2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Dynamic History-driven Evolutionary Algorithm

Chi Kin Chow and Shiu Yin Yuen

Abstract—**Dynamic objective problem (DOP) raises two challenging issues to evolutionary algorithm: comparing two individuals evaluated at different time instances and tracing the jumping global optimum. This paper presents a dynamic objective evolutionary algorithm (DOEA) that handles these issues through search history. The presented algorithm, namely dynamic objective history driven evolutionary algorithm (DyHdEA), stores the entire search history including the position, the fitness and the evaluated time of the solutions in a dynamic fitness tree. In the experiment section, DyHdEA is examined on a 10-dimensional DOP that is composed of five basis problems ranging from uni-modal to multi-modal, and from separable to non- separable. Meanwhile, the performance of DyHdEA is compared with five benchmark DOEAs including artificial immune algorithm, differential evolution, evolutionary programming, and particle swarm optimization. Seen from the result, DyHdEA effectively traces the dynamic global optimum with jumping transitions.**

I. Introduction

THE WORLD is full of uncertainty. When we take into account a real world optimization problem subjected to dynamic environments, it is called dynamic optimization problem (DOP). An ideal dynamic objective evolutionary algorithm (DOEA) should be able to continuously guess the accurate optimal solution at any time instance. Since the global optimal solution of a DOP probably jumps from one position to another, the search by DOEA should be explorative enough to trace this kind of suddenly appearing optimum.

In this paper, we propose a novel evolutionary algorithm, namely dynamic history-driven evolutionary algorithm (DyHdEA), which optimizes DOP with the guidance retrieved from search history. Note that as the objective landscape as well as the global optimum is dynamic, the search history contains time information. The more recent is the search history, the more reliable information it can provide for interrupting the environment at the current instance. Thus, in DyHdEA, search history is defined as the position, the fitness value, and the evaluated time of the evaluated solutions. In [1]-[4], this set of history is organized by a binary space partitioning (BSP) tree. In this paper, a novel BSP tree called *dynamic fitness tree* that includes time information is reported. The term *dynamic* emphasizes that all information in the tree has timelines. For example, while following the idea of estimating objective landscape from search history in [4] and considering also the evaluation time, *dynamic fitness tree* acts as a dynamic objective landscape estimator. The current fitness value of past evaluated individuals could be estimated by the tree. It solves the problem of time bias in fitness comparison without increasing the computation load on fitness re-evaluation. The most related work of landscape estimation is coupled map lattices (CML) [5], which normally works on low-dimensional DOP as the number of cells in CML exponentially increases along with the problem dimensions.

Originated from the linkage between mutation step size and solution density in [2], we extend it to DOP by simultaneously considering solution density and its reliability. The corresponding history-driven mutation operator adaptively switches the search strategy between exploration and exploitation. Given an individual to be mutated, the proposed mutation operator would intensively search its neighborhood region if it is surrounded by many recently evaluated solutions. On the other hand, if its neighborhood region only has anciently evaluated solutions, no matter how numerous these evaluated solutions are, the operator would perform an explorative search on it.

DyHdEA has been compared with five benchmark DOEAs. Superior performance is observed in both leadership and accuracy.

The rest of this paper is organized as follows: Section II presents the details of dynamic fitness tree. Section III describes an adaptive and parameter-less mutation operator. Section IV reports the details of DyHdEA. Section V reports the experimental results and Section VI gives the conclusion.

II. Dynamic Fitness Tree

Dynamic fitness tree is a BSP tree structure memory archive. It structurally memorizes the positions, the fitness and the evaluated time of the solutions. Cooperating with specific procedures described in details below, it would serve as 1) a dynamic objective landscape prediction model, and 2) a solution density estimator.

BSP tree represents a space partitioning scheme organizing a set of data points in a space. When the data points are the evaluated solutions of an evolutionary algorithm, the corresponding partitioning scheme represents the distribution of the solutions. [1], [2] use it to control the mutation step size in an adaptive and parameter-less manner. In [4], as the BSP tree stores also the fitness values of the evaluated solutions, it serves as a static fitness landscape estimator. Likewise, [3] uses BSP tree as a solution density estimator to adaptively diversify the population. For full details and an illustrative example of how the BSP tree is constructed and operates, please refer to [1].

The authors are with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, China; e-mail: chowchi821@yahoo.com.hk, kelviny.ee@cityu.edu.hk.

The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 125313].

In this paper, we generalize BSP tree to dynamic fitness tree that includes temporal information. In dynamic fitness tree, a node represents a partitioned rectangular sub-region of the solution space. Suppose a parent node has two child nodes **l** and **r**, a decision boundary L binary partitions the parent sub-region into two sub-regions represented by **l** and **r**. The partitioned sub-regions are disjoint and their union is the sub-region of the parent, i.e., $L(\mathbf{x}) \leq 0$ *iff* **x** belongs to the sub-region represented by **l**, and $L(\mathbf{x}) > 0$ *iff* **x** belongs to the sub-region represented by **r**. Moreover, if the partitioned sub-region is represented by leaf node, it contains only one evaluated solution.

Definition 1. The sub-region of **x**

Suppose solution space X is partitioned into the sub-region set $H = \cup_i h_i$ by BSP Tree. Let $\mathbf{x}=[x_1, x_2, \ldots, x_D]$ be a point in X, where D is the dimension of the problem. We define the sub-region $h \subseteq H$ as the '*sub-region of* **x**' if $\mathbf{x} \in h$ and h is represented by a leaf node of the Tree.

In static objective function, every evaluated solution describes the same objective landscape. BSP tree records it by inserting a tree node. However, as dynamic fitness tree deals with dynamic objective problem, it records not only the position but also the fitness value and the evaluation time of the solution. We use the recorded temporal information to identify the outdated solutions and discard them from the tree. Given that a D-dimensional sub-region h contains two solutions: a recorded solution $\mathbf{a} = [a_1, a_2, \ldots, a_D]$ and a newly evaluated solution $\mathbf{b} = [b_1, b_2, \ldots, b_D]$. We denote by t_a and t_b as the evaluation time of **a** and **b** respectively, with $t_b > t_a$. We define that, as long as the temporal difference between **a** and **b** is not larger than their difference along any positional dimension, i.e., $t_b - t_a < \max_{k\in[1,D]}|b_k - a_k|$, $F(\mathbf{a},t_a)$ agrees with the current objective landscape, i.e., $F(\mathbf{a}, t_a) \approx F(\mathbf{a}, t_b)$. Thus, we preserve **a** and insert **b** to the tree. This is equivalent to partitioning h into two non-overlapping and adjacent sub-regions. The decision boundary between them is expressed in the same manner as in [1]:

$$L(\mathbf{x}) = x_j - p, \text{ where } j = \arg\max_{k\in[1,D]}|b_k - a_k| \text{ and}$$
$$p = (a_j + b_j)/2$$

where $\mathbf{x}=[x_1, x_2, \ldots, x_D]$ is a solution.

On the other hand, if their temporal difference is larger than that along any positional dimension, solution **a** is outdated and should be discarded from the memory. Thus, the node that originally records **a** should now record **b**.

While partitioning a sub-region, we compare the difference of two solutions at different dimensions. It is common that they have different physical meanings and different ranges. These differences may introduce a source of sub-region selection bias. For example, suppose a solution space is defined as $[0, 4] \times [0, 100]$ and we are going to partition it by two solutions $[a_1, a_2]$ and $[b_1, b_2]$ that are randomly generated in the space. Because the range of the second dimension is larger, $|b_2 - a_2|$ is usually larger than $|b_1 - a_1|$. Thus the partition strategy in [1] is intrinsically biased to cut at the second dimension. Generally speaking, the larger range is the dimension, the higher chance the dimension is selected for partitioning. Meanwhile, while identifying outdated solutions, we also compare the temporal information as if it is one of the spatial dimensions, which is clearly inadequate because space and time are different entities.

In this paper, we remove the comparison bias stated above by normalizing the temporal dimension as well as all dimensions of solution space to [0, 1]. Normalization is straightforward if the upper bound and the lower bound of the corresponding range are fixed (i.e., the dimensions of solution space). However, as the dynamic fitness tree constantly records solutions, the maximum evaluation time amongst the recorded solutions monotonically increases. Meanwhile, during the optimization process, some solutions are identified as outdated and are discarded from the tree. The minimum evaluation time amongst the stored solutions (the lower bound of the temporal dimension) expectedly increases too but not at the same speed as the maximum evaluation time. Thus, the range of temporal dimension naturally varies from time to time in the whole optimization process, and the corresponding normalization is not an easy task.

To normalize the temporal dimension, we have to keep track of the range of temporal dimension. The upper bound is obviously equivalent to the current time instance. On the other hand, the lower bound can be computed through manipulating the tree data structure: Suppose a tree node **a** has two sub-trees Ω_1 and Ω_2. The sub-tree Ω_1 records the solution set H_1 and the minimum time stamp (i.e., time stamp refers to the evaluation time instance of an individual) of the set is t_1. The sub-tree Ω_2 records the solution set H_2 and the minimum time stamp of the set is t_2. Since the tree rooted at **a** records the union of H_1 and H_2, the minimum time stamp of **a** is simply the minimum of t_1 and t_2.

Definition 2. Minimum time stamp of tree node

Suppose **a** is a node of *dynamic fitness tree* Ω and Ω_a is a sub-tree of Ω rooted at **a**, we define the minimum time stamp $t'(\mathbf{a})$ of **a** as the evaluation time of the earliest solution in Ω_a. Given that $\mathbf{Z} = \{\mathbf{z}_j\}$ is the set of solutions in Ω_a and t_j is the evaluation time of $\mathbf{z}_j$, i.e., $t'(\mathbf{a}) = \min\{t_j\}$.

Algorithm A1. MinimumTimeStampUpdate(**z**, t, Ω)
Input: 1) an individual **z**, 2) its evaluation time t, and 3) dynamic fitness tree Ω

1. **p**:= the leaf node of Ω that represents the sub-region of **z**
2. $t'(\mathbf{p}) := t$
3. While **p** is not a root node
4. **p**:= the parent node of **p**
5. **a**:= left child node of **p**
6. **b**:= right child node of **p**
7. $t'(\mathbf{p}) := \min\{t'(\mathbf{a}), t'(\mathbf{b})\}$
8. Loop

Output: the dynamic fitness tree Ω with updated minimum time stamp

The minimum time stamps can be obtained bottom-up from the leaf nodes. By definition, the minimum time stamp of the root node represents the minimum time stamp amongst all solutions. Note that when the sub-regions of some formerly evaluated solutions would be replaced by those of more recently evaluated ones, the minimum time stamp of some tree nodes should be re-computed. Suppose **p** is a newly inserted tree node (or a replaced node), the minimum time stamp re-computation is only involved at the ancestor nodes of **p**. Algorithm A1 summarizes the procedure of minimum

time stamp re-computation.

