Cooperative Charged Particle Swarm Optimiser

Anna Rakitianskaia and Andries P. Engelbrecht, *Member, IEEE*

***Abstract*— Most optimisation algorithms from the Computational Intelligence field assume that the search landscape is static. However, this assumption is not valid for many real-world problems. Therefore, there is a need for efficient optimisation algorithms that can track changing optima. A number of variants of Particle Swarm Optimisation (PSO) have been developed for dynamic environments. Recently, the cooperative PSO [1] has been shown to significantly improve performance of PSO in static environments, especially for high-dimensional problems. This paper investigates the performance of a cooperative version of the charged PSO on a benchmark of dynamic optimisation problems. Empirical results show that the cooperative charged PSO is an excellent alternative to track dynamically changing optima.**

I. Introduction

Computational Intelligence (CI) techniques are becoming more widespread, and the range of problems where these techniques can be of aid is growing. A new category of complex optimisation problems has recently emerged, including problems where the search space adapts over time. Optimising such problems is not an easy task, since the algorithm should not just locate the optima, but also detect changes in the optima and track optima to new locations, as well as automatically locate new optima.

PSO has shown to be a simple, but very efficient optimisation technique [2], [3]. Due to its success in static environments, a number of variations have been developed such that PSO can be applied to dynamic environments [4]–[8]. One of the most successful PSO algorithms for dynamic environments is the charged PSO developed by Blackwell and Bentley [4]. The charged PSO continually searches for better solutions while refining the current solution. This is done by achieving a good balance between exploration and exploitation.

Recent research in PSO produced many ways in which algorithm performance can be boosted [9]. The cooperative split PSO (CSPSO) [1], [9], [10] is one such improvement, where a divide-and-conquer approach to problem solving is followed. The search space is divided into smaller subspaces, with each subspace being optimised by a separate swarm. The best solutions from all subswarms are combined to form the overall solution. The CSPSO has been shown to reduce computational complexity and to improve the accuracy of solutions found [1], [9], [10], especially for high-dimensional problems.

A. Rakitianskaia is with the Department of Computer Science, School of Information Technology, University of Pretoria, Pretoria 0002, South Africa (e-mail: annar@cs.up.ac.za).

A. P. Engelbrecht is with the Department of Computer Science, School of Information Technology, University of Pretoria, Pretoria 0002, South Africa (e-mail: engel@cs.up.ac.za).

Due to the success of the CSPSO on static problems, and the charged PSO on dynamic problems, this paper proposes a hybrid between these two algorithms, referred to as the cooperative charged PSO. This is achieved by using the CSPSO divide-and-conquer mechanism, and by using the charged PSO in each subswarm. The paper conducts an empirical investigation into the advantages of this hybrid algorithm.

The rest of the paper is structured as follows: Section II provides background information on PSO, CSPSO, dynamic environments, and the charged PSO. Section III introduces the cooperative charged PSO. Section IV discusses the experimental setup, problems used in experiments, performance measurements specific to dynamic environments, and analyses experimental results. Section V concludes the paper, summarising the observations. Topics for further research are also mentioned in this section.

II. Background

This section briefly discusses various algorithms used in the paper, existing approaches to PSO in dynamic environments, and properties of dynamic environments.

A. Particle Swarm Optimisation Algorithm

Particle Swarm Optimisation (PSO) (introduced in [2]) is a nature-inspired population-based optimisation technique. As the name suggests, PSO operates on a swarm of particles, where each particle represents a candidate solution to the problem. The swarm is "flown" through the search space. At each time step t, the position $\vec{x}_i(t)$ of a particle i is modified by adding particle velocity $\vec{v}_i(t)$ to the previous position vector:

