Control and Cybernetics

vol. **39** (2010) No. 2

Multi-swarm that learns*

by

Krzysztof Trojanowski

Institute of Computer Science
Polish Academy of Sciences, Warsaw, Poland

Abstract: This paper studies particle swarm optimization approach enriched by two versions of an extension aimed at gathering information during the optimization process. Application of these extensions, called memory mechanisms, increases computational cost, but it is spent to a benefit by incorporating the knowledge about the problem into the algorithm and this way improving its search abilities. The first mechanism is based on the idea of storing explicit solutions while the second one applies one-pass clustering algorithm to build clusters containing search experiences. The main disadvantage of the former mechanism is lack of good rules for identification of outdated solutions among the remembered ones and as a consequence unlimited growth of the memory structures as the optimization process goes. The latter mechanism uses other form of knowledge representation and thus allows us to control the amount of allocated resources more efficiently than the former one. Both mechanisms have been experimentally verified and their advantages and disadvantages in application for different types of optimized environments are discussed.

Keywords: particle swarm optimization, multi-swarm, dynamic optimization, memory, clusters, clustering evolving data streams, quantum particles.

1. Introduction

Effective optimization in dynamic environments with heuristic algorithms always requires modifications and extensions in the base versions of the algorithms to cope with dynamics present in the environment. There is a large group of modifications, which introduce diverse forms of learning from the experiences gained during optimization. One of these forms is a memory mechanism. Information about the past can be stored in the memory structures in implicit or explicit form. The mechanism with implicit memory can be represented as some extra strategies, based on statistical factors updated according to the

*Submitted: February 2009; Accepted: September 2009.

current progress in the optimization process. The factors can define, for example, directions of further search in the search space. The mechanism with explicit memory stores complete solutions and makes use of them according to rules defined for remembering, recalling and forgetting. The latter form of memory has already been tested in evolutionary algorithms (Mori et al., 1997; Trojanowski, Michalewicz and Xiao, 1997; Branke, 1999; Trojanowski and Michalewicz, 1999) or immune based optimization algorithms (Trojanowski and Wierzchoń, 2002a,b,c). These approaches proved their efficiency especially in the case of cyclic or semi-cyclic changes in the optimized problem. In case of random changes, however, their influence was negligible. Besides, they introduce additional computation cost because for every of these mechanisms the step of recalling procedure always includes reevaluation of the entire memory content. This extra computational cost is hard to ignore, especially with regard to the assumption that the number of evaluation function calls between subsequent changes should be constant. Execution of the recalling procedure consumes a significant share of the given limit of calls, and so the more solutions to reevaluate in the memory, the smaller the number of evaluation function calls between subsequent changes. Although this assumption limits the search capabilities of the algorithm, it is usually respected because it allows for a fair comparison of efficiency of different algorithms.

In this paper we present a novel approach, avoiding additional evaluation function calls in the recalling procedure. The proposed rules of memory management calculate just the distance between solutions and no function evaluation is needed. So the limit of evaluation function calls between subsequent changes remains untouched. There are two types of memory mechanisms studied in this paper: with explicit and implicit form of information stored in the memory structures. The first type of the mechanisms uses a buffer with selected complete solutions accumulated during the ongoing optimization. Mechanism of the second type creates, first, a number of clusters and then continuously adapts them by absorption of solutions selected during the optimization. Both mechanism are discussed and experimentally verified in this paper.

In the presented research memory mechanisms are applied to particle swarm optimization (PSO). Our PSO algorithm is an extension of the constriction model (Clerc and Kennedy, 2002) with the following modifications useful for coping with dynamic optimization tasks:

1. particles are split into sub-swarms (Blackwell and Branke, 2004),
2. two types of particles are applied in the sub-swarms: classic (Clerc and Kennedy, 2002) and quantum ones (Blackwell and Branke, 2006),
3. exclusion mechanism (Blackwell and Branke, 2006) protects sub-swarms against following the same optimum.

There are two novelties proposed and studied in this paper: application of distance between solutions instead of the evaluation function value in memory management rules (a), and application of clusters as a representation of infor-

mation gathered during the optimization process (b). The paper is organized as follows. A brief description of PSO algorithm and particularly of the selected version of PSO is given in Section 2. Section 3 describes the two proposed memory mechanisms: explicit memory structures (Section 3.1) and memory structures with clusters (Section 3.2). In Section 4, we present the performance measure and the benchmark. Section 5 includes the results obtained with the first type of the memory mechanism, while Section 6 — the results obtained with the second type of the memory mechanism. Section 7 concludes.

2. Quantum multi-swarm

Particle swarm optimization is one of approaches in the field of metaheuristics. The idea proposed by Kennedy and Eberhart (1995) originates from observations of behavior of school of fish or flock of birds searching for food. In PSO, a swarm of particles just like a flock moves throughout the domain of possible solutions in search for optimum and changes location and velocity of its members by means of linear kinematic laws.

2.1. Basic scheme

The basic scheme of PSO is given below. The algorithm works with a set of particles $\mathbf{x}_i$ where $i \in 1, \ldots N$ represent solutions in an l-dimensional real valued search space D. The quality of particles is evaluated with a function $f(\cdot)$ defined on D. The best solution found by $\mathbf{x}_i$ is denoted by $\mathbf{y}_i$ (the personal best solution called the particle attractor), whereas the best solution found by the swarm is denoted by $\mathbf{y}^*$ (the neighborhood best solution called the swarm attractor).

Algorithm 1 the particle swarm optimization algorithm

```
1: Create and initialize the swarm
2: repeat
3:    for i = 1 to N do
4:       if f(x_i) > f(y_i) then
5:          y_i = x_i
6:       end if
7:       if f(y_i) > f(y*) then
8:          y* = y_i
9:       end if
10:   end for
11:   update velocity and location of all the particles
12: until stop condition is satisfied
```

In the main loop of Algorithm 1, values of attractors are updated first. Then, the step "update the velocity and location" involves two main actions: first, the velocity of each of the particles is updated, then all the particles change their location in the search space. According to the constriction model proposed by

Clerc and Kennedy (2002), both the velocity and the coordinates of the location undergo the following transformation in every iteration t:

$$v_j^{t+1} = \chi(v_j^t + U[0,c_1]_j^t(y_j^t - x_j^t) + U[0,c_2]_j^t(y_j^{*t} - x_j^t)) \,, \quad (1)$$

$$x_j^{t+1} = x_j^t + v_j^{t+1} \,, \quad (2)$$

where $\mathbf{v}^t = (v_1^t, \ldots, v_l^t)$ is particle velocity in t-th iteration, and $\mathbf{x}^t = (x_1^t, \ldots, x_l^t)$ is particle location. $U[0,c_1]_j^t$ and $U[0,c_2]_j^t$ are random variates uniformly distributed on $[0,c_1]$ and $[0,c_2]$, respectively, generated for the j-coordinate $j \in 1, \ldots l$ in the t-th iteration. The factor c_1 controls the attraction to the personal best, and c_2 to the global best, and χ is the constriction factor, $\chi < 1$. The constriction model has been selected because of its advantageous feature, that is, convergence to stable states when the parameters χ, c_1 and c_2 have the appropriate values, namely $c_1 = c_2 = 2.05$, $\chi = 0.7298$.

2.2. Our PSO algorithm

In our version of PSO, we have extended the basic idea presented in Algorithm 1 by mechanisms suitable for dynamic optimization. The selected mechanisms are briefly discussed below.

Multi-swarm There have been proposed solutions both with adaptive (Li, 2004; Li, Branke and Blackwell, 2006; Parrot and Li, 2006) and static number of subs-warms (Blackwell and Branke, 2004, 2006). In our research, we selected the version with the number of sub-swarms specified in advance (Blackwell and Branke, 2004). In this approach, the sub-swarms are defined before the beginning of optimization. The sub-swarms have their own sub-swarm attractors and there is no exchange of information between the sub-swarms.

Exclusion This mechanism was proposed for the multi-swarm version of PSO. The exclusion (Blackwell and Branke, 2006) guarantees the appropriate distribution of the sub-swarms over the entire search space by elimination of sub-swarms being located too close to each other. When the sub-swarms attractor are closer than the distance r_{excl}, the occupation of the same optimum is most likely to occur. In this case, one of the sub-swarms is selected to be eliminated and a new one is generated from scratch.

Quantum swarm In our research, sub-swarms consist of two types of particles governed by different movement rules: the location of the first type of particles is evaluated according to the constriction model principles (Clerc and Kennedy, 2002), that is, eq. (2), whereas the remaining ones are treated as quantum particles (Trojanowski, 2009) and change their location according to the analogy with the quantum dynamics of particles. At the beginning of each subsequent execution of the main loop the new swarm-attractor is selected based on the solutions found by both the classic and quantum particles.

Our model (Trojanowski, 2009) of quantum particle movement extends the one originally proposed by Blackwell and Branke (2004). As in the original model, the swarm attractor takes the role of the current location of moved particle, however, the new location is evaluated with a two-phase mechanism. In the first phase, a new point from a uniform distribution inside the quantum cloud of the range $\delta = r \cdot (D_{\mathrm{w}}/2)$ is found, where r is a control parameter and D_{w} is a width of the feasible part of the domain, that is, a distance between the lower and the upper boundary of D. In the second phase, the new point is moved forward according to the direction defined by the original location and the point. The new point is located in the new distance d' from the current location, which is calculated as follows:

$$d' = d \cdot \exp(-f'(\mathbf{x}_i)) \cdot S\alpha S(0,1), \tag{3}$$

where d is a distance from the origin obtained in the first phase, $f'(\mathbf{x}_i)$ is the value of the i-th solution $\mathbf{x}_i$ normalized in $[0,1]$ with respect to the values of all the solutions in P, and $S\alpha S(0,1)$ is an α-stable symmetric distribution variate [1]. This method returns an isotropic distribution, where locations are distributed equally in all directions, that is, none of directions is distinctive in any sense.

The α-stable distribution is controlled by four parameters: stability index α ($\alpha \in (0,2]$), skewness parameter β, scale parameter σ and location parameter μ. In the symmetric version of this distribution (called $S\alpha S$, that is, symmetric α-stable distribution), β is set to 0. In the practical implementation the Chambers-Mallows-Stuck method of generation of the α-stable symmetric random variables (Chambers, Mallows and Stuck, 1976) can be used. For $\alpha = 2$ the $S\alpha S(\mu,\sigma)$ distribution reduces to the Gaussian $N(\mu,\sigma)$, and for $\alpha = 1$ the Cauchy $C(\mu,\sigma)$ is obtained.

2.3. Feasible and unfeasible solutions

The principles of quantum particle movement allow us to generate new location candidates outside D. Therefore, we need to make the algorithm ready for appearance of such a situation, that is, we need to enrich the algorithm by a constrained optimization mechanism. For more information about constrained optimization with PSO the reader is referred to, for example, the publications by Pulido and Coello (2004); Muñoz Zavala, Aguirre and Villa Diharce (2005); Lu and Chen (2006); Paquet and Engelbrecht (2007). We are not particularly interested in studying constrained optimization in this research, therefore, we selected just a very simple procedure of immediate repairing unfeasible particles. Clearly, the j-th coordinate of the solution $\mathbf{x}$, violating its box constraints is trimmed to the exceeded limit, that is:

$$\texttt{if } x_j < lo \texttt{ then } x_j = lo, \quad \text{and} \quad \texttt{if } x_j > hi \texttt{ then } x_j = hi, \tag{4}$$

[1] α-stable laws were introduced by Paul Levy during his investigations of behavior of sums of independent random variables in the early 1920s. For a computer software see the web page by J. P. Nolan: http://academic2.american.edu/∼jpnolan/stable/stable.html

where lo is the lower boundary of the search space and hi – the upper one. The coordinates of both quantum and classic particles are repaired in the same way. In case of neutral particles, the velocity vector $\mathbf{v}$ of the repaired particle remains unchanged, even if it still leads the particle outside the acceptable search space.

3. Memory mechanisms

There are two novelties proposed and studied in this paper: the memory mechanism for explicit solutions and the mechanism for solutions stored in implicit form, that is, in the form of clusters. The two mechanisms are discussed below.

3.1. Memory structures for explicit solutions

The mQSO algorithm equipped with memory structures for explicit solutions ($\text{mQSO}_{\text{ExplMem}}$) stores complete solutions in a form of an unordered set. At the very beginning the structure for solutions, that is, the memory buffer is empty.

3.1.1. Rules of reminding

When the change occurs in the environment, current values of the particles are most probably out of date and reevaluation is needed. Just before the reevaluation, however, it is the most appropriate moment for injection of the solutions from the memory buffer into the sub-swarms. More precisely, in this step randomly selected solutions from the memory replace some of the solutions in each of the sub-swarms. Only one solution is selected to be replaced within each of the sub-swarms. For the minimal loss of information in the sub-swarm we select a pair of solutions being the closest to each other among all the solutions in the sub-swarm. Then random solution from the memory buffer is selected and its copy takes place of one of the solutions from the pair. The copy overwrites that one from the pair, which is located closer to the solution from the memory buffer.

3.1.2. Rules of remembering

Eventually, when the step of reminding is finished, we can reevaluate all the solutions in the sub-swarms and write the swarm attractors to the memory buffer. Keeping all the versions of attractors in the memory during the entire optimization process would cost lots of resources and in fact is not necessary. A suboptimal solution or even a solution being quite far from the optimum, but remaining in its basin of attraction, gives the swarm sufficient information to find the right way to the optimum. Therefore, when a new solution is to be written into the memory, a solution in the buffer is searched, which would be the closest to the newcomer. If the distance is less than a defined neighborhood threshold υ, it is assumed that both solutions belong to the basin of attraction of the same optimum. The threshold υ has to be tuned with regard to the severity

of changes. When the solution in the memory buffer is found, it is probable that it represents the area of the previous location of the optimum, whereas the newcomer represents the area of its current location. If so, there is no sense to keep the outdated information and the remembered solution is replaced by the new one. In the opposite case, when no solutions in the range of υ have been found, the new solution is simply added to the buffer. In this case, it is assumed that a new local optimum appeared in the search space.

The strategy of memory update described above allows us to refresh the information about existing peaks in the memory every time the change occurs, but also to extend the stored set of solutions when the new peak appeared. It is important to stress that the proposed strategy assumes specific type of changes, that is, the changes of limited severity. This means that it is expected that reallocation of each of the optima should be always of limited range and especially no catastrophic changes, which completely reconstruct the optimized environment are expected. The value of υ is based on this maximum range of reallocations.

3.1.3. Some additional remarks

The first remark concerns the issue of forgetting. The procedures described above do not specify any step where the number of solutions in the buffer is verified. So, theoretically, the remembered information can expand its volume continuously and to infinity. This phenomenon of unconstrained growth could be under control with use of the threshold υ, which defines maximum density of points stored in the memory. The optimal value of υ should be close to the maximum range of a single change of the moving optimum location. After some time, points stored in the memory fill the area of the feasible search space where the optima appear. When the remembered points saturate the area with themselves, almost all the newly remembered points usually replace the existing ones and do not increase their overall number.

The next remark is about the computational cost of memory operations. The cost is concerned only with computation of distances between points. The cost grows as the number of remembered solutions increases. However, the procedures do not involve evaluation function calls, so we can assume that the full number of evaluation function calls between subsequent changes is still available.