Algorithm A2 summarizes the procedure of dynamic fitness tree node insertion. Suppose **z** is the individual to be inserted and it is normalized to the space $[0, 1]^D$; t is the evaluation time of **z** and Ω is the dynamic fitness tree. The tree insertion starts by searching the leaf node **c** in Ω that represents the sub-region of **z**. Let $\check{t}(\mathbf{r})$ be the minimum time stamp of Ω, we update the evaluation time of **c**, $\check{t}(\mathbf{c})$, as $\check{t}(\mathbf{c}) = (t(\mathbf{c}) - \check{t}(\mathbf{r})) / (t - \check{t}(\mathbf{r}))$. Since **z** is the latest evaluated solution, its normalized evaluation time is one. If the normalized temporal difference of **z** and **c**, i.e., $(1- \check{t}(\mathbf{c}))$, is larger than the distance between **z** and **c** at any positional dimension, i.e., $1- \check{t}(\mathbf{c}) > \max_{k\in[1,D]}|z_k - c_k|$ we overwrite the information stored in **c** to **z**. Otherwise, we insert child node under **c** to record **z**.

Algorithm A2. DynamicFitnessTreeNodeInsertion(**z**, f, t, Ω)
Input: 1) a normalized individual **z**, 2) it fitness value f, 3) its evaluation time t, and 4) dynamic fitness tree Ω
1. **c**:= root node of Ω //Line 1–9 is the same as in [1]
2. While(**c** has two child nodes: **a** and **b**)
3. $L(\cdot)$:= the decision boundary of **c**
4. If($L(\mathbf{z}) \le 0$)
5. **c**:= child node **a**
6. Else
7. **c**:= child node **b**
8. EndIf
9. Loop
10. Compute the updated time stamp $\check{t}(\mathbf{c})$ of **c**
11. If $1 - \check{t}(\mathbf{c}) > \max_{k\in[1,D]}|z_k - c_k|$ then
12. Overwrite the stored information in **c** (its position, fitness and evaluation time) to **z**, f and t
13. MinimumTimeStampUpdate(**z**, t, Ω)
14. Else
15. Insert **a** child node under **c** to record **z**, f and t
16. Endif
Output: the updated dynamic fitness tree

One purpose of dynamic fitness tree is to estimate the fitness value of any point in solution space at the current time instance. Let t be the current time instance; $\mathbf{Z} = \{\mathbf{z_i} \mid i = 1, 2, \ldots, t\text{-}1\}$ be the entire solution set; t_i be the evaluation time of $\mathbf{z_i}$; $Z = \prod_{k=1}^{D}[L_k, U_k]$ be the solution space and Ω be the dynamic fitness tree that memories **Z**, the fitness value of a solution **s** at current instance t is estimated as $F(\mathbf{z_k}, t_k)$:

$$F(\bar{\mathbf{s}}, t) \approx \tilde{F}(\mathbf{s}|\Omega) = F(\bar{\mathbf{z}}_k, t_k) \quad (1)$$

where $\bar{\mathbf{s}}$ (the normalized **s** by X) is in the sub-region of $\bar{\mathbf{z}}_k$ (the normalized $\mathbf{z}_k$ by X).

III. Adaptive Mutation for Dynamic Environment

In [2], the evaluated solution density provides a good guidance on setting the mutation step size; the corresponding mutation operators are adaptive and applied to handle static optimization problems. While optimizing DOP, the guidance on mutation step size by solution density involves temporal information as well. Intuitively, the reliability of the guidance is higher if the corresponding solution density is computed from a set of more recent solutions, and vice versa. We formally define the reliability of a tree node below:

Definition 3. Reliability of tree node

Suppose **a** is a node of *dynamic fitness tree* Ω and Ω_a is the sub-tree of Ω rooted at **a**. Given that $\mathbf{Z_a} = \{\mathbf{z_i}\}$ is the solution set in Ω_a and t_i is the evaluation time of $\mathbf{z_i}$, we define the reliability $\mathbf{r}(\mathbf{a})$ of **a** as the evaluation time of the latest solution in the sub-region represented by **a**, i.e., $r(\mathbf{a}) = \max\{t_i\}$.

Expectedly, after recording a solution or overwriting node information, the reliabilities of some nodes in the dynamic fitness tree should be re-computed. This re-computation is only applied to the newly inserted tree node (or the replaced node) and all its ancestor nodes. Given that **a** is the newly inserted tree node and t is the time stamp of **a**, the reliabilities of **a** and all its ancestor nodes are re-assigned as t. Algorithm A3 summarizes the procedure of reliability update.

Algorithm A3. ReliabilityUpdate(**z**, t, Ω)
Input: 1) an individual **z**, 2) its evaluation time t and 3) dynamic fitness tree Ω
1. **a**:= the leaf node of Ω that contains **z**
2. While **a** is not a root node
3. $r(\mathbf{a})$:= t
4. **a**:= the parent node of **a**
5. Loop
Output: the dynamic fitness tree Ω with updated reliability

Note that the mutation operator commonly performs a local search. The locally optimal strategy to balance between the exploration (expanding mutation region) and the exploitation (increasing the reliability of the region) of mutation operator is to expand the region until the corresponding reliability is locally (not necessary globally) maximal. We formally define the locally optimal mutation region below:

Definition 4. Locally optimal mutation region of solution **s**

Suppose **s** belongs to leaf node **a**; the node **a** together with all its n ancestor nodes form **a** node sequence $\{\mathbf{P_0}, \mathbf{P_1}, \mathbf{P_2}, \ldots, \mathbf{P_n}\}$ where $\mathbf{P_0} = \mathbf{a}$ and $\mathbf{P_i}$ is the parent node of $\mathbf{P_{i-1}}$ for all $i = 1, 2, \ldots, n$. Meanwhile, we denote by r_i the reliability of $\mathbf{P_i}$. The locally optimal mutation region of **s** is defined as the sub-region represented by $\mathbf{P_k}$ where $k = \min_{j=0,1,\ldots n-1} j$ such that $r_j = r_{j+1}$ or if it is not satisfied, then $k = n$.