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t)$$

Particle velocity determines the speed and direction of the particle. Velocity is updated using

$$\begin{aligned}\vec{v}_i(t) = \omega\vec{v}_i(t-1) &+ c_1 r_1(\vec{x}_{pbest}(t) - \vec{x}_i(t)) \\ &+ c_2 r_2(\vec{x}_{gbest}(t) - \vec{x}_i(t)),\end{aligned} \tag{1}$$

where ω is the inertia weight, controlling the influence of previous velocity values on the new velocity; $\vec{x}_i(t)$ is the position of particle i; c_1 and c_2 are acceleration coefficients used to scale the influence of the *congnitive* (second term in equation (1)) and *social* (third term in equation (1)) components; r_1 and r_2 are random numbers sampled from a uniform $U(0,1)$ distribution; $\vec{x}_{pbest}(t)$ is the *personal best* of particle i, or, in other words, the best position encountered by this particle so far; similarly, $\vec{x}_{gbest}(t)$ is the *global best* of particle i, or the best position found by any of the particles in the neighbourhood of particle i. Thus, each particle is

attracted to both the best position encountered by itself so far, as well as the overall best position found by the neighbourhood so far.

PSO can be written out in pseudocode as follows:

1) Initialise the swarm of N particles.
2) For each particle $i \in 1..N$, set personal and global best positions, and update particle velocity and position.
3) Repeat step 2 until a stopping condition is met.

B. Cooperative PSO

The cooperative PSO was first introduced by Van den Bergh and Engelbrecht in [1], and analysed in [10].

The main idea behind cooperative PSO (also known as the split PSO and split-cooperative PSO) is to split the search space dimension-wise into K mutually-exclusive portions and to assign each portion to a separate subswarm. Each subswarm then optimises the problem over a limited set of dimensions. In order to get the complete solution vector, the best particles (partial solutions) from all the subswarms have to be combined together. Two issues arise at this point: (1) How can one select the best solution from a subswarm if the subswarm works on a limited number of dimensions, thereby optimising only a part of the bigger problem? There is no fitness function defined for the smaller subproblems. (2) How can the solutions from the subswarms be combined?

Both of these issues are resolved by introducing a *context vector*. The context vector maintains the complete solution, and each subswarm uses the context vector to evaluate its particles. This is done by substituting values into the context vector only for the dimensions that a subswarm is responsible for, and keeping the rest of the context vector fixed at the best values from the other subswarms. Each subswarm evaluates the context vector for each of its particles substituted into it, and returns as the best solution the particle that, when substituted into the context vector, yielded the best fitness. After application of this procedure to each subswarm, the context vector will contain the optimal solution found so far.

Pseudocode for the CSPSO is given below:

1) Initialise K sub-swarms. Assuming that the problem dimension is n and n is divisible by K, each subswarm will be of dimension $\frac{n}{K}$.
2) For each sub-swarm:
   a) For each particle:
      i) Evaluate fitness using the context vector.
      ii) Adjust velocity and position.
      iii) Determine personal best.
   b) Determine global best.
3) Repeat step 2 until a stopping condition is met.

What are the advantages of splitting the search space into subsets? First of all, it reduces problem complexity. Empirical studies have shown that PSO performance decreases with increase in problem dimension [10]. One reason for this might be the fact that fitness is calculated for the complete solution vector, which may average out contributions of different dimensions. Suppose a multidimensional candidate solution vector is obtained, where the value in a certain dimension is optimal, however, the value in another dimension is very bad. The bad value will yield bad overall fitness of the current solution vector. As a result, other particles in the neighbourhood will not be attracted to this position, and the optimal value found in a certain dimension will be dismissed. However, when the problem is split dimension-wise, subswarms optimise smaller dimensional problems, thus each dimension has a larger contribution to the final solution. This decreases the possibility of losing valuable information. This property implies that cooperative PSO would be able to better adapt to local environment change, i.e. change in a limited number of dimensions. Reduced complexity also improves performance in high-dimensional search spaces [10].

Another advantage is that the CSPSO can easily be parallelised, thereby decreasing computational costs. This is significant for optimisation in dynamic environments, where the time taken to adapt to an environment change is crucial.