3.2. Structures for implicit solutions — clusters

Problems with continuous increase of allocated memory resources accompanying the memory approach make this approach useless for long lasting dynamic optimization processes. Therefore, another idea of learning from previous experiences is proposed. The idea originates from the algorithm for clustering evolving data streams (Aggarwal et al., 2003). This is a one-pass clustering algorithm, able to accept large volumes of data arriving in a stream and continu-

ously evolving in time. Consequently, the output clusters also vary considerably with time respectively to the current state of the optimization. Aggarwal et al. (2003) proposed the stream clustering algorithm consisting of two parts: micro- and macro-clustering. In our approach, just the the micro-clustering part was applied. For simplicity, all the micro-clusters are called just clusters in the entire further text.

The novelty in the mQSO algorithm equipped with cluster structures (labeled as $\text{mQSO}_{\text{ClStrm}}$) comes from the combination of the two algorithms into one. The cooperation between them is based on the use of the data selected to remember as an input data stream for the clustering algorithm. There are many similarities with the explicit memory mechanism described in the previous section. The step of remembering conforms to the one-pass clustering algorithm. The memory structure storing explicit solutions conforms to the structure with clusters evolving respectively to the current state of optimization and the assumed time horizon. There is also a recalling procedure, which is adjusted to the current structure with clusters. Detailed description how the clusters look like, how to calculate them, and how to make use of them is given below.

3.2.1. Cluster structure

Clusters are defined in exactly the same way as by Aggarwal et al. (2003). A single cluster aggregates information for a selected set of points in the search space. A cluster for a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_m$ with time-stamps $t_1, \ldots, t_m$ is defined as a tuple ($\mathbf{CF2}^x$, $\mathbf{CF1}^x$, $CF2^t$, $CF1^t$, m), wherein the components are defined as follows:

- $\mathbf{CF2}^x$ consists of the sum of the squares of data values, i.e., $CF2^x_j = \sum_{i=1}^{m} (x_{ji})^2$,
- $\mathbf{CF1}^x$ consists of the sum of data values, i.e., $CF1^x_j = \sum_{i=1}^{m} x_{ji}$,
- $CF2^t$ consists of the sum of the squares of the time-stamps, i.e., $CF2^t = \sum_{i=1}^{m} (t_i)^2$,
- $CF1^x$ consists of the sum of the time-stamps, i.e., $CF1^t = \sum_{i=1}^{m} t_i$,
- m equals the number of points assigned to the current cluster.

Besides, for every cluster, a centroid $\mathcal{M}$ is assigned. The centroid represents a geometrical center of the set of points, that is, $\mathcal{M}_j = 1/m \sum_{i=1}^{m} x_{ji}$.

3.2.2. Absorption of new solutions

At the very beginning of the optimization process the memory is empty, that is, there are no clusters. They have to be created before the initial phase of the process is over. The initial phase lasts as long as the value of offline error calculated on-line has an extremely high level. Usually, a few preliminary experiments allow to estimate the number of first iterations of the algorithm

necessary to reach a reasonably low level of off-line error. Within these iterations there is no point to calculate offline error, however, information about the progress of optimization can be successfully gathered. More precisely, within this period all the sub-swarm attractors and particles are collected in a stream buffer. They are saved in the buffer every time a change occurs in the environment. When the initial phase is over, the initial set of clusters is generated with a standard k-means clustering algorithm based on the data in the buffer. The initial number of clusters is given to the algorithm before the experiment starts and depends on the size of available memory resources.

In the main phase of the experiment the clusters are updated at every change in the environment. When the change occurs, all the sub-swarm attractors are copied into the input stream of the clustering algorithm. Then, the solutions in the stream are treated one by one. First, Euclidean distance between the input solution and each of the cluster centroids is found. For the closest cluster, we check if the point falls within the cluster maximum boundary, that is, if the distance to the point is less than R_{cluster}. In our experiments, R_{cluster} is by default set to 10 when the cluster has just one point assigned, otherwise it equals RMS deviation of the cluster points from the centroid $\mathcal{M}$, that is:

$$R_{\text{cluster}} = \sqrt{\sum_{j=1}^{n}(1/m\sum_{i=1}^{m}(\mathcal{M}_j - x_{ji})^2)}. \tag{5}$$

Every solution in the stream can be absorbed by existing clusters or constitute a new one. When the solution belongs to the closest cluster maximum boundary, it is absorbed by the cluster. More precisely, both all the components of the tuple representing the cluster and its centroid are updated, that is, m is incremented and sums are increased by respective data and a time-stamp of the solution. In the opposite case, that is, when none of the clusters is close enough to the input solution, a new cluster has to be created.

3.2.3. Cluster validation

Just after the absorption of the input solution the clusters are validated. There are two types of validation. The first one check the size of clusters, whereas the second one checks if they represent no outdated information. Depending on the case, the number of clusters can be increased or decreased. Both cases are discussed below.

First, the size of the cluster, which absorbed the input solution is compared with the limit MAX_{size}. If the size exceeds the limit, the cluster has to be split into two. Two points with the biggest Euclidean distance to each other, are found in the cluster, and then used as initializers for two new clusters. The remaining points are divided into two groups, the first one with points closer to the first initializer, and the other one with points closer to another initializer.

