Two Phased Cellular PSO: A New Collaborative Cellular Algorithm for Optimization in Dynamic Environments

Ali Sharifi[1], Vahid Noroozi[1], Masoud Bashiri[1], Ali B. Hashemi[2], Mohammad Reza Meybodi[1]

[1] Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran

[2] Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada

alish@aut.ac.ir, vnoroozi@aut.ac.ir, ashkan_bashiri@aut.ac.ir, ali.b.hashemi@utoronto.ca, mmeybodi@aut.ac.ir

Abstract— **Many real world optimization problems are dynamic in which the fitness landscape is time dependent and the optima change over time such as dynamic economic modeling, dynamic resource scheduling, and dynamic vehicle routing. Such problems challenge traditional optimization methods as well as conventional evolutionary optimization algorithms. For such environments, optimization algorithms not only have to find the global optimum but also closely track its trajectory. In this paper, we propose a collaborative version of cellular PSO, named Two Phased cellular PSO to address dynamic optimization problems. The proposed algorithm introduces two search phases in order to create a more efficient balance between exploration and exploitation in cellular PSO. The conventional PSO in cellular PSO is replaced by a proposed PSO to increase the exploration capability and an exploitation phase is added to increase exploitation is the promising cells. Moreover, the cell capacity threshold which is a key parameter of cellular PSO is eliminated due to these modifications. To demonstrate the performance and robustness of the proposed algorithm, it is evaluated in various dynamic environment modeled by Moving Peaks Benchmark. The results show that for all the experimented dynamic environments, TP-CPSO outperforms all compared algorithms including cellular PSO.**

Keywords-Particle Swarm Optimization; Dynamic Environment; Cellular PSO.

I. INTRODUCTION

In dynamic environments, the fitness function changes over time and consequently the optimum points may change. Hence the optimization algorithm has to track the changes in the environment and find the new optima quickly. The dynamic optimization problems have challenged traditional optimization algorithms which were designed for non-stationary problems. These algorithms converge to the fixed global optimum, which results in losing diversity and decreasing the algorithm's ability to search for the new optimum after the environment changes.

Many nature-inspired algorithms have been proposed for optimization in dynamic environments. Particle Swarm Optimization (PSO) algorithm is considered as one of the major algorithms that is widely used for optimization in stationary environments, yet it still needs some modifications to better cope with dynamic environments.

The disadvantages of PSO in dynamic environments come from two major sources: the outdated memory or experiences of particles after the changes due to the dynamism of environment; and loss of diversity due to convergence. The outdated memory problem occur when the optima move(s) and/or optimum value changes. Particle memory which conveys the best place visited and its relevant fitness may not be valid after the change which leads to a misdirected search. The outdated memory problem can be solved either by clearing or by re-evaluating all the memories. The loss of diversity problem is rather serious. The time spent for swarm convergence and re-diversification, finding the moved optima and re-convergence lead to severe decrease in the performance of the algorithm.

Hashemi and Meybodi proposed cellular PSO, a hybrid model of particle swarm optimization and cellular automata and showed it produces promising results [1, 2]. In cellular PSO, a cellular automaton is embedded into the search space and partitions the search space such that each partition corresponds to a cell. At any time, in some cells of the cellular automaton, a group of particles are present and search for an optimum. In the search for an optimum, particles in a cell use their best personal experience and the best solution found in their neighborhood cells. Moreover, in order to prevent losing diversity of particles, a limit on the number of particles searching in each cell is imposed.

In this paper, a two phased version of cellular PSO (TP-CPSO) is proposed for optimization in dynamic environments. The proposed algorithm addresses the disadvantages of cellular PSO algorithm and introduces solutions to these issues. Two different phases of exploration and exploitation are defined for each cell which leads to eliminating the cell capacity threshold which is a key parameter of cellular PSO. A powerful local search (Naïve Direct Search) is also introduced in order to follow the moving peaks more efficiently.

The rest of this paper is organized as follows. The next section provides an overview of the state of the art PSO approaches proposed for dynamic optimization problems. Sections 3 and 4 provide brief introduction to Particle Swarm Optimization and Cellular Automata as the foundations of our approach followed by a detailed explanation of the proposed algorithm in section 5. Section 6 provides the experimental

results of the proposed algorithm. Finally, section 7 concludes the paper.

II. Previous Works

All the PSO based optimization algorithms for dynamic environments can be divided into five categories: 1) Randomization, 2) Mutual Repulsion, 3) Dynamic Information Networks, 4) Multi-Population and 5) Hybrid.

A. Randomization

One of the simplest approaches to address dynamic problems by PSO is reinitializing. Hu and Eberhart used re-diversification of the population after the change [3]. Their proposed method included randomly relocating all or part of the memories by calculating the fitness in one or more points of particles' memories after detecting the environment change. Since randomization leads to missing information, there is a chance that a large amount of information goes missing as if the algorithm runs from scratch with no previous memory. Also, lack of randomization may decrease diversity to an extent which is not enough to track the optimum.