C. Dynamic Environments

Dynamic environments can often be observed in real life, for example, the stock exchange, or traffic conditions on roads. Given a problem in such an environment (for example, constructing an optimal traffic lights schedule), it is clear that a solution once found, may become suboptimal, because the characteristics of the environment will change over time. Thus, optimisation algorithms for dynamic environments have two goals: to detect optima and to track optima as the environment changes. Additionally, it should be possible to detect new optima.

Considering the search space as a hypersurface, it can be argued that environment changes are basically perturbations of this hypersurface. Perturbations are realised as floating optima. An optimum may change its position and magnitude, existing optima may disappear while new optima may appear somewhere else on the hypersurface.

Based on similar hypotheses, Eberhart *et al.* [11], [12] defined three types of dynamic environments:

1) *Type I:* These are the environments where the location of optima is subject to change.
2) *Type II:* These are the environments where the location of optima remains fixed, but optima values are subject to change.
3) *Type III:* These are the environments where both the location and the value of optima are subject to change.

This paper deals with optimisation in dynamic environments, and algorithms were tested on the three types of dynamic environments in order to make the sample of experiments more representative.

D. Charged PSO

A number of PSO algorithms for dynamic environments have been developed. These include split adaptive PSO [7], fine-grained PSO [8], charged PSO [4], and quantum PSO [6].

All dynamic PSO variants have to face the following two major problems:

1) *Outdated memory:* Once the hypersurface of the objective function changes, previous values stored in PSO memory (personal best and global best positions) are no longer representative of the new environment [13].
2) *Loss of diversity:* It was formally proven [9], [14] that all particles will converge on a point, which is a weighted average of personal best and global best positions. Once converged, PSO will not explore any longer, even if the environment changes [13].

Simple solutions to the above problems are:

- Reevaluate particle positions when a change occurs. Reevaluation is effective only when particles have not yet converged.
- Reinitialise a percentage of particles when a change occurs. This will increase diversity, but at the same time valuable information about the search space will be lost.

One of the most successful PSO algorithms for dynamic environments is the charged PSO. The main objective of this paper is to propose an improvement to the charged PSO, by combining the charged PSO with the CSPSO.

The charged PSO [4] is based on electrostatic principles. All particles in a charged PSO store a charge, represented by a positive scalar value. A charge magnitude equal to zero means that a particle is neutral (i.e. does not bear a charge), and a value greater than zero indicates a charged particle. Charge magnitude can not be negative, and does not change during algorithm execution.

Charged particles repel from one another if the distance between them is small enough. This prevents charged particles from converging to a single point, thus facilitating exploration. Repulsive forces are introduced by adding an acceleration term, $\vec{a}_i$, to the standard velocity equation:

$$\vec{v}_i(t) = \omega\vec{v}_i(t-1) + c_1 r_1(\vec{x}_{pbest} - \vec{x}_i(t)) + c_2 r_2(\vec{x}_{gbest} - \vec{x}_i(t)) + \vec{a}_i(t)$$

Repulsion between two particles i and j at time step t is defined as [4]

$$\vec{a}_{ij}(t) = \begin{cases} \left(\frac{Q_i Q_j}{||\vec{d}_{ij}(t)||^3}\right)(\vec{d}_{ij}(t)), & \text{if } R_c \leq ||\vec{d}_{ij}(t)|| \leq R_p; \\ 0, & \text{otherwise.} \end{cases}$$

In the above equation, $\vec{d}_{ij}(t) = \vec{x}_i(t) - \vec{x}_j(t)$, Q_i is the charge magnitude of particle i, R_c is the core radius and R_p is the perception limit of a particle. These two limits define the distance range $[R_c, R_p]$ at which charged particles would repel.

As the empirical results have shown [4], the most efficient charged PSO architecture has 50% of the swarm charged, and the rest of the particles are neutral. Neutral particles are normal PSO particles that obey standard PSO position and velocity update rules. Thus, one half of the population acts as a standard PSO swarm and refines found solutions, in this way facilitating exploitation. Hence, charged PSO achieves a balance between exploration and exploitation.

The problem of outdated swarm memory is addressed by re-evaluating the fitness of each particle in the swarm, the personal best of each particle, and the global best of each neighbourhood, whenever a change occurs.

Blackwell and Branke introduced multiswarms in [6]. In a multiswarm, cooperation is achieved by running a number of swarms where each swarm is defined over all problem dimensions; the task of a single swarm is to track a single peak. An obvious drawback of such an approach is that the number of peaks in the objective function has to be known or estimated in advance, which might not always be possible in real life. CSPSO uses dimension-wise cooperation and makes no assumptions about the environment it is working in.

III. Cooperative Charged PSO

This paper proposes a hybridisation between CSPSO and the charged PSO. For the cooperative charged PSO, each subswarm makes use of a charged PSO. The cooperative charged PSO will have the following advantages:

- better accuracy and resistance to premature convergence due to refined search,
- ability to re-optimise the solution only in the dimensions where the change has occurred, and
- faster execution due to reduced problem complexity and parallel-ready architecture

IV. Experimental Results

The different algorithm and problem parameters used in the experiments that were carried out are outlined in this section. All algorithms were implemented using the CILib (Computational Intelligence Library) framework. CILib is an open-source project written in the Java programming language. The latest version that includes implementations of all the algorithms analysed in this paper, together with problem simulations, can be downloaded from *http://cilib.sourceforge.net*.

A. Experimental Setup

The aim of the experiments was to compare the performance of a standard charged PSO to a cooperative charged PSO. For this purpose, the dynamic moving peaks function [15] was used as a benchmark problem. Parameters used for both the algorithms and the problem are outlined below.

1) Charged PSO: In all experiments, the inertia weight ω was set to 0.729844 while the values of the acceleration coefficients c_1 and c_2 were set to 1.496180. This choice is based on [16], where it was shown that such parameter settings give convergent behaviour. Velocity was not constrained, and a Von Neumann swarm topology was used, since it is known to promote diversity [8], which is beneficial for dynamic problems.

Parameters for the charged PSO were based on empirical and theoretical studies published in [6]. Radii of repulsion R_c and R_p were set to 1 and 1000, respectively. Such a wide range was chosen in order to constrain acceleration the least. All charged particles bore a charge magnitude of $Q = 0.3$;

TABLE I

COMPARISON RESULTS ON MOVING PEAKS TYPE I, II AND III FOR DIFFERENT CPSO AND CCPSO CONFIGURATIONS

10-Dimensional Results									
Function	30 Particles			40 Particles			50 Particles		
	CPSO	$CCPSO_2$	$CCPSO_3$	CPSO	$CCPSO_2$	$CCPSO_4$	CPSO	$CCPSO_2$	$CCPSO_5$
Type I	5.174864	2.770571	2.746289	3.726819	2.480963	2.578542	8.740904	2.598303	2.541954
Type II	10.22702	10.75957	9.501665	9.359644	11.11443	11.05367	11.48007	9.628640	10.40601
Type III	13.21649	12.14580	13.46634	11.08900	12.09872	13.53482	19.26544	12.20411	13.54199
30-Dimensional Results									
Function	30 Particles			40 Particles			50 Particles		
	CPSO	$CCPSO_2$	$CCPSO_3$	CPSO	$CCPSO_2$	$CCPSO_4$	CPSO	$CCPSO_2$	$CCPSO_5$
Type I	18.96331	10.38750	8.322251	17.91513	10.24580	6.554991	22.02454	10.07373	5.463228
Type II	24.40358	17.15510	15.60511	20.59032	16.34300	16.34128	39.84675	16.90977	13.43909
Type III	25.12467	19.73833	16.43730	23.91404	19.30508	18.20369	40.53018	22.38823	16.01231
100-Dimensional Results									
Function	30 Particles			40 Particles			50 Particles		
	CPSO	$CCPSO_2$	$CCPSO_3$	CPSO	$CCPSO_2$	$CCPSO_4$	CPSO	$CCPSO_2$	$CCPSO_5$
Type I	131.8274	75.08864	63.14444	113.8889	67.43745	43.62751	120.7774	61.21762	32.34882
Type II	155.2341	98.64216	85.27976	130.2408	89.27339	50.32634	231.1876	87.80434	38.53914
Type III	150.8090	106.1653	83.42042	129.7577	88.33909	64.60076	231.4262	93.57236	47.67687

50% of each swarm was charged with another 50% kept neutral.

