History-driven firefly algorithm for optimisation in dynamic and uncertain environments

Babak Nasiri*

Department of Computer Engineering and Information Technology,
Islamic Azad University,
Qazvin Branch, Qazvin, Iran
Email: nasiri.babak@qiau.ac.ir
*Corresponding author

Mohammad Reza Meybodi

Department of Computer Engineering and Information Technology,
Amirkabir University of Technology,
Tehran, Iran
Email: mmeybodi@aut.ac.ir

Abstract: Due to dynamic and uncertain nature of many optimisation problems in real-world, the applied algorithm in this environment must be able to continuously track the changing optima over the time. In this paper, we report a novel speciation-based firefly algorithm for dynamic optimisation, which improved its performance by employing prior landscape historical information. The proposed algorithm, namely history-driven speciation-based firefly algorithm (HdSFA), uses a binary space partitioning (BSP) tree to capture the important information about the landscape during the optimisation process. By utilising this tree, the algorithm can approximate the fitness landscape and avoid wasting the fitness evaluation for some unsuitable solutions. The proposed algorithm is evaluated on the most well-known dynamic benchmark problem, moving peaks benchmark (MPB), and also on a modified version of it, called MPB with pendulum-like motion among the environments (PMPB), and its performance is compared with that of several state-of-the-art algorithms in the literature. The experimental results and statistical test prove that HdSFA outperforms the most of the algorithms in different scenarios.

Keywords: dynamic environment; uncertain environment; firefly algorithm; history-driven approach; speciation-based algorithm; binary space partitioning; BSP; swarm intelligence.

Reference to this paper should be made as follows: Nasiri, B. and Meybodi, M.R. (xxxx) 'History-driven firefly algorithm for optimisation in dynamic and uncertain environments', *Int. J. Bio-Inspired Computation*, Vol. X, No. Y, pp.xxx–xxx.

Biographical notes: Babak Nasiri received his BS and MS in Computer science in Iran, in 2002 and 2004, respectively. He has been a PhD candidate in Computer Science from Qazvin Islamic Azad University, Qazvin, Iran from 2010. Prior that, he was the faculty member of Computer Engineering and IT Department at Qazvin Islamic Azad University from 2008. His research areas include soft computing, machine learning, nature-inspired algorithms, data mining and web mining.

Mohammad Reza Meybodi received his BS and MS in Economics from National University in Iran, in 1974 and 1977, respectively. He also received his MS and PhD degree from Oklahoma University, USA, in 1980 and 1983, respectively in Computer Science. Currently, he is a Full Professor in Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran. Prior to current position, he worked from 1983 to 1985 as an Assistant Professor at Western Michigan University, and from 1985 to 1991 as an Associate Professor at Ohio University, USA. His research interests include soft computing, learning system, wireless networks, parallel algorithms, sensor networks, grid computing and software development.

1 Introduction

Over the past decade, there has been a growing interest in proposing new methods for optimisation in dynamic and uncertain environments (Nguyen et al., 2012). The most important reason for this growing trend can be due to the dynamic nature of many optimisation problems in real-world. Having several dynamic problems, World Wide Web (WWW) is a perfect example of such environments. Web page clustering, web personalisation and web community identification are a number of examples from this dynamic environment.

Generally, the goal, challenges, performance metrics and benchmark problems are completely different for optimisation in dynamic environments rather than the static ones. Finding the global optima as close as possible is the only goal of optimisation in static environments. In dynamic ones, however, detecting changes and tracking optima over the time are the other goals as well. In many dynamic real-world optimisation problems, the old and new environments are somehow correlated to each other. As a result, adding the ability for the algorithm to learn from the previous one (Nguyen, 2011) can be another goal for a good optimisation algorithm to function more efficiently.

Among the most important challenges for optimisation in static environments, premature convergence and imbalance between exploration and exploitation can be mentioned. In dynamic ones, however, there are some more challenges due to happening changes in environment. They include transforming a local optima to a global one or vice versa, losing the position of all optima, losing diversity and out-dated memory are a number of them. Besides them, tracking one optima by more than one subpopulation (specie) at a time in multi-population approaches and having a short time interval between two changes in the environment are the other challenges as well.

Moreover, there are a number of performance metrics and benchmark problems that are commonly used for evaluating the efficiency of optimisation algorithms in dynamic environments different from the static ones. Offline error (OE) (Branke and Schmeck, 2002), offline performance (Nguyen et al., 2012) and best-error-before-change (Li et al., 2014) are three of the most widely used metrics in literature. However, moving peaks benchmark (MPB) (Branke, 1999b), DF1 (Morrison and De Jong, 1999) and XOR dynamic problem generator (Yang, 2003) are some of the most well-known benchmarks, as well. In this paper, MPB and OE are used as the benchmark problem and the performance metric, respectively.

Over the past two decades, a large number of nature-inspired algorithms are proposed for optimisation in multimodal, time-varying and dynamic environments. The most widely used algorithms are PSO (Li and Yang, 2009, 2011; Yang and Li, 2010; Branke and Schmeck, 2002; Blackwell and Branke, 2004, 2006; Li et al., 2006; Yazdani et al., 2011, 2013a, 2013b), GA (Oppacher and Wineberg, 1999; Yang, 2003; Rand et al., 2006; Andersen, 1991), ACO (Wang et al., 2007; Mavrovouniotis and Yang, 2011, 2013) and AFSA (Yazdani et al., 2012, 2014). In this paper, a novel firefly algorithm is proposed for optimisation in dynamic environments.

Being inspired from flashing behaviour of fireflies in the nature, firefly algorithm was introduced by Yang in 2008 which uses three idealised rules:

1 all fireflies are unisex

2 degree of their attractiveness is proportional to their brightness

3 brightness can be achieved by the objective function value.

This algorithm has been shown to perform well in variety of applications such as clustering (Senthilnath et al., 2011), dynamic optimisation (Nasiri and Meybodi, 2012) and image processing (Horng and Liou, 2011). Besides, some different modified versions of this algorithm are proposed in Farahani et al. (2011), Fister et al. (2013b), and Tilahun and Ong (2012). A comprehensive review of firefly algorithm and also a summary of its latest developments is provided in Fister et al. (2013a) and Yang (2014) as well.

The proposed approaches for optimisation in dynamic environments can be divided into six different categories:

1 maintaining diversity during the change

2 introducing diversity after change

3 multi-population approaches

4 memory-based approaches

5 prediction approaches

6 self-adaptive approaches (Nguyen, 2011).

Among them, the multi-population and memory-based approaches are the most commonly used methods in the literature. In this paper, a combination of the second, third and the forth approaches is proposed to take advantage of all of them.

As mentioned earlier, in many dynamic real-world optimisation problems, the current state is often similar or related to the previously seen environments. In such circumstances, keeping the past useful information may speedup the search process for similar solutions after a change in the environment. One way to maintain the previous information is the use of memory either in implicit or explicit types. Implicit memory stores the past information as part of an individual, whereas explicit one stores information separated from the population. Explicit memory has been much more widely studied and has produced much better performance on dynamic optimisation problems rather than implicit one (Barlow, 2009).

Explicit memory can be divided into two different categories: direct memory and associative memory. In most cases the direct memories are the previous good solutions (Yang and Yao, 2008; Yu and Suganthan, 2009). However, associative memory can include various types of information, such as the probability vector that created the best solutions (Yang and Yao, 2008), the probability of the

occurrence of good solutions in the landscape (Richter and Yang, 2008).

Unlike most of the direct and associative memory approaches, the proposed method stores all the past individuals, which are evaluated during the computation in each environment. At the first glance, it seems that it needs a large amount of memory and is not an economic method. However, in many real-world optimisation problems the fitness evaluation cost is much higher than individual generation cost. For such problems, the total number of fitness evaluation is limited and it is not efficient to throw away the information achieved from the past fitness evaluations (Chow and Yuen, 2011).