Then, for each of the two groups their centroids are calculated and eventually new clusters are generated.

The next validation verifies the age of data in the cluster. For each of the clusters we find the time of arrival of the q_{relev}-th quantile of the solutions, assuming that their time-stamps are normally distributed:

$$q_{\text{relev}} = \begin{cases} l/(2 \cdot m) & \text{iff } l \leq m, \\ 0.5 & \text{otherwise.} \end{cases} \tag{6}$$

The obtained value of arrival time is called relevance stamp s_{relev}. When s_{relev} is below a user-defined threshold $t - t_{\text{relev}}$ where t is the current time, the cluster is regarded as too old and can be eliminated to make room for other clusters.

3.2.4. Drawing on the cluster knowledge

Application of continuously updated clusters allows us to draw on the knowledge collected during the optimization process by injection of solutions originating from the clusters into sub-swarms. This is done just after the change but before the reevaluation of solutions in the sub-swarm. In the first step, we need to find a solution to remove. This can be done according to the same principle as for the memory with explicit solutions. Simply, we select a pair of two solutions being the closest to each other among all the solutions in the sub-swarm. Then a copy of centroid from the randomly selected cluster overwrites this one from the two in the pair which is located closer to the centroid. The overwritten solution is lost. The procedure of data transfer is repeated for each of the sub-swarms in the manner, which does not allow to select any of the centroids more than once during a single step of reminding.

3.3. Parameters

Application of clusters requires a few parameters to be defined. The first parameter is the range of the cluster maximum boundary R_{cluster} which is equal to 10 when the cluster consists of just one solution, and calculated with eq. (5) for clusters with number of solutions greater than one. Then, we need to define also the relevance threshold for the time window t_{relev}, the minimum number of points in the cluster $l = 10$, which is used in calculation of q_{relev} (eq. (6)), and maximum size of a cluster MAX_{size}.

4. Plan of experiments

4.1. Measure of results applied

For evaluation of the algorithm effectiveness we used the *offline error* measure (Branke, 1999, 2002) called briefly *oe*. The offline error represents the average deviation from the optimum value of the best individual evaluated since the last

change of the optimized environment. As mentioned earlier, during the optimization process *oe* starts to be evaluated only after some number of changes in the environment. We thus minimize the measurement error caused by extremely big values of *oe* appearing in the initial phase.

4.2. Benchmark

As a dynamic test-bed the MPB generator (Branke, 1999) was selected. In MPB the environment is built of a set of unimodal functions. Appropriate tuning of the function parameters allows for generation of diverse types of changes in the environment. In our research we were particularly interested in studying two types of them, requiring completely different response from the optimizer. The first one is the case when a global optimum hill actually occupied by the particles moves, however, it still remains the global optimum and some of the particles remain in its basin of attraction after the change. The second one occurs when after the change the occupied hill stops to be the global optimum and none of the hills currently controlled by the other particles takes over the role of the global optimum. In the latter case, the immediate extensive exploration of the search space is needed.

Intensity of appearance of both types of situations in the search space can be controlled by the number of peaks in the environment and their maximum shift distance. Since the number of sub-swarms is constant and is set to ten, the first situation appears for ten moving peaks in the search space. In this case, it is expected that every peak is controlled by another sub-swarm and all we need is to make the sub-swarms able to follow their hills. The second type of changes occurs when the number of moving peaks is much higher than the number of sub-swarms, for example, there are 50 moving peaks and ten sub-swarms. Every change causes not only a move of the peaks but also changes of their elevation so it hardly even happens that the highest peak remains the highest after the change.

The remaining parameters of MPB were set exactly the same as specified in scenario 2 of this benchmark. The evaluation function was defined for the five-dimensional search space with boundaries for each of dimensions set to $[0; 100]$. For the search space there exist a set of moving peaks of randomly varying altitude within the interval $[30; 70]$, width within $[1; 12]$, and coordinates of the peak by one.

4.3. Static parameter settings for the algorithm

The algorithm parameter settings applied for the experiments presented below originate from Blackwell and Branke (2006). Those authors present results of experiments obtained for different configurations of mQSO, tested with the MPB benchmark. Among many tested configurations the best results for the optimization problem with ten moving peaks were obtained when there were ten

sub-swarms and each of them consisted of five neutral particles and five quantum ones (see Table III in Blackwell and Branke, 2006). The total population of particles consisted of 100 solutions divided equally into ten sub-swarms. The values of pure PSO parameters were: $c_{1,2} = 2.05$ and $\chi = 0.7298$. The QSO parameter r_{excl} depends on the number of peaks and the volume of the search space in the benchmark (Blackwell and Branke, 2006). For ten moving peaks $r_{\text{excl}} = 31.55$, whereas for 50 moving peaks $r_{\text{excl}} = 22.87$.

4.4. Initialization and re-initialization of particles

At the beginning of each subsequent experiment, all the particles are randomized. First, all the particle coordinates are generated within feasible search space boundaries with a uniform random number generator. Then, the velocity vector coordinates are also randomly generated within the range $[-1, 1]$. Eventually, particle attractors are initialized with coordinates of their locations, whereas sub-swarm attractors are initialized with coordinates of the best solution in sub-swarms.