B. Mutual Repulsion

Applying mutual repulsion between particles, swarms or extracted optima can maintain a desirable amount of diversity during the search process. For instance, Vesterstrom and Krink [4] studied particles with limited size to avoid premature convergence. Parsopoulos and Vrahatis [5] introduced a repulsive operator inside a found optimum point to repel the swarm from that point which allows the swarm to extract other unfound peaks. The atomic model [6-9] is another example of using repulsion for dynamic problems. This model defines a swarm of particles as two sub-swarms of charged and neutral particles. Atomic model can be pictured as a cloud of charged particles orbiting around a neutral nucleus. Charged particles increase the diversity around the converging neutral sub-swarm (nucleus).

C. Dynamic Information Networks

Another approach is applying modifications to the topology of information sharing between the particles of the swarm. This can lead to a temporal decrease in tendency to move towards the best found positions which maintains the diversity. Li and Dum [10] proposed a neighborhood model with a four-cell grid structure and Johnson and Middendorf [11] studied a hierarchical structure and reported improvements in compare to canonical PSO algorithm.

D. Multi-Population

One of the most successful approaches to tackle dynamic environments is multi-population methods. For multi-population algorithms it is usually desired to divide the particles into sub-swarms some designed to converge to the best positions found while others designed to look for new optimum points. This becomes more useful when global optimum alters between local optima. This method increases the odd that a sub-swarm would always be around the new global optimum after the change.

Lunge and Dumitrescu [12] use two collaborative populations with the same as a solution to avoid premature convergence and effectively track the optimum. One population is responsible for maintaining diversity using Crowding Differential Evolution algorithm while the other population tracks the optimum using PSO algorithm. The collaboration system is activated every time an environment change is detected or the best member of the second population is too close to the best solution found, that is the second population in reinitialized with the particles from the first population.

In [13] Du and Li suggested that particles should be divided into two parts, one part uses the canonical PSO equipped with a local Gaussian search and the other uses a differential mutation to do the search around the first swarm in order to expand the search regions of the algorithm and reaching the moving peak. These two parts improve the convergence and local optima avoidance of the algorithm respectively.

E. Hybrid

Blackwell and Branke proposed an algorithm that combines the advantages of repulsion and multi-population methods. Inspired by multi-population methods, the algorithm divides the population into multiple sub-swarms. Similar to the atomic model each sub-swarm includes charged particles, and applies the repulsion method by employing repulsion and anti-convergence operators. Charged particles maintain a suitable amount of diversity around found optima. Repulsion operator tunes the mutual interaction between the swarms. If the distance between two swarms falls below a predetermined threshold, positions of the weaker swarm's particles are reinitialized randomly. Repulsion operator on the other hand, helps the algorithm maintain diversity during the optimization process by avoiding over-convergence to one particular point. However, given the fact that the number of peaks may be larger than the number of swarms, it is necessary to put a number of particles in charge of finding new and better peaks. An anti-convergence operator is embedded in the algorithm to do this task. In [14] Blackwell and Branke introduced the basic version of the algorithm and later proposed the two self-adaptive variants of the algorithm [15]. The first variant starts with one swarm and more swarms will be added based on the needs with regard to a predetermined number of particles. The second variant starts with a random number of swarms and slowly converges to a number of swarms desirable to cover the peaks. This method uses a prefixed number of particles and dynamically assigns particles to swarms.

Hashemi and Meybodi [1] proposed an algorithm, named cellular PSO, that combines the features from multi-populations, information networks and randomization methods. cellular PSO is based on the idea of partitioning the search space and mapping a Cellular Automata (CA) onto this divided space in a way that each cell is assigned to one part of the partitioned space. A threshold on the maximum number of particles in each cell is applied which prevents from the crowding of many particles on a single optimum. Extra particles are moved to random cells to explore other points of the search space.

Noroozi et al. [16] also proposed another cellular based optimization algorithm named, CellularDE, which used the

similar idea of space partitioning but instead of PSO it utilized Differential Evolution (DE) with a new proposed DE scheme suitable for dynamic optimization. They reported that CellularDE outperforms compared algorithms including cellular PSO in most tested environments.

III. Particle Swarm Optimization

Particle Swarm Optimization (PSO) algorithm was first introduced by Eberhart and Kennedy [17] which is based on the social behavior of self-organizing swarms. In PSO, a potential solution for a problem is considered as a bird without quality and volume, which is called a particle. This so called bird flies through a *d*-dimensional space, adjusts its position in search space according to its own experience and its neighbors. A set of particles is called swarm.

Each particle in the *d*-dimensional space is represented as $p_i = (p_{i1}, p_{i2}, \ldots, p_{id})$, where p_i is the *i*th particle. The velocity of the *i*th particle is represented as $v_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$ that is in the range of $(0, V_{max})$, where V_{max} is a parameter set by the user. At each time step, particles calculate their new velocities from (1) and update their position using (2).

$$v_i(t+1) = wv_k(t) + c_1 r_1(t)\left(pBest_i - p_i(t)\right) + c_2 r_2(t)\left(lBest_i - p_i(t)\right) \quad (1)$$

$$p_i(t+1) = p_i(t) + v_i(t+1) \quad (2)$$

In above equations, $w < 1$ is the inertial weight, c_1 and c_2 are positive constants known as acceleration constants that determine the amount of contribution that social and cognitional factors make to the particle's navigation. r_1 and r_2 are *d*-dimensional vectors filled with values generated from a random uniform distribution over [0,1]. $pBest_i$ is the best so far place that the *i*th particle visited and $lBest_i$ is the best visited place by particles that are considered as the *i*th particle's neighbors. The term $(pBest_i - p_i(t))$ is the cognitive component of the *i*th particle's velocity, and the term $(lBest_i - p_i(t))$ conveys the social component of the particle's velocity. In other words, the particle tends to move based on two attractors: its own best memory and the best memory of its neighbors. The coefficient *w* is the inertial weight that indicates the importance of the particle's past velocity to the particle.

IV. Cellular Automata

Cellular Automata (CA) are mathematical models for systems consisting of large number of simple identical components with local interactions in which space and time are discrete. It is called cellular because it is made up of cells like points of a lattice or squares of checker boards, and it is called automata because each cell follows a simple rule [18]. Informally, a *d*-dimensional CA consists of a *d*-dimensional lattice of identical cells. The simple components act together to produce complicated patterns of behavior. Each cell can assume a state from a finite set of states. The cells update their states synchronously on discrete steps according to a local rule. The new state of each cell depends on the previous states of a set of cells, including the cell itself, and constitutes its neighborhood. Moore and Von Newman are the two common neighborhood styles which are illustrated in Fig. 1 for a 2-dimensional space with the neighborhood size of one.

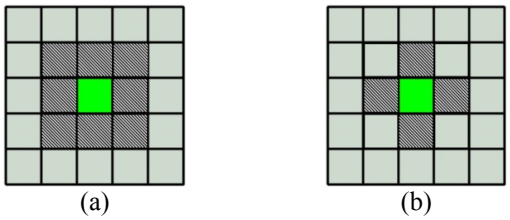

Figure 1. Neighborhood in a 2-D cellular automaton: (a) Moore, (b) von Neumann Neighborhood.

V. Proposed Algorithm

The proposed algorithm is based on the idea of partitioning the search space and then performing the search in each partition locally. To do so, a cellular automaton is embedded into the search space and partitions it such that each cell is correspondent with a partition and is responsible to control the search process in it. Each cell performs the search process within its boundary using a proposed version of PSO, named Mutative PSO (MPSO). This structure implicitly distributes particles into sub-swarms wherein particles are responsible for finding the most promising optimum in the neighborhood of their corresponding cells. All the procedures of the proposed algorithm such as controlling the search process, interaction between neighboring cells, and controlling diversity of the swarm are performed by each cell as the rules of the cellular automata.

A. Initialization

For a *d*-dimensional search space each dimension is partitioned equally into N_p segments, then each cell in a d-dimensional CA belongs to cells collection $C=\{cell_i | 1 \le i \le (N_p)^d\}$. Fig. 2 illustrates a 2-dimensional search space where each dimension is partitioned into four equal segments and a CA of the same size is embedded into the partitioned space. Moreover, in the proposed algorithm, neighborhood of range *r* of a cell is defined by Moore neighborhood, which for a cell located at $(z_1, z_2, \ldots, z_d)$ in the cellular automata are cells defined by (3).

$$N^{r}_{cell(z_1, z_2, \ldots, z_d)} = \left\{ cell_{(m_1, m_2, \ldots, m_d)} : |z_i - m_i| \le r \right\} \quad (3)$$

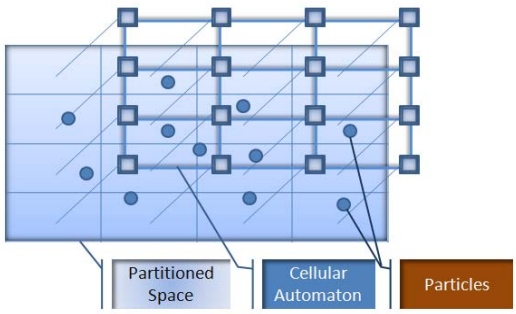

Figure 2. Embedding a CA in a two-dimensional space which is partitioned into 4^2 equal partirions.

Initially, a swarm of *M* particles, $P = \{p_1, p_2, \ldots, p_M\}$ are randomly distributed in the search space. Particle p_i which is

located in (x_1, x_2,…, x_d) is assigned to $cell_{z_1,z_2,\cdots,z_d}$ where z_i is calculated by (4).

$$z_i = \lfloor x_{ki} / N_p \rfloor + 1 \tag{4}$$

B. Cell Memory

To reduce unnecessary searches in the regions which have been already searched as well as to use the results of the previous search efforts, a memory, Mem_i, is defined for each $cell_i$ as in (5). This memory contains the best position found in the cell since the last change in the environment. These cell memories are used to calculate the best visited place in each cell and its neighborhood from the last change until the current iteration. Equation (6) shows this calculation where f is the fitness function of the optimization problem. LB_i, is the best solution found in the vicinity of cell i.

