A Generalized Approach to Construct Benchmark Problems for Dynamic Optimization

Changhe Li and Shengxiang Yang

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
{cl160, s.yang}@mcs.le.ac.uk

Abstract. There has been a growing interest in studying evolutionary algorithms in dynamic environments in recent years due to its importance in real applications. However, different dynamic test problems have been used to test and compare the performance of algorithms. This paper proposes a generalized dynamic benchmark generator (GDBG) that can be instantiated into the binary space, real space and combinatory space. This generator can present a set of different properties to test algorithms by tuning some control parameters. Some experiments are carried out on the real space to study the performance of the generator.

1 Introduction

In recent years, there has been a growing interest in studying evolutionary algorithms for dynamic optimization problems (DOPs) due to its importance in real world applications. In order to study the performance of EAs in dynamic environments, one important task is to develop proper dynamic benchmark problems. Over the years, researchers have applied a number of dynamic test problems to compare the performance of EAs in dynamic environments. Generally speaking, they can be roughly divided into two types.

For the first type, the environment is just switched between several stationary problems or several states of a problem. For example, many researchers tested their approaches on the dynamic knapsack problem where the weight capacity of the knapsack changes over time, usually oscillating between two or more fixed values [3, 7]. Cobb and Grefenstette [2] used a significantly changing environment that oscillates between two different landscapes.

The second type of DOP generators construct dynamic environments by reshaping a predefined fitness landscape. For example, Branke [1] suggested a dynamic benchmark problem, called the "moving peaks" benchmark (MPB) problem. It consists of a multi-dimensional landscape with several peaks, where the height, width and position of each peak is altered a little every time the environment changes. This function is capable of generating a given number of peaks in a given number of dimensions that vary both spatially (position and shape of a peak) and in terms of fitness. Morrison and De Jong [10] also defined a similar dynamic generator as the MPB problem. Yang and Yao [12–14]

proposed a DOP generator that can generate dynamic environments from any binary encoded stationary problem using a bitwise exclusive-or (XOR) operator.

Though a number of DOP generators exist in the literature, there is no a unified approach of constructing dynamic problems across the binary space, real space, and combinatory space so far. This paper proposes a generalized dynamic benchmark generator (GDBG) to construct dynamic environments for all the three solution spaces. GDBG provides six properties of the environmental dynamics, which are random change, small step change, large step change, chaotic change, recurrent change, and recurrent change with noisy. Especially, in the real space, we introduce a rotation method instead of shifting the positions of peaks as in [10] and [1]. The rotation method can overcome the problem of unequal challenge per change for algorithms of the MPB generator, which happens when the peak positions bounce back from the boundary of the landscape. In this paper, some experiments are carried out on the real space using the particle swarm optimizaiton (PSO) [4, 6] algorithm and fast evolutionary programming (FEP) [15] to test the performance of the GDBG system.

The rest of the paper is organized as follows. Section 2 describes the GDBG system in details. Section 3 presents several instances of the GDBG in the binary space, the real space and the combinatory space. Then some experiments based on the real space are carried out to test the performance of the GDBG system in Section 4. Finally, Section 5 concludes the paper.

2 The Generalized Dynamic Benchmark Generator

In this section, we first define DOPs and then introduces the GDBG system. DOPs can be defined as follows:

$$F = f(x, \phi, t) \tag{1}$$

where F is the optimization problem, f is the cost function, x is a feasible solution in the solution set $\mathbf{X}$, t is the real-world time, and ϕ is the system control parameter, which determines the solution distribution in the fitness landscape. The objective is to find a global optimal solution x^* such that $f(x^*) \leq f(x) \forall x \in \mathbf{X}$ (without loss of generality, minimization problems are considered in the paper).

First, we classify the environmental changes into two categories: the dimensional changes and the non-dimensional changes. Dimensional changes correspond to adding or removing variables from the optimization problem. For example, the number of cities increases or decreases in the traveling salesman problem (TSP), the dimensions in the function optimization problem and the number of objects in the knapsack problems increases or decreases. This kind of changes requires the alternation of the representation of solutions.