Swarms of 30, 40 and 50 particles were used in experiments.

2) Cooperative PSO: In order to be comparable with the standard charged PSO, the cooperative charged PSO used the same number of particles. For example, a charged PSO of 30 particles was compared to a cooperative charged PSO with 2 subswarms of 15 particles each, or 3 subswarms of 10 particles each.

3) Moving Peaks Benchmark Function: The publicly available moving peaks benchmark [15] was used in experiments. The moving peaks is a function containing a specified number of peaks that dynamically move around the search space, with changing heights and widths. The moving peaks function with n dimensions and m peaks can be formulated as

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

where $B(\vec{x})$ is a time-invariant "basis" hypersurface, and P is the peak function, where each peak, i, such that $i \in [1,m]$, has its own time-variant height, $h_i(t)$, width, $w_i(t)$, and location, $\vec{p}_i(t)$. Coordinates, height, and width of each peak are randomised within specified intervals at $t=0$. Each Δe evaluations, the location of every peak i is moved by adding a vector $\vec{\nu}_i(t)$ of a fixed length s to it. The direction of $\vec{\nu}_i(t)$ depends on the correlation parameter, $\lambda \in R[0,1]$. If $\lambda = 0$, the direction will be random, otherwise, if $\lambda > 0$, the new direction will depend on the previous direction. The height and width of each peak are changed by adding a random gaussian number. Thus, a change of a single peak can be formally defined as

$$\begin{aligned} \sigma &\in N(0,1) \\ h_i(t) &= h_i(t-1) + height\_severity \cdot \sigma \\ w_i(t) &= w_i(t-1) + width\_severity \cdot \sigma \\ \vec{p}_i(t) &= \vec{p}_i(t-1) + \vec{\nu}_i(t), \end{aligned}$$

where the shift vector $\vec{\nu}_i(t)$ is a linear combination of the previous shift vector and a normal random vector $\vec{r}$, normalised to the length s:

$$\vec{\nu}_i(t) = \frac{s}{||\vec{r} + \vec{\nu}_i(t-1)||}((1-\lambda)\vec{r} + \lambda\vec{\nu}_i(t-1))$$

The severity of changes can be controlled by setting s, $height\_severity$, and $width\_severity$ to appropriate values.

In all the experiments, a flat basis surface $B(\vec{x}) = 0$ was used. The cone peak function was used on all the peaks, defined as

$$P(\vec{x}, h_i(t), w_i(t), \vec{p}_i(t)) = h_i(t) - w_i(t) \cdot ||\vec{x} - \vec{p}_i(t)||$$

Table II summarises the parameter values used to generate problem examples for each type of dynamic environment.

TABLE II

PARAMETERS FOR MOVING PEAKS BENCHMARK

Parameter	Type I	Type II	Type III
Number of Peaks p	10	10	10
Peak heights	$\in [30, 70]$	$\in [30, 70]$	$\in [30, 70]$
Peak widths	$\in [1, 12]$	$\in [1, 12]$	$\in [1, 12]$
Δe	5000	5000	5000
s	5	0	5
Height change severity	0.1	7	7
Width change severity	0.1	1	1
Correlation coefficient λ	0.5	0	0.5

The experiments were run for moving peaks in 10, 30 and 100 dimensions. All reported results are averages over 30 simulations. Each algorithm ran for 1000 iterations.

B. Performance Measurement

The moving peaks function is an artificial maximisation problem. At each time step the real maximum is known, hence the difference between the real maximum and the function value at the position of a particle was used to measure particle fitness.

