A Cooperative Approach to Particle Swarm Optimization

Frans van den Bergh and Andries P. Engelbrecht, *Member, IEEE*

***Abstract*—The particle swarm optimizer (PSO) is a stochastic, population-based optimization technique that can be applied to a wide range of problems, including neural network training. This paper presents a variation on the traditional PSO algorithm, called the cooperative particle swarm optimizer, