEC 2005. The parameters of rECGA we use in this section are population size = 250, probability of crossover = 0.975, tournament size = 8, $\gamma = 0.5$, $\epsilon = 0.998$, and $L = 5$.

Table 2: The mean SoD code length is 13.23.

		F_1	F_2	F_3	F_4
SoD	mean	**2.84E–05**	**8.46E+00**	**1.23E–03**	**1.09E–02**
	var.	2.12E–09	5.79E+01	2.96E–07	2.44E–04
FHH	mean	6.65E+01	3.05E+01	4.32E+00	1.59E+00
10	var.	1.30E+03	1.70E+02	5.41E–01	8.12E–02
	t-value	1.30E+01	1.03E+01	4.15E+01	3.92E+01
FHH	mean	6.44E+00	1.51E+01	1.84E+00	8.71E–01
15	var.	2.52E+01	1.35E+02	3.53E–01	4.88E–02
	t-value	9.07E+00	3.40E+00	2.19E+01	2.74E+01
FWH	mean	3.56E+02	1.67E+01	8.29E+00	4.51E+00
10	var.	1.18E+04	4.64E+00	4.99E–01	6.72E–01
	t-value	2.32E+01	7.39E+00	8.30E+01	3.88E+01
FWH	mean	5.81E+01	1.06E+01	4.53E+00	1.56E+00
15	var.	6.20E+02	9.26E+00	2.47E–01	6.27E–02
	t-value	1.65E+01	1.88E+00	6.45E+01	4.37E+01
		F_5	F_6	F_7	F_8
SoD	mean	**1.03E–01**	**1.24E+00**	**4.04E+00**	**1.55E–04**
	var.	1.13E–03	1.22E+00	2.11E+00	1.03E–09
FHH	mean	1.12E+00	4.82E+00	5.88E+00	5.97E+01
10	var.	8.39E–02	4.82E+00	2.65E+00	3.59E+03
	t-value	2.48E+01	1.03E+01	5.98E+00	7.05E+00
FHH	mean	3.67E–01	3.29E+00	4.38E+00	1.12E+01
15	var.	2.90E–02	2.46E+00	1.82E+00	3.14E+02
	t-value	1.07E+01	7.58E+00	1.22E+00	4.46E+00
FWH	mean	4.86E+00	2.78E+01	2.24E+01	4.09E+02
10	var.	3.46E–01	2.91E+01	9.37E+01	1.32E+04
	t-value	5.71E+01	3.42E+01	1.33E+01	2.51E+01
FWH	mean	2.61E+00	1.76E+01	1.56E+01	4.10E+02
15	var.	5.33E–02	1.02E+01	1.35E+01	1.12E+04
	t-value	7.59E+01	3.42E+01	2.07E+01	2.73E+01

The error values, $f(x) - f(x^*)$, are presented in Table 4 for the 25 test functions. The number of dimensions for each problem is 10. The error values are recorded after 100,000 function evaluations (FEs) for each of the 25 runs. A run is considered a *success* if the final solution reaches within the given fixed accuracy level. The predefined accuracy levels are 1×10^{-6} for functions 1–5, 1×10^{-2} for functions 6–14, and 1×10^{-1} for functions 15–25. The error values of the 25 runs on one function are sorted, and the table presents the following items: the 1st (best), the 13th (median), the 25th (worst), and the average (mean). The notations * and ** put after the function number denote that the function is considered solved or rECGA obtains comparable results against other algorithms, respectively. According to the benchmark, if an algorithm successfully reaches within the given accuracy level for a function in at least one out of the 25 runs, the function is considered solved by the algorithm. "Comparable results" mean that the performance of rECGA is equally good compared to that of other algorithms.

The experimental results indicate that rECGA can solve functions 1, 2, 3, 4, 6, 7, 9, 12, 13, and 15, which are denoted with *. Functions 1 and 2 are easy and can be solved in every run. Function 3 is the shifted rotated high conditioned elliptic function, which magnifies the error of input. Even if the error of input is quite small, the error value will be huge due to a huge multiplier. By utilizing a good local search operator, rECGA

Table 3: The mean SoD code length is 22.81.