Algorithm A4 summarizes the procedure of computing the locally optimal mutation region.

Algorithm A4. LocallyOptimalMutationRegion(**s**, Ω)
Input: 1) solution **s** and 2) dynamic fitness tree Ω
1. **a**:= the leaf node of Ω that contains **s**
2. If **a** is the root node of Ω
3. M:= the sub-region of **a**
4. Else
5. While(**a** is not a root node of Ω) ^ ($r(\mathbf{b}) > r(\mathbf{a})$) where **b** is the sibling node of **a**
6. **a**:= the parent node of **a**
7. Loop
8. M:= the sub-region of **a**
9. EndIf
Output: the active mutation region M of **s**

After computing the locally optimal mutation region $M = \prod_{k=1}^{D}[l_k, u_k]$ of individual **s** to be mutated, we randomly select a dimension j of the solution space. The j^{th} element of **s**, s_j, is replaced by a uniformly distributed random number in the range $[l_j, u_j]$. The individual after element replacement is regarded as the mutant of **s**. This procedure implements a One-Gene-Flip (OGF) mutation, which is proposed in [2]. Comparing to the adaptive mutation in [1] that randomly selects a point in M, OGF mutation is less disruptive to the

schemata structure.

IV. Dynamic Objective History-Driven Evolutionary Algorithm

In this section, a new dynamic objective evolutionary algorithm, DyHdEA, is presented. Given a D-dimensional DOP $F(\cdot)$, the corresponding optimization by DyHdEA starts from a sequence of initializations: 1) presetting the evaluation time stamp t to zero, and 2) presetting dynamic fitness tree Ω to contain the root node only. Moreover, a population of μ individuals $\{\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_\mu}\}$ is generated randomly inside the solution space X. After the initialization, the μ individuals are evaluated one by one. At each solution evaluation, the position, as well as its fitness value and evaluation time, is inserted in Ω. Meanwhile, the minimum time stamp and the reliabilities of tree nodes are updated.

During the evolution, the offspring of each individual $\mathbf{z_i}$ in the current population is generated by randomly selecting an individual $\mathbf{y}$ distinct from $\mathbf{z_i}$. A uniform crossover operator is applied on $\mathbf{z_i}$ and $\mathbf{y}$ to generate a new individual $\mathbf{s}$. Afterwards, we compute the locally optimal mutation region M of $\mathbf{s}$, and perform OGF mutation to generate an offspring $\mathbf{m_i}$ of $\mathbf{z_i}$. Note that as M relates to the normalized search space, we should scale it while performing the OGF mutation. After evaluating $\mathbf{m_i}$ and inserting it into Ω, the minimum time stamps and the reliabilities of tree nodes are updated.

When all offspring has been evaluated, we select μ individuals from the pool of the current and the offspring populations. Instead of elitism selection of which each individual in the combined population pool would be compared with all remaining ones, individual in the current population is compared only with its offspring, and the fitter one of them would survive. This selection scheme is the same as that used in differential evolution [6]. As the comparison involves only two individuals, the diversity of the survived population is higher than that through elitism selection. During the parent-offspring selection, it may happen that the parent is anciently evaluated but the offspring is recently evaluated. Directly comparing their fitness is not a proper way to choose a fitter individual. Rather, it is better to compare the fitness values of parent and offspring at the current time instance. Since re-evaluation of parent individual is costly, estimating the current fitness value of parent individual from search history by (1) is a more practical approach. This is because falling short of evaluating the parent individual once again, the most up to date estimate is the current fitness tree after the time stamp updates. Note in particular that this update has included the information provided by all the newly generated offspring. Suppose $\tilde{f}_i$ is the estimated fitness value of parent individual $\mathbf{z_i}$ at current time instance (i.e., $\tilde{f}_i = \tilde{F}(\mathbf{z_i}|\Omega)$ in (1)), and q_i is the fitness value of offspring individual $\mathbf{m_i}$, $\mathbf{z_i}$ would be replaced by $\mathbf{m_i}$ if q_i is better than $\tilde{f}_i$. The evolution by DyHdEA is repeated until a given stopping criterion is satisfied. At the end of each generation, the optimal individual amongst the selected population is recorded. Algorithm A5 summarizes the procedure of DyHdEA. Without loss of generality, the presented procedure is for a minimization problem.

Algorithm A5. DyHdEA
Input: 1) a D-dimensional DOP $F(\cdot)$, 2) population size μ, 3) crossover rate R_x and 4) solution space $X = \prod_{k=1}^{D}[L_k, U_k]$