Non-dimensional changes result from the change of the value of variables within the problem constraints. For example, the capacity of the knapsack changes and the weight or profit of objects changes in the knapsack problem, the positions of cities change in the TSP, the position of peaks change in the function optimization problem, and the processing time and ready date change in a scheduling

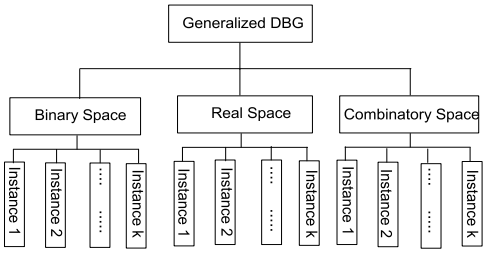

Fig. 1. Model of the GDBG system.

problem. Non-dimensional changes generally can be regarded harder than dimensional changes. We just simply delete or add variables when dimensional changes occur. While for non-dimensional changes, some special techniques need to be taken to adapt algorithms to this kind of changes. In this paper, we only consider non-dimensional changes in the GDBG system.

In the GDBG system, the dynamism results from a deviation of solution distribution from the current environment by tuning the system control parameters. It can be described as follows:

$$\phi(t+1) = \phi(t) \oplus \Delta\phi \tag{2}$$

where $\Delta\phi$ is a deviation from the current system control parameters. Then, we can get the new environment at the next moment $t+1$ as follows:

$$f(x, \phi, t+1) = f(x, \phi(t) \oplus \Delta\phi, t) \tag{3}$$

The system control parameters decide the distribution of solutions in the solution space. They may be different from one specific instance to another instance. For example, the only distance matrix determines the solution distribution in TSP, while the capacity of the knapsack and the weights and profits of objects together determine the solution distribution in the knapsack problem. The GDBG system constructs dynamic environments by changing the values of these system control parameters. Fig. 1 shows the model of the GDBG system.

There are six change types of the system control parameters in the GDBG system. They are small step change, large step change, random change, chaotic change, recurrent change, and recurrent change with noise. By controlling the values of the system control parameters using the six change types, GDBG can present six different dynamic properties. This easily enables the test and comparison of the adaptibility of algorithms in different dynamistic types. The framework of the six change types are described as follows:

Framework of DynamicChanges
switch(change type)
  case small step: $\Delta\phi = \alpha \cdot \|\phi\| \cdot rand()$
  case large step: $\Delta\phi = \|\phi\| \cdot (\alpha + (1-\alpha)rand())$

case random: $\Delta\phi = \|\phi\| \cdot rand()$
case chaotic: $\phi(t+1) = A \cdot \phi(t) \cdot (1 - \phi(t)/\|\phi\|)$
case recurrent: $\phi(t+1) = \phi(t\%P)$
case recurrent with nosy: $\phi(t+1) = \phi(t\%P) + \alpha \cdot \|\phi\| \cdot rand()$

where $\|\phi\|$ is the change range of ϕ, $\alpha \in (0, 1)$ is a constant of the step severity, which is set to 0.01 in the GDBG system. A logistics function is used in the chaotic change type, where A is a positive constant between $(1.0, 4.0)$. P is the periodicity of recurrent change and recurrent change with noise, $rand()$ is a random number in $(0, 1)$. By changing the value of the system control parameters according to the above six change types, we can easily obtain a new problem with corresponding properties for any real specific problems.

In the following section, some specific instances from the GDBG system in the binary space, real space, and combinatory space are described respectively.

3 Generator Instances from the GDBG system

3.1 Generator Instance in the Binary Space

The XOR DOP generator proposed in [12–14] can generate DOPs from any binary encoded stationary problem. Given a stationary problem $f(\boldsymbol{x}, \phi)(\boldsymbol{x} \in \{0, 1\}^l)$, where l is the length of the binary representation, $\phi \in [0, l]$ is the number of ones in x. In the GDBG system, we can also use the XOR operator to construct a new environment from the current fitness landscape as follows:

Step 1. $\phi(t+1)$=DynamicChanges($\phi(t)$)
Step 2. Generate a binary string $\boldsymbol{m}(t)$ of length l containing $\phi(t+1)$ ones
Step 3. $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) \oplus \boldsymbol{m}(t)$

where "$\oplus$" is a XOR operator, i.e., $1 \oplus 1 = 0, 1 \oplus 0 = 1, 0 \oplus 0 = 0$. The difference between $\phi(t+1)$ and $\phi(t)$ controls the severity of each environmental change.

3.2 Generator Instances in the Real Space

In this section, two different real DBGs will be constructed using different methods. In [1] and [10], two real DBGs were constructed that contain several peaks. The height, width and position of each peak may change every time an environmental change occurs. They have been tested by many researchers. However, both have a disadvantages of unequal challenge per change for algorithms when the position of a peak bounces back from the search boundary. This paper proposes a rotation method for the peak position to overcome that shortcoming.