Our algorithm has no embedded strategy for detecting changes in the fitness landscapes. We assumed that such a strategy would just introduce another unnecessary bias into the obtained values of *oe*, and make their analysis more difficult. Therefore, our optimization system is informed of the change as soon as it occurs, and no additional computational effort for its detection is needed. When the change appears, all the solutions stored in both classic and quantum particles are reevaluated. Then, classic particle attractors are overwritten by the current solutions represented by these particles, and sub-swarm attractors are overwritten by the current best solutions in the sub-swarms.

For better compatibility with experiments published in the literature, we use the limit of 5000 evaluation function calls between the subsequent changes in the fitness landscape. Every experiment was performed for 110 changes in the environment, however, it is important to stress that *oe* was not evaluated from the beginning but just after the tenth change. Every experiment was repeated 50 times, and then the average value of *oe* was calculated.

5. Results of experiments with explicit memory structures

Four cases were selected for comparing: two versions of the algorithm (the base version – $\text{mQSO}_{\text{base}}$, and the version with memory – $\text{mQSO}_{\text{ExplMem}}$) and two versions of the optimized environment (with 10 and 50 moving peaks). A set of tests was performed for each of the cases. Each set was based on the same variation of values of two parameters: α and r. The former parameter varied from 0.05 to 2 with step 0.05 while the latter – from 0.001 to 0.15 with step 0.001. This gives 6000 configurations of the two parameters per set and allows us to build reliable graphs with characteristics of the search engines. The graphs are presented in Figs. 1 and 2.

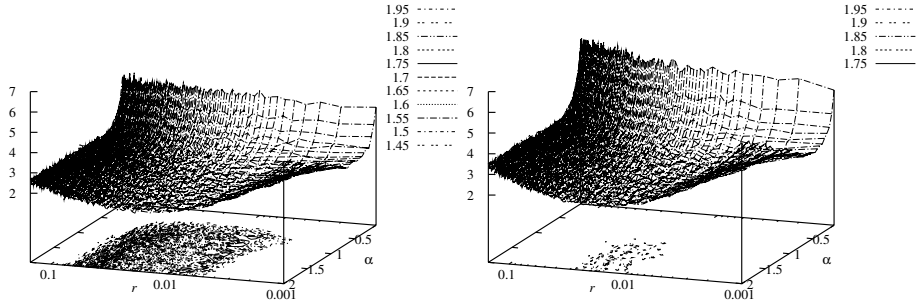

Figure 1. Characteristics of *oe* for mQSO without (graph on the left) and with (graph on the right) memory for the case with 10 moving peaks

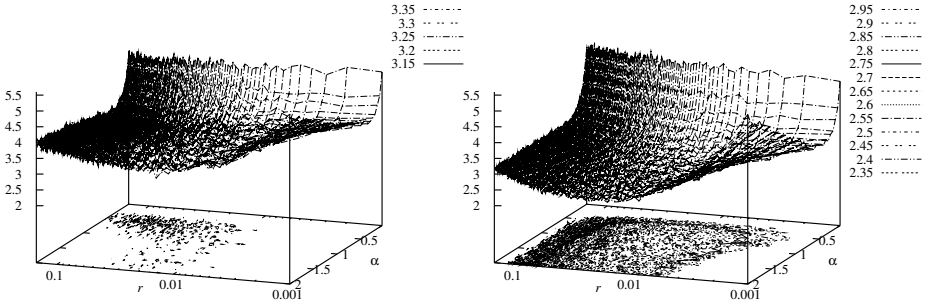

Figure 2. Characteristics of *oe* for mQSO without (graph on the left) and with (graph on the right) memory for the case with 50 moving peaks

For the case, where the number of sub-swarms is comparable with the number of peaks the results obtained with $\text{mQSO}_{\text{base}}$ are the best. In the other case, $\text{mQSO}_{\text{ExplMem}}$ outperformed $\text{mQSO}_{\text{base}}$. The best mean values of *oe* for each of the four cases are presented in Table 1. In case of high number of peaks, the best value of the scale parameter in $S\alpha S$ is higher for $\text{mQSO}_{\text{base}}$ because the global optimum can be lost easily after every change. Therefore, sub-swarms have to exploit current area of their residence and explore for new promising areas as well. When the memory is added, however, the optimal value of r goes back down because the memory takes over just the exploration part.

Unfortunately, a harmful side effect of continuous increase of allocated memory during the optimization process accompanies the memory mechanism. Sample graphs with mean sizes of volume allocated are presented in Fig. 3. Fig. 3 shows the sizes for a series of ten repeated experiments for the two best configurations of $\text{mQSO}_{\text{ExplMem}}$ from Table 1.

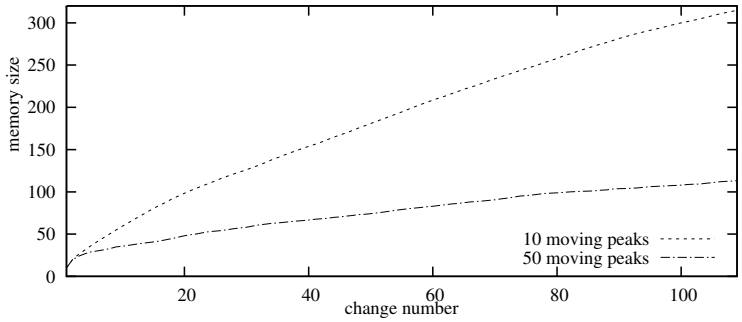

Figure 3. Number of solutions stored in the memory buffer after subsequent changes for two types of environment: with 10 and with 50 moving peaks

Table 1. The best mean values of *oe* obtained for each of the six groups of experiments

no. of peaks; version of mQSO	*oe*	std. dev.	r	α
10 peaks; $\text{mQSO}_{\text{base}}$	**1.4293**	0.371	0.016	1.65
10 peaks; $\text{mQSO}_{\text{ExplMem}}$	**1.7389**	0.334	0.018	1.75
50 peaks; $\text{mQSO}_{\text{base}}$	**3.1321**	0.472	0.031	0.80
50 peaks; $\text{mQSO}_{\text{ExplMem}}$	**2.3060**	0.303	0.018	1.05

In the case of ten moving peaks, usually seven of ten newcomers replace the older members. In the case of 50 moving peaks, the replacement level is even higher – nine per ten. However, in both cases the volume never stabilizes until the end of the experiment. This continuous increase of allocated memory can be explained by the large volume of the entire search space in comparison to the volume of a point neighborhood. The distance between lower and upper boundaries of the search space equals 100, whereas the neighborhood threshold υ of the point is set to one. The volume of such a search space equals 100^5, whereas the volume of the neighborhood equals $8\pi^2/15$, which makes this neighborhood approximately $19 \cdot 10^8$ smaller than the entire search space.

6. Results of experiments with clusters

For $\text{mQSO}_{\text{ClStrm}}$, two groups of experiments were conducted: with 10 and 50 moving peaks in the optimized environment. In both cases, we applied such configurations of the algorithm parameters which gave the best values of *oe* for $\text{mQSO}_{\text{base}}$ (Table 1). The cluster memory mechanism was controlled by two parameters: MAX_{size} and t_{relev}. We varied these parameters within some

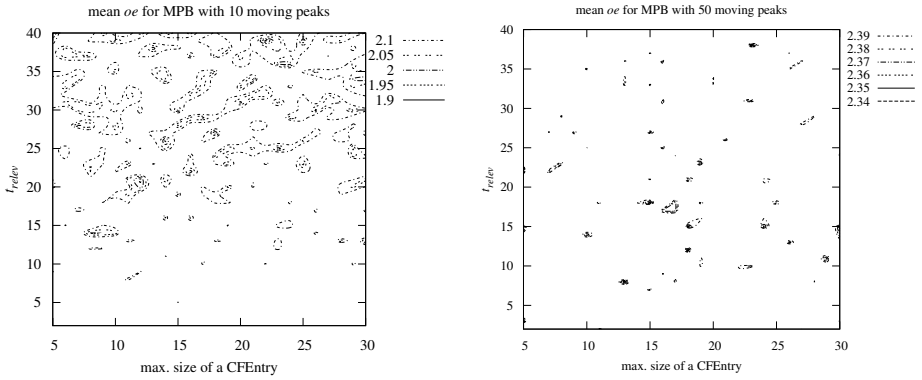

Figure 4. Mean values of *oe* for $\text{mQSO}_{\text{ClStrm}}$ with varying number of clusters. Two cases: for ten (left figure) and 50 (right figure) moving peaks

ranges, and evaluated *oe* for each of the parameter configurations. As in the case of $\text{mQSO}_{\text{ExplMem}}$, the configurations were uniformly distributed over the subspace of feasible configurations, which was as a result covered by a uniform grid of testing points. The configurations were defined as follows: MAX_{size} varied from 5 to 30 with step one, and t_{relev} varied from 2 to 40 with step one. The means of *oe* obtained for series of 50 experiments are depicted in Fig. 4.

The best mean values of *oe* are equal to: 1.888 for 10 moving peaks (obtained for $MAX_{\text{size}} = 22$ and $t_{\text{relev}} = 39$), and 2.332 for 50 moving peaks (obtained for $MAX_{\text{size}} = 23$ and $t_{\text{relev}} = 38$). For the case with 10 moving peaks, the performance of $\text{mQSO}_{\text{ClStrm}}$ is always worse than of $\text{mQSO}_{\text{base}}$ (Table 1). It is worth noting, however, that the performance is directly dependent on the t_{relev}. For the case with 50 moving peaks, the performance of $\text{mQSO}_{\text{ClStrm}}$ is worse than of $\text{mQSO}_{\text{ExplMem}}$, but still much better than of $\text{mQSO}_{\text{base}}$.

7. Conclusions

This paper studies the properties of the multi-swarm approach extended by the memory mechanisms of two types: with explicit memory structure and with memory represented as a set of clusters. Performance of mQSO equipped with both mechanisms was experimentally verified. For the case, where the number of moving peaks is close to the number of sub-swarms, the best results were returned by $\text{mQSO}_{\text{base}}$. For the case, where the number of moving peaks is much higher than the number of sub-swarms, both proposed versions of mQSO equipped with memory structures outperformed the base version of mQSO. The best results were obtained with $\text{mQSO}_{\text{ExplMem}}$. Unfortunately, application of this version of mQSO is accompanied with disadvantageous side effect of continuous increase of allocated resources. The results returned by $\text{mQSO}_{\text{ClStrm}}$ are

worse than by $\text{mQSO}_{\text{ExplMem}}$, but this time the number of allocated resources remains under control. In spite of the fact that in $\text{mQSO}_{\text{ClStrm}}$ the number of clusters is not limited, the proposed principles of cluster validation allow us to stabilize the number of clusters on a fixed level.