/* Initialization */
Time stamp t:= 0
Initialize Ω to contain a root node only
For i = 1 to μ
  Randomly generates μ individuals $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_\mu\}$
  t:= t + 1
  f_i:= $F(\mathbf{z}_i,t)$
  Normalize $\mathbf{z}_i$ as $\mathbf{v} = [v_1, v_2, \ldots, v_D]$ where $v_k = (z_{i,k} - L_k)/(U_k - L_k)$
  Ω:= DynamicFitnessTreeNodeInsertion($\mathbf{v}, f_i, t, \Omega$)
  Ω:= MinimumTimeStampUpdate($\mathbf{v}, t, \Omega$)
  Ω:= ReliabilityUpdate($\mathbf{v}, t, \Omega$)
Next i
/* Evolution by DyHdEA */
While stopping criterion is not satisfied
  /* Offspring generation */
  For i = 1 to μ
    $\mathbf{y}$:= $\mathbf{z}_k$ where k = Rand($\{1, 2, \ldots, \mu\}$ / i)
    For j = 1 to D
      If Rand([0,1]) < R_x
        s_j:= z_j
      Else
        s_j:= y_j
      EndIf
    Next j
    $M = \prod_{k=1}^{D}[l_k, u_k]$:= LocallyOptimalMutationRegion(s, Ω)
    d = Rand($\{1, 2, \ldots, D\}$)
    $$m_{i,j} = \begin{cases} s_j & if\ j \neq d \\ \text{Rand}([0,1])(u_d - l_d)(U_d - L_d) + l_d(U_d - L_d) + L_d & if\ j = d \end{cases}$$
  Next i
  For i = 1 to μ
    t:= t + 1
    q_i:= $F(\mathbf{m}_i, t)$
    Normalize $\mathbf{m}_i$ as $\mathbf{v}$=$[v_1, v_2, \ldots, v_D]$
    Ω:= DynamicFitnessTreeNodeInsertion($\mathbf{v}, q_i, t, \Omega$)
    Ω:= MinimumTimeStampUpdate($\mathbf{v}, t, \Omega$)
    Ω:= ReliabilityUpdate($\mathbf{v}, t, \Omega$)
  Next i
  /* Selection */
  For i = 1 to μ
    Update the fitness value of $\mathbf{z_i}$ by (1)
    If $\tilde{f}_i \geq q_i$ then
      $\mathbf{z_i}$:= $\mathbf{m_i}$
      $\tilde{f}_i$:= q_i
    EndIf
  Next i
Loop
Output: the optimal solution $\mathbf{x_{best}} \in X$

V. Experimental Results

A. Test Problem

In this experiment, the test DOP, $F(\cdot)$, is a weighted sum of basis functions $\{b_i(\cdot)\}$. Its general form is shown in (2) (see below). D is the problem dimension; n is the number of basis functions and N is the maximum number of fitness evaluations. This equation contains common types of dynamics of a DOP: 1) the optimal point moves along a non-linearly locus and 2) it would suddenly jump from one point to another.

$$F(\mathbf{x}, t|\alpha_0, \{\mathbf{R_i}\}, \{b_i(\mathbf{x})\}) = \sum_{i=1}^{n} H_i(t) b_i\left(\left(\mathbf{x} - \mathbf{o_i}(t)\right)\mathbf{R_i}\right) \quad (2)$$

where

$$\mathbf{o_i}(t) = \left[o_{i,1}(t), o_{i,2}(t), \ldots o_{i,D}(t)\right]$$

$$o_{i,j}(t) = \begin{cases} \cos^{j-1}\alpha_i(t)\sin\alpha_i(t) & if\ j < D \\ \cos^{D}\alpha_i(t) & otherwise \end{cases}$$

$$\alpha_i(t) = 2\pi\left(\frac{i}{n} + \frac{t}{N} + \alpha_{0,i}\right)$$

N is the total number of fitness evaluations.

$$H_i(t) = \begin{cases} \frac{Nt}{n} - (i-2) & if\ \frac{(i-2)N}{n} \le t \le \frac{(i-1)N}{n} \\ i - \frac{Nt}{n} & if\ \frac{(i-1)N}{n} \le t \le \frac{iN}{n} \\ 0 & otherwise \end{cases}$$

Problem F consists of four parts: 1) basis function set $\{b_i(.)\}$, 2) weight functions $\{H_i(t)\}$, 3) translation function set $\{\mathbf{o_i}(t)\}$, and 4) rotation matrix set $\{\mathbf{R_i}\}$. The dynamics of F arises from two sources: the time-varying weight functions $\{H_i(t)\}$ and the time-varying translation functions $\{\mathbf{o_i}(t)\}$. The objective landscape changes continuously from one bias function to another, and the global optimum of each bias function keeps moving in a circular path.

The behavior of the test DOP is described as follows: Initially, the objective landscape of each basis functions b_i is rotated at the origin of the solution space with a distinct and arbitrary rotation matrix $\mathbf{R_i}$. Once the test problem starts to be evaluated, the landscape of each basis function keeps translating such that the global optima of the functions sparsely shift in X. The translation paths are high dimensional and circular. It returns to its original position at the end of the evolution. Meanwhile, to implement the dramatic jump of global optimum, each basis function is weighted by a time-varying function $H_i(t)$ activated at particular time intervals. In summary, the test DOP is multi-modal, non-separable and continuously changing. Meanwhile, its global optimum is also continuously shifting and occasionally jumping.

In this experiment, DyHdEA is examined on three DOPs: F_1, F_2 and F_3. Each of them consists of five distinct basis functions and totally fifteen basis functions f_1-f_{15} are used. These basis functions cover a wide range of problem classes: from uni-modal to multi-modal and from separable to non-separable.

f_1: Schwefel's problem 2.22 [7]
f_2: Generalized Griewank function [7]
f_3: Inverted cosine mixture problem [8]
f_4: Levy and Montalvo 2 problem [8]
f_5. Sinusoidal problem [8]
f_6. Schwefel's problem 2.21 [7]
f_7. Ackley function [7]
f_8. Epistatic Michalewicz problem [8]
f_9. Neumaier 3 problem [8]
f_{10}. Shubert problem [8]
f_{11}. Generalized Rosenbrock function [7]
f_{12}. Weierstrass's function [9]
f_{13}. Zakharov function [10]
f_{14}. Pathological function [10]
f_{15}. Odd Square problem [8]

The three tested DOPs are as follows:

$$F_1(\mathbf{x}, t) = F\left(\mathbf{x}, t|\boldsymbol{\alpha}_\mathbf{0}^{(1)}, \left\{\mathbf{R}_\mathbf{i}^{(1)}\right\}, \left\{b_i^{(1)} = f_i(\mathbf{x})\right\}\right)$$

$$F_2(\mathbf{x}, t) = F\left(\mathbf{x}, t|\boldsymbol{\alpha}_\mathbf{0}^{(2)}, \left\{\mathbf{R}_\mathbf{i}^{(2)}\right\}, \left\{b_i^{(2)} = f_{i+5}(\mathbf{x})\right\}\right)$$

$$F_3(\mathbf{x}, t) = F\left(\mathbf{x}, t|\boldsymbol{\alpha}_\mathbf{0}^{(3)}, \left\{\mathbf{R}_\mathbf{i}^{(3)}\right\}, \left\{b_i^{(3)} = f_{i+10}(\mathbf{x})\right\}\right)$$

where i = 1,2,…,5; all $\boldsymbol{\alpha}_\mathbf{0}^{(j)}$ and $\mathbf{R}_\mathbf{i}^{(j)}$ are distinct.

The employed basis functions are with different sizes of search space. To combine them as the DOPs, in this experiment, each dimension of every search space is normalized to [0, 1].

B. Algorithms for Comparison

To evaluate the impact of the proposed algorithm, we compare the performance of DyHdEA with five benchmark DOEAs.

Test algorithm 1 – Dynamic History-driven Evolutionary Algorithm (DyHdEA)

Test algorithm 2 – Artificial Immune Algorithm (AIA) [11]

Test algorithm 3 – Self-Adaptive Differential Evolution (jDE) [12]

Test algorithm 4 – Evolutionary Programming (UEP) [13]

Test algorithm 5 – Clustering Particle Swarm Optimizer for Dynamic Optimization (CPSO) [14]

Test algorithm 6 – Particle Swarm Optimization with Composite Particles in Dynamic Environment (PSO-CP) [15]

TABLE I
THE PARAMETER SETTINGS OF THE TEST ALGORITHMS

Algorithm	Parameter setting
DyHdEA	Population size: 10 Crossover rate: 0.5
AIA	Initial population size: 10 Maximum population size: 50 Cell suppression threshold: 5
jDE	Initial F: 0.5 Initial CR: 0.9 Population size: 50 Number of sub-population: 10
UEP	Population size: 100 Percentage of fitness evaluation assigned to local search: 20% Number of possible directions in local search: 5 Number of individuals in local search: 4 Tournament size: 10 Interval of archive updating: 10 generations Initial temperature: 6 Clearing radii: 5 Threshold with regard to premature convergence: f_{ini} / 100 where f_{ini} is the initial fitness deviation Threshold with regard to complete loss of diversity: f_{ini} / 1000
CPSO	Population size: 20D num_rgh: 5 num_ref: 3 $\eta_1 = \eta_2 = 1.7$ Inertia weight: linearly decrease from 0.6 to 0.3
PSO-CP	Population size: 100 $c_1 = c_2 = 2.05$ Inertia weight: 0.729844 Initial reflection step size: 6 Initial diversity threshold: 3

The detailed parameter settings of the test algorithms for all conducted experiments can be found in Table I. The parameters of all test algorithms follow their recommended settings in the literature [11]-[15]. All simulations are performed on a PC with 3.2GHz CPU and 1GB memory. The test algorithms are implemented in MATLAB language version 6.0.

C. Performance Measure

In this experiment, for each of the test DOEAs, we record the best individual $\mathbf{x}_{\mathbf{best}}(i)$ as well as its time stamp $t_{best}(i)$ at each generation i. The list $\mathbf{P} = \{[t_{best}(i), \mathbf{x}_{\mathbf{best}}(i)]\}$ represents the discrete locus of the global optimal solutions computed by the DOEAs. Since an ideal DOEA should be able to compute an accurate optimal solution at *any* time instance, a discrete locus is not sufficient to judge the performance of the DOEA. Moreover, because DOEA is commonly population-based, it can only identify the best individual at every generation but not at every time stamp. Thus, we denote by $\tilde{\mathbf{x}}_{\mathbf{best}}(t)$ as the continuous version of $\mathbf{P}$, which is estimated as follows: $\tilde{\mathbf{x}}_{\mathbf{best}}(t) \approx \mathbf{x}_{\mathbf{best}}(j)$ where $j = \arg\min_{k=1,\ldots,|\mathbf{P}|} |t_{best}(k) - t|$. $\tilde{\mathbf{x}}_{\mathbf{best}}(t)$ is estimated to be the best solution. For example, if the $\mathbf{P}$ of a DOEA is {[100, $\mathbf{x}_{\mathbf{best}}(1)$], [200, $\mathbf{x}_{\mathbf{best}}(2)$], [300, $\mathbf{x}_{\mathbf{best}}(3)$]}, $\tilde{\mathbf{x}}_{\mathbf{best}}(t)$ is as follows:

$$\tilde{\mathbf{x}}_{\mathbf{best}}(t) = \begin{cases} \mathbf{x}_{\mathbf{best}}(\mathbf{1}) & if\ 0 < t < 150 \\ \mathbf{x}_{\mathbf{best}}(\mathbf{2}) & if\ 149 < t < 250 \\ \mathbf{x}_{\mathbf{best}}(\mathbf{3}) & if\ t > 249 \end{cases}$$

Under the definition of $\tilde{\mathbf{x}}_{\mathbf{best}}(t)$, we define the convergence curve G of a DOEA as the locus of the fitness values of $\tilde{\mathbf{x}}_{\mathbf{best}}(t)$: $G(t) = F(\tilde{\mathbf{x}}_{\mathbf{best}}(t), t)$.

In a group of test algorithms, ideally the best one is superior to others at all considered time instances. Meanwhile the obtained best fitness should be significantly smaller (as we are dealing with minimization) than those of others. In this paper, we compare the test algorithms on two measures: 1) the number of time instances that one is superior to others, C_1 and 2) relative fitness C_2 (see (3) below) The first measure reflects the frequency of superiority of one algorithm and the second measure indicates the significance of the superiority. Given a pool of investigated DOEAs $\{A_i\}$ and convergence curves of $\{A_i\}$ $\{G(t|A_i)\}$, the first measure C_1 can be obtained by simply counting the number of time instances that the algorithm ranks first. It may happen that the actual optimal fitness of a DOP varies widely as time goes on. If we directly represent the accuracy of a DOEA as the averaged fitness values of all instances, it will be dominated by large fitness. Thus, the second measure C_2 is represented as the average of normalized best fitness:

$$F_{\sum}(A_i) = \frac{1}{N}\sum_{t=1}^{N} \frac{G_+(t) - G(t|A_i)}{G_+(t) - G_-(t)} \quad (3)$$

where $G_+(t) = \max_j G(t|A_j)$ and $G_-(t) = \min_j G(t|A_j)$. An algorithm performs better if it is closer to zero.

D. Simulation Results

In this experiment, the test DOPs are 10 dimensional (i.e., $D = 10$). The number of fitness evaluations is fixed at 80,000 for each test DOP. Since the test algorithms are stochastic, their performances are evaluated based on statistics obtained from 100 independent runs. The parameters $\{\mathbf{R}_\mathbf{j}^{(\mathbf{i})}\}$ and $\{\boldsymbol{\alpha}_\mathbf{0}^{(\mathbf{i})}\}$ for $i = 1, 2, 3$ and $j = 1, 2, \ldots, 5$ in the DOPs are fixed for all EA in all 100 independent runs.

To illustrate the significance of DyHdEA, the leadership of the test algorithms is presented in Tables 2-4. Seen from the tables, DyHdEA is superior to the other five test algorithms in more than 87%, 69% and 63% time instances on F_1, F_2 and F_3 respectively. Compared to the second longest leading algorithm which is superior to the others on F_1 in around 9.71% time instances; on F_2 in around 9.51% and on F_3 in around 17.34%, the leading time of DyHdEA is sufficiently large to demonstrate the superior leadership of DyHdEA. Apart from demonstrating how frequently DyHdEA performs better than the other test algorithms, it is also important to show how frequently DyHdEA performs worse than the other algorithms. According to Tables II-IV, DyHdEA ranks 4th or lower in only 4.2% of the total time instances for F_1, which is the smallest portion amongst the test algorithms (i.e., 11.9% for AIA, 9.8 for jDE, 81% for UEP, 94.8% for CPSO, and 98.3% for PSO-CP). It ranks 4th or lower in around 3% and 8% of total time instances for F_2 and F_3, which are also the smallest portions amongst the test algorithms.

TABLE II
THE AVERAGED RATES OF RANKING OF THE TEST ALGORITHMS: F_1.

	1	2	3	4	5	6
DyHdEA	87.65%	5.76%	2.42%	1.77%	2.39%	0.00%
AIA	9.71%	24.81%	53.56%	7.71%	3.02%	1.19%
jDE	2.11%	61.82%	26.25%	8.64%	0.52%	0.66%
UEP	0.05%	7.23%	11.72%	53.73%	7.88%	19.39%
CPSO	0.00%	0.01%	5.21%	8.87%	82.07%	3.84%
PSO-CP	0.48%	0.36%	0.83%	19.28%	4.13%	74.92%

TABLE III
THE AVERAGED RATES OF RANKING OF THE TEST ALGORITHMS: F_2.

	1	2	3	4	5	6
DyHdEA	69.29%	19.37%	7.62%	2.28%	0.76%	0.68%
AIA	9.51%	23.05%	45.99%	21.40%	0.06%	0.00%
jDE	5.38%	53.89%	24.04%	6.78%	4.03%	5.88%
UEP	0.34%	2.03%	1.97%	28.38%	28.16%	39.12%
CPSO	0.02%	0.24%	0.54%	26.23%	53.18%	19.79%
PSO-CP	15.47%	1.41%	19.85%	14.93%	13.81%	34.53%

TABLE IV
THE AVERAGED RATES OF RANKING OF THE TEST ALGORITHMS: F_3.

	1	2	3	4	5	6
DyHdEA	63.59%	25.41%	2.79%	4.95%	3.03%	0.23%
AIA	15.64%	36.92%	16.70%	26.64%	2.69%	1.41%
jDE	17.34%	28.36%	48.15%	3.99%	1.47%	0.69%
UEP	1.50%	2.73%	3.19%	50.75%	11.58%	30.24%
CPSO	0.46%	3.88%	10.29%	11.43%	73.31%	0.63%
PSO-CP	1.47%	2.70%	18.87%	2.24%	7.92%	66.80%

TABLE V
THE RELATIVE FITNESS OF THE TEST ALGORITHMS.