Real Rotation DBG

The fitness landscape of the rotation DBG also consists of several peaks that can be artificially controlled. The height, width, and position of each peak are system control parameters, which are altered according to the above six change

types. Given a problem $f(x, \phi, t), \phi = (\boldsymbol{H}, \boldsymbol{W}, \boldsymbol{X})$, where $\boldsymbol{H}, \boldsymbol{W}$, and $\boldsymbol{X}$ denote the peak height, width and position respectively. The function of $f(x, \phi, t)$ is defined as follows:

$$f(x, \phi, t) = \min_{i=1}^{m} (\boldsymbol{H}_i(t) + \boldsymbol{W}_i(t) \cdot (exp(\sqrt{\sum_{j=1}^{n} \frac{(x_j - \boldsymbol{X}_j^i(t))^2}{n}}) - 1)), \quad (4)$$

where m is the number of peaks, n is the number of dimensions. $\boldsymbol{H}$ and $\boldsymbol{W}$ change as follows:

$$\boldsymbol{H}(t+1) = \text{DynamicChanges}(\boldsymbol{H}(t))$$
$$\boldsymbol{W}(t+1) = \text{DynamicChanges}(\boldsymbol{W}(t))$$

Instead of shifting peak position as in MPB[1], we borrow the idea from [11] and use rotation matrices to change the peak position in the GDBG system.

A rotation matrix $R_{ij}(\theta)$ is obtained by rotating the projection of $\overrightarrow{x}$ in the plane $i - j$ by an angle θ from the i-th axis to the j-th axis. The peak position $\boldsymbol{X}$ is changed by the following algorithm:

Step 1. Randomly select l dimensions (l is an even number) from the n dimensions to compose a vector $r = [r_1, r_2, ..., r_l]$.
Step 2. For each pair of dimension $r[i]$ and dimension $r[i+1]$, construct a rotation matrix $R_{r[i],r[i+1]}(\theta(t))$, $\theta(t)$=DynamicChanges($\theta(t-1)$).
Step 3. A transformation matrix $A(t)$ is obtained by:
$A(t) = R_{r[1],r[2]}(\theta(t)) \cdot R_{r[3],r[4]}(\theta(t)) \cdots R_{r[l-1],r[l]}(\theta(t))$,$\theta(t) \in (0, 2\pi)$
Step 4. $\boldsymbol{X}(t+1) = \boldsymbol{X}(t) \cdot A(t)$

By changing the height, width and position of each peak, we can easily construct a new fitness landscape with different properties. However, this artificial fitness landscape has a problem that the shape of each peak in the fitness landscape is symmetrical at their peak position. It is easy to obtain the local optima for some algorithms using this property. For example, PSO is effective to search the global optimum in symmetrical fitness landscapes. We can see this from the experimental results presented in the paper. So, a composition real DBG based on several static benchmark problems is proposed in this paper.

Real composition DBG

A composition function construction method was proposed in [9]. The idea is to compose the standard benchmark functions to construct a more challenging function with a randomly located global optimum and several randomly located local optima. By shifting, rotating and composing the global optimum of standard functions, we can get more challenging test functions possessing many desirable properties. In the GDBG system, we get dynamism by controlling the values and locations of these global and local optima. The composition function can be described as:

$$F(x, \phi, t) = \sum_{i=1}^{m} (w_i \cdot (f_i'((x - \boldsymbol{O}_i(t) + O_{iold})/\lambda_i \cdot \boldsymbol{M}_i) + \boldsymbol{H}_i(t))) \quad (5)$$

where the system control parameter $\phi = (\boldsymbol{O}, \boldsymbol{M}, \boldsymbol{H})$, $F(x)$ is the composition function, $f_i(x)$ is i-th basic function used to construct the composition function. m is the number of basic functions, $\boldsymbol{M}_i$ is orthogonal rotation matrix for each $f_i(x)$, $\boldsymbol{O}_i$ and $\boldsymbol{O}_{iold}$ are the shifted and old optimum position for each $f_i(x)$. The weight value w_i for each $f_i(x)$ is calculated as:

$$w_i = exp(-sqrt(\frac{\sum_{k=1}^{n}(x_k - o_i^k + o_{iold}^k)^2}{2n\sigma_i^2}))$$