		F_1	F_2	F_3	F_4
SoD	mean	**6.86E–08**	**6.99E+00**	**2.47E–04**	**1.24E–02**
	var.	4.86E–15	2.68E+00	1.27E–08	1.34E–04
FHH	mean	2.31E+00	1.13E+01	1.16E+00	5.93E–01
20	var.	5.03E+00	1.30E+02	5.24E–01	6.67E–02
	t-value	7.27E+00	2.65E+00	1.13E+01	1.59E+01
FHH	mean	1.33E+00	8.30E+00	1.23E+00	5.55E–01
25	var.	1.46E+00	5.00E+00	4.25E–01	6.95E–02
	t-value	7.81E+00	3.36E+00	1.34E+01	1.45E+01
FWH	mean	1.02E+02	1.11E+01	5.46E+00	1.94E+00
20	var.	4.69E+02	1.57E+01	2.19E–01	2.82E–02
	t-value	3.32E+01	6.74E+00	8.25E+01	8.07E+01
FWH	mean	5.42E+01	1.11E+01	4.35E+00	1.46E+00
25	var.	1.14E+03	5.14E+01	6.16E–01	1.11E–01
	t-value	1.14E+01	3.92E+00	3.92E+01	3.08E+01
		F_5	F_6	F_7	F_8
SoD	mean	**9.35E–02**	**1.32E+00**	**2.49E+00**	**1.27E–04**
	var.	1.18E–03	1.23E+00	2.00E+00	1.22E–14
FHH	mean	2.32E–01	2.19E+00	3.71E+00	3.94E+00
20	var.	2.00E–02	1.27E+00	1.34E+00	1.53E+01
	t-value	6.74E+00	3.89E+00	4.70E+00	7.13E+00
FHH	mean	1.99E–01	1.90E+00	2.71E+00	3.52E+00
25	var.	1.08E–02	1.14E+00	1.46E+00	1.41E+01
	t-value	6.83E+00	2.66E+00	*8.21E–01	6.64E+00
FWH	mean	3.34E+00	3.72E+01	1.32E+01	9.45E+01
20	var.	1.12E–01	6.27E+01	3.94E+01	1.18E+03
	t-value	6.84E+01	3.18E+01	1.18E+01	1.94E+01
FWH	mean	1.89E+00	7.97E+00	6.50E+00	1.83E+02
25	var.	4.78E–02	2.75E+00	3.98E+00	2.15E+03
	t-value	5.75E+01	2.35E+01	1.16E+01	2.78E+01

Note: *Less than the critical value (1.68).

is able to solve function 3. Function 4 is the shifted Schwefel's problem 1.2 with noise in fitness. rECGA can solve this problem because SoD can decrease the noise effect by splitting the real number interval at random. Function 5 is Schwefel's problem 2.6 with global optimum on bounds. The special property of function 5 is that function 5 can be solved if the individuals are located at bounds. Due to SoD's characteristics, rECGA fails to satisfy the success criterion, although rECGA can provide comparable results.

Functions 6–14 are basic and expanded multimodal problems. Although rECGA cannot solve all these problems, most of the results are comparable to that of other evolutionary algorithms. The optimum of function 8 is within a very narrow valley, and the parameter setting used in this experiment does not allow rECGA to have a sufficient resolution to accurately find the valley. Functions 15–25 are composites of the basic functions, and they are big challenges to search algorithms. rECGA successfully solves only function 15. Some of the results of rECGA for functions 16–25 are comparable to other algorithms. rECGA and many other algorithms can only find the local optima. For an overall comparison, Table 5 lists the performance indexes of the algorithms compared to the benchmark in terms of the number of solved functions. We note that rECGA is ranked second, between two versions of the most advanced evolutionary algorithm for

Table 4: Error values for functions 1–25 (100,000 function evaluations).

Function	1*	2*	3*	4*	5**	6*	7*
Best	5.68E–14	5.68E–14	1.14E–13	3.45E–08	1.06E–04	1.71E–13	9.38E–13
Median	6.82E–13	1.02E–12	1.25E–12	2.33E–04	1.33E–03	1.24E–11	1.03E–01
Worst	4.43E–12	1.68E–11	1.93E–04	2.58E–02	1.54E+00	5.54E+01	5.34E–01
Mean	1.29E–12	2.11E–12	7.72E–06	2.13E–03	8.77E–02	5.77E+00	1.42E–01
Function	8**	9*	10**	11	12*	13*	14**
Best	2.00E+01	1.71E–13	2.98E+00	5.23E–01	5.68E–14	9.91E–03	2.16E+00
Median	2.00E+01	6.82E–13	7.96E+00	2.21E+00	1.76E–12	3.85E–01	3.06E+00
Worst	2.00E+01	8.01E–12	1.39E+01	5.53E+00	1.56E+03	8.54E–01	4.00E+00
Mean	2.00E+01	1.33E–12	7.84E+00	2.35E+00	6.39E+01	4.37E–01	2.97E+00
Function	15*	16**	17	18	19	20	21
Best	2.84E–14	9.43E+01	1.13E+02	3.00E+02	3.00E+02	3.00E+02	3.00E+02
Median	1.72–12	1.10E+02	1.25E+02	8.00E+02	8.00E+02	8.00E+02	8.00E+02
Worst	4.24E+02	1.30E+02	1.73E+02	9.51E+02	9.68E+02	9.28E+02	1.13E+03
Mean	1.55E+02	1.11E+02	1.30E+02	7.28E+02	6.61E+02	5.86E+02	6.87E+02
Function	22**	23**	24**	25			
Best	7.27E+02	5.59E+02	2.00E+02	4.21E+02			
Median	7.40E+02	9.71E+02	2.00E+02	4.40E+02			
Worst	8.00E+02	1.24E+03	2.00E+02	1.48E+03			
Mean	7.47E+02	8.14E+02	2.00E+02	6.35E+02			

*Considered solved according to the given accuracy.
**Comparable to the results obtained by other algorithms.

Table 5: The number of solved functions in the benchmark for various evolutionary algorithms. U: Unimodal functions (f_1 to f_5); B: Basic functions (f_6 to f_{12}); E: Expanded functions (f_{13} and f_{14}); H: Hybrid composition functions (f_{15} to f_{25}).