	G_1	G_2	G_3
DyHdEA	0.0296	0.1371	0.0412
AIA	0.1322	0.4709	0.1837
jDE	0.0827	0.4191	0.0942
UEP	0.3869	0.7715	0.4317
CPSO	0.5048	0.7713	0.2533
PSO-CP	0.8449	0.6466	0.7747

Table V lists the relative fitness $F_{\sum}$ of the test algorithms on F_1, F_2 and F_3. The relative fitness of DyHdEA on F_1, F_2 and F_3 are 0.0296, 0.1371 and 0.0412 respectively. The relative fitness of the second ranked and the third ranked algorithms on F_1 are 0.0827 and 0.1322; the relative fitness of the second ranked and the third ranked algorithms on F_2 are 0.4191 and 0.4709, and the relative fitness of the second ranked and the third ranked algorithms on F_3 are 0.0942 and 0.1837. Seen from the results shown in Tables II-V, DyHdEA is significantly superior to other test algorithms in terms of

leadership and accuracy.

VI. Conclusion

Dynamic objective problem (DOP) raises two challenges to evolutionary algorithm. These challenges are comparing the individuals at different environment, and designing a search strategy that traces suddenly appearing optimum. In this paper, we propose a dynamic objective evolutionary algorithm (DOEA) that tackles these challenges through search history. The proposed DOEA, namely dynamic objective history driven evolutionary algorithm (DyHdEA), has the following features:

1) It does not assume the dynamics of DOP to be changing at discrete instances nor are periodic.
2) It enhances the space partitioning strategy in the binary space partitioning (BSP) tree, which eliminates the sub-region selection bias induced from the solution space with different lengths.
3) It estimates the objective value and solution density at any position of solution space from search history.
4) The proposed mutation operator adaptively switches amongst exploitation for fine tuning the currently found optimum, exploration for bringing individuals out of a local optimum and exploration for tracking the dynamic global optimum.
5) Both the estimation of objective landscape and solution density, as well as the adaptive mutation, are parameter-less.

DyHdEA is examined on three 10-dimensional DOPs: F_1, F_2 and F_3. Each of the DOPs is composed of five distinctive benchmarked basis functions. The DOP linearly transforms from one function to another as the number of fitness evaluations increases. Meanwhile, the objective landscape of each basis function keeps translating. The performance of DyHdEA is compared with five benchmark DOEAs including artificial immune algorithm, differential evolution, evolutionary programming, and particle swarm optimization. Seen from the experimental results, DyHdEA is superior to the other test algorithms in terms of both leadership and accuracy. The relative fitness of DyHdEA on F_1, F_2 and F_3 are 0.0296, 0.1371 and 0.0412 respectively. They are very close to zero (i.e., the optimal value) and are the highest values amongst the test algorithms. Meanwhile, they are significantly smaller than the second highest values: 2.8 times smaller on F_1, 3.1 times on F_2 and 2.3 times smaller on F_3. Thus, the superiority of DyHdEA, in terms of both leadership and accuracy, emphatically underlies the contribution of search history to DOP optimization.

References

[1] S. Y. Yuen and C. K. Chow, "A Genetic algorithm that adaptively mutates and never revisits," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 454-472, Apr. 2009.

[2] C. K. Chow and S. Y. Yuen, "Continuous non-revisiting genetic algorithm with random search space re-partitioning and one-gene-flip mutation," in *Proc. Congr. Evol. Comput.*, Jul. 2010, pp. 1 – 8.

[3] C. K. Chow and S. Y. Yuen, "A multi-objective evolutionary algorithm that diversifies population by its density," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 149-172, 2012.

[4] C. K. Chow and S. Y. Yuen, "An evolutionary algorithm that makes decision based on the entire previous search history," *IEEE Trans. Evol. Comput.*, vol. 15, no. 6, pp. 741-769, 2011.

[5] H. Richter, "Evolutionary Optimization in Spatio-temporal Fitness Landscapes" in *Parallel Problem Solving from Nature*, T. P. Runarsson et al. (Eds). Heidelberg: Spring-Verlag, 2006, pp. 1-10.

[6] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, pp. 341–359, 1997.

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

[8] M. M. Ali, C. Khompatraporn and Z. B. Zabinsky, "A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems," *J. Global Optim.*, vol. 31, no. 4, pp. 635-672, 2005.

[9] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization," Technical Report, Nanyang Technological University, Singapore, KanGAL Report #2005005, IIT Kanpur, India, 2005.

[10] V. K. Koumousis and C. P. Katsaras, "A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance," *IEEE Trans. Evol. Comput.*, vol. 10, no. 1, pp. 19-28, Feb. 2006.

[11] F. O. de Franca and F. J. Von Zuben, "A Dynamic Artificial Immune Algorithm Applied to Challenging Benchmarking Problems," in *Proc. Congr. Evol. Comput.*, Jul. 2009, pp. 423 – 430.

[12] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec and V. Zumer, "Dynamic Optimization using Self-Adaptive Differential Evolution," in *Proc. Congr. Evol. Comput.*, Jul. 2009, pp. 415 – 422.

[13] E. L. Yu and P. N. Suganthan, "Evolutionary Programming with Ensemble of Explicit Memories for Dynamic Optimization," in *Proc. Congr. Evol. Comput.*, Jul. 2009, pp. 431 – 438.

[14] C. Li and S. Yang, "A clustering particle swarm optimizer for dynamic optimization," in *Proc. Congr. Evol. Comput.*, Jul. 2009, pp. 439-446.

[15] L. Liu, S. Yang, and D. Wang, "Particle swarm optimization with composite particles in dynamic environments," *IEEE Trans. System, Man and Cybernetics*, Part B: Cybernetics, vol. 40, no. 6, pp. 1634-1648, Dec. 2010.