$$w_i = \begin{cases} w_i & \text{if } w_i = max(w_i) \\ w_i \cdot (1 - max(w_i)^{10}) & \text{if } w_i \neq max(w_i) \end{cases}$$

$$w_i = w_i / \sum_{i=1}^{m} w_i$$

where σ_i is the converge range factor of $f_i(x)$, whose default value is 1.0 in the paper, λ_i is the stretch factor for each $f_i(x)$, which is defined as:

$$\lambda_i = \sigma_i \cdot \frac{X_{max} - X_{min}}{x_{max}^i - x_{min}^i}$$

where $[X_{max}, X_{min}]^n$ is the search range of $F(x)$ and $[x_{max}^i, x_{min}^i]^n$ is the search range of $f_i(x)$.

In Eq. (5), $f_i'(x) = C \cdot f_i(x)/|f_{max}^i|$, where C is a predefined constant, which is set to 2000 as in [9], and f_{max}^i is the estimated maximum value of $f_i(x)$, which is estimated as:

$$f_{max}^i = f_i(x_{max} \cdot M_i)$$

In the composition DBG, $\boldsymbol{M}$ is randomly initialized using the above transformation matrix construction algorithm and then remains unchanged. The dynamism of the system control parameter $\boldsymbol{H}$ and $\boldsymbol{O}$ can be described as follows:

$$\boldsymbol{H}(t+1) = \text{DynamicChanges}(\boldsymbol{H}(t))$$
$$\boldsymbol{O}(t+1) = \text{DynamicChanges}(\boldsymbol{O}(t))$$

Five basic benchmark functions are used in the GDBG system. Table 1 shows the details of the five functions.

3.3 Generator Instances in the Combinatory Space

In this section, the dynamic multi-dimensional knapsack problem (DMKP) and dynamic TSP (DTSP) are instantiated from the GDBG system.

Dynamic Multi-dimensional Knapsack Problem (DMKP)

The knapsack problem [5] is a classical combinatorial benchmark problem to test the performance of EAs. The static multi-dimensional knapsack problem (MKP) belongs to the class of NP-complete problems. It has a wide range of

Table 1. Details of the basic benchmark functions

name	function	range
Sphere	$f(x) = \sum_{i=1}^{n} x_i^2$	[-100,100]
Rastrigin	$f(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	[-5,5]
Weierstrass	$f(x) = \sum_{i=1}^{n} (\sum_{k=0}^{k_{max}} [a^k \cos(2\pi b^k (x_i + 0.5))]) - n \sum_{k=0}^{k_{max}} [a^k \cos(\pi b^k)]$	
	$a = 0.5, b = 3, k_{max} = 20$	[-0.5,0.5]
Griewank	$f(x) = \frac{1}{4000} \sum_{i=1}^{n} (x_i)^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	[-100,100]
Ackley	$f(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)) + 20 + e$	[-32,32]

real world applications, such as cargo loading, selecting projects to fund, budget management, etc. The DMKP can be defined as $f(x, \phi, t)$, $\phi = (\boldsymbol{R}, \boldsymbol{P}, \boldsymbol{C})$ and $\boldsymbol{R}$, $\boldsymbol{P}$, and $\boldsymbol{C}$ are the vector of resources, profits, and capacities respectively. The DMKP can be formalized as:

$$f(x, \phi, t) = max \sum_{i=1}^{n} p_i(t) \cdot x_i(t) \tag{6}$$

$$subject\ to \sum_{i=1}^{n} r_{ij}(t) \cdot x_i(t) \leq c_i(t), j = 1, 2, \cdots, m \tag{7}$$

where n is the number of items, m is the number of resources, $x_i \in \{0, 1\}$ indicates whether item i is included in the subset or not, p_i is the profit of item i, r_{ij} shows the resource consumption of item i for resource j, and c_i is the capacity constraint of resource i. The system control parameters are changed as:

$$\begin{aligned} \boldsymbol{P}(t+1) &= \text{DynamicChanges}(\boldsymbol{P}(t)) \\ \boldsymbol{C}(t+1) &= \text{DynamicChanges}(\boldsymbol{C}(t)) \\ \boldsymbol{R}(t+1) &= \text{DynamicChanges}(\boldsymbol{R}(t)) \end{aligned}$$

where the item profits, resources, and consumption constraints are bounded in the range of $[l_p, u_p]$, $[l_r, u_r]$, and $[l_c, u_c]$ respectively.

