Cooperative Particle Swarm Optimization in Dynamic Environments

Nikolas J. Unger
Department of Computer Science
Brock University
St. Catharines, Ontario L2S 3A1
Email: nu08ty@brocku.ca

Beatrice M. Ombuki-Berman
Department of Computer Science
Brock University
St. Catharines, Ontario L2S 3A1
Email: bombuki@brocku.ca

Andries P. Engelbrecht
Department of Computer Science
School of Information Technology
University of Pretoria
Pretoria 0028
Email: engel@cs.up.ac.za

***Abstract*—Most optimization algorithms are designed to solve static, unchanging problems. However, many real-world problems exhibit dynamic behavior. Particle swarm optimization (PSO) is a successful metaheuristic methodology which has been adapted for locating and tracking optima in dynamic environments. Recently, a powerful new class of PSO strategies using cooperative principles was shown to improve PSO performance in static environments. While there exist many PSO algorithms designed for dynamic optimization problems, only one cooperative PSO strategy has been introduced for this purpose, and it has only been studied under one type of dynamism. This study proposes a new cooperative PSO strategy designed for dynamic environments. The newly proposed algorithm is shown to achieve significantly lower error rates when compared to well-known algorithms across problems with varying dimensionalities, temporal change severities, and spatial change severities.**

I. Introduction

Optimization algorithms have historically focused on problems that do not change while agents interact with them. However, real-world problems often change as the algorithm searches for a solution. An algorithm applied to a dynamic optimization problem must not only locate optima, but also track them as they move. Algorithms must maintain diversity in order to respond to environmental changes, and minimize reliance on possibly invalidated state.

Particle swarm optimization (PSO) refers to a category of optimization algorithms that are inspired by swarming behavior in nature (e.g., the movement of flocks of birds or schools of fish). PSO algorithms can find good solutions for many problems [21], [26]. While the original PSO was mainly used in static environments, its core ideas have been adapted to dynamic optimization problems [12]. Well-known PSO strategies for use in dynamic environments include the re-evaluating PSO [20], the charged PSO [4], and the quantum PSO (QSO) [1].

Concepts from cooperative systems were recently applied to PSO in order to improve its performance in static environments [16]. In particular, Van den Bergh and Engelbrecht [32], [33] introduced the cooperative split PSO (CPSO-S_K) and the hybrid cooperative split PSO (CPSO-H_K) algorithms. These strategies subdivide a search space into multiple subspaces and then optimize them with swarms that work together. CPSO-S_K and CPSO-H_K have been shown to improve solution quality and computational complexity when compared to the vanilla PSO, particularly for high-dimensional problems [32], [33]. Recent extensions by Li and Yao support large-scale optimization of problems with up to 2000 variables [27]. An alternative approach to improve scalability using orthogonal learning was presented in [36].

Despite the improvements offered by the cooperative PSO strategies, only one previous attempt has been made to adapt them for use in dynamic environments. The resulting algorithm, known as the cooperative charged PSO (CCPSO), was introduced by Rakitianskaia and Engelbrecht [31]. By replacing the vanilla PSO in subswarms of CPSO-S_K with Blackwell and Benteley's charged PSO, CCPSO was shown to significantly improve charged PSO's performance [31]. However, the performance of the CCPSO has only been studied in one type of dynamic environment.

This paper proposes a new cooperative PSO strategy for use in dynamic environments. An empirical study comparing the performance of this new strategy and CCPSO to several popular algorithms is performed. Unlike many previous studies, this work includes comparisons across environments of varying dimensionality, temporal change severities, and spatial change severities. The performance of CCPSO is validated across these environments while the newly proposed variant is shown to be superior to the other approaches in general.

The remainder of this paper is organized as follows: Section II provides the necessary background information for the study, Section III surveys previous work in the field, Section IV proposes the new variant, Section V describes the experimental framework, and Section VI presents and discusses the results from the experiments. Finally, Section VII offers conclusions and future work.

II. Background

This section provides a brief overview of dynamic environments and the challenges they pose, particle swarm optimization, and previous work in this field.

A. Dynamic Environments

In general, a dynamic environment (also called a nonstationary environment) is one whose properties change while a learning agent is acting [6], [9]. In contrast, a static environment is one that does not change during interaction. In the context of function optimization, a dynamic environment is one whose fitness landscape changes over time. There are many

978-1-4673-6004-3/13/$31.00

different ways in which changes can occur over time. Research in this field often presents new algorithms in the context of a single dynamic environment without considering performance if the environment changes in a different manner. Several attempts have been made to classify varieties of dynamic environments [10], [15], [19], [34].

Recently, Duhain [13] collected several definitions into a unified classification framework. Temporal severity and spatial severity of changes – how often changes occur and the degree of the changes, respectively – can be set as axes on a plane and visualized as in Figure 1. The quadrants of the grid in the visualization are defined as *quasi-static*, *progressive*, *abrupt*, and *chaotic*. An absolute measurement for this classification is not possible [12], so one must be careful to specify environmental parameters precisely when testing algorithms.