Acknowledgment

The author would like to thank Krzysztof Ciesielski for inspiration and very useful discussions on clustering evolving data streams.

References

Aggarwal, C.C., Han, J., Wang, J. and Yu, P.S. (2003) A Framework for Clustering Evolving Data Streams. In: *VLDB 2003: Proc. of 29th Int. Conf. on Very Large Data Bases, September 9–12, 2003, Berlin, Germany*, Morgan Kaufmann Publishers, 81–92.

Blackwell, T. and Branke, J. (2004) Multi-swarm Optimization in Dynamic Environments. In: *Applications of Evolutionary Computing, Evo Workshops 2004.* **LNCS 3005**, Springer, 489–500.

Blackwell, T. and Branke, J. (2006) Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Trans. Evol. Comput.*, **10**(4), 459–472.

Branke, J. (1999) Memory enhanced evolutionary algorithms for changing optimization problems. In: *Congress on Evolutionary Computation.* **3**, IEEE Press, 1875–1882.

Branke, J. (2002) *Evolutionary Optimization in Dynamic Environments.* Kluwer Academic Publishers.

Chambers, J.M., Mallows, C.L. and Stuck, B.W. (1976) A Method for Simulating Stable Random Variables. *J. Amer. Statist. Assoc.* **71** (354), 340–344.

Clerc, M. and Kennedy, J. (2002) The particle swarm-explosion, stability, and convergence in a multi-dimensional complex space. *IEEE Trans. Evol. Comput.* **6** (1), 58–73.

Kennedy, J. and Eberhart, R.C. (1995) Particle Swarm Optimization. In: *Proc. IEEE Int. Conf. on Neural Networks (ICNN'95)*, **4**, IEEE Service Center, 1942–1948.

Li, X. (2004) Adaptively choosing neighborhood bests in a particle swarm optimizer for multimodal function optimization. In: *GECCO 2004: Conf. on Genetic and Evolutionary Computation*, **LNCS 3102**, Springer, 105–116.

Li, X., Branke, J. and Blackwell, T. (2006) Particle swarm with speciation and adaptation in a dynamic environment. In: *GECCO 2006: Conf. on Genetic and Evolutionary Computation.* ACM Press, 51–58.

Lu, H. and Chen, W. (2006) Dynamic-objective particle swarm optimization for constrained optimization problems. *J. Comb. Optim.* **12** (4), 409–419.

Mori, N., Imanishi, S., Kita, H. and Nishikawa, Y. (1997) Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In: *7th Int. Conf. on Genetic Algorithms*. Morgan Kaufman, 299–306.

Muñoz Zavala, A.E., Aguirre, A.H. and Villa Diharce, E.R. (2005) Constrained optimization via particle evolutionary swarm optimization algorithm (PESO). In: H.-G. Beyer et al., eds., *GECCO 2005: Proc. Conf. on Genetic and Evolutionary Computation*, **1**, ACM Press, 209–216.

Paquet, U. and Engelbrecht, A.P. (2007) Particle Swarms for Linearly Constrained Optimisation. *Fund. Inform.*, **76**(1-2), 147–170.

Parrot, D. and Li, X. (2006) Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Trans. Evol. Comput.* **10** (4), 440–458.

Pulido, G.T. and Coello, C.A. (2004) A Constraint Handling Mechanism for Particle Swarm Optimization. In: *Proc. Congress on Evolutionary Computation*, **2**, IEEE, 1396–1403.

Trojanowski, K. (2009) Properties of Quantum Particles in Multi-Swarms for Dynamic Optimization. *Fund. Inform.* **95** (2-3), 349–380.

Trojanowski, K. and Michalewicz, Z. (1999) Searching for Optima in Non-Stationary Environments. In: *Congress on Evolutionary Computation*, **3**, IEEE Press, 1843–1850.

Trojanowski, K., Michalewicz, Z. and Xiao, J. (1997) Adding Memory to the Evolutionary Planner/Navigator. In: *4th IEEE Int. Congress on Evolutionary Computation*. IEEE Publishing, 483–487.

Trojanowski, K. and Wierzchoń, S.T. (2002a) Control of Immune Memory in Artificial Immune System. In: *WAE 2002: 3rd Nat. Workshop on Evolutionary Computation and Global Optimisation*. Warsaw Univ. of Technology Publishing House, 111–118.

Trojanowski, K. and Wierzchoń, S.T. (2002b) Memory Management in Artificial Immune System. In: *6th Int. Conf. on Neural Networks and Soft Computing ICNNSC 2002. Adv. in Soft Computing*, **19**, Physica/Springer, 650–655.

Trojanowski, K. and Wierzchoń, S.T. (2002c) Searching for Memory in Artificial Immune System. In: *IIS 2002: 11th Int. Symposium on Intelligent Information Systems. Adv. in Soft Computing*, **17**, Physica/Springer, 175–183.