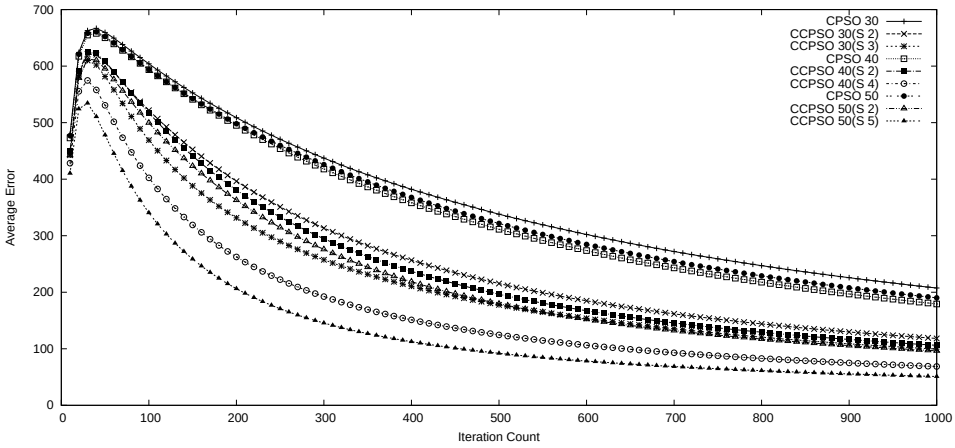

Fig. 1. Results obtained for 100-Dimensional Moving Peaks Type I

Morrison [17] showed that a representative performance measurement in a dynamic environment should reflect algorithm performance "across the entire range of landscape dynamics". Standard fitness measure reflects the current algorithm state only, therefore it can not be used. Morrison [17] suggested using mean fitness instead. Mean fitness is the average over all previous fitness values:

$$F_{mean}(T) = \frac{\sum_{t=1}^{T} F_{best}}{T}$$

where T is the total number of iterations, and F_{best} is the fitness of the best particle after iteration t. The mean fitness represents the entire algorithm performance history, hence it is valid in dynamic environments. This performance measurement was used in all the experiments described below.

C. Analysis of Empirical Data

Table I summarises the mean fitness results obtained after 1000 iterations. In addition to this, three graphs were drawn to visualise the results obtained on moving peaks type I, II and III in 100 dimensions (see Figures 1, 2 and 3). The following naming conventions are used:

- In Table I, CPSO stands for charged PSO, and CCPSO$_k$ stands for cooperative charged PSO with k subswarms.
- In all figures, CPSO N stands for charged PSO with N particles; CCPSO N(S K) stands for cooperative charged PSO with N particles and K subswarms.

Algorithm performance on moving peaks in 10, 30 and 100 dimensions is discussed below.

1) Dimension 10: As seen from Table I, the easiest problem to track changes in is the type I problem, since the smallest error was obtained for this problem. Here, cooperation was clearly beneficial; it produced consistent results for various number of particles and splits, as opposed to the standard charged PSO.

The type II problem, where optima remains static and only heights vary, is more difficult to optimise. Here, it is hard to say which configuration won: both CPSO and CCPSO produced similar results, none significantly better than the other.

The type III problem that combines optima movement with height changes is, as expected, the hardest one to optimise. Once again, cooperation did not introduce any significant improvements, except for consistency of results, which can not be said about the results produced by the standard charged PSO.

Hence, on a 10-dimensional problem cooperation helps to better track the moving optima. However, it does not help to track the changing optima heights.

2) Dimension 30: CCPSO performed remarkably well on the type I problem, producing consistent results that are twice as good as those produced by CPSO. Also, another tendency can be observed here: higher splits tend to produce better results. This can be explained by the fact that solutions in different dimensions have a lesser impact on one another, and good solutions in some dimensions are not dominated by bad solutions in other dimensions, as discussed earlier in the paper.

Results on the type II problem are a little less impressive; however, CCPSO kept producing consistently better results than CPSO. Higher splits again gave better results for swarms of 30 and 50 particles.

Large errors were obtained for the type III problem. However, compared to the 10-dimensional problem, the CCPSO performed consistently better than CPSO.