B. Dynamic Environment Benchmarks

While real-world problems can be used to evaluate the performance of new algorithms, synthetic benchmarks offer researchers a greater degree of control over the experiment. Additionally, properties of synthetic benchmarks are often fully known, allowing performance metrics such as error measurements to be used.

Branke [5] introduced the moving peaks benchmark (MPB) function. MPB is extensively used in the literature [2], [3], [12], among others. MPB constructs landscapes out of peaks with user-defined parameters controlling spatial severity, temporal severity, and the strength of patterns in the movement. Morrison and De Jong introduced a peak-based benchmark similar to the MPB, called DF1 [29]. Other dynamic environment benchmarks include static functions with time-varying offsets, the generalized dynamic benchmark generator [24], and the CEC'09 benchmark functions [25].

C. Vanilla PSO

The original PSO was proposed by Kennedy and Eberhart [21]. PSO is a biologically inspired population-based iterative metaheuristic which has been used in various applications including neural network training [21], function optimization [14], manufacturing [14] and power systems [11].

The vanilla PSO [21] consists of a *swarm* of entities known as *particles*. The particles search through an n-dimensional search space. Each particle stores a position in the search space (i.e., a candidate solution). The function determines the quality of candidate solutions. Each particle also stores a velocity vector and a personal best location (*pbest*) – the best candidate solution encountered by the particle so far along with its function value. The swarm consists of s particles, and it maintains the overall best solution found, the *gbest*, and its function value. Algorithm 2 outlines the asynchronous vanilla PSO.

The velocity update in line 4 is defined for particle i by

$$\mathbf{v}_i = w\mathbf{v}_i + c_1\mathbf{r_1}(\mathbf{y}_i - \mathbf{x}_i) + c_2\mathbf{r_2}(\hat{\mathbf{y}} - \mathbf{x}_i) \qquad (1)$$

where $\mathbf{v}_i$ is the velocity for particle i, $\mathbf{x}_i$ is its position in the search space, $\mathbf{y}_i$ is its personal best location, and $\hat{\mathbf{y}}$ is the swarm's best location. w, c_1, and c_2 – the *inertia weight*, *cognitive coefficient*, and *social coefficient* respectively – are

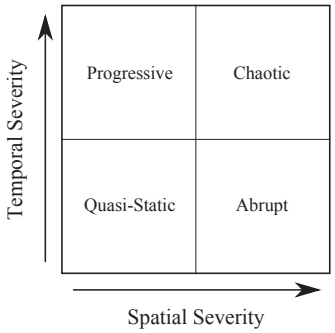

Figure 1. Dynamic environments by change type

all parameters of the algorithm. $\mathbf{r_1}$ and $\mathbf{r_2}$ are random vectors where each component is uniformly drawn from the range 0 to 1, inclusively. Work done by Clerc [8] provides theoretically derived guidance on the selection of w, c_1, and c_2 through the use of a *constriction factor*. w, c_1, and c_2 are typically set to 0.729, 1.494, and 1.494 respectively in order to satisfy Clerc's formula [7], [14]. The position update on line 5 is defined by

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i \qquad (2)$$

Initial particle positions are assigned randomly. One approach is to randomize the locations throughout the search space [14] and to set the initial velocities to zero [17]. Typical stopping criteria are an elapsed number of iterations or a desired fitness threshold.

Figure 2 summarizes the global best (gbest) PSO, where the social component attracts particles to the best position in the swarm. In other words, the neighborhood topology is the star topology (i.e. one position is the social attractor of all particles). There exist many alternative neighborhood topologies that can be used instead [14]. The Von Neumann topology introduced by Kennedy and Mendes [22] achieves consistently good performance. Figure 2 also uses *asynchronous* updates. If the best positions are updated as a group after the position updates, then the algorithm uses *synchronous* updates [6].

D. Cooperative PSO

Van den Bergh and Engelbrecht [32] introduced the cooperative split PSO (CPSO-S), which extends PSO using principles of cooperative systems. CPSO-S was developed as an approach to improve the scalability of the vanilla PSO.

CPSO-S moves individual components in beneficial directions without affecting the rest of the vector. It accomplishes this by splitting the n-dimensional search space into n 1-dimensional spaces, each of which is optimized independently by a *subswarm*. Each subswarm is a complete implementation of a PSO variant.

It is necessary to construct a mechanism to map 1-D positions to a complete candidate solution in order to evaluate particles in the subswarms. CPSO-S maintains a *context vector* – a vector with the same dimensionality as the original problem whose components are the global best positions from the corresponding subswarms. When a particle of a subswarm needs to be evaluated, the corresponding component of the context vector is temporarily replaced with the particle's partial

```
1: Randomly initialize s particles
2: while stopping criteria are not met do
3:     for each particle do
4:         Update velocity
5:         Update position
6:         Update pbest if new position is better
7:         Update gbest if new position is better
8:     end for
9: end while
10: Solution is the global best position
```

Figure 2. Vanilla PSO Algorithm

solution and the resultant vector is evaluated using the function. The context vector is initialized with random components from each subswarm.