$$Mem_i(t) \leftarrow \underset{\forall k,\ \text{particle } p_k \text{ is in } cell_i}{argmax} \{f(p_k), f(Mem_i(t-1))\} \tag{5}$$

$$LB_i \leftarrow \underset{\forall j, cell_j \text{ is a neighbor of } cell_i}{argmax} \{f(Mem_i(t-1)), f(Mem_j(t-1))\} \tag{6}$$

C. Evolution by Mutative PSO

In cellular PSO, the conventional PSO is used which causes stagnation or premature convergence for swarms with few particles. This is due to the fact that in small-sized swarms, most particles may become too close to their attractors before convergence which drastically slows down the progress of the swarm toward the local optima. In order to tackle this problem, a modified PSO algorithm is proposed that uses a new velocity updating equation. A mutation term is introduced that adds a random vector to the velocity vector. The proposed velocity update equation for the particle p_i, which is located in $cell_j$, is shown in (7) where the mutation term (N_i) is calculated by (8) and (9).

$$v_i(t+1) = wv_k(t) + c_1 r_1 (pBest_i - p_i(t)) + c_2 r_2 (LB_j - p_i(t)) + \sigma_i N(0,1) \tag{7}$$

$$\sigma_i = \min\left(\frac{2e^{-ds_i}}{1+e^{-ds_i}}, \sigma_{\max}\right) \tag{8}$$

$$ds_i = \|p_i - pBest_i\| + \|p_i - LB_j\| \tag{9}$$

Note that for all particles in $cell_j$, LB_j is used as their *lBest* in (1). Although the proposed update equation eliminates the stagnation problem and premature convergence, the mutation term slows down convergence of particles which is unnecessary. In addition, when the particles in a cell are converged to the local optima, any further searching would have a little influence on the performance and only force many futile fitness evaluations to the algorithm. The proposed PSO algorithm is able to explore the search space though lacks necessary exploitation abilities, so it's necessary to use another search method after the convergence in order to perform exploitation appropriately.

D. Exploration and Exploitation Phases

In contrast to cellular PSO where all the search process is always conducted by PSO, the proposed method defines a two-phase search for each cell: one exploration phase and one exploitation phase. Each cell has an operational status variable that determines which phase the cell is functioning in. At the beginning, all cells are in exploration phase in which the search is performed by the proposed PSO algorithm until a convergence to local optima is detected. A convergence in a cell is detected when for at least k consecutive fitness evaluations it contains at least one particle and the best position found in the cell is the best position in the neighborhood of the cell. After a convergence is detected in a cell, the cell enters the exploitation phase. In the exploitation phase, all particles in a cell, except the best performing particle, are reinitialized to random cells in the cellular automata. Then, instead of PSO, the search process is continued by a proposed local search with a high capability of exploitation called Naive Directed Search (NDS). The NDS algorithm is based on directed movements in all dimensions with a defined step size. The step size is reduced if no improvement is achieved in any dimension. This local search algorithm works as follows.

For cell i which is in exploitation state, a step size ss_i is defined that specifies the step length of local search. Initially, for all cells ss_i is set to a specified value ss_{init}. In addition, a search direction vector $sd_i=(sd_i^1, sd_i^2, \ldots, sd_i^d)$ is used to determines the current direction of the local search in each dimension, where $sd_i^1 \in \{-1,1\}$. Initially, the search direction in each dimension is selected randomly. At each iteration, the $LB_i(t)$, which point to the best position found in the cell i, is considered as the base point for the local search. The local search is performed by evaluating a point in the search space which is located at the $LB_i(t)+sd_i \times ss_i^T$. If the evaluated point results in a better fitness value compared to the fitness value of $LB_i(t)$, the search is a success and the base point, i.e. $LB_i(t)$, is updated; otherwise the direction in that dimension is flipped to the opposite direction. The obtained movements are preserved to be used in the next iteration. In case of a movement failure, if there has been at least one success in the current direction for the current step size, movement stops for its relative particle, otherwise it tries the reverse direction for the next movement. When the search is stopped in all dimensions, the search process starts again in all dimensions and the search counter sn_i is increased by one. At each iteration, the step size of movement (ss_i) is calculated according to (10)

$$ss_i = ss_{init}.b^{sn_i} \tag{10}$$

where b is a fixed discount factor and determines the speed of reduction. It should be noted that all the failure counters are reset to their initial values after any change detection in the environment.

Since the proposed local search method is able to perform exploitation using only one particle, cell capacity is set to one during the exploitation state. As long as a cell is in this state, if any particle enters the cell in the next iterations, all particles except than the best one are reinitialized to random cells. As a result, a number of particles that were previously converged to these cells will be free and ready to explore the search space.

A cell would remain in the state of exploitation until the base point of local search exceeds the cell bounds. Afterwards the cell would go back to the exploration state.

In cellular PSO, a threshold on the maximum number of allowed particles in each cell is defined which is sensitive to the characteristics of the environment. But in the proposed algorithm, capacity of the cell is either unlimited when a cell is in the exploration phase or equals to 1 when the cell is in exploitation phase. Hence, unlike cellular PSO, there is no need to adjust capacity threshold.

E. Movement of Particles in CA

Due to the limit of access to other cells, which is originated from the locality of CA rules, it is not possible to move the particles freely between cells. A propagation mechanism is used to move particles between different cells. Particles which are marked to move are made inactive and will not participate in the search process. A counter, Hop_j, is defined for each inactive particle, p_j, which is initialized to the distance of the source, $cell_{s_1,s_2,\cdots,s_d}$, and destination cell, $cell_{q_1,q_2,\cdots,q_d}$, as in (11). At each iteration, all cells copy the inactive particles with non-zero counters which are in their neighborhood cells. Their counters are decreased by one and the ones with zero counters are checked to have the same destination as the current cell. Inactive particles which reach their destination become active again in order to participate in the search process.

$$Hop_j = \max_{k=1}^{d}\{|q_k - s_k|\} / S_N \tag{11}$$

F. Dealing with Change

Each time a change is detected in the environment, all particle memories are re-evaluated and cell memories are cleared. Also, the failure counters for the local search are reset to their initial values. As a result of utilizing the combination of this local search and the proposed PSO, in contrast with cellular PSO, there is no need to perform any other additional local search for tracking moving optima after any change detection in the environment.

VI. EXPERIMENTS

A. Moving Peaks Benchmark

The performance of the proposed algorithm is evaluated in various dynamic environments modeled by Moving Peaks Benchmark (MPB) [19]. MPB has been widely used in the literature to study the performance of optimization algorithms in dynamic environments. MPB generates functions in multidimensional landscapes consisting of several peaks where the height, the width, and the position of the peaks are altered every time a change in the environment occurs. The MPB function with m peaks in an n-dimensional environment.

$$F(\vec{x},t) = \max(B(\vec{x}), \max_{i=1\ldots m} P(\vec{x}, h_i(t), w_i(t), \vec{p}_i(t))) \tag{12}$$

where $B(\vec{x})$ is a time-invariant basis landscape, P is a function defining a peak shape, $h_i(t)$, $w_i(t)$, and $p_i(t)$ are the height, width, and position of the respective peaks respectively and are defined as:

$$\sigma \in N(0,1)$$

$$h_i(t) = h_i(t-1) + height\_severity \cdot \sigma \tag{13}$$

$$w_i(t) = w_i(t-1) + width\_serverity \cdot \sigma \tag{14}$$

$$\vec{p}_i(t) = \vec{p}_i(t-1) + \vec{v}_i(t) \tag{15}$$

In above equations, h_s and w_s are the maximum amount with which the width and height of peaks can change and $\vec{v}_i(t)$ is the amount of peak movement defined in (15).

$$\vec{v}_i(t) = \frac{s}{\left|\vec{r} + \vec{v}_i(t-1)\right|}((1-\lambda)\vec{r} + \lambda\vec{v}_i(t-1)) \tag{16}$$

where $\vec{r}$ is a random vector, s controls the severity of the shift, and is the degree of correlation to previous shifts.

TABLE I. THE DEFAULT PARAMETERS OF MPB FOR SCENARIO II

Parameter	Value
Number of dimensions (d)	5
Number of peaks (m)	10
Change frequency (f)	5000
Height severity (h_s)	0.7
Width severity (w_s)	0.1
Peak shape	Cone
Shift length (s)	1.0
A	[0, 100]
H	[30, 70]
W	[1, 12]
I	50

Unless otherwise mentioned, the value for the parameters of MPB are set to a well-known scenario known as Scenario II (Table I0). Shift length s is the radius of peak movement after environment changes. m is the number of peaks. f is the frequency of the changes in environment as number of fitness evaluations. H and W denote range of the height and width of peaks which change by height severity (h_s) and width severity (w_s), respectively. I is the initial height of the peaks. Parameter A denotes the range of the search space for all dimensions.

B. Evaluation Criterion

Since there are no specific constant optima for dynamic optimization problems, the goal is not just to find the optima, but to track the trajectories of optimum points in the search space. A common method for illustrating an algorithm's prominence over others is to demonstrate convergence diagrams for best or average fitness and visually comparing them. The most common measurement criterion in dynamic optimization problems is offline-error which can be calculated by (17).

$$offline-error = \frac{1}{T}\sum_{t=1}^{T} e_t' \tag{17}$$

where T is the total number of fitness evaluations, $e'_t = min\{e_\tau, e_{\tau+1}, \ldots, e_t\}$ and τ represents the last time step at which the last change in the environment occurred.

C. Experimental Settings

For all experiments the default values for parameters of Two Phased cellular PSO are experimentally set according to Table II. The search space is partitioned by embedding a cellular automata with 12^5 cells. Moreover, in the cellular automata, the Moore neighborhood with radius of two cells is used, which is large enough to handle the exploration of the sub-populations in a neighborhood and is small enough to prevent the convergence of the population to a local optimum. Initial step size (ss_{init}), discount factor (b) and NDS are empirically set to 0.5 and 0.4, respectively. Not that for satisfaction fairness the values of the parameters of TP-CPSO and other compared algorithms are selected so that algorithms perform best for the scenario II of MPB.

TABLE II. DEFAULT VALUES FOR PARAMETERS OF TP-CPSO

Parameter	Value
Swarm size	30
Inertial weight (w)	0.7298
Acceleration constants (c_1=c_2)	1.4961
Length of the cellular automata in each dimension (N_p)	12
Neighborhood radius	2
Neighborhood type	Moore
Maximum variance of mutation ($_{max}$)	0.2
Discount factor (b)	0.4
Initial step size of NDS (ss_{init})	0.5

D. Experiment: Comparison with other algorithms

The results of the proposed algorithm are compared with cellular PSO [18, 19], CellularDE [16], HmSO [20], and adaptive mQSO [15]. To the best of our knowledge, adaptive mQSO and HmSO are two of the best-performing PSO-based optimization algorithms introduced for dynamic environments. cellular PSO [18, 19] and CellularDE share the idea of embedding cellular automata in the search space. Moreover, to show the effectiveness of the proposed Mutative PSO (section V-C), the results of a version of cellular PSO (Cellular MuPSO) in which canonical PSO is replaced by Mutative PSO is provided.

For all experiments, parameters of cellular PSO, CellularDE, HmSO, and adaptive mQSO are set to the values reported in [2], [16], [20], and [15], respectively. All experiments were performed for 100 changes in environment. Note that in MPB the changes occurs every f fitness evaluation. Hence, regardless of the size of swarm, all evaluated algorithms perform the same number of fitness evaluations. The average offline errors of the algorithms in 100 runs with 95% confidence interval for various dynamic environments are depicted in Table III to Table VI. For each environment, t-tests with significance level of 0.05 have been applied and the result of the best performing algorithm(s) is printed in bold. When the offline errors of the best performing algorithms are not significantly different, all are printed in bold.

The results of the experiments show that TP-CPSO outperforms all other compared algorithms, for most of the tested dynamic environments. Only in environments where changes occurs frequently (f=500) and there are more than 40 peaks, CellularDE outperforms TP-CPSO. This is because in such environments exploration is more required than exploitation. Hence, not only the exploitation capability of TP-CPSO is not any help, but also the exploration capability of DE helps CellularDE to outperform TP-CPSO.

Moreover, the results illustrate that the Cellular MuPSO performs better than cellular PSO in all tested dynamic environments. Note that in the Cellular MuPSO the swarm size and the capacity of cells are set to 40 and 4, respectively, while these parameters in cellular PSO are set to 100 and 10, respectively.

In addition, as depicted in Fig. 3 for an environment with 50 peaks, TP-CPSO not only results in less offline error than cellular PSO and CellularDE but also it can find better solutions than them much faster. This is due to the fact that TP-CPSO takes advantage of two separate search phases which impose an efficient balance between exploration and exploitation in the search process.

VII. CONCLUSIONS

In this paper, we proposed a two phased version of cellular PSO algorithm to tackle dynamic optimization problems. Like cellular PSO, in TP-CPSO, the search space is partitioned by embedding cellular automata into it. Then, each cell performs a two-phased search using particles residing in it, i.e. an exploration phase followed by an exploitation phase. TP-CPSO replaces conventional PSO used in cellular PSO by a proposed PSO to increase the exploration capability of the algorithm. Moreover, to increase the exploitation capability in the promising cells, a naïve directed search has been used in the exploitation phase.

The proposed PSO algorithm helps TP-CPSO to be able to perform better than cellular PSO when there are few particles in the swarm. In addition, a sensitive parameter of cellular PSO, i.e. cell capacity, has been eliminated in TP-CPSO.

To evaluate the performance of the proposed algorithm, extensive experiments in various dynamic environments modeled by moving peaks benchmark were conducted. The results show significant prominence of the proposed algorithm over the best PSO-based algorithms known in the literature.

TABLE III. OFFLINE ERROR FOR DIFFERENT NUMBER OF PEAKS (f = 500)

m	TP-CPSO	Cellular MuPSO	Cellular PSO	CellularDE	HmSO	Adaptive mQSO
1	5.96±0.22	8.42±0.49	11.62±0.77	8.20±0.19	8.53±0.49	**5.08±0.27**
5	**5.12±0.11**	6.36±0.24	8.59±0.36	6.06±0.05	7.40±0.31	**5.14±0.09**
10	**5.56±0.11**	6.39±0.22	8.78±0.28	5.93±0.04	7.56±0.27	6.20±0.11
20	**5.67±0.10**	6.62±0.23	8.67±0.25	**5.60±0.03**	7.81±0.20	6.94±0.18
30	**5.59±0.08**	6.58±0.17	8.24±0.22	**5.56±0.03**	8.33±0.18	7.23±0.16
40	**5.53±0.07**	6.79±0.18	8.50±0.20	**5.48±0.02**	8.45±0.18	7.43±0.17
50	5.57±0.08	6.60±0.17	8.37±0.20	**5.47±0.02**	8.83±0.17	7.49±0.09
100	5.36±0.06	6.68±0.14	7.91±0.18	**5.29±0.02**	8.85±0.16	7.29±0.15
200	5.33±0.05	6.52±0.12	7.71±0.14	**5.07±0.02**	8.85±0.16	6.82±0.14