Algorithm	U	B	EH	Total
IPOP-CMA-ES (Auger and Hansen, 2005b)	1,2,3,4,5	6,7,9,10,11,12	—	11
rECGA	1,2,3,4	6,7,9,12,13	15	10
DMS-L-PSO (Liang and Suganthan, 2005)	1,2,3,5	6,7,9,12	15	9
LR-CMA-ES (Auger and Hansen, 2005a)	1,2,3,4,5	6,7,12	—	8
PSGES (Hsieh et al., 2007)	1,2,4	6,7,9,12,13	—	8
BLX-MA (Molina et al., 2005)	1,2,4,5	9,11,12	—	7
DE (Ronkkonen et al., 2005)	1,2,3,4,5	6,9	—	7
K-PCX (Sinha et al., 2005)	1,2,4	6,9,10,12	—	7
SPC-PNX (Ballester et al., 2005)	1,2,4,5	6,7,11	—	7
Co-EVO (Posik, 2005)	1,2,3,4	7	—	5
EDA (Yuan and Gallagher, 2005)	1,2,3,4	—	—	4

real-parameter optimization, CMA-ES (Hansen, 2006). Such a result encourages us to conduct further investigations and to apply rECGA to real-world applications in the next section. For further information, the detailed numerical results on the test functions can be found at http://www.nclab.tw/SM/2009/01.

6 Real-World Applications

After verifying the discretization capability of SoD, we are interested in putting the proposed framework, rECGA, into action. In this section, we employ rECGA to handle

the economic dispatch (ED) problem, which is an essential topic in the power system because several important facets of power systems are involved. Thanks to the importance of the ED problem, researchers have been making numerous attempts to find good solutions. Among evolutionary optimization methods for tackling the ED problem are genetic algorithms (Walters and Sheble, 1993; Sheble and Brittig, 1995; Chen and Chang, 1995; Baskar et al., 2003; Yalcinoz et al., 2001), evolutionary programming (Jayabarathia et al., 2005; Sinha et al., 2003; Yang et al., 1996; Wong and Yuryevich, 1998), and particle swarm optimization (Gaing, 2003; Victoire and Jeyakumar, 2004a; Park et al., 2005; Victoire and Jeyakumar, 2004b). In order to obtain superior solutions, we apply rECGA on the ED problem in this section. First, we briefly introduce the ED problem, and then the high quality solutions offered by rECGA are presented.

6.1 The Problem: Economic Dispatch

With the development of modern power systems, the economic dispatch (ED) problem has been receiving increased attention. The ED problem is essential for the real-time control of power system operations. It consists of allocating the total generation required among the available thermal generating units, assuming that a thermal unit commitment is previously determined. The problem aims to minimize the fuel cost subject to the physical and operational constraints. As a result, the ED problem is to find the optimal combination of generations that minimizes the total generation cost while satisfying the specified constraints. In order to model the ED problem, a simplified cost function (Allen and Bruce, 1984) of each generator, represented as a quadratic function, can be given as:

$$C = \sum_{j \in J} F_j(P_j),$$

$$F_j(P_j) = a_j P_j^2 + b_j P_j + c_j,$$

where C is the total generation cost; J is the set for all generators; P_j is the electrical output of generator j; F_j is the cost function for generator j; and a_j, b_j, and c_j are the cost coefficients for generator j.

In the real world, the total generation should be equal to the total system demand plus the transmission network loss. However, in this study, the network loss is not considered for simplicity as in many previous studies. Thus, the constraints of the problem include two parts. The first part is the equality constraint. The total system power demand must be equal to the summation of the output of each generator:

$$D = \sum_{j \in J} P_j, \tag{3}$$

where D is the total system demand.

Moreover, the generation output of each unit should be within the respective minimum and maximum limits. Such a condition introduces the inequality constraint for each generation unit, such as for generator j:

$$P_{j\,\min} \leq P_j \leq P_{j\,\max},$$

where $P_{j\,\min}$ and $P_{j\,\max}$ are the minimum and maximum output of generator j, and P_j is the desired output for generator j.

In reality, the objective function of economic dispatch is more complicated due to the valve point effects, the change of fuels, and other potential practical factors. Therefore, nonsmooth cost functions should be taken into consideration instead of the simplistic form of Equation (3). The inclusion of the valve point effects makes the modeling of the incremental fuel cost function of the generation units more practical. Such a modification increases the nonlinearity as well as the number of local optima in the search space. Hence, the employed search algorithm may be trapped at the local optima more easily. The incremental fuel cost function of the generator with the valve point effects can be given as (Walters and Sheble, 1993)

$$F_j(P_j) = a_j P_j^2 + b_j P_j + c_j + |e_j \sin(f_j \times (P_{jmin} - P_j))|, \tag{4}$$

where e_j and f_j are the coefficients to reflect the valve point loading effects.

In this study, we focus on solving the ED problem with the valve point loading effects, which is modeled in Equation (4). We employed rECGA as an optimization tool. The equality and inequality constraints are handled through a repair operator. Descriptions in detail are given in the following section.

6.2 Our Solution: rECGA for Economic Dispatch

The integration of ECGA and SoD, rECGA, can generally handle global optimization problems. However, certain extra efforts have to be made for employing rECGA to tackle the ED problem. Among the most important topics for solving the ED problem may be the equality and inequality constraints. These constraints divide the problem search space into complicated areas. This condition renders the optional local searcher (the Nelder-Mead method, included in the framework of rECGA), inefficient and ineffective (Nelder and Mead, 1965). As a consequence, the optional local search operator is not applied when rECGA is used to solve the ED problem.