A non-revisiting genetic algorithm (NrGA) is proposed in Yuen and Chow (2007, 2009) which memorise all the solutions by a binary space partitioning (BSP) tree structure. This scheme uses the BSP tree as an explicit memory in a static optimisation problem to prevent the solutions from being evaluated more than one time during the run. The BSP tree is also utilised (Chow and Yuen, 2011) in continuous search space to guide the search by doing some adaptation on the mutation operator. In Leung et al. (2010, 2012), the BSP tree is utilised for creating an adaptive parameter control system. It automatically adjusts the parameters based on entire search historical information. In this paper, the BSP tree structure is utilised as an explicit memory to store the past useful information during the computation, as well.

Memory can improve the performance of the algorithms for optimisation in dynamic environments in several ways including speeding up the search process after an environmental change, building a dynamic model of the problem over the time, and also guiding the search direction to the promising areas. Even though, many memory-based optimisation algorithms are proposed for dynamic environments, there remains a need for an efficient algorithm that can utilise the memory in a way to accomplish most of the above benefits. In this paper, a novel swarm intelligence algorithm, namely history-driven speciation-based firefly algorithm (HdSFA) is proposed for this purpose.

The remainder of this paper is organised as follows: Section 2 presents the fundamental of the proposed HdSFA, and the structure of the explicit memories which used in this paper in detail. Section 3 investigates the experimental study of the proposed algorithm on MPB and a modified version of it named MPB with pendulum-like motion among the environments. Finally, Section 4 presents some concluding remarks.

2 Proposed algorithm

In the following subsections, first, a brief description of speciation-based firefly algorithm (SFA) is provided. Then, the explicit memory scheme and the proposed approach – HdSFA – are presented in detail.

2.1 Speciation-based firefly algorithm

In standard firefly algorithm, the swarm consists of a population of the fireflies each of which represents a solution in the search space. The fireflies mutually attracting each other proportion to their brightness and inversely to their distance. In a D-dimensional search space, the current position of firefly i and its intensity are represented as $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D})$ and $I(x_i) \sim f(x_i)$, respectively. The position of firefly i at time step $t + 1$ is updated based on the positions of firefly i and firefly j at time step t, if $f(x_j^t)$ is better than $f(x_i^t)$:

$$x_i^{(t+1)} = x_i^{(t)} + \beta_0 e^{-\gamma r_{i,j}^2} \left(x_j - x_i \right) + \alpha\varepsilon \tag{1}$$

where β_0 is the initial attractiveness, $r_{i,j}$ is the distance between firefly i and firefly j, α is a significance factor of randomisation parameter, and ε is a random number obtained from a uniform distribution $U(0, 1)$.

In Nasiri and Meybodi (2012), a speciation-based firefly algorithm (SFA) is introduced for optimisation in dynamic environments. This algorithm utilised the inherent capability of firefly algorithm in converging to more than one optima at a time; the authors called it 'auto-speciation behaviour'. Taking advantage of this behaviour, the firefly algorithms developed for optimisation in dynamic and multimodal environments does not need any extra working such as proximity based speciation (Cho et al., 2011) or clustering (Yang and Li, 2010) to generate multiple sub-populations (species). Furthermore, SFA does not need any exclusion mechanism between the species to make them away from each other. In most of multi-modal dynamic optimisation algorithm, exclusion is an inseparable mechanism to ensure that each local optima is covered only by one specie.

In SFA, the original firefly algorithm is modified as follows. First, by inspiration from the PSO algorithm, Pbest, the personal best of each firefly, is added to the algorithm. Second, the firefly moves from one position to another if and only if the fitness value of new position is better than the previous one. Third, the randomisation parameter of the firefly (α) is adapted in a way that improves the exploration and exploitation capability of the algorithm during each environment.

2.2 Explicit memory scheme

As mentioned before, memory can improve the performance of optimisation algorithms in dynamic environments, especially for the cyclic ones. Figure 1, shows the block diagram of the explicit memory scheme used in this paper and its relationship with the proposed approach. It contains two memories: short-term memory and long-term memory.

Figure 1 Block diagram of explicit memory scheme and its relationship with the proposed approach

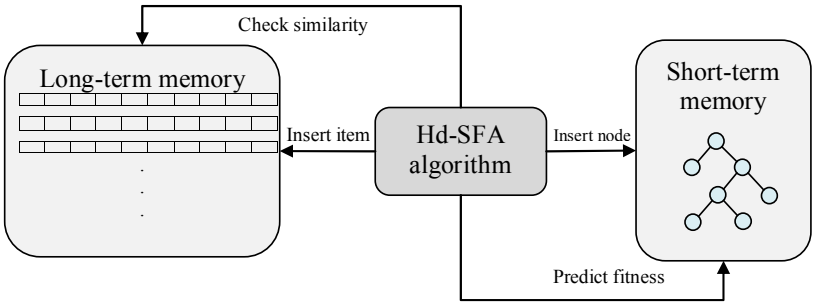
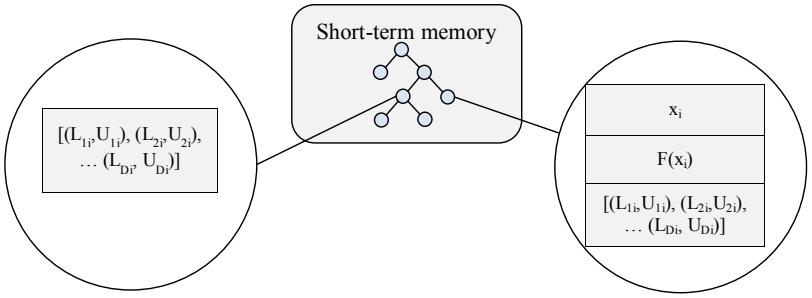


Figure 2 The structure of the each terminal and non-terminal node in short-term memory scheme



2.2.1 Short-term memory scheme

The short-term memory is responsible for storing useful information about the individuals in the current environment, temporarily. After each environmental change, the useful information will transfer to the long-term memory in a summarised format. The aim of this memory is guiding the search direction to the promising areas by building a dynamic model of the problem over the time.

This memory scheme uses the BSP tree for storing the valuable information during the run. In this tree, the whole search space is partitioned based on the distribution of the individuals in the environment. Each node in this tree represents a unique hyper-rectangular box in the search space and contains a representative solution for this sub-area. Suppose a parent node has two child nodes. The sub-areas represented by the child nodes are disjoint and their union is the sub-area of the parent. Figure 2 shows the structure of the short-term memory scheme used in this paper, in detail.

The process of adding a new solution to the short-term memory is shown in Algorithm 1. After each fitness evaluation, first, the short-term memory – BSP tree – must be traversed to find the sub-area which the new solution belongs to (*cur_node*). Then, the new solution must be compared with the representative solution of that sub-area (x_{cur_node}) to find a cut-off point. The middle of the largest difference between the all dimensions of these two solutions (x and x_{cur_node}) will be selected as a cut-off point for creating two new children nodes. For more detail on the BSP tree construction as well as a numerical example, please refer to Chow and Yuen (2011).

Algorithm 1 ShortTermMem_insertNode()

```
    Input: [x, f(x)]
1   begin
2       set cur_node to root node
3       while(cur_node has two child nodes: a and b)
4           Set Dimension j = arg max |a(k) – b(k)|
            where k ∈ [1, D]
5           if (|a(j) – x(j)| ≤ |b(j) – x(j)|) then
6               cur_node = a
7           else
8               cur_node = b
9           end if
10      end while
11      add two child nodes [x_cur_node, F(x_cur_node)] and [x, F(x)]
        to the parent node cur_node
12      convert the cur_node to a non-terminal node by
        removing [xcur_node, F(x_cur_node)]
13  end
```

The proposed algorithm utilises this dynamic model (BSP tree) to approximate the fitness value of each individual exactly before actual fitness evaluation; this is done by Algorithm 2. Taking advantage of this dynamic model, the number of wasted fitness evaluations will decrease significantly and as a result the proposed algorithm is able to guide the search direction to the promising areas in the search space.