3) Dimension 100: Referring to Figure 1, for the type I problem, all algorithms produced smooth graphs with gradually decreasing average error. From this figure it is clear that the standard CPSO performed the worst. All variants of the CCPSO performed better and, similar to what was observed for the 30-dimensional problem, algorithms with most splits performed the best.

Figure 2 displays the results obtained for the 100-dimensional problem of type II. Here, the non-cooperative algorithms not only performed the worst, but also produced the least smooth graphs, which means that the changes were

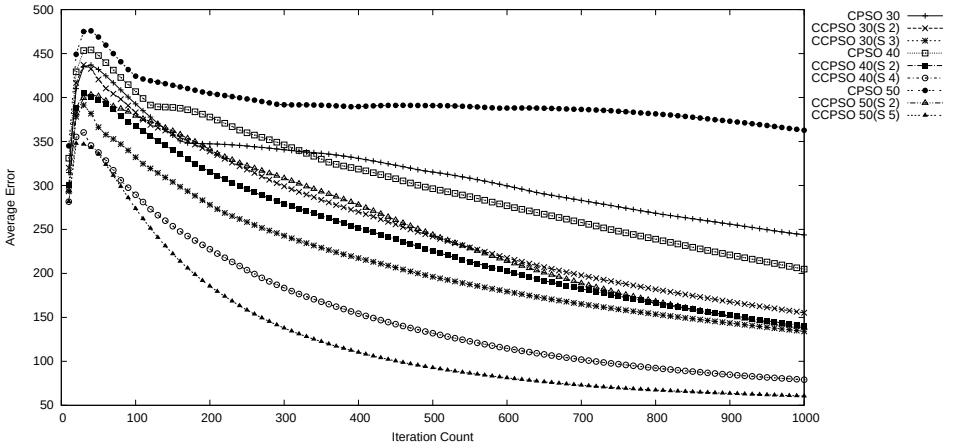

Fig. 2. Results obtained for 100-Dimensional Moving Peaks Type II

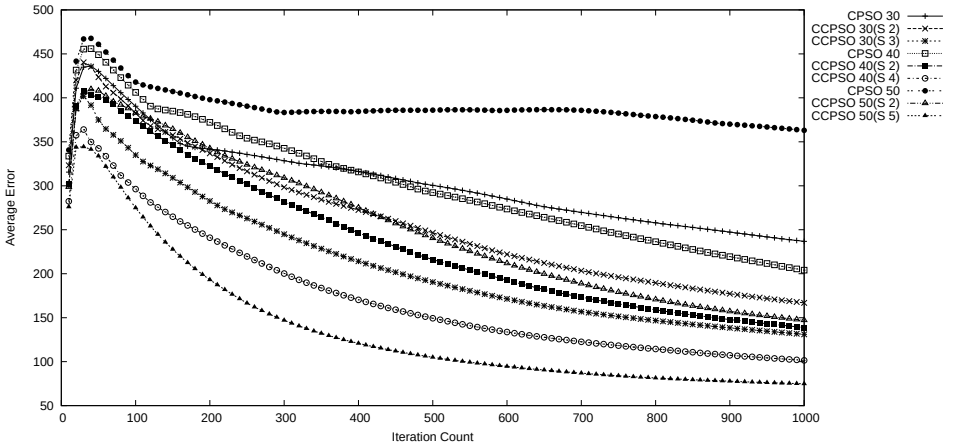

Fig. 3. Results obtained for 100-Dimensional Moving Peaks Type III

poorly tracked. On the other hand, the CCPSO variations with a higher split factor were significantly better than others. In general, all cooperative algorithms performed better than their non-cooperative counterparts.

Figure 3 shows results for the 100-dimensional problem of type III. Results are similar to that of the type II problem, however, the distinction between higher and lower split factor is less significant here. The cooperative algorithms performed visibly better than the non-cooperative algorithms, which again hardly managed to track the changing optima, especially when more particles were used.