TABLE IV. OFFLINE ERROR FOR DIFFERENT NUMBER OF PEAKS (f = 1000)

m	TP-CPSO	Cellular MuPSO	Cellular PSO	CellularDE	HmSO	Adaptive mQSO
1	**2.66±0.08**	5.10±0.27	5.86±0.42	4.98±0.35	4.46±0.26	**2.68±0.14**
5	**2.67±0.09**	4.23±0.21	5.26±0.26	3.96±0.04	4.27±0.08	3.22±0.07
10	**3.17±0.06**	4.34±0.17	5.75±0.23	3.98±0.03	4.61±0.07	4.11±0.08
20	**3.33±0.06**	4.76±0.14	5.74±0.19	4.53±0.02	4.66±0.12	4.75±0.14
30	**3.45±0.06**	4.81±0.13	5.84±0.16	4.77±0.02	4.83±0.09	4.98±0.10
40	**3.58±0.05**	5.02±0.15	5.84±0.17	4.87±0.02	4.82±0.09	5.10±0.11
50	**3.57±0.05**	4.92±0.11	5.84±0.14	4.87±0.02	4.96±0.03	5.12±0.05
100	**3.52±0.04**	4.92±0.11	5.73±0.11	4.85±0.02	5.14±0.08	5.03±0.09
200	**3.51±0.04**	4.87±0.11	5.48±0.11	4.46±0.01	5.25±0.08	4.65±0.09

TABLE V. OFFLINE ERROR FOR DIFFERENT NUMBER OF PEAKS (f = 2500)

m	TP-CPSO	Cellular MuPSO	Cellular PSO	CellularDE	HmSO	Adaptive mQSO
1	**0.92±0.03**	2.32±0.13	3.78±0.25	2.38±0.78	1.75±0.10	**1.09±0.06**
5	**1.17±0.05**	2.42±0.16	2.91±0.14	2.12±0.02	1.92±0.11	1.58±0.13
10	**1.59±0.06**	2.41±0.12	3.18±0.16	2.42±0.02	2.39±0.16	2.33±0.11
20	**1.82±0.04**	3.13±0.13	3.65±0.13	3.05±0.04	2.46±0.09	2.84±0.09
30	**1.99±0.04**	3.44±0.11	3.90±0.11	3.29±0.03	2.57±0.05	3.13±0.09
40	**1.96±0.03**	3.56±0.11	4.20±0.13	3.43±0.03	2.56±0.06	3.23±0.08
50	**2.01±0.03**	3.59±0.10	4.08±0.11	3.44±0.02	2.65±0.05	3.24±0.07
100	**2.09±0.03**	3.75±0.11	4.23±0.09	3.36±0.01	2.72±0.04	3.20±0.06
200	**2.06±0.02**	3.66±0.11	4.09±0.10	3.13±0.01	2.81±0.04	3.00±0.05

TABLE VI. OFFLINE ERROR FOR DIFFERENT NUMBER OF PEAKS (f = 5000)

m	TP-CPSO	Cellular MuPSO	Cellular PSO	CellularDE	HmSO	Adaptive mQSO
1	**0.40±0.01**	1.27±0.07	2.79±0.18	1.53±0.07	0.87±0.05	0.55±0.02
5	**0.75±0.07**	1.36±0.13	2.03±0.17	1.50±0.04	1.18±0.04	1.00±0.04
10	**0.96±0.05**	1.67±0.10	2.06±0.12	1.64±0.03	1.42±0.04	1.43±0.04
20	**1.18±0.03**	2.36±0.12	2.99±0.13	2.46±0.05	1.50±0.06	1.95±0.05
30	**1.32±0.03**	2.72±0.10	3.21±0.11	2.62±0.05	1.65±0.04	2.15±0.05
40	**1.36±0.03**	2.86±0.10	3.35±0.11	2.76±0.05	1.65±0.05	2.28±0.04
50	**1.40±0.03**	3.15±0.11	3.37±0.12	2.75±0.05	1.66±0.02	2.28±0.02
100	**1.50±0.02**	3.21±0.10	3.35±0.10	2.73±0.03	1.68±0.03	2.31±0.03
200	**1.56±0.02**	3.08±0.09	3.32±0.09	2.61±0.02	1.71±0.02	2.11±0.03

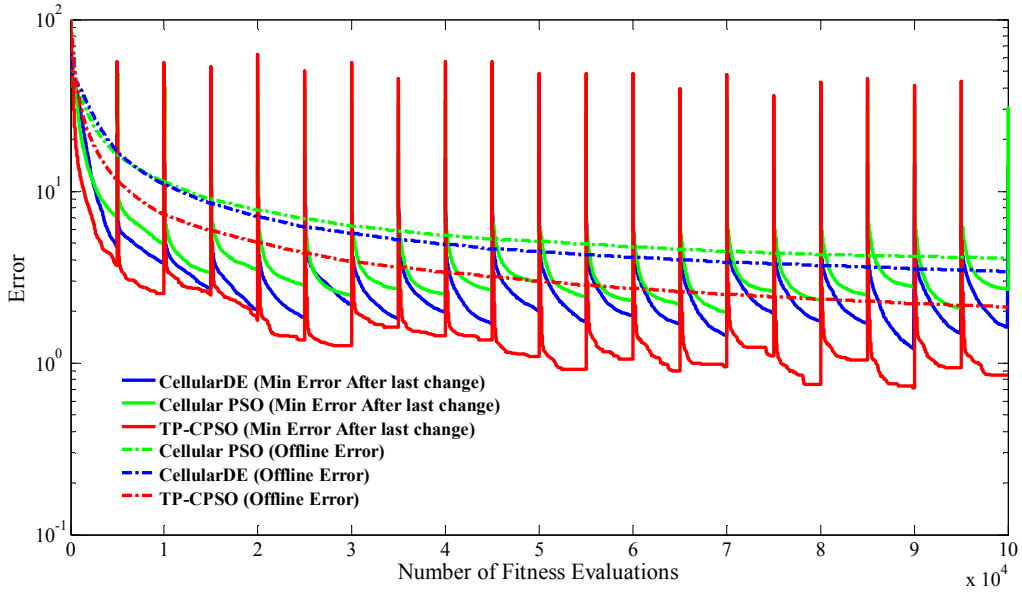

Figure 3. Offline error and current error for MPB in scenario II with 50 peaks.

REFERENCES

[1] A. B. Hashemi and M. R. Meybodi, "Cellular PSO: A PSO for Dynamic Environments," *Advances in Computation and Intelligence*, pp. 422-433, 2009.

[2] A. B. Hashemi and M. R. Meybodi, "A Multi-Role Cellular PSO for Dynamic Environments," in *14th International CSI Computer Conference*, Tehran, Iran, 2009, pp. 412-417.

[3] X. Hu and R. C. Eberhart, "Adaptive Particle Swarm Optimisation: Detection and Response to Dynamic Systems," in *Congress on Evolutionary Computation*, 2002, pp. 1666-1670.

[4] J. Vesterstrom, T. Krink, and J. Riget, "Particle Swarm Optimisation with Spatial Particle Extension," in *Congress on Evolutionary Computation*, 2002, pp. 1474-1479.

[5] K. E. Parsopoulos and M. N. Vrahatis, "Particle Swarm Optimizer in Noisy and Continuously Changing Environments," in *IASTED International Conference on Artificial Intelligence and Soft Computing*, Cancun, Mexico, 2001, pp. 289-294.

[6] T. M. Blackwell and P.Bentley, "Don't Push Me! Collesion Avoiding Swarms," in *Congress on Evolutionary Computation*, 2002, pp. 1691-1696.

[7] T. M. Blackwell and J. Branke, "Dynamic Search with Charged Swarms," in *Genetic and Evolutionary Computation Conference*, 2002, pp. 19-26.

[8] T. M. Blackwell and J. Brank, "Multi-Swarm Optimization in Dynamic Environments," in *Application of Evolutionary Computing*, 2004, pp. 489-500.

[9] T. M. Blackwell, "Swarms in Dynamic Environments," in *Genetic and Evolotionary Computation Conference*, 2003, pp. 1-12.

[10] X. Li and K. H. Dam, "Comparing Particle Swarm for Tracking Exterma in Dynamic Environments," in *Congress on Evolutionary Computation*, 2003, pp. 1772-1779.

[11] S. Janson and M. Middendorf, "A Hierachical Particle Swarm Optimizer for Dynamic Optimization Problems," in *Applications of evolutionary computating*, 2004, pp. 513-524.

[12] R. I. Lung and D. Dumitrescu, "A Collaborative Model for Tracking Optima in Dynamic Environments," in *IEEE Congress on Evolutionary Computation*, 2007, pp. 564-567.

[13] W. Du and B. Li, "Multi-Strategy Ensemble Particle Swarm Optimization for Dynamic Optimization," *Information Sciences: an International Journal*, vol. 178, no. 15, pp. 3096-3109, 2008.

[14] T. M. Blackwell and J. Branke, "Multi-Swarm, Exlusion and Anti-Convergence in Dynamic Environments," *IEEE transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 459-472, 2004.

[15] T. Blackwell, "Particle Swarm Optimization in Dynamic Environments," in *Evolutionary Computation in Dynamic and Uncertain Environments*, vol. 51, Springer Berlin, 2007, pp. 29-49.

[16] V. Noroozi, A. B. Hashemi, and M. R. Meybodi, "CellularDE: A Cellular Based Deifferential Evolution for Dynamic Optimization Problems," in *Adaptive and Natural Computing Algorithms, Lecture Notes in Computer Science*, 2011, pp. 340-349.

[17] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," in *Proceeding of IEEE International Conference of Neural Networks*, 1995, pp. 1942-1948.

[18] E. Fredkin, "Digital Mechanics: An Information Process Based on Reversible Universal Cellular Automata," *Physica*, vol. D45, pp. 254-270, 1990.

[19] J. Branke, "Memory Enhanced Evolutionary Algorethms for Changing Optimization Problems," in *Congress on Evolutionary Computation CEC99*, 1999, pp. 1875-1882.

[20] M. Kamosi, A. B. Hashemi, and M. Meybodi, "A Hibernating Multi-Swarm Optimization Algorithm for Dynamic Environments," in *the Second World Congress on Nature and Biologically Inspired Computing*, 2010, pp. 363-369.