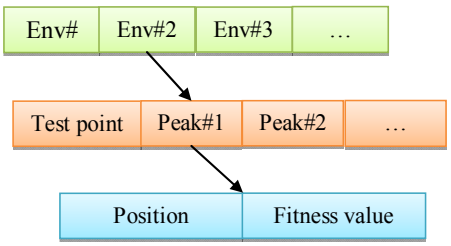
As we can see in Algorithm 2, the preliminary steps are completely similar to the Algorithm 1. The only difference is from line 11 to 15. If the short-term memory is mature enough, the algorithm will return the fitness value of the

representative for the area which x belongs to, as a predicted value for solution x. The short-term memory is mature if the percentage of correct prediction is more than a pre-defined threshold value (*maturity_threshold*); this percentage can be calculated after each fitness evaluation.

Algorithm 2 ShortTermMem_PredictIndividual()

Input: [x]
begin
  set *cur_node* to root node
  while(*cur_node* has two child nodes: a and b)
    Set Dimension j = arg max $|a(k) - b(k)|$ where $k \in [1, D]$
    if $(|a(j) - x(j)| \leq |b(j) - x(j)|)$ **then**
      cur_node = a
    else
      cur_node = b
    end if
  end while
  if the short-term memory is mature enough **then**
    return f_{cur_node} as a predicted value for x
  else
    return ∞
  end if
end
Output: [f_{cur_node}]

Figure 3 The long-term memory scheme structure (see online version for colours)



2.2.2 *Long-term memory scheme*

The long-term memory is responsible for storing useful information about the previous environments in a summarised format. This useful information includes the positions and fitness values of the all discovered optima and also a pre-specified test solution in the past environments. The aim of this summarised memory is to provide the ability for the algorithm to remember the previous environments if re-occur in the future. The long-term memory is not destroyed by changes happening in the environment and it is permanent during the computation. However, it becomes more mature and more informative over the time. Figure 3 shows the structure of this summarised long-term memory scheme in detail.

Algorithm 3 LongTermMem_CheckSimilarity()

Input: $[x_{testPoint}, f(x_{testPoint})]$
begin
  foreach *env#i* in long-term memory **do**
    if $|f(x_{testPoint}) - f(x_{testPoint})^{env\#i}| < sim\_thr$ **then**
      $f(x_{gOptima})$ = calculateFitness($x_{gOptima}{}^{env\#i}$)
      if $|f(x_{gOptima}) - f(x_{gOptima})^{env\#i}| < sim\_thr$ **then**
        return [true, env #i]
      end if
    end if
  end
  return [false, –1];
end
Output: [flagSimilarity, SimilarEnv#]

In order to take advantage of this long-term memory, after each environmental change, the pre-specified test solution should be evaluated and the result must be compared with all the previous fitness values for this test solution in previous environments. If the difference was less than a predefined threshold then one can say maybe a previous environment is appeared again. To ensure this event, the algorithm will re-evaluate the global optima ($x_{gOptima}$) for the most similar environment. By comparing the fitness values of the global optima in current environment and the most similar environment, we can ensure that a previous environment will appear again or not; this process is shown in Algorithm 3.

2.3 *History-driven speciation-based firefly algorithm*

In this section, a novel swarm intelligence algorithm, namely HdSFA is presented for optimisation in dynamic environments. This algorithm store the landscape historical information in two distinct explicit memories and build a dynamic model of the problem over the time to improve the accuracy and convergence speed of the algorithm as more as possible. Also, taking advantage of these memories, the algorithm is able to remember the previous environment if re-occur in the future.

HdSFA is a real-coded swarm intelligence algorithm. For a D-dimensional landscape $S \subset R^D$, a solution x of HdSFA is a 1 by D real-valued vector, i.e., $x \in R^D$. This algorithm employs two different populations – discoverer and tracker swarm – and also two different explicit memories to overcome the most challenges which exist in dynamic environments. Among them, we can mention to changes happening in the environment, transforming a local optima to a global, losing the position of all optima, losing diversity, converging two sub-species to one local optima, out-dated memory and short interval of time between two changes in the environment.

The proposed algorithm begins with initialising the swarms and short-term and long-term memories. Afterwards, specie identification, change identification, discoverer and tracker swarm processing and also fine

tuning process are repeated until the number of fitness evaluations exceeds a pre-determined value. Algorithm 4 shows the block diagram of the HdSFA. In the following, each step in this algorithm is described in detail.

Algorithm 4 Proposed algorithm – HdSFA

```
1  begin
2    call Initialisation()
3    repeat
4      call specie_identification()
5      call change_identification()
6      /*----discoverer swarm process begin------*/
7      if discoverer swarm is active then
8        call Swarm_Moving(Discoverer)
9        call Exclusion(Discoverer)
10       call Discoverer_Convergence_Checking()
11     end if
12     /*------ discoverer swarm process end------*/
13     /*------ tracker swarm process begin------*/
14     foreach active swarm s_i in tracker S do
15       call Swarm_Moving(si)
16     end
17     call Specie_Freezing()
18     /*------ tracker swarm process end------*/
19     call fine_Tuning()
20   until stopping criterion is met
21 end
```

Algorithm 5 Initialisation()

```
1  begin
2    Initialise the discoverer swarm S_discoverer = [x_i]
3    for i=1, 2, …, Discoverer_number.
4      Initialise the tracker swarm S_tracker = [x_i]
5    end
6    for i=1, 2, …, Tracker_number.
7      Initialise the short-term memory STM to consist of
       a root node only.
8    end
9    Initialise the long-term memory LTM.
10   Initialise the x_test-point and calculate f(x_test-point).
11   Activate the discoverer swarm.
12   foreach Firefly i in discoverer swarm F do
13     f(x_i) = calculateFitness(xi)
14     ShortTermMem_insertNode([x_i, f(x_i)])
15   end
16 end
```

The initialisation process is shown in Algorithm 5. It utilises two firefly swarms with different configurations and purposes for optimisation in dynamic environment. The discoverer swarm is configured somehow to explore the new promising areas in the search space effectively and efficiently. Once the discoverer swarm converged to a local optima, the k-best fireflies in it should be migrated to the tracker swarm, and the discoverer swarm should be re-initialised. The tracker swarm is responsible for chasing the discovered optima in the environment and does not have any firefly initially. Also, the short-term memory will initialise by adding the root node of the BSP tree. However, the long-term memory does not have any entry at first.

Algorithm 6 Specie_identification()

```
   input: [tracker swarm sw, r_excl]
1  begin
2    Calculate distance matrix M where M_i,j is Euclidean
     distance between firefly i and j.
3    Calculate binary matrix B form M, where B_i,j is one
4    if M_i,j is less than r_excl
5    Create a graph from matrix B and consider each
     connected graph as one specie.
6    Return the best firefly in each specie as a Gbest firefly
7  end
8  Output: [list of Gbest firefly in each specie]
```

Algorithm 7 Change_identification()

```
1  begin
2    /* checking behavioural change in the algorithm*/
3    if the offline error increased more than k times
     sequentially then
4      /* ------ re-evaluate a test solution -------*/
5      f(x_test solution)^new = calculate_fitness(x_test solution)
6      if f(x_test solution)^new <> f(x_test solution)old then
7        /*------ change happened ------*/
8        Add all discovered optima and pre-specified test
         solution into long-term memory.
9        Reinitialise short-term memory.
10       Re-evaluate the pre-specified test solution.
11       [flagSimilarity, SimilarEnv#] =
         LongTermMem_CheckSimilarity()
12       if flagSimilarity is true then
13         Restore position of all local optima from
           SimilarEnv# into current environment.
14       else
15         Diversify all the fireflies in tracker swarm by
           adding a random value to the GBestPosition
           of each swarm between [-Vlength*P,
           Vlength*P]
16       end if
17     end if
18   end if
19 end
```

Specie identification is the first process in the main loop of the algorithm. The aim of this process is to identify the whole tracker swarms in the environment by doing a simple clustering as shown in Algorithm 7.