V. Conclusion

The aim of this paper was to examine the applicability of cooperation to the charged PSO. A hybrid algorithm was proposed that combined the properties of both the CSPSO and the charged PSO. The proposed cooperative charged PSO uses the CSPSO divide-and-conquer mechanism, and each subswarm makes use of the charged PSO.

Experiments with the cooperative charged PSO were ran on dynamic problems of three different types. For each type, the algorithm was tested on search spaces of different dimensionality. Analysis of the experimental results has shown that cooperation significantly improves the performance of the charged PSO. In low-dimensional search spaces, cooperation is more useful for tracking optima location changes than optima height changes. In high-dimensional search spaces, however, cooperation improves overall performance of the charged PSO regardless of the nature of the dynamic changes. A higher split factor seems to be beneficial to algorithm performance, possibly because it reduces the interdependency between different dimensions.

In general, it can be said that cooperation introduces a significant improvement to the charged PSO. Dimension-wise search is more refined and more sensitive to changes in distinct dimensions. It produces robust results of higher accuracy, and it is less computationally expensive.

Further research would involve an in-depth study of cooperation applied to dynamic environments at large, and the applicability of cooperation to other dynamic algorithms. Also, the influence of the split factor on the performance of the cooperative charged PSO would need to be further investigated.

REFERENCES

[1] F. Van den Bergh and A. Engelbrecht, "Cooperative learning in neural networks using particle swarm optimizers," South African Computer Journal, vol. 26, pp. 84–90, 2000.

[2] J. Kennedy and R. Eberhart, "Particle swarm optimization", in *In Proc. of the IEEE International Joint Conference on Neural Networks*, IEEE Press, 1995, pp. 1942–1948.

[3] J. Kennedy and R. Eberhart, *Swarm Intelligence*, Morgan Kaufmann Publishers, 2001.

[4] T. M. Blackwell and P. J. Bentley, "Dynamic Search with Charged Swarms," *In Proc. of the Genetic and Evolutionary Computation Conference*, pp. 19–26, 2002.

[5] A. J. Carlisle, *Applying the Particle Swarm Optimizer to Non-Stationary Environments*, Ph.D. Dissertation, Auburn University, Auburn, Alabama, USA, 2002.

[6] T. Blackwell and J. Branke, "Multiswarms, Exclusion, and Anti-Convergence in Dynamic Environments", *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 459–472, 2006.

[7] J. P. Coelho, P. B. De Moura Oliviera, and J. Boa Ventura Cunha, "Non-Linear Concentration Control System Design using a New Adaptive PSO", *In Proc. of the 5th Portuguese Conference on Automatic Control*, 2002.

[8] X. Li, K. H. Dam, "Comparing Particle Swarms for Tracking Extrema in Dynamic Environments," in *In Proc. of the Congress on Evolutionary Computation*, vol. 3, pp. 1772–1779, 2003.

[9] F. Van den Bergh, *An Analysis of Particle Swarm Optimizers*, Ph.D. Dissertation, University of Pretoria, Pretoria, South Africa, 2001.

[10] F. Van den Bergh and A. Engelbrecht, "A Cooperative approach to particle swarm optimization", *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[11] R. C. Eberhart and Y. Shi, "Tracking and Optimizing Dynamic Systems with Particle Swarms," *In Proc. of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 94–100, 2001.

[12] X. Hu and R. C. Eberhart, "Tracking Dynamic Systems with PSO: Where's the Cheese?", *In Proc. of the Workshop on Particle Swarm Optimization*, pp. 80–83, 2001.

[13] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons, Ltd, 2005.

[14] M. Clerc and J. Kennedy, "The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional Complex Space", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[15] J. Branke. *The moving peaks benchmark website*. [Online]. Available: http://www.aifb.uni-karlsruhe.de/∼jbr/MovPeaks/

[16] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization", *In Proc. of the Congress on Evolutionary Computation*, pp. 84–88, 2000.

[17] R. W. Morrison, "Performance Measurement in Dynamic Environments", *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, Edited by J. Branke, Ed. 5-8, 2003.