Dynamic Traveling Salesman Problem (DTSP)

TSP is another classical NP-complete combinatory problem. DTSP [8] has a wide range of real applications, especially in the optimization of dynamic networks, like network planning and designing, load-balance routing, and traffic management.

DTSP is a TSP determined by a dynamic cost (distance) matrix as follows:

$$D(t) = \{d_{ij}(t)\}_{n*n} \tag{8}$$

where $d_{ij}(t)$ is the cost from city i to city j, n is the number of cities. DTSP can be defined as $f(x, \phi, t)$, $\phi = \boldsymbol{D}$, the objective of DTSP is to find a minimum-cost

route containing all cities at time t. It can be described as:

$$f(x, \phi, t) = Min(\sum_{i=1}^{n} d_{T_i, T_{i+1}}(t)) \tag{9}$$

where $T \in 1, 2, \cdots, n$, if $i \neq j$,then $T_i \neq T_j$, $T_{n+1} = T_1$. The dynamism of the cost matrix $\boldsymbol{D}$ is described as:

$$\boldsymbol{D}(t+1) = \text{DynamicChanges}(\boldsymbol{D}(t)).$$

4 Experimental Study

In this section, some experiments based on PSO and FEP [15] algorithms are carried out on the two instances of real space to test the performance of the GDBG system. The number of dimensions $n = 10$, peak number and basic function number $m = 10$ in both instances, the 10 basic functions selected are all sphere function in the composition DBG, the search range is set to $x \in [-5, 5]$ for both DBGs. The population size is set to 50 for both PSO and FEP. For the PSO algorithm, acceleration constants η_1 and η_2 are both set to be 1.496180 and the inertia weight $\omega = 0.729844$. In FEP, the tournament size is 5 for selection and the initial standard deviation is 3.0 as used in [15]. Both algorithms were run 30 times independently for all the results.

For evaluating the efficiency of the algorithms, we use the offline performance measure defined as follows:

$$e_t = \frac{h_t}{f_t}, \tag{10}$$

where f_t is the best solution got by an algorithm just before the t-th environmental change, h_t is the optimum value at time t, and e_t is the relative value of h_t and f_t. Fig. 2 shows the offline performance of PSO and FEP in the composition DBG and rotation DBG environments.

From Fig. 2, it can be seen that algorithms show a different performance in different dynamic types. Algorithms give a much better performance in small step changes than large step changes. Both PSO and FEP show the recurrent performance in the recurrent and recurrent with noise environments. On the other hand, the composition DBG environment is much harder to search than the rotation DBG environment for all change types. Composition DBG environments can give a challenging complexity for algorithms to test, but the rotation DBG is easier than the composition DBG to be controlled.