Generally, there are two methods for discovering changes in dynamic environment, including re-evaluating a pre-specified test solution, and monitoring behavioural changes in the algorithm during the run. The current paper presents a hybrid approach based on two mentioned methods to discover any changes in the environment as shown in Algorithm 7. In this algorithm first, the trend of a performance KPI should be checked during k-latest fitness evaluations. This KPI is the mean of all fitness evaluations after the latest change in environment. If the trend of this KPI is increasing, the algorithm needs to re-evaluate the pre-specified test solution to ensure that any changes have happened in the environment. This hybrid approach can help the algorithm to decrease the number of wasted fitness evaluations for discovering changes in the environment. In this paper, we used 4 for the *k* value.

After the change is discovered in the environment, two actions should be done. First, all discovered optima should be added to the long-term memory and then the fireflies in each tracker species should be diversified. This diversification can be done by adding a random value in range of [–Vlength*P, Vlength*P] to the Gbest Position of each firefly specie.

Algorithm 8 Swarm_Moving()

```
    Input: [swarm sw]
1   begin
2     foreach firefly I in SW
3       foreach firefly j in SW
4         if f(x_i) > f(x_j)
5           Move firefly i towards j based Equation (1)
6         end if
7         p(x_i^new)=ShortTermMem_PredictIndividual(x_i^new)
8         if p(x_i^new) <=Pbest(x_i^old) then
9           f(x_i) = calculate_fitness(x_i^new)
10          Pbest(x_i) = f(x_i)
11        else
12          x_i = Pbest(x_i)
13        end
14      end for
15    end for
16  end
```

After the change identification, the algorithm will initiate the discoverer swarm process if this swarm is active in the environment. The discoverer swarm process includes three actions including swarm moving, exclusion and convergence checking.

The swarm moving, as shown in Algorithm 8, is very similar to the standard firefly algorithm with some minor changes as below:

1 Personal-best solution (Pbest) is added for each firefly by inspiration from particle swarm optimisation.

2 The fitness value of the each firefly should be predicted by the BSP tree whenever the firefly moves to another position. If the predicted value for this new position is equal or better than its Pbest fitness value, the actual fitness evaluation will be happened on new position of the firefly.

3 The position of each firefly will change if the fitness value of the new position is better than the current position.

Algorithm 9 exclusion()

```
1  begin
2    foreach firefly i in S_firefly do
3      if || x_i, G_f || <r_excl then
4        re-initialise discoverer swarm F
5      end
6    end
7  end
```

Also, the exclusion and convergence checking algorithms are shown in Algorithms 9 and 10. In exclusion algorithm, the discoverer swarm will be re-initialised if the distance between the global best firefly in discoverer swarm and fireflies in tracker specie is less than r_{excl}.

Algorithm 10 discovererConvergenceChecking()

```
1  begin
2    if (f(G_f)^(t–2)–f(G_f)^t) < r_conv) or (||(G_f)^(t–2), (G_f)^t|| < r_conv/5)
     then
3      add m-best fireflies in the discoverer swarm to the
       tracker swarm
4      re-initialise the discoverer swarm
5    end
6  end
```

Based on Algorithm 10, the discoverer swarm is converged if the difference between the fitness values of best global fireflies during the last two iterations is less than r_{conv} or the distance between the best global fireflies during the last two iterations is less than $r_{conv/5}$. After the convergence of discoverer swarm, the *m* best fireflies in it will move to the tracker swarm and the discoverer swarm will be re-initialised.

After discoverer swarm process, the tracker swarm will begin by swarm moving, freezing and fine-tuning process. The swarm moving for the tracker swarm uses the same algorithm as discoverer swarm (Algorithm 8) but, with different configuration.

The specie freezing, as shown in Algorithm 11, is an action which used for freezing the species which have not any influence in improvement of the algorithm's result. In this manner, the algorithm does not waste the fitness evaluations to improve some fireflies which have not any influence on performance of the whole algorithm. A specie will be frozen if its GbestPosition or GbestFitness did not change considerably during their last three movements.

Algorithm 11 specie_freezing()

```
1  begin
2    foreach specie i in tracker swarm
3      if (f(Gbest_i)^(t-3) – f(Gbest_i)^t) < r_conv) or (||(Gbest_i)^(t-3),
       (Gbest_i)^t|| < r_conv/5) then
4        freeze the firefly i
5      end
6    end for
7  end
```

Algorithm 12 Fine-tuning()

```
   Input: [Firefly Gbest_firefly]
1  begin
2    x_i = Gbest_position
3    for counter =1 to attempt_number
4      calculate x_i^new based on Equation (2)
5      p(x_i^new) = ShortTermMem_PredictIndividual(x_i^new)
6      if p(x_i^new) <= f(Gbest) then
7        f(x_i) = calculate_fitness(x_i^new)
8        Pbest(x_i) = f(x_i)
9      else
10       x_i = Pbest(x_i)
11     end
12   end
13 end
```

The goal of fine-tuning process (Algorithm 12) is to improve the overall result of the algorithm as much as possible. That is why this process tries to enhance the quality of global best firefly in environment by doing some local search around it. Predicting new solutions after each movement can help the algorithm to reduce the number of wasted fitness evaluation during this phase. New position can be achieved by using equation (2).

$$x_i^{new} = Gbest\_position + \left(r_{cloud} \times R_j\right) \tag{2}$$

where R_j is a random number with uniform distribution in [–1, 1]. Hence, the x_i^{new} is located in the radius of r_{cloud} from *Gbest*. Also, r_{cloud} will decrease based on equation (3).

$$r_{cloud} = r_{cloud} \times \left(w_{\min} + \left(rand \times \left(w_{\max} - w_{\min}\right)\right)\right) \tag{3}$$

where $W_{\min}$, $W_{\max}$ are the lower and the upper bounds for multiplying in r_{cloud}.

3 Experimental study

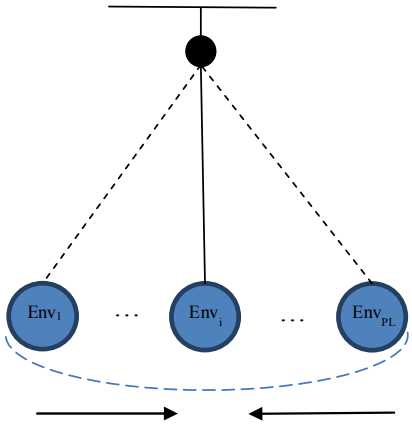
3.1 Benchmark problems

Due to time consuming evaluations of fitness functions in the real-world problems, the computation cost of an algorithm in most applications is expressed in terms of the number of evaluations; as is done in this paper.

The first benchmark used in this paper is the MPB (Branke, 1999a, 1999b), which is the most commonly used benchmark for evaluating algorithms in dynamic environments (Yaochu and Branke, 2005). The reason for this popularity among the researchers is related to the large number of adjustable parameters in this benchmark for generating a wide range of dynamic environments. Using this benchmark, we can study algorithms from a number of perspectives.

In MPB, there are a number of peaks in a D-dimensional space which their widths, heights and positions will change by a fixed amount s (shift severity), every α fitness evaluation. The goal of the problem is to locate and track the global peak (optima) in environment during the all changes which happen to environment over the time. More detail about MPB, can be fined in Branke (1999b).

Figure 4 Pendulum motion of environments (see online version for colours)



The second benchmark used in this paper, is a modified version of MPB with pendulum motion characteristic (Figure 4), namely pendulum-moving peaks benchmark (PMPB). In fact, most of the dynamic optimisation algorithms that enhanced with memory are expected to work better on environments that re-appear in the future. In this benchmark, pendulum length (*PL*) parameter is introduced to control the linear pendulum motion accurately and correctly.

The implementation for PMPB is as follows. For the first *PL* changes in environments, the whole peak positions should be stored in an array memory for the future use. Afterwards, for every change in the environment, the position of the peaks should be achieved from the next environment returned from Algorithm 13. It should be mentioned that the first value for the *direction* parameter should be set to *right*.

Algorithm 13 Environment_selector()

```
Input: Environment i, Direction dir,
Pendulum length PL
begin
    if dir == right
        if i < PL then
            i = i + 1
        else if i == PL then
            dir = left
            i = i – 1
        end if
    else if dir == left then
        if i >1 then
            i = i – 1
        else if i ==1 then
            dir = right
            i = i + 1
        end if
    end if
    return i //the index of the next environment
end
Output: next environment
```

3.2 Performance metric

There are several performance metrics used in the literature to measure the efficiency of optimisation algorithm in dynamic environments (Weicker, 2002). In order to make our results comparable with other state-of-the-art algorithms, the metric selected in this paper is the *OE* which is the average of the differences between the values found so far by the algorithm and the global optimum value (Branke, 199a, 1999b).

$$OE = \frac{1}{FE}\sum_{t=1}^{FE}\left(f\left(gbest(t)\right) - f\left(globalOptimum(t)\right)\right) \quad (4)$$

where *FEs* is the total number of fitness evaluations, and *globalOptimum*(*t*) and *gbest*(*t*) are the global optimum value and the best value found by the algorithm at the t^{th} fitness evaluation, respectively.

3.3 Parameter setting

In this section, the required parameters' setting for the benchmarks and algorithm is provided. For the MPB problem, the parameters are set according to scenario 2 of this benchmark (Branke, 1999b) in Table 1. However, in order to study the various aspects of the algorithm for some important parameters such as number of peaks, change frequency, dimensions, and change severity, several values are specified in this table.

In addition, for the PMPB, the *PL* parameter is set to 5, 10 and 20. It means that the environments will re-appear in pendulum manner after 5, 10 and 20 changes in the environment.

Table 1 MPB parameters setting

Parameter	*Value*
Number of peaks, M	1, 5, 10, 20, 30, 50, 100, 200
Change frequency, α	500, 1,000, 2,500, 5,000, 10,000
Height change	7.0
Width change	1.0
Peaks shape	Cone
Basic function	No
Change severity, s	1, 2, 3, 5
Number of dimensions, D	5
Correlation coefficient, λ	0
Peaks location range	[0–100]
Peak height	[30.0–70.0]
Peak width	[1–12]
Initial value of peaks	50.0
PL	5, 10, 20

Furthermore, the initial configuration of the HdSFA is as follows.

The number of fireflies in discoverer swarm is 10 and in the tracker swarms is 5 as mentioned in Blackwell et al. (2008). Also, the [α, γ] for the discoverer and tracker swarms are [1, 0] and [10, 1], respectively. The r_{excl} and r_{conv} parameters are considered according to Blackwell and Branke (2006). Furthermore, the value of P, $W_{\min}$ and $W_{\max}$ are set to 0.5, 0.6 and 0.9respectively and r_{cloud} is set to 0.2*s (change severity). Also, *maturity_thresholdandsim_thr* are set to 0.7 and 0.9, respectively.

3.4 Comparison between HdSFA and other state-of-the-art algorithms on MPB problem

In this part, the efficiency and performance of HdSFA is tested on different MPB configurations and is compared with several state-of-the-art algorithms. The experimental results are obtained by the average of 50 executions of algorithm. Also, in each of them, the initial position of the fireflies and the peaks are determined randomly and with different random seeds. Termination condition of the algorithm is set to 100 changes in an environment which is equal to 100 * α. Some of the presented results of the state-of-art algorithms are obtained by implementing the methods and some of them are extracted from the related references. In Table 2 to 6, the efficiency of seven related algorithms including: mQSO (Blackwell and Branke, 2006), AmQSO (Blackwell et al., 2008), mPSO (Kamosi et al., 2010a), HmPSO (Kamosi et al., 2010b) APSO (Rezazadeh et al., 2011), FTMPSO (Yazdani et al., 2013b) and SFA (Nasiri and Meybodi, 2012) is compared with that of HdSFA. Furthermore, for each environment, t-tests with a significance level of 0.05 have been applied and the result of the best performing algorithms is printed in bold. When the results for the best performing algorithms are not significantly different, all are printed in bold.

Table 2 Comparison of OE (standard error) on MP problem with different number of peaks, dimension = 5, change frequency = 500 and shift severity = 1

Algorithm	*Number of peaks*							
	1	*5*	*10*	*20*	*30*	*50*	*100*	*200*
mQSO(5, 5q)	33.67 (3.42)	11.91 (0.76)	9.62 (0.34)	9.07 (0.25)	8.80 (0.21)	8.72 (0.20)	8.54 (0.16)	*8.19 (0.17)*
AmQSO	3.02 (0.32)	5.77 (0.56)	5.37 (0.42)	6.82 (0.34)	7.10 (0.39)	7.57 (0.32)	7.34 (0.31)	*7.48 (0.19)*
mPSO	8.71 (0.48)	6.69 (0.26)	7.19 (0.23)	8.01 (0.19)	8.43 (0.17)	8.76 (0.18)	8.91 (0.17)	*8.88 (0.14)*
HmPSO	8.53 (0.49)	7.40 (0.31)	7.56 (0.27)	7.81 (0.20)	8.33 (0.18)	8.83 (0.17)	8.85 (0.16)	*8.85 (0.16)*
APSO	4.81 (0.14)	4.95 (0.11)	5.16 (0.11)	5.81 (0.08)	6.03 (0.07)	5.95 (0.06)	6.08 (0.06)	*6.20 (0.04)*
FTMPSO	1.76 (0.09)	2.93 (0.18)	3.91 (0.19)	4.83 (0.19)	*5.05 (0.21)*	*4.98 (0.15)*	*5.31 (0.11)*	*5.52 (0.21)*
SFA	4.72 (0.12)	4.88 (0.12)	5.11 (0.14)	5.72 (0.13)	5.97 (0.12)	5.94 (0.15)	6.15 (0.08)	*6.18 (0.11)*
HdSFA	*1.55 (0.03)*	*2.61 (0.03)*	*3.49 (0.02)*	*4.45 (0.03)*	*5.18 (0.03)*	6.00 (0.02)	6.56 (0.02)	*6.40 (0.02)*

Table 3 Comparison of OE (standard error) on MP problem with different number of peaks, dimension = 5, change frequency = 1,000 and shift severity = 1

Algorithm	*Number of peaks*							
	1	*5*	*10*	*20*	*30*	*50*	*100*	*200*
mQSO(5, 5q)	18.60 (1.63)	6.56 (0.38)	5.71 (0.22)	5.85 (0.15)	5.81 (0.15)	5.87 (0.13)	5.83 (0.13)	*5.54 (0.11)*
AmQSO	2.33 (0.31)	2.90 (0.32)	4.56 (0.40)	5.36 (0.47)	5.20 (0.38)	6.06 (0.14)	4.77 (0.45)	*5.75 (0.26)*
mPSO	4.44 (0.24)	3.93 (0.16)	4.57 (0.18)	4.97 (0.13)	5.15 (0.12)	5.33 (0.10)	5.60 (0.09)	*5.78 (0.09)*
HmPSO	4.46 (0.26)	4.27 (0.08)	4.61 (0.07)	4.66 (0.12)	4.83 (0.09)	4.96 (0.03)	5.14 (0.08)	*5.25 (0.08)*
APSO	2.72 (0.04)	2.99 (0.09)	3.87 (0.08)	4.13 (0.06)	4.12 (0.04)	4.11 (0.03)	4.26 (0.04)	*4.21 (0.02)*
FTMPSO	0.89 (0.05)	1.70 (0.10)	2.36 (0.09)	3.01 (0.12)	*3.06 (0.10)*	3.29 (0.10)	3.63 (0.09)	*3.74 (0.09)*
SFA	2.45 (0.12)	2.71 (0.06)	3.64 (0.04)	4.01 (0.07)	4.02 (0.08)	4.12 (0.07)	4.40 (0.07)	*4.43 (0.07)*
HdSFA	*0.80 (0.01)*	*1.23 (0.02)*	*1.78 (0.02)*	*2.44 (0.01)*	*2.96 (0.02)*	*3.04 (0.02)*	*3.48 (0.02)*	*3.62 (0.01)*