The CPSO-S is summarized in Figure 3 [33]. Each subswarm P_j has the following parameters: $P_j.s$ is the number of particles in the subswarm, $P_j.\hat{\mathbf{y}}$ is the subswarm's best position, $P_j.\mathbf{x}_i$ is the position of the i^{th} particle in the subswarm, and $P_j.\mathbf{y}_i$ is the i^{th} particle's personal best position. The fitness function for the problem is f. Function b is meant to capture the notion of a context vector and is defined by

$$\mathbf{b}(j, \mathbf{z}) \equiv (P_1.\hat{\mathbf{y}},\ P_2.\hat{\mathbf{y}},\ ...,\ P_{j-1}.\hat{\mathbf{y}},\ \mathbf{z},\ P_{j+1}.\hat{\mathbf{y}},\ ...,\ P_n.\hat{\mathbf{y}})$$

The position updates referenced by line 12 in Figure 3 are defined by (1) and (2). It is important to note that CPSO-S requires significantly more function evaluations per iteration than vanilla PSO, since one evaluation will occur per iteration for each particle in each subswarm. In order to keep comparisons between algorithms consistent, the total number of function evaluations across all subswarms should be equal to the number of function evaluations of algorithms against which comparisons are made [32].

It was noted in [33] that CPSO-S is negatively affected when components of the solution are correlated; independent changes made by different subswarms have a detrimental effect. Figure 3 does not prevent the subswarms from having more than one dimension, so it is advantageous to place correlated variables in the same subswarm. However, in real-world problems correlations are not always known in advance. CPSO-S_K was designed to address these concerns. It is identical to CPSO-S except that its subswarms have more than one dimension. The n dimensions of the solution space are blindly split into k parts, each having $\frac{n}{k}$ dimensions, in the hope that correlated variables are grouped. Recently, Li and Yao [27] proposed variants which utilize more sophisticated grouping techniques.

A third cooperative PSO variant, CPSO-H_K, was also introduced in [33]. This hybrid algorithm was created in order to address the problem of *pseudo-optima* – non-optima in deceptive functions that can trap CPSO-S_K. Many additional approaches for combining PSO with cooperative systems have been proposed and analyzed by El-Abd in [16].

III. PSO for Dynamic Environments

This section gives a brief overview of some popular dynamic PSO variants that are found in the literature and that are relevant to this work.

```
1: Initialize n 1-D swarms: P_j, j ∈ [1, .., n]
2: while stopping criterea are not met do
3:     for each swarm j ∈ [1, .., n] do
4:         for each particle i ∈ [1, .., P_j.s] do
5:             if f(b(j, P_j.x_i)) < f(b(j, P_j.y_i)) then
6:                 P_j.y_i = P_j.x_i
7:             end if
8:             if then f(b(j, P_j.y_i)) < f(b(j, P_j.ŷ))
9:                 P_j.ŷ = P_j.y_i
10:            end if
11:        end for
12:        Perform PSO velocity and position updates on P_j
13:    end for
14: end while
```

Figure 3. CPSO-S Algorithm

A. Challenges for PSO

PSO-based algorithms experience two major issues in dynamic environments: outdated memory, and loss of diversity [2], [6]. If either is unaddressed, the algorithm will perform poorly after a sustained or severe environmental change.

Outdated memory occurs due to stored personal best and global best positions. When the environment changes, the superiority of these positions may become invalid, leading to undesirable particle attraction. Re-evaluating personal best and global best positions after an environmental change is a simple solution [2], [12]. To accomplish this re-evaluation, the algorithm must know when a change occurs. The personal bests could be re-evaluated during each iteration at the expense of extra fitness function calls. Alternatively, a change detection mechanism such as a sentry particle can be used. Sentry particles re-evaluate their position after each iteration to determine if the environment has changed beyond a given threshold [6]. Other change detection mechanisms are also available [20].

Diversity loss occurs as the swarm converges. When a change occurs, the swarm may not have enough diversity to locate and track the new optima [4]. One class of solutions reintroduces diversity after a change is detected, while another class maintains diversity at all times [2].

B. Re-evaluating PSO

Re-evaluating PSO (RPSO) is a simple strategy introduced by Hu and Eberhart in [20]. It is a vanilla PSO that re-evaluates all personal best values and the global best value when a change has occurred. A change detection mechanism can be used to limit the number of re-evaluations. This variant does not prevent diversity loss.

C. Charged PSO

Charged PSO [4] is inspired by principles of electromagnetism. In this variant of PSO, some proportion of the swarm is given a "charge." All charges are the same and thus charged particles repel each other (i.e., there is no notion of negative charges). In order to implement the repulsing behavior, the velocity update formula in (1) is ammended to

$$\mathbf{v}_i = w\mathbf{v}_i + c_1\mathbf{r_1}(\mathbf{y}_i - \mathbf{x}_i) + c_2\mathbf{r_2}(\hat{\mathbf{y}} - \mathbf{x}_i) + \sum_{j \neq i} \mathbf{a}_{ij} \quad (3)$$

The last term of (3) represents the particle's acceleration. $\mathbf{a}_{ij}$, the acceleration between particle i and particle j, is defined by

$$\mathbf{a}_{ij} = \begin{cases} \frac{Q_i Q_j}{{\mathbf{r}_{ij}}^3}(\mathbf{x}_i - \mathbf{x}_j) & \text{if } p_{core} < \mathbf{r}_{ij} \leq p \quad (4a) \\ 0 & \text{otherwise} \quad (4b) \end{cases}$$

where Q_i is the charge of a given particle and $\mathbf{r}_{ij}$ is the distance between the two particles given by

$$\mathbf{r}_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||$$

Neutral particles have $Q_i = 0$ whereas charged particles have $Q_i > 0$. p_{core} and p are parameters to the algorithm which prevent explosive acceleration and simplify computations, respectively. The magnitude of the acceleration may be clamped instead of introducing p_{core} [2]. An additional strategy to address outdated memory is required when implementing charged PSO.

D. Quantum PSO

Quantum PSO (QSO), introduced in [1], was designed to simplify charged PSO and remove its computational overhead. QSO has been shown to outperform charged PSO [1]. It is inspired by the quantum model of the atom. A proportion of the particles are selected to use alternative update mechanisms while the remainder behave normally. The quantum particles no longer use a velocity calculation; their positions are determined directly. The quantum cloud is distributed in an n-dimensional ball centered on the gbest position. The position calculation for these quantum particles is given by

$$\mathbf{x}_i = d(\hat{\mathbf{y}},\, r_{cloud}) \quad (5)$$

where d is function which returns a position within a cloud centered on the given position with a given radius according to some probability distribution, such as a Gaussian distribution, a uniform volume distribution, or a non-uniform volume distribution [3]. r_{cloud} is a tunable parameter [12]. Similarly to charged PSO, a strategy to address outdated memory is required.

E. Cooperative Charged PSO

The only previous attempt to adapt cooperative PSO to dynamic environments was performed by Rakitianskaia and Engelbrecht [31], where the cooperative charged PSO (CCPSO) was introduced. This algorithm is CPSO-S_K where each subswarm implements charged PSO from [4] rather than vanilla PSO.

IV. Proposed Cooperative Quantum PSO for Dynamic Environments

The cooperative quantum PSO (CQSO-S_K) is based on the cooperative principles of the cooperative split PSO (CPSO-S_K). This new strategy is designed to locate and track a single peak. Instead of a typical CPSO-S_K structure where each subswarm uses vanilla PSO, the subswarms instead use the QSO. The CQSO-S_K has the following control parameters in addition to that of a vanilla PSO: number of subswarms, K; quantum radius, r_{cloud}; number of neutral particles per subswarm, N; and number of quantum particles per subswarm, N^q. Note that s, the total swarm size, is equal to $K(N + N^q)$.

V. Experimental Setup

In order to evaluate the effectiveness of the proposed algorithm, an empirical study comparing it with several well-known algorithms from Section III was performed. The experiments were repeated in multiple different dynamic environments. These environments were selected to analyze sensitivity with respect to change frequency, change severity, and problem dimensionality. All experiments were performed using the Computational Intelligence Library [30]. Section V-A defines the test environments used in the study. Section V-B describes a performance metric designed for dynamic environments. Section V-C lists the control parameters for the algorithms used in the study. Finally, Section V-D describes the statistical tests used.

A. Environments

Four environment types – static, progressive, abrupt, and chaotic by the definitions in Section II-A – were evaluated. A static environment was included in order to determine if the addition of dynamic functionality impacts the ability of the algorithms to optimize unchanging problems. Each environment was instantiated with a single peak in 5, 10, 30, and 50 dimensions. Benchmark parameters were set according to the proposal in [12]. Table I lists the MPB parameters that define these environments. *width_severity*, *height_severity*, and s control the spatial severity of changes while Δe controls the temporal severity and λ controls movement patterns. The complete MPB definition is available in [5].

B. Performance Metric

When considering dynamic environments, specially designed performance metrics must be used [12]. Various metrics exist to measure independent aspects of performance [12], [35]. In this study, the *collective mean error* (CME) was calculated for all experiments. The CME is the mean error of the best candidate solution across the entire experiment. It is commonly used and incorporates several aspects of performance for an algorithm: stability, reactivity, detection capacity, tracking capacity, and accuracy. The CME will drop if any of these measures drop [12].

C. Parameters

For comparison purposes, the number of function evaluations was selected as a fair time measure [33]. All of the algorithms were given a fixed budget of function evaluations for each iteration, allowing the algorithm performance to be compared using iterations as the independent variable. For simplicity, all algorithms were configured to use the same number of particles performing the same number of function evaluations in each iteration. A relatively large number of particles (i.e., 200) was chosen in order to support the cooperative subswarms in high-dimensional environments – with 5 subswarms, 40 particles can be allocated for each subswarm.