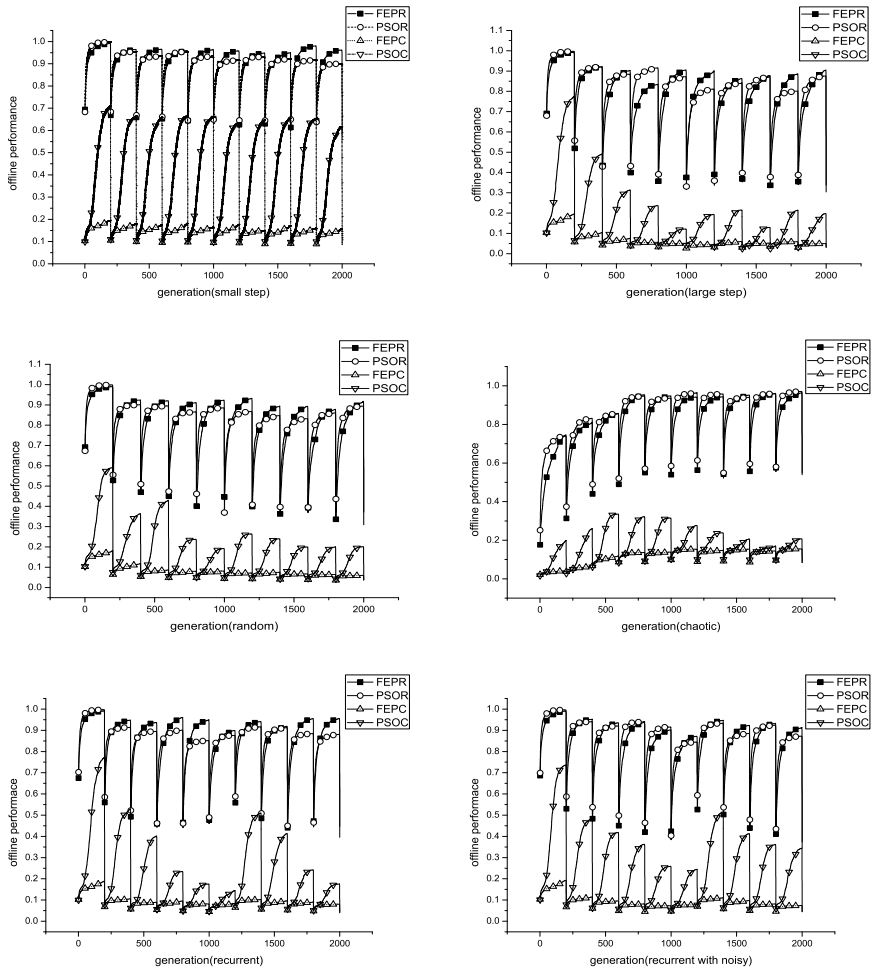

Fig. 2. The offline performance of PSO and FEP in the composition and rotation DBG environments.

5 Conclusions

Constructing benchmark problems is an important task in studying EAs in dynamic environments. This paper proposes a unified method, GDBG, to construct dynamic environments across the binary, real, and combinatory solution spaces. GDBG provides six dynamism types of random change, small step change, large step change, recurrent change, chaotic change and recurrent change with noise. In this paper, we present several instatiations of generators in the binary, real, and combinatory spaces respectively from the GDBG system. Especially, in the

real space, we introduce a rotation method instead of shifting peak positions as in the MPB. The rotation method can overcome the shortcomings of unequal challenge per change for algorithms of the MPB model, which occurs when the peak positions bounce back from the boundary of the landscape. In order to test the GDBG system, experiments were carried out using the PSO and FEP algorithms under the composition DBG and rotation DBG environments. Experimental results show that the GDBG system can give different properties by simply setting the environment change type.

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/1.

References

1. J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, *Proc. of the 1999 Congr. on Evol. Comput*, pp. 1875-1882, 1999.
2. H. G. Cobb, J. J. Grefenstette, Genetic algorithms for tracking changing environments, *in Proc. 5th Int. Conf. Genetic Algorithms*, pp. 523-530, 1993.
3. D. Dasgupta and DR. McGregor, Nonstationary function optimization using the structured genetic algorithm, *Proc. of the 2nd Int. Conf. on Parallel Problem Solving from Nature*, pp. 145-154, 1992.
4. R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proc. of the 6th Int. Symp. on Micro Machine and Human Science*, pp. 39-43, 1995.
5. H. Kellerer, U. Pferschy and D. Pisinger, Knapsack Problems. Springer, 2004.
6. J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. *Proc. of the 1995 IEEE Int. Conf. on Neural Networks*, pp. 1942-1948, 1995.
7. J. Lewis, E. Hart and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems, *Proc. of the 5th Int. Conf. on Parallel Problem Solving from Nature*, pp. 139-148, 1998.
8. C. Li, M. Yang, and L. Kang. A new approach to solving dynamic TSP, *Proc of the 6th Int. Conf. on Simulated Evolution and Learning*, pp. 236-243, 2006.
9. J. J. Liang, P. N. Suganthan, and K. Deb. Novel composition test functions for numerical global optimization, *Proc. of the 2005 IEEE Congr. on Evol. Comput.*, pp. 68-75, 2005.
10. R. W. Morrison and K. A. De Jong. A test problem generator for non-stationary environments, *Proc. of the 1999 Congr. on Evol. Comput.*, pp. 2047-2053, 1999.
11. K. Weicker and N. Weicker. Dynamic rotation and partial visibility, *Proc. of the IEEE 2003 Congr. on Evol. Comput.*, pp. 1125-1131, 2003.
12. S. Yang. Non-stationary problem optimization using the primal-dual genetic algorithm, *Proc. of the 2003 IEEE Congr. on Evol. Comput.*, pp. 2246-2253, 2003.
13. S. Yang and X. Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Comput.*, 9(11): 815-834, 2005.
14. S. Yang and X. Yao. Population-based incremental learning with associative memory for dynamic environments. *IEEE Trans. on Evol. Comput.*, 2008.
15. X. Yao and Y. Liu. Fast evolutionary programming, *Proc. of the 5th Annual Conf. on Evolutionary Programming*, pp. 451-460, 1996.