Table 4 Comparison of OE (standard error) on MP problem with different number of peaks, dimension = 5, change frequency = 2,500 and shift severity = 1

Algorithm	*Number of peaks*							
	1	*5*	*10*	*20*	*30*	*50*	*100*	*200*
mQSO(5, 5q)	7.64 (0.64)	3.26 (0.21)	3.12 (0.14)	3.58 (0.13)	3.63 (0.10)	3.63 (0.10)	3.58 (0.08)	*3.30 (0.06)*
AmQSO	0.87 (0.11)	2.16 (0.19)	2.49 (0.10)	2.73 (0.11)	3.24 (0.18)	3.68 (0.15)	3.53 (0.14)	*3.07 (0.12)*
mPSO	1.79 (0.10)	2.04 (0.12)	2.66 (0.16)	3.07 (0.11)	3.15 (0.08)	3.26 (0.07)	3.31 (0.05)	*3.36 (0.05)*
HmPSO	1.75 (0.10)	1.92 (0.11)	2.39 (0.16)	2.46 (0.09)	2.57 (0.05)	2.65 (0.05)	2.72 (0.04)	*2.81 (0.04)*
APSO	1.06 (0.03)	1.55 (0.05)	2.17 (0.07)	2.51 (0.05)	2.61 (0.02)	2.66 (0.02)	2.62 (0.02)	*2.64 (0.01)*
FTMPSO	0.39 (0.02)	0.91 (0.08)	1.21 (0.06)	1.66 (0.05)	1.87 (0.05)	2.09 (0.07)	2.22 (0.06)	*2.22 (0.07)*
SFA	0.85 (0.06)	1.49 (0.07)	1.95 (0.04)	2.28 (0.05)	2.46 (0.05)	2.57 (0.05)	2.74 (0.04)	*2.76 (0.04)*
HdSFA	*0.30 (0.01)*	*0.59 (0.01)*	*0.81 (0.01)*	*1.23 (0.01)*	*1.46 (0.01)*	*1.55 (0.01)*	*1.83 (0.01)*	*1.88 (0.01)*

Table 5 Comparison of OE (standard error) on MP problem with different number of peaks, dimension = 5, change frequency = 5,000 and shift severity = 1

Algorithm	*Number of peaks*							
	1	*5*	*10*	*20*	*30*	*50*	*100*	*200*
mQSO(5, 5q)	2.24 (0.05)	1.82 (0.08)	1.85 (0.08)	2.48 (0.09)	2.51 (0.10)	2.53 (0.08)	2.35 (0.06)	*2.24 (0.05)*
AmQSO	2.62 (0.10)	1.01 (0.09)	1.51 (0.10)	2.00 (0.15)	2.19 (0.17)	2.43 (0.13)	2.68 (0.12)	*2.62 (0.10)*
mPSO	2.24 (0.05)	1.82 (0.08)	1.85 (0.08)	2.48 (0.09)	2.51 (0.10)	2.53 (0.08)	2.35 (0.06)	*2.24 (0.05)*
HmPSO	2.62 (0.10)	1.01 (0.09)	1.51 (0.10)	2.00 (0.15)	2.19 (0.17)	2.43 (0.13)	2.68 (0.12)	*2.62 (0.10)*
APSO	0.53 (0.01)	1.05 (0.06)	1.31 (0.03)	1.69 (0.05)	1.78 (0.02)	1.95 (0.02)	1.95 (0.01)	*1.90 (0.01)*
FTMPSO	0.18 (0.01)	0.47 (0.05)	0.67 (0.04)	0.93 (0.04)	1.14 (0.04)	1.32 (0.04)	1.61 (0.03)	*1.67 (0.03)*
SFA	0.42 (0.03)	0.89 (0.07)	1.05 (0.04)	1.48 (0.05)	1.56 (0.06)	1.87 (0.05)	2.01 (0.04)	*1.99 (0.06)*
HdSFA	*0.14 (0.00)*	*0.30 (0.01)*	*0.51 (0.01)*	*0.7 (0.01)*	*0.88 (0.01)*	*1.17 (0.01)*	*1.23 (0.01)*	*1.26 (0.01)*

Table 6 Comparison of OE (standard error) on MP problem with different number of peaks, dimension = 5, change frequency = 10,000 and shift severity = 1

Algorithm	*Number of peaks*							
	1	*5*	*10*	*20*	*30*	*50*	*100*	*200*
mQSO(5, 5q)	1.90 (0.18)	1.03 (0.06)	1.10 (0.07)	1.84 (0.08)	2.00 (0.09)	1.99 (0.07)	1.85 (0.05)	*1.71 (0.04)*
AmQSO	0.19 (0.02)	0.45 (0.04)	0.76 (0.06)	1.28 (0.12)	1.78 (0.09)	1.55 (0.08)	1.89 (0.14)	*2.52 (0.10)*
mPSO	0.27 (0.02)	0.70 (0.10)	0.97 (0.04)	1.34 (0.08)	1.43 (0.05)	1.47 (0.04)	1.50 (0.03)	*1.48 (0.02)*
HmPSO	-	-	-	-	-	-	-	-
APSO	0.25 (0.01)	0.57 (0.03)	0.82 (0.02)	1.23 (0.02)	1.39 (0.02)	1.46 (0.01)	1.38 (0.01)	*1.36 (0.01)*
FTMPSO	0.09 (0.00)	*0.31 (0.04)*	0.43 (0.03)	0.56 (0.01)	0.69 (0.09)	0.86 (0.02)	1.08 (0.03)	*1.13 (0.04)*
SFA	0.26 (0.03)	0.53 (0.04)	0.72 (0.02)	0.91 (0.03)	0.99 (0.04)	1.19 (0.04)	1.44 (0.04)	*1.52 (0.03)*
HdSFA	*0.06 (0.00)*	*0.27 (0.01)*	*0.30 (0.01)*	*0.4 (0.00)*	*0.59 (0.00)*	*0.76 (0.00)*	*0.87 (0.00)*	*0.99 (0.01)*

Table 7 Comparison of OE (standard error) on MP problem with different shift severities, dimension = 5, peak number = 10, change frequency = 5,000

Algorithm	*Number of peaks*			
	1	*2*	*3*	*5*
mQSO(5, 5q)	1.85 (0.08)	2.40 (0.06)	3.00 (0.06)	*4.24 (0.10)*
AmQSO	1.51 (0.10)	2.09 (0.08)	2.72 (0.09)	*3.71 (0.11)*
mCPSO	4.89 (0.11)	3.57 (0.08)	2.80 (0.07)	*2.08 (0.07)*
SPSO	2.51 (0.09)	3.78 (0.09)	4.96 (0.12)	*6.76 (0.15)*
rSPSO	1.50 (0.08)	1.87 (0.05)	2.40 (0.08)	*3.25 (0.09)*
PSO-CP	1.31 (0.06)	1.98 (0.06)	2.21 (0.06)	*3.20 (0.13)*
FTMPSO	0.67 (0.04)	1.20 (0.06)	*1.40 (0.09)*	*1.69 (0.07)*
SFA	1.05 (0.04)	1.44 (0.06)	2.06 (0.07)	*2.89 (0.13)*
HdSFA	*0.51 (0.01)*	*0.82 (0.01)*	1.50 (0.03)	*2.03 (0.02)*

Table 8 The rank of algorithms on MP problem based on different change frequency

Algorithm	*Change frequency*					
	500	*1,000*	*2,500*	*5,000*	*10,000*	*Overall ranking*
mQSO(5, 5q)	8	8	8	7	7	*8*
AmQSO	5	5	6	5	5	*5*
mPSO	7	7	7	7	6	*7*
HmPSO	6	6	5	5	8	*6*
APSO	4	4	4	4	4	*4*
FTMPSO	1	2	2	2	2	*2*
SFA	3	3	3	3	3	*3*
HdSFA	2	1	1	1	1	*1*