All algorithms used Von Neumann topologies since it generally outperforms other strategies in static environments [22] and maintains higher swarm diversity in dynamic environments [26]. Velocity update parameters were set to $w = 0.729844$, and $c_1 = c_2 = 1.496180$ for all algorithms in accordance with Clerc's constriction factor [8]. All algorithms

Table I. TEST ENVIRONMENTS

	STATIC	PROGRESSIVE	ABRUPT	CHAOTIC
Peaks	1	1	1	1
Peak heights	$\in [30, 70]$	$\in [30, 70]$	$\in [30, 70]$	$\in [30, 70]$
Peak widths	$\in [1, 12]$	$\in [1, 12]$	$\in [1, 12]$	$\in [1, 12]$
height_severity	0	1	10	10
width_severity	0	0.05	0.05	0.05
s	0	1	50	50
λ	0	0	0	0
Δe	∞	1	200	5

Table II. ALGORITHM PARAMETERS

Parameter	RPSO	QSO	CCPSO	CQSO-S_K
s	200	200	200	200
K	-	-	5	5
d	-	UVD	-	UVD
r_{cloud}	-	5	-	5
Q	-	-	1	-
p	-	-	1000	-
p_{core}	-	-	1	-
N	-	180	20	20
N^q	-	20	-	20
N^+	-	-	20	-

Table III. CME RANKS

Algorithm	Score
CQSO-S_K	34
CCPSO	17
QSO	-17
RPSO	-34

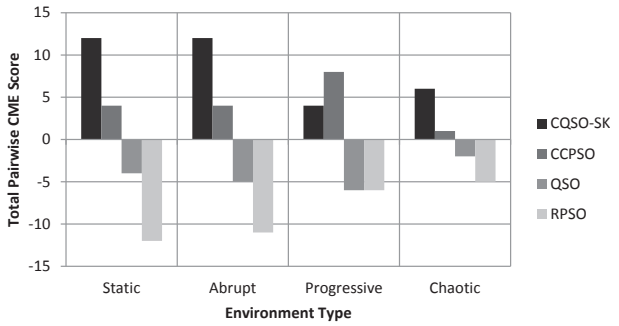

Figure 4. CME ranks by environment type

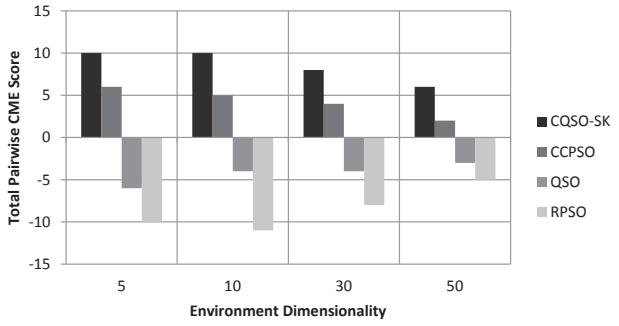

Figure 5. CME ranks by environment dimensionality

were permitted to re-initialize pbest values on every iteration to exclude the performance influence of change detection strategies. Algorithm-specific parameters are listed in Table II. Parameter settings for previously known algorithms were chosen based on recommendations from their authors as well as the empirical results from [12].

D. Statistical Methods

All experiments were executed 30 times and the Kruskal-Wallis test [23] was used to check for statistical significance in the difference of CME values. After concluding that there was a difference, pairwise Mann-Whitney-Wilcoxon rank sum tests [28] with Holm correction [18] were performed in order to determine which algorithms outperformed others. For each pair of algorithms where a statistically significant difference existed, the median CME values were compared to determine which algorithm performed best. An algorithm received a point when it outperformed another in this comparison, and lost a point if it was outperformed. At the end of the process, the algorithms were ranked based on their overall scores. All statistical tests were performed with a 95% significance level.

VI. EXPERIMENTAL RESULTS AND DISCUSSION

This section presents the results of the experiments comparing the performance of the algorithms. First, the impact of cooperation on PSO strategies for dynamic environments is examined. Next, the sensitivity of the algorithms to spatial and temporal change severities is tested. A similar sensitivity analysis is then performed with respect to problem dimensionality. Following these analyses, the error of the algorithms is reported. Finally, a behavioral analysis of a representative environment is performed to highlight the cause of improved performance. In all figures throughout this section, the proposed strategy is bolded.

A. Cooperative Versus Non-Cooperative

Table III lists the ranks resulting from the statistical process described in Section V-D. From the table, the proposed CQSO-S_K strategy clearly outperformed the other algorithms by a factor of 2 or more. However, this overall ranking is very coarse. In order to assess the exact types of problems in which each algorithm excels, the CME rankings were grouped by type of dynamism and dimensionality in Figures 4 and 5 respectively. The groups are ordered such that the total pairwise score decreases (i.e., relative performance becomes more uncertain) from left to right.

As depicted in Figures 4 and 5, the ranks of the leftmost two algorithms (i.e., the cooperative algorithms) significantly exceed the values of the rightmost two algorithms (i.e., the previously known non-cooperative algorithms) consistently across all groups. Rakitianskaia and Engelbrecht [31] showed that cooperation produced superior results for swarms in dynamic environments, but that study focused on only one type of spatial and temporal change severity. Figures 4 and 5 confirm this observation for a variety of environments. Both well-known algorithms (QSO and RPSO) consistently perform comparatively poorly. The following sections examine performance

differences between the algorithms as environments become more challenging and as the problem dimensionality increases.

B. Sensitivity to Spatial and Temporal Severity

Figure 4 provides additional insights when solely examining the cooperative algorithms. First, it is shown that the proposed CQSO-S_K outperforms all other strategies in static, abrupt, and chaotic environments in general. When considering only progressive environments, the CCPSO strategy was superior.