Furthermore, in order to deal with the constraints, based on the repair concept, we developed a constraint handling technique specifically for the ED problem. Repairing solutions stands for transforming infeasible solutions into feasible ones via a certain procedure. For the equality constraint, Equation (3) in the ED problem, we repair infeasible solutions in the following way. Firstly, we create a sequence from 1 to the number of generator in a random order. Each number in the sequence represents one designated generator. The sequence indicates the order in which we adjust the output of the generation for making a solution feasible. For example, if the generated sequence is 4, 2, 1, 5, 3, we will first process generator 4, then generator 2, and so on. To process a generator, we check the equality constraint, that is, the sum of the output has to be equal to the total power demand. If the equality constraint is not satisfied, the output of the generator under processing is modified according to:

$$P_i' = \min(\text{UBound}(P_i), \max((D - \sum_{j=1, j\neq i}^{n} P_j), \text{LBound}(P_i))), \tag{5}$$

where D is the system power demand, $\text{LBound}(P_i)$ and $\text{UBound}(P_i)$ are the lower bound and upper bound of P_i, that is, the inequality constraint of P_i.

Table 6: Parameters for test case I (three-unit system) with the valve point loading effect. a, b, c, e, and f are the cost coefficients in the fuel cost function: $F_j(P_j) = a_j P_j^2 + b_j P_j + c_j + |e_j \sin(f_j \times (P_{j\min} - P_j))|$.

Generator	$P_{\min}$(MW)	$P_{\max}$(MW)	a	b	c	e	f
1	100	600	0.001562	7.92	561	300	0.0315
2	100	400	0.00482	7.97	78	150	0.063
3	50	200	0.00194	7.85	310	200	0.042

The proposed algorithm, rECGA, incorporating the constraint handling technique is capable of tackling the ED problem effectively. For simplicity, the optional local search method is not used. With the adoption of the proposed repair mechanism, rECGA for solving the ED problem can be outlined as:

1. Initialize a population of N individuals at random according to the constraints posed to the generator output.
2. Apply tournament selection of size S.
3. Use SoD to encode each dimension of the continuous variables.
4. Model the selected, encoded individuals with a greedy MPM search.
5. Stop if the MPM model has converged.
6. Generate a new population by sampling the built MPM model.
7. Repair the infeasible solutions in the population.
8. Return to Step 2.

In order to observe the effectiveness and to verify the performance of rECGA for ED, two ED problem instances, one consisting of three generators and the other consisting of 40 generators, serve as a testbed. The experimental results on the two ED problems are presented in the next section.

6.3 Verification: Numerical Experiments

We focus on solving the ED problem with nonsmooth cost functions considering the valve point loading effect for verifying the utility of the proposed framework, rECGA. The nonsmooth cost functions were described as Equation (4). In order to examine the performance, rECGA for ED was applied to two ED problems that were adopted as test problems in the literature (Walters and Sheble, 1993; Sinha et al., 2003) for the purpose of comparison. One consists of three generation units, and the other consists of 40 generation units. The input data for the three-generator system are given by Walters and Sheble (1993), and those for the 40-generator system are given by Sinha et al. (2003). The detailed problem parameters for the two test problems, including the lower bound and upper bound for the output of each generator as well as the coefficients for computing the cost functions, are given in Tables 6 and 7. The total system demand for the three-unit system is 850 MW, and that for the 40-unit system is 10,500 MW. It has been proven that for the three-unit system, the global optimal solution is 8234.07 (Lin et al.,

Table 7: Parameters for test case II (40-unit system) with the valve point loading effect. a, b, c, e, and f are the cost coefficients in the fuel cost function: $F_j(P_j) = a_j P_j^2 + b_j P_j + c_j + |e_j \sin(f_j \times (P_{j\min} - P_j))|$.

Generator	$P_{\min}$(MW)	$P_{\max}$(MW)	a	b	c	e	f
1	36	114	0.0069	6.73	94.705	100	0.084
2	36	114	0.0069	6.73	94.705	100	0.084
3	60	120	0.2028	7.07	309.54	100	0.084
4	80	190	0.00942	8.18	369.03	150	0.063
5	47	97	0.0114	5.35	148.89	120	0.077
6	68	140	0.01142	8.05	222.33	100	0.084
7	110	300	0.00357	8.03	287.71	200	0.042
8	135	300	0.00492	6.99	391.98	200	0.042
9	135	300	0.00573	6.6	455.76	200	0.042
10	130	300	0.00605	12.9	722.82	200	0.042
11	94	375	0.00515	12.9	635.2	200	0.042
12	94	375	0.00569	12.8	654.69	200	0.042
13	125	500	0.00421	12.5	913.4	300	0.035
14	125	500	0.00752	8.84	1,760.4	300	0.035
15	125	500	0.00708	9.15	1,728.3	300	0.035
16	125	500	0.00708	9.15	1,728.3	300	0.035
17	220	500	0.00313	7.97	647.85	300	0.035
18	220	500	0.00313	7.95	649.69	300	0.035
19	242	550	0.00313	7.97	647.83	300	0.035
20	242	550	0.00313	7.97	647.81	300	0.035
21	254	550	0.00298	6.63	785.96	300	0.035
22	254	550	0.00298	6.63	785.96	300	0.035
23	254	550	0.00284	6.66	794.53	300	0.035
24	254	550	0.00284	6.66	794.53	300	0.035
25	254	550	0.00277	7.1	801.32	300	0.035
26	254	550	0.00277	7.1	801.32	300	0.035
27	10	150	0.52124	3.33	1,055.1	120	0.077
28	10	150	0.52124	3.33	1,055.1	120	0.077
29	10	150	0.52124	3.33	1,055.1	120	0.077
30	47	97	0.0114	5.35	148.89	120	0.077
31	60	190	0.0016	6.43	222.92	150	0.063
32	60	190	0.0016	6.43	222.92	150	0.063
33	60	190	0.0016	6.43	222.92	150	0.063
34	90	200	0.0001	8.95	107.87	200	0.042
35	90	200	0.0001	8.62	116.58	200	0.042
36	90	200	0.0001	8.62	116.58	200	0.042
37	25	110	0.0161	5.88	307.45	80	0.098
38	25	110	0.0161	5.88	307.45	80	0.098
39	25	110	0.0161	5.88	307.45	80	0.098
40	242	550	0.00313	7.97	647.83	300	0.035