Table 9 Comparison of OE (standard error) on PMPB and MPB problems with different number of peaks, dimension = 5, change frequency = 5,000 and shift severity = 1

Algorithm	*Problem*	*Number of peaks*							
		1	*5*	*10*	*20*	*30*	*50*	*100*	*200*
HdSFA	MPB	0.14 (0.00)	0.30 (0.01)	0.51 (0.01)	0.70 (0.01)	0.88 (0.01)	1.17 (0.01)	1.23 (0.01)	1.26 (0.01)
HdSFA	PMPB l = 5	0.05 (0.00)	0.12 (0.01)	0.20 (0.01)	0.29 (0.01)	0.34 (0.01)	0.39 (0.01)	0.43 (0.01)	0.45 (0.01)
HdSFA	PMPB l = 10	0.07 (0.00)	0.16 (0.01)	0.27 (0.01)	0.38 (0.01)	0.44 (0.01)	0.54 (0.01)	0.65 (0.01)	0.66 (0.01)
HdSFA	PMPB l = 20	0.11 (0.00)	0.24 (0.01)	0.38 (0.01)	0.55 (0.01)	0.62 (0.01)	0.72 (0.01)	0.83 (0.01)	0.90 (0.01)

As can be seen from Tables 2 to 6, the efficiency of the HdSFA, in most of the cases outperform that of other seven state-of-the-art algorithms. Utilising the new change identification method in comparison with techniques that used before on other state-of-the-art algorithms as like as (Yazdani et al., 2013b), is one of the most prominent reasons for superiority of the proposed method. In most of them, a test solution is used to discover the change in the environment. When the time interval between two consecutive changes in environment is extremely high, re-evaluating a test solution in each iteration can waste several fitness evaluations during the computation. E.g., if the mean of fitness evaluations in each iteration and the change frequency are 100 and 10,000, respectively, then at least (10,000/100–1 = 99) fitness evaluation will be wasted in each environment. By multiplying 99 by 100 (number of change in environment), 9,900 fitness evaluation will be wasted during the 100 times of changes in environment.

In Table 7, the efficiency of the proposed algorithm on MPB with 10 peaks, shift severities of 1, 2, 3, 5, dimension of 5 and a change frequency of 5,000 is compared with that of other 8 algorithms including: mQSO (Blackwell and Branke, 2006), AmQSO (Blackwell et al., 2008), mCPSO (Blackwell and Branke, 2006), SPSO (Bird and Li, 2008), rSPSO (Du and Li, 2008), PSO-CP (Liu et al., 2010), FTMPSO (Yazdani et al., 2013b), and SFA (Nasiri and Meybodi, 2012). By increasing the value of shift severity, the process of tracking peaks becomes more complex, since the peaks move to further distances after each change in the environment. As can be seen in Table 7, the more the shift severity increases, the more the efficiency of all algorithms decreases. However, less degradation is involved in HdSFA and FTMPSO by increasing shift severity, compared to other algorithms. Also, FTMPSO will outperform the HdSFA when the shift severity is more than two.

Table 8 shows the rank of algorithms on MP problem based on different change frequency. Each value is calculated as follows. First, rank all the OEs in Tables 2 to 6 in column order for different algorithms. Then, add the previous ranks in each row and in each table for different number of peaks. Finally, rank the values of achieved column for each table (change frequency). As can be seen, the rank of proposed algorithm is one at the total. Also, FTMPSO and SFA are in the second and third places in this ranking.

3.5 Comparison of HdSFA on PMPB problem

In this section, the performance of the proposed algorithm is evaluated on PMPB problem. This benchmark is defined for the first time in this paper and is described in Section 3.1 completely. As we said before, the PL parameter is set to 5, 10 and 20. It means that the environments will re-appear in pendulum manner after 5, 10 and 20 changes in an environment. Table 9 shows the performance of HdSFA on MPB and PMPB. As we can see in Table 9, the more the pendulum length parameter increases, the more efficiency of all algorithms decreases.

4 Conclusions

In this paper, a novel speciation based firefly algorithm was proposed for optimisation in dynamic environments. An explicit memory is utilised to store the past useful information during the computation, as well. Taking advantage of this explicit memory, the number of wasted fitness evaluations will decrease significantly and as a result the proposed algorithm is able to guide the search direction to the promising areas in the search space. The proposed algorithm could quickly find the peaks in the problem space and follow them after an environmental change. In the proposed algorithm, swarms in the problem space were categorised into discoverer and tracker swarms which were configured somehow to complete each other during the search process. Also, the proposed algorithm benefited from the inherent capability of firefly algorithm in converging to more than one peak during the run for removing the exclusion method between each species in the tracker swarm.

Efficiency of the proposed algorithm has been evaluated on MPB, and also on a modified version of it, called MPB with pendulum-like motion among the environments (PMPB), and its performance is compared with that of several state-of-the-art algorithms in the literature. The experimental results and comparative studies showed the superiority of the proposed method. Diverse experiments showed that the proposed algorithm involved a high convergence speed which is significantly important in designing any optimisation algorithm in dynamic environments.

We would like to use the proposed algorithm on different dynamic benchmark problems and also dynamic real-world applications. Currently, we are working on its applications to the clustering in web, and to the personalisation of web pages dynamically. Adapting the algorithm to dynamic multi-objective optimisation is one of the goals that will be pursued in near future. Also, we are working to make the more significant parameters adaptive in HdSFA for more convenient use.

References

Andersen, H.C. (1991) *An Investigation into Genetic Algorithms, and the Relationship between Speciation and the Tracking of Optima in Dynamic Functions*, Honors, Queensland Univ., Brisbane, Australia.

Barlow, G.J. (2009) *Generalized Density-Estimate Memory for Dynamic Problems*, PhD dissertation, Carnegie Mellon University.

Bird, S. and Li, X. (2008) 'Using regression to improve local convergence', *IEEE Congress on Evolutionary Computation, 2007, CEC 200*7, pp.592–599.

Blackwell, T. and Branke, J. (2004) 'Multi-swarm optimization in dynamic environments', *Applications of Evolutionary Computing*, Vol. 3005, pp.489–500.

Blackwell, T. and Branke, J. (2006) 'Multiswarms, exclusion, and anti-convergence in dynamic environments', *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 4, pp.459–472.

Blackwell, T., Branke, J. and Li, X. (2008) 'Particle swarms for dynamic optimization problems', *Swarm Intelligence*, pp.193–217.

Branke, J. (1999a) 'Memory enhanced evolutionary algorithms for changing optimization problems', *Congress on Evolutionary Computation*.

Branke, J. (1999b) *The Moving Peaks Benchmark* [online] http://people.aifb.kit.edu/jbr/MovPeaks/ (accessed 08/11/2008).

Branke, J. and Schmeck, H. (2002) 'Designing evolutionary algorithms for dynamic optimization problems', *Advances in Evolutionary Computing: Theory and Applications*, pp.239–262.

Cho, H., Kim, D., Olivera, F. et al. (2011) 'Enhanced speciation in particle swarm optimization for multi-modal problems,' *European Journal of Operational Research*, Vol. 213, No. 1, pp.15–23.

Chow, C.K. and Yuen, S.Y. (2011) 'An evolutionary algorithm that makes decision based on the entire previous search history', *IEEE Transactions on Evolutionary Computation*, Vol. 15, No. 9, pp.741–769.

Du, W. and Li, B. (2008) 'Multi-strategy ensemble particle swarm optimization for dynamic optimization', *Information Sciences*, Vol. 178, No. 15, pp.3096–3109.

Farahani, S.M., Abshouri, A., Nasiri, B. et al. (2011) 'A Gaussian firefly algorithm', *International Journal of Machine Learning and Computing*, Vol. 1, pp.448–453.

Fister, I., Fister Jr., I., Yang, X-S. et al. (2013a) 'A comprehensive review of firefly algorithms', *Swarm and Evolutionary Computation*.

Fister, I., Yang, X-S., Brest, J. et al. (2013b) 'Modified firefly algorithm using quaternion representation', *Expert Systems with Applications*, Vol. 40, No. 18, pp.7220–7230.

Horng, M-H. and Liou, R-J. (2011) 'Multilevel minimum cross entropy threshold selection based on the firefly algorithm', *Expert Systems with Applications*, Vol. 38, No. 12, pp.14805–14811.

Kamosi, M., Hashemi, A. and Meybodi, M. (2010a) 'A new particle swarm optimization algorithm for dynamic environments', *Swarm, Evolutionary, and Memetic Computing*, Vol. 6466, pp.129–138.

Kamosi, M., Hashemi, A.B. and Meybodi, M.R. (2010b) 'A hibernating multi-swarm optimization algorithm for dynamic environments', *Proceedings of World Congress on Nature and Biologically Inspired Computing (NaBIC2010)*, Kitakyushu, Japan, pp.370–376.

Leung, S.W., Yuen, S.Y. and Chow, C.K. (2010) 'Parameter control by the entire search history: case study of history-driven evolutionary algorithm', *IEEE Congress on Evolutionary Computation, CEC 2010*, pp.1–8.

Leung, S.W., Yuen, S.Y. and Chow, C.K. (2012) 'Parameter control system of evolutionary algorithm that is aided by the entire search history', *Applied Soft Computing*, Vol. 12, No. 9, pp.3063–3078.

Li, C. and Yang, S. (2009) 'A clustering particle swarm optimizer for dynamic optimization', *IEEE Congress on Evolutionary Computation, CEC 2009*, pp.439–446.

Li, C. and Yang, S. (2011) 'A general framework of multi-population methods with clustering in undetectable dynamic environments', *IEEE Transactions on Evolutionary Computation*, Vol. 16, No. 4, pp.556–577.

Li, C., Yang, S. and Yang, M. (2014) 'An adaptive multi-swarm optimizer for dynamic optimization problems', *Evolutionary Computation*, pp.1–36.

Li, X., Branke, J. and Blackwell, T. (2006) 'Particle swarm with speciation and adaptation in a dynamic environment', *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pp.51–58.

Liu, L., Yang, S. and Wang, D. (2010) 'Particle swarm optimization with composite particles in dynamic environments', *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 40, No. 6, pp.1634–1648.

Mavrovouniotis, M. and Yang, S. (2011) 'Ant colony optimization algorithms with immigrants schemes for the dynamic travelling salesman problem', *Evolutionary Computation for Dynamic Optimization Problems*, Vol. 490, pp.317–341.

Mavrovouniotis, M. and Yang, S. (2013) 'Dynamic vehicle routing: a memetic ant colony optimization approach', *Automated Scheduling and Planning*, Vol. 505, pp.283–301.

Morrison, R.W. and De Jong, K.A. (1999) 'A test problem generator for non-stationary environments', *IEEE Congress on Evolutionary Computation, CEC 1999*, IEEE, p.2053.

Nasiri, B. and Meybodi, M. (2012) 'Speciation based firefly algorithm for optimization in dynamic environments', *International Journal of Artificial Intelligence*, Vol. 8, pp.118–132.

Nguyen, T.T. (2011) *Continuous Dynamic Optimisation using Evolutionary Algorithms*, PhD dissertation, University of Birmingham.

Nguyen, T.T., Yang, S. and Branke, J. (2012) 'Evolutionary dynamic optimization: a survey of the state of the art', *Swarm and Evolutionary Computation*, Vol. 6, pp.1–24.

Oppacher, F. and Wineberg, M. (1999) 'The shifting balance genetic algorithm: improving the GA in a dynamic environment', *The Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pp.504–510.

Rand, W., Riolo, R. and Holland, J.H. (2006) 'The effect of crossover on the behavior of the GA in dynamic environments: a case study using the shaky ladder hyperplane-defined functions', *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO 2006)*, pp.1289–1296.

Rezazadeh, I., Meybodi, M. and Naebi, A. (2011) 'Adaptive particle swarm optimization algorithm for dynamic environments', *Advances in Swarm Intelligence*, Vol. 6728, pp.120–129.

Richter, H. and Yang, S. (2008) 'Memory based on abstraction for dynamic fitness functions', *Applications of Evolutionary Computing,* Vol. 4974, pp.596–605, Springer-Verlag.

Senthilnath, J., Omkar, S.N. and Mani, V. (2011) 'Clustering using firefly algorithm: performance study', *Swarm and Evolutionary Computation*, Vol. 1, No. 3, pp.164–171.

Tilahun, S.L. and Ong, H.C. (2012) 'Modified firefly algorithm', *Journal of Applied Mathematics*, p.12.

Wang, X., Gao, X.Z. and Ovaska, S.J. (2007) 'An immune-based ant colony algorithm for static and dynamic optimization', *IEEE International Conference on Systems, Man and Cybernetics*, pp.1249–1255.

Weicker, K. (2002) 'Performance measures for dynamic environments', *Parallel Problem Solving from Nature – PPSN VII*, pp.64–73, Springer.

Yang, S. (2003) 'Non-stationary problem optimization using the primal-dual genetic algorithm', *The 2003 Congress on Evolutionary Computation, CEC 2003*, pp.2246–2253.

Yang, S. and Li, C. (2010) 'A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments', *IEEE Transactions on Evolutionary Computation*, p.1.

Yang, S. and Yao, X. (2008) 'Population-based incremental learning with associative memory for dynamic environments', *IEEE Transactions on Evolutionary Computation*, Vol. 12, No. 5, pp.542–561.

Yang, X-S. (2008) *Nature-Inspired Metaheuristic Algorithms*, Luniver Press.

Yang, X-S. (2014) *Cuckoo Search and Firefly Algorithm, Theory and Applications*, Springer Press.

Yaochu, J. and Branke, J. (2005) 'Evolutionary optimization in uncertain environments – a survey', *IEEE Transactions on Evolutionary Computation*, Vol. 9, No. 3, pp.303–317.

Yazdani, D., Akbarzadeh-Totonchi, M.R., Nasiri, B. et al. (2012) 'A new artificial fish swarm algorithm for dynamic optimization problems', *IEEE Congress on Evolutionary Computation, CEC 2012*, IEEE, pp.1–8.

Yazdani, D., Nasiri, B., Azizi, R. et al. (2013a) 'Optimization in dynamic environments utilizing a novel method based on particle swarm optimization', *International Journal of Artificial Intelligence*, Vol. 11, pp.170–192.

Yazdani, D., Nasiri, B., Sepas-Moghaddam, A. et al. (2013b) 'A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization', *Applied Soft Computing*, Vol. 13, No. 4, pp.2144–2158.

Yazdani, D., Nasiri, B., Sepas-Moghaddam, A. et al. (2014) 'mNAFSA: a novel approach for optimization in dynamic environments with global changes', *Swarm and Evolutionary Computation*, Vol. 18, pp.38–53.

Yazdani, D., Nasiri, B. and Meybodi, M.R. (2011) 'A new adaptive optimization method for dynamic environment based on particle swarm optimization', *Proceedings of the 16th Annual CSI Computer Conference (CSI 2011)*, University of Sharif, Tehran, Iran.

Yu, E.L. and Suganthan, P.N. (2009) 'Evolutionary programming with ensemble of explicit memories for dynamic optimization', *IEEE Congress on Evolutionary Computation, CEC 2009*, pp.431–438.

Yuen, S.Y. and Chow, C.K. (2007) 'A non-revisiting genetic algorithm', *IEEE Congress on Evolutionary Computation, CEC 2007 IEEE*, pp.4583–4590.

Yuen, S.Y. and Chow, C.K. (2009) 'A genetic algorithm that adaptively mutates and never revisits', *IEEE Transactions on Evolutionary Computation*, Vol. 13, No. 2, pp.454–472.