The progressive and chaotic environments were more difficult than static and abrupt environments for all algorithms, with chaotic environments being the most difficult. This observation is due to the increasing uncertainty in groups toward the right of the figure. Since it is expected that problem difficulty increases as either spatial or temporal severity increases, the observation that fewer significant differences in algorithm performance appear in environments with severe or frequent changes is consistent with expectations. Since the progressive and chaotic environments are the most difficult in general, it follows that an increased temporal severity is more challenging than an increased spatial severity.

C. Sensitivity to Dimensionality

When examining the results grouped by environment dimensionality in Figure 5, similar trends to those in Section VI-B can be observed. As expected, the statistical significance between CME values decreases as the dimensionality increases, suggesting that problem difficulty is growing. The superiority of the cooperative algorithms in all groups is clearly shown by the negative values for the QSO and RPSO algorithms. The proposed CQSO-S_K algorithm was significantly superior to all other algorithms in all problem dimensionalities.

D. Error Rates

Although Figures 4 and 5 give insight into the relative performance of the algorithms, they do not quantify the quality of the results. Table IV lists the mean CME across all simulations for each algorithm in each environment. Since the peak heights are in the range $[30, 70]$, it is possible – though extremely unlikely – to achieve CME of 30 in some environments through random guessing. For this reason, values of 30 or lower are indicative of successful optimization and thus are written in boldface. Hyphens in the "Score" column indicate that the Kruskal-Wallis detected no significant difference between algorithms. Where a significant difference was detected, the sum of absolute pairwise Mann-Whitney-Wilcoxon scores is listed in the "Score" column. Higher scores indicate that there were more statistically significant differences between algorithms.

Based on the data in Table IV, the cooperative algorithms perform very well in the 5-D environments. Only the chaotic environment presents a difficulty at this low dimensionality. The CQSO-S_K achieves a CME equal to roughly half the height of the smallest possible peak for this 5-D chaotic environment. In 10 dimensions, the cooperative algorithms continue to do well. The progressive environment becomes much more difficult than it was in 5-D. All algorithms perform much poorer on the chaotic environment in 10-D when compared to the other environment types. In 30 dimensions, the performance in the progressive environment grows much worse. At this dimensionality, there is no significant difference among the algorithms for the chaotic environment. However, the cooperative algorithms are still able to locate and track peaks in the static and abrupt environments. This trend continues for the 50-D problems. Only the static problem can be efficiently solved in 50-dimensions, and only by the cooperative algorithms. While the cooperative algorithms are able to achieve some acceptable results in the 50-D abrupt environment, the 50-D progressive and 50-D chaotic environments become far too difficult for any algorithm. The non-cooperative algorithms struggle across all environments with the exception of relatively simple ones: non-chaotic 5-D environments, static and abrupt 10-D environments, and 30-D static environments.

Table IV. CME BY ENVIRONMENT

	Score	**CQSO-S_K**	CCPSO	QSO	RPSO
5D Static	8	**0.05**	**0.11**	**1.18**	**1.57**
5D Abrupt	8	**0.87**	**1.23**	**6.67**	**11.81**
5D Progressive	8	**0.51**	**0.40**	**24.41**	**23.63**
5D Chaotic	8	**16.43**	**24.25**	47.28	53.74
10D Static	8	**0.54**	**0.72**	**2.70**	**3.59**
10D Abrupt	8	**2.86**	**3.54**	**15.13**	**21.19**
10D Progressive	8	**12.64**	**10.17**	41.53	42.20
10D Chaotic	6	48.02	45.89	47.71	44.64
30D Static	8	**4.71**	**5.28**	**15.41**	**21.90**
30D Abrupt	8	**17.52**	**19.50**	48.71	52.40
30D Progressive	8	47.74	48.00	54.09	52.47
30D Chaotic	-	51.18	51.54	49.16	47.43
50D Static	8	**8.55**	**10.25**	30.93	44.09
50D Abrupt	8	31.52	35.52	56.50	54.88
50D Progressive	-	52.78	54.01	54.79	54.34
50D Chaotic	-	50.58	54.58	49.80	50.59
Total	**-**	**10 / 16**	**10 / 16**	**6 / 16**	**6 / 16**

E. Behavioral Analysis

In order to gain insight into the operation of the algorithms in the various environment types, plots were produced of the mean error, rather than the mean CME, across all simulations for the four 10-D problems. Figure 6 plots the error for 10-D static environment, while Figures 7, 8, and 9 plot the error for the 10-D abrupt, 10-D progressive, and 10-D chaotic environments respectively.