2002). As for the 40-unit system, the global optimal solution has not been determined. To the best of our limited knowledge, the known best solution previously obtained by other methods is 122,252.265 as reported by Park et al. (2005) with MPSO.

The parameter settings in rECGA for ED are population size = 400, crossover probability = 0.975, tournament size = 8, $\gamma = 0.5$, $\epsilon = 0.999$, and the maximum function

Table 8: Comparison of the results obtained by various methods on the nonsmooth cost function considering the valve point loading effect. For the three-unit system, EP, CMA-ES, MPSO, and rECGA were able to find the global optimum (Lin et al., 2002).

Generator	GA	IEP (pop = 20)	EP	MPSO (par = 20)	CMA-ES	rECGA
1	300	300.23	300.26	300.27	300.27	300.27
2	400	400	400	400	400	400
3	150	149.77	149.74	149.73	149.73	149.73
TP	850	850	850	850	850	850
TC	8237.6	8234.09	**8234.07**	**8234.07**	**8234.07**	**8234.07**

Table 9: Comparison of the results obtained by various methods on the nonsmooth cost function considering the valve point loading effect. For the 40-unit system, rECGA was able to find the best solution.

Minimum Cost	Method
123,488.3	CEP
122,679.7	FEP
122,647.6	MFEP
122,624.35	IFEP
122,252.27	MPSO
122,160.19	CMA-ES
121,462.3591	rECGA

evaluations is 200,000. One hundred independent trials were conducted for each problem to collect statistically significant results. The obtained results for the three-unit system are shown in Table 8 and are compared to those obtained by IEP (Park et al., 1998), EP (Yang et al., 1996), MPSO (Park et al., 2005), and CMA-ES (Hansen, 2006).[1] The results for this small ED problem indicate that rECGA was able to find the global optimal solution determined by Lin et al. (2002).

In the case of the 40-unit system, the results are compared with those obtained by using other methods given by Sinha et al. (2003) such as classical EP (CEP), fast EP (FEP), modified FEP (MFEP), and improved FEP (IFEP). The results for MPSO provided in Park et al. (2005) and obtained by CMA-ES (Hansen, 2006) are also included. The minimum costs, that is, the best solutions, achieved by these methods are presented in Table 9. We can see that the best solution delivered by rECGA is 121,462.3591, which is better than the known, published best solution, 122,252.27, presented by Park et al. (2005) and that obtained by CMA-ES, 122,160.19. For the purpose of access and verification, the generation outputs (the values of the decision variables) and the corresponding cost (the objective values) of the solution found by rECGA are given in Table 10.

Because of the stochastic nature of evolutionary computation methods, to avoid reporting the results of a "lucky shot," comparison of the experimental results in a statistical manner has to be conducted. First of all, Table 11 shows the range of the results in the 100 trials obtained by CEP, FEP, MFEP, IFEP, MPSO, CMA-ES, and rECGA, where the listed results except for those of rECGA and CMA-ES are given elsewhere

[1] We downloaded the source code and conducted the experiments for CMA-ES.

Table 10: The generator outputs and the costs of the best solution obtained by rECGA.

Generator	$P_{\min}$(MW)	$P_{\max}$(MW)	$Output$	Cost
1	36	114	110.80098	925.11565
2	36	114	110.88806	926.56631
3	60	120	97.40449	1,190.63739
4	80	190	179.73300	2,143.55011
5	47	97	96.15215	840.66343
6	68	140	140.00000	1,596.46432
7	110	300	299.99898	3,216.41474
8	135	300	284.62219	2,780.24662
9	135	300	284.61234	2,798.46198
10	130	300	130.00001	2,502.06532
11	94	375	94.00003	1,893.30606
12	94	375	94.00027	1,908.17291
13	125	500	214.76169	3,792.11715
14	125	500	394.27878	6,414.85790
15	125	500	304.52026	5,171.21428
16	125	500	394.28449	6,436.71537
17	220	500	489.27966	5,296.71703
18	220	500	489.27855	5,288.76474
19	242	550	511.27996	5,540.94200
20	242	550	511.28163	5,540.95823
21	254	550	523.28030	5,071.30855
22	254	550	523.28419	5,071.38735
23	254	550	523.28495	5,057.33548
24	254	550	523.28151	5,057.26621
25	254	550	523.28214	5,275.14526
26	254	550	523.27977	5,275.09678
27	10	150	10.00013	1,140.52698
28	10	150	10.00517	1,140.64280
29	10	150	10.00018	1,140.52812
30	47	97	87.84287	707.21302
31	60	190	189.99927	1,643.98840
32	60	190	189.99996	1,643.99109
33	60	190	189.99993	1,643.99098
34	90	200	199.99994	2,101.01644
35	90	200	199.99993	2,043.72638
36	90	200	199.99972	2,043.72436
37	25	110	110.00000	1,220.16612
38	25	110	109.99978	1,220.16484
39	25	110	109.99871	1,220.15859
40	242	550	511.28401	5,541.02984
Total generation and total cost			10,500	121,462.3591

(Sinha et al., 2003; Park et al., 2005). As we observe from Table 11, the distribution of the rECGA results may be considered better than those obtained by the other methods.

Furthermore, to more carefully and accurately compare the performance of rECGA and MPSO (Park et al., 2005) on the 40-unit problem, the t-test was conducted for the statistical significance of the obtained experimental results. Since the actual results of the 100 trials for MPSO are not available, in order to get a fair performance comparison and capability assessment, we set up two conditions under which the t-test was conducted.

Table 11: Comparison of methods on relative frequency of convergence in the cost range.

	Range of cost											
Method	126.5–∞	126.0–126.5	125.5–126.0	125.0–125.5	124.5–125.0	124.0–124.5	123.5–124.0	123.0–123.5	122.5–123.0	122.0–122.5	121.5–122.0	121.0–121.5
CEP	10	4	—	16	22	42	4	2	—	—	—	—
FEP	6	—	4	2	10	20	26	24	6	—	—	—
MFEP	—	—	—	—	—	14	26	50	10	—	—	—
IFEP	—	—	2	—	4	4	18	50	22	—	—	—
MPSO	—	—	—	—	—	—	—	—	53	47	—	—
CMA-ES	9	1	6	12	19	19	24	4	5	1	—	—
rECGA	—	—	—	—	—	—	—	—	—	2	97	1

Table 12: The t-test for the experimental results obtained by rECGA and MPSO under condition 1, where the rECGA dataset contains the actual results, and the MPSO dataset contains 47 values of 122,252.265 and 53 values of 122,750.

	rECGA	MPSO
mean	121,777.649963	122,516.06455
t-value	27.8068829451749	
p-value	2.2645299161711E–55	

Table 13: The t-test for the experimental results obtained by rECGA and MPSO under condition 2, where the rECGA dataset contains the actual results, and the MPSO dataset contains 47 values of 122,252.265 and 53 values of 122,500.

	rECGA	MPSO
mean	121,777.649963	122,383.56455
t-value	39.4214198098397	
p-value	9.0857670116394E–91	

Based on the data given in Table 11, the first condition is that the MPSO results contain 47 values of 122,252.265, which is the optimum reported for MPSO (Park et al., 2005), and 53 values of 122,750, which is the mean value of 122,500 and 123,000. Table 12 demonstrates the t-test results for condition 1. Given the p-value: 2.26×10^{-55}, which is smaller than the commonly used statistically significant levels, such as .05 (5%), .01 (1%), or .001 (0.1%), we can conclude that the performance of rECGA on the 40-unit ED problem is statistically significantly better than that of MPSO on the same problem. For condition 2, the MPSO results contain 47 values of 122,252.265, which is the optimum reported for MPSO (Park et al., 2005), and 53 values of 122,500, which is the best value in the range from 122,500 to 123,000. The t-test results under condition 2 are presented in Table 13. Due to the change of the standard deviation, the p-value becomes 9.09×10^{-91}. The small p-value prevents us from accepting the null hypothesis, which is interpreted as that the performance of rECGA and MPSO on the problem is equivalent.

According to the experimental results, we know that the proposed algorithm, rECGA (ECGA+SoD), performs well on the two ED problems. Particularly, for the 40-unit ED problem, we improved the known best solution from 122,252.265 (Park et al., 2005) to 121,462.3591. Moreover, from Tables 11, 12, and 13, we observe that rECGA statistically significantly outperformed MPSO on the 40-unit ED problem. Therefore, rECGA is capable of solving ED problems effectively.

7 Summary and Conclusions

In this study, we proposed an adaptive discretization method, SoD, to enable EDAs designed for handling discrete variables to tackle real-parameter optimization. SoD was described in detail with its procedure, effect, and usage. In order to show the utility of SoD, an ECGA was employed as an optimization engine, and SoD was used as a variable-type interface. By combining ECGA and SoD, the real-coded ECGA (rECGA) were applied to solve a set of benchmark functions and two ED problems. The results on benchmark functions indicated that SoD was better than two well-known discretization methods, FHH and FWH. The results on the ED problems demonstrated that rECGA successfully achieved the global optimal solution of the three-unit ED problem and

was able to obtain the solutions better than the known best solution obtained by other methods and reported in the literature for the 40-unit ED problem.

The outcome of this study indicates that it is not only possible but also practical to employ an optimization method designed for handling discrete variables to tackle problems consisting of continuous variables, as long as an appropriate interface is adopted. Although many researchers in the field of evolutionary computation do not consider variable-type transformation to be an issue, in practice, except for some limited cases, most algorithms designed for discrete variables do not perform well on continuous problems and vice versa. By comparing the real-coded ECGA to the algorithms specifically designed for handling continuous variables, such as particle swarm optimization (MPSO), evolutionary programming (IFEP, MFEP, FEP, CEP), and evolution strategies (CMA-ES), this paper provides the experimental results to serve as the proof of principle for transforming the variable type while retaining the capability of the back end optimization algorithm.

Given the nature of discretization, although good results were obtained in this study, we believe that the proposed framework might not be suitable for applications that require very high precision. As we can see in Table 5 in Section 5.2, rECGA seems to perform as well as the state of the art method, CMA-ES. However, for the benchmark, since there is a predefined accuracy level for each function (1×10^{-6}, 1×10^{-2}, and 1×10^{-1}), the experimental results in terms of the number of solved functions should be appropriately interpreted instead of being considered to demonstrate that rECGA can perform very well under any situations. For applications that do not require very high precision, such as the control of robots, evolutionary arts, or machine learning, the proposed discretization technique may come in handy for handling real-parameter problems with the discrete-type optimization algorithm that is already in use.

Finally, future work of this study includes applying rECGA to handle other important problems as well as developing different integrations of optimization algorithms and transforming techniques. Moreover, theoretical understandings for the quality of the transforming techniques, such as SoD, FHH, and FWH, as well as for the interaction between the engine and the interface, should also be considered.

Acknowledgments

The work was supported in part by the National Science Council of Taiwan under Grant NSC-96-2221-E-009-196. The authors are grateful to the National Center for High-Performance Computing for computer time and facilities.

References

Ahn, C. W., Ramakrishna, R. S., and Goldberg, D. E. (2004). Real-coded Bayesian optimization algorithm, bringing the strength of BOA into the continuous world. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2004)*, pp. 840–851.

Allen, J. W., and Bruce, F. W. (1984). *Power generation, operation, and control*. New York: Wiley.

Auger, A., and Hansen, N. (2005a). Performance evaluation of an advanced local search evolutionary algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pp. 1777–1784.

Auger, A., and Hansen, N. (2005b). A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pp. 1769–1776.

Ballester, P. J., Stephenson, J., Carter, J. N., and Gallagher, K. (2005). Real-parameter optimization performance study on the CEC-2005 benchmark with SPC-PNX. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pp. 498–505.

Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Baluja, S., and Davies, S. (1997). Using optimal dependency-trees for combinational optimization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 30–38.

Bambha, N. K., Bhattacharyya, S. S., Teich, J., and Zitzler, E. (2004). Systematic integration of parameterized local search into evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):137–154.

Baskar, S., Subbaraj, P., and Rao, M. V. C. (2003). Hybrid real coded genetic algorithm solution to economic dispatch problem. *Computers and Electrical Engineering*, 29(3):407–419.

Bosman, P. A., and Thierens, D. (2000). Continuous iterated density estimation evolutionary algorithms within the idea framework. In *Workshop Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2000)*, pp. 197–200.

Cantú-Paz, E. (2001). Supervised and unsupervised discretization methods for evolutionary algorithms. In *Workshop Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 213–216.

Chen, C.-H., Liu, W.-N., and Chen, Y.-p. (2006). Adaptive discretization for probabilistic model building genetic algorithms. In *Proceedings of ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2006)*, pp. 1103–1110.

Chen, P. H., and Chang, H. C. (1995). Large-scale economic-dispatch by genetic algorithm. *IEEE Transactions on Power Systems*, 10(4):1919–1926.

de Bonet, J., Isbell, C., and Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems*, 9:424–430.

Etxeberria, R., and Larrañaga, P. (1999). Global optimization using Bayesian networks. In *Proceedings of the Second Symposium on Artificial Intelligence*, pp. 332–339.

Gaing, Z. L. (2003). Particle swarm optimization to solving the economic dispatch considering the generator constraints. *IEEE Transactions on Power Systems*, 18(3):1187–1195.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*, Vol. 7 of *Genetic algorithms and evolutionary computation*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Goldberg, D. E., Korb, B., and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530.

Hansen, N. (2006). The CMA evolution strategy: A comparing review. In J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea (Eds.), *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, Vol. 192 of *Studies in fuzziness and soft computing*, pp. 75–102. Berlin: Springer.

Harik, G. R. (1999). Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana.

Harik, G. R., Lobo, F. G., and Goldberg, D. E. (1999). The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287.

Hart, W. E. (1994). Adaptive global optimization with local search. PhD thesis, University of California, San Diego.

Hsieh, C.-T., Chen, C.-M., and Chen, Y.-p. (2007). Particle swarm guided evolution strategy. In *Proceedings of ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2007)*, pp. 650–657.

Jayabarathia, T., Jayaprakasha, K., Jeyakumarb, D. N., and Raghunathan, T. (2005). Evolutionary programming techniques for different kinds of economic dispatch problems. *Electric Power Systems Research*, 73(2):169–176.

Larrañaga, P., and Lozano, J. A. (2001). *Estimation of distribution algorithms: A new tool for evolutionary computation*, Vol. 2 of *Genetic algorithms and evolutionary computation*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Li, B., and Jiang, W. (2000). A novel stochastic optimization algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, 30(1):193–198.

Liang, J. J., and Suganthan, P. N. (2005). Dynamic multi-swarm particle swarm optimizer with local search. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pp. 522–528.

Lin, W. M., Cheng, F. S., and Tsay, M. T. (2002). An improved TABU search for economic dispatch with multiple minima. *IEEE Transactions on Power Systems*, 17(1):108–112.

Molina, D., Herrera, F., and Lozano, M. (2005). Adaptive local search parameters for real-coded memetic algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pp. 888–895.

Mühlenbein, H., and Höns, R. (2005). The estimation of distributions and the minimum relative entropy principle. *Evolutionary Computation*, 13(1):1–27.

Mühlenbein, H., and Mahnig, T. (1999). FDA: A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376.

Mühlenbein, H., and Paaß, G. (1996). From recombination of genes to the estimation of distributions I. binary parameters. In *Proceedings of PPSN IV*, pp. 178–187.

Nelder, J. A., and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7:308–315.

Ong, Y. S., Lim, M. H., Zhu, N., and Wong, K. W. (2006). Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 36(1):141–152.

Park, J. B., Lee, K. S., Shin, J. R., and Lee, K. Y. (2005). A particle swarm optimization for economic dispatch with nonsmooth cost functions. *IEEE Transactions on Power Systems*, 20(1):34–42.

Park, Y.-M., Won, J. R., and Park, J. B. (1998). New approach to economic load dispatch based on improved evolutionary programming. *Engineering Intelligent Systems for Electrical Engineering and Communications*, 6(2):103–110.

Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-99)*, pp. 525–532.

Pelikan, M., Goldberg, D. E., and Lobo, F. G. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20.

Pelikan, M., Goldberg, D. E., and Tsutsui, S. (2003). Getting the best of both worlds: Discrete and continuous genetic and evolutionary algorithms in concert. *Information Science*, 156:147–171.

Pelikan, M., and Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In *Advances in Soft Computing*, pp. 521–535.

Posik, P. (2005). Real parameter optimization using mutation step co-evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pp. 872–879.

Rissanen, J. (1989). *Stochastic complexity in statistical inquiry*. River Edge, NJ: World Scientific.

Ronkkonen, J., Kukkonen, S., and Price, K. V. (2005). Real-parameter optimization with differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pp. 506–513.

Sebag, M., and Ducoulombier, A. (1998). Extending population-based incremental learning to continuous search spaces. In *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature (PPSN V)*, pp. 418–427.

Servet, I. L., Trave-Massuyes, L., and Stern, D. (1997). Telephone network traffic overloading diagnosis and evolutionary computation techniques. In *Proceedings of the Third European Conference on Artificial Evolution (AE 97)*, pp. 137–144.

Sheble, G. B., and Brittig, K. (1995). Refined genetic algorithm—Economic-dispatch example. *IEEE Transactions on Power Systems*, 10(1):117–124.

Shin, S.-Y., and Zhang, B.-T. (2001). Bayesian evolutionary algorithms for continuous function optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001)*, pp. 508–515.

Sinha, A., Tiwari, S., and Deb, K. (2005). A population-based, steady-state procedure for real-parameter optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pp. 514–521.

Sinha, N., Chakrabarti, R., and Chattopadhyay, P. K. (2003). Evolutionary programming technique for economic load dispatch. *IEEE Transactions on Evolutionary Computation*, 7(1):83–94.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-p., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore.

Tsutsui, S., Pelikan, M., and Goldberg, D. E. (2001). Evolutionary algorithm using marginal histogram models in continuous domain. IlliGAL Report No. 2001019, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana.

Victoire, T. A. A., and Jeyakumar, A. E. (2004a). Hybrid PSO-SQP for economic dispatch with valve-point effect. *Electric Power Systems Research*, 71(1):51–59.

Victoire, T. A. A., and Jeyakumar, A. E. (2004b). Particle swarm optimization to solving the economic dispatch considering the generator constraints. *IEEE Transactions on Power Systems*, 19(4):2121–2122.

Walters, D. C., and Sheble, G. B. (1993). Genetic algorithm solution of economic dispatch with valve point loading. *IEEE Transaction on Power Systems*, 8(3):1325–1332.

Wong, K. P., and Yuryevich, J. (1998). Evolutionary-programming-based algorithm for environmentally-constrained economic dispatch. *IEEE Transactions on Power Systems*, 13(2):301–306.

Yalcinoz, T., Altun, H., and Uzam, M. (2001). Economic dispatch solution using a genetic algorithm based on arithmetic crossover. Paper presented at IEEE Power Tech Conference, Porto, Portugal.

Yang, H. T., Yang, P. C., and Huang, C. L. (1996). Evolutionary programming based economic dispatch for units with non-smooth fuel cost functions. *IEEE Transactions on Power Systems*, 11(1):112–117.

Yuan, B., and Gallagher, M. (2005). Experimental results for the special session on real-parameter optimization at CEC 2005: A simple, continuous EDA. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pp. 1792–1799.

This article has been cited by:

1. Vui Ann Shim, Kay Chen Tan, Jun Yong Chia, Abdullah Al Mamun. 2013. Multi-Objective Optimization with Estimation of Distribution Algorithm in a Noisy Environment. *Evolutionary Computation* **21**:1, 149-177. [Abstract] [Full Text] [PDF] [PDF Plus]
2. Mark Hauschild, Martin Pelikan. 2011. An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation* . [CrossRef]
3. Martin PelikanGenetic Algorithms . [CrossRef]