The static and abrupt environments have very similar profiles in Figures 6 and 7 – the only difference is the sawtooth pattern created in the abrupt environment due to changes in the landscape. From this pattern, algorithms which perform well in abrupt environments can be expected to perform similarly in static environments since static environments effectively represent the first peak in the sawtooth pattern. The variations designed to enable use in dynamic environments (e.g., quantum particles) do not impact the algorithms' usability for these simpler problems. The cooperative algorithms both converge on the peak very rapidly while the non-cooperative QSO and RPSO take a longer time to do so. Another pattern in the profile of error in abrupt environments is the increasingly large mean error after each subsequent environmental change. A

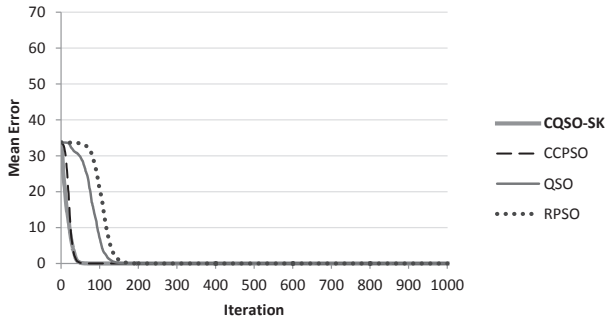

Figure 6. Mean error in 10-D static environment

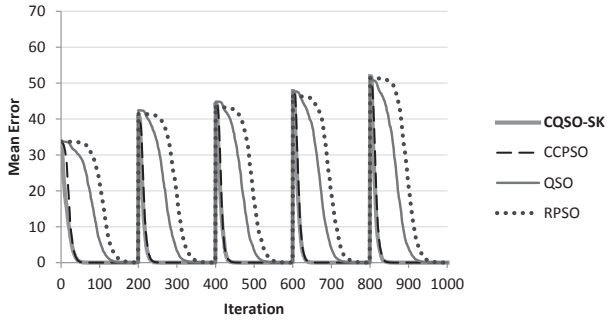

Figure 7. Mean error in 10-D abrupt environment

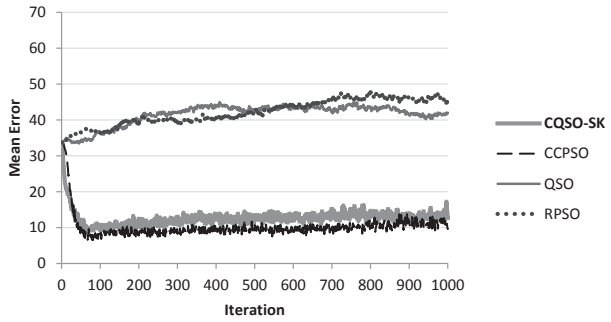

Figure 8. Mean error in 10-D progressive environment

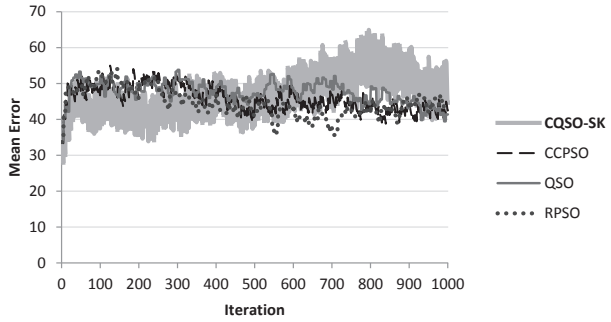

Figure 9. Mean error in 10-D chaotic environment

deteriorating stability measure caused by swarm convergence explains this observation. With the exception of RPSO, all algorithms have a mechanism which maintains a minimum diversity level. This decrease in stability is expected to halt after sufficient iterations have elapsed due to the minimum diversity level imparted by charged and quantum particles. The profile in Figure 7 shows that these negative stability impacts nearly stop growing by the fourth environmental change.

The progressive and chaotic environments also form similar profiles to each other in Figures 8 and 9. For the 10-D progressive environment, the cooperative algorithms quickly locate the peak and successfully track it while maintaining a moderate error level. QSO and RPSO maintain a relatively large error without the initial drop-off expected of an algorithm that is climbing a peak, suggesting that they never seem to locate the peak. The 10-D chaotic environment continues to prove challenging – all of the algorithms perform similarly poorly for this environment.

VII. Conclusions and Future Work

Through this study, cooperation has been demonstrated to be a powerful component in the creation of particle swarm optimizers for a variety of dynamic environments. A consistent result throughout the experiments was the superiority of cooperative algorithms in terms of CME. Both cooperative strategies exhibited much faster convergence speeds than the non-cooperative strategies. The proposed CQSO-S_K strategy was superior to the other three variants in all but the 5- and 10-dimensional progressive environments, where CCPSO performed the best. Despite being outperformed by the proposed CQSO-S_K, the CCPSO was also shown to be superior to the well-known strategies across a range of dynamism classes. For abrupt environments, both cooperative algorithms were able to perform well even in 30- and 50-dimensional uni-modal environments. In addition, environments with high temporal change severity were more challenging for all algorithms than those with high spatial change severity. As a corollary to these observations, the importance of testing new algorithms in environments exhibiting varying types of dynamism is emphasized.

There are many opportunities for future work in this area. The effect of parameter selection for the QSO and charged PSO subswarms of CPSO-S_K is still not well-understood. Variants designed for dynamic environments using the hybrid approach of CPSO-H_K could be investigated. Cooperative principles could also be applied to multi-modal dynamic problems. Finally, while these experiments are consistent with functions most commonly used in the dynamic PSO literature, our goal is to test these algorithms on more difficult functions. Benchmarks containing more peaks, complex peak functions, or deceptive functions would be excellent candidates for analysis; the impact of pseudo-optima in dynamic environments has not yet been investigated. Despite these unknowns, these early indications show that dynamic cooperative PSO algorithms hold great promise.

Acknowledgment

We thank the anonymous reviewers for their valuable insight and comments.

REFERENCES

[1] T. Blackwell and J. Branke, "Multi-swarm optimization in dynamic environments," *Applications of Evolutionary Computing*, pp. 489–500, 2004.

[2] ——, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 4, pp. 459–472, aug 2006.

[3] T. Blackwell, J. Branke, and X. Li, "Particle swarms for dynamic optimization problems," in *Swarm Intelligence*, ser. Natural Computing Series, C. Blum and D. Merkle, Eds. Springer Berlin Heidelberg, 2008, pp. 193–217.

[4] T. Blackwell and P. Bentley, "Dynamic search with charged swarms," in *Proceedings of the genetic and evolutionary computation conference*. Citeseer, 2002, pp. 19–26.

[5] J. Branke. (1999, dec) The moving peaks benchmark. [Online]. Available: http://people.aifb.kit.edu/jbr/MovPeaks/movpeaks/

[6] A. Carlisle, "Applying the particle swarm optimizer to non-stationary environments," Ph.D. dissertation, Auburn, AL, USA, 2002, aAI3070763.

[7] A. Carlisle and G. Dozier, "An off-the-shelf pso," in *Proceedings of the workshop on particle swarm optimization*, vol. 1. Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI, 2001.

[8] M. Clerc, "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3, 1999, pp. 1951–1957.

[9] H. Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," DTIC Document, Tech. Rep., 1990.

[10] K. De Jong, "Evolving in a changing world," *Foundations of Intelligent Systems*, pp. 512–519, 1999.

[11] Y. del Valle, G. Venayagamoorthy, S. Mohagheghi, J. Hernandez, and R. Harley, "Particle swarm optimization: Basic concepts, variants and applications in power systems," *Evolutionary Computation, IEEE Transactions on*, vol. 12, no. 2, pp. 171–195, apr 2008.

[12] J. Duhain, "Particle swarm optimization in dynamically changing environment - an empircal study," Master's thesis, University of Pretoria, Pretoria, 2011.

[13] J. Duhain and A. Engelbrecht, "Towards a more complete classification system for dynamically changing environments," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*. IEEE, 2012, pp. 1–8.

[14] R. Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, 2001, pp. 81–86.

[15] ——, "Tracking and optimizing dynamic systems with particle swarms," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1. IEEE, 2001, pp. 94–100.

[16] M. El-Abd, "Cooperative models of particle swarm optimizers," Ph.D. dissertation, University of Waterloo, 2008.

[17] A. Engelbrecht, "Particle swarm optimization: Velocity initialization," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*. IEEE, 2012, pp. 1–8.

[18] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian journal of statistics*, pp. 65–70, 1979.

[19] X. Hu and R. Eberhart, "Tracking dynamic systems with pso: where's the cheese," in *Proceedings of the workshop on particle swarm optimization*, 2001, pp. 80–83.

[20] ——, "Adaptive particle swarm optimization: detection and response to dynamic systems," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 2, 2002, pp. 1666–1670.

[21] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, 1995, pp. 1942–1948.

[22] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 2, 2002, pp. 1671–1676.

[23] W. Kruskal and W. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, pp. 583–621, 1952.

[24] C. Li and S. Yang, "A generalized approach to construct benchmark problems for dynamic optimization," *Simulated Evolution and Learning*, pp. 391–400, 2008.

[25] C. Li, S. Yang, T. Nguyen, E. Yu, X. Yao, Y. Jin, H. Beyer, and P. Suganthan, "Benchmark generator for cec'2009 competition on dynamic optimization," *Dept. Comput. Sci., Univ. Leicester, Leicester, UK, Tech. Rep*, 2008.

[26] X. Li and K. Dam, "Comparing particle swarms for tracking extrema in dynamic environments," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, vol. 3, dec 2003, pp. 1772–1779.

[27] X. Li and Y. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 2, pp. 210–224, apr 2012.

[28] H. Mann and D. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[29] R. Morrison and K. De Jong, "A test problem generator for non-stationary environments," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3, 1999.

[30] G. Pampara, A. Engelbrecht, and T. Cloete, "Cilib: A collaborative framework for computational intelligence algorithms - part i," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, jun 2008, pp. 1750–1757.

[31] A. Rakitianskaia and A. Engelbrecht, "Cooperative charged particle swarm optimiser," in *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, jun 2008, pp. 933–939.

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

[33] ——, "A cooperative approach to particle swarm optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 3, pp. 225–239, jun 2004.

[34] K. Weicker, "An analysis of dynamic severity and population size," in *Parallel Problem Solving from Nature PPSN VI*. Springer, 2000, pp. 159–168.

[35] ——, "Performance measures for dynamic environments," *Parallel Problem Solving from Natureï£¡PPSN VII*, pp. 64–73, 2002.

[36] Z.-H. Zhan, J. Zhang, Y. Li, and Y.-H. Shi, "Orthogonal learning particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 6, pp. 832–847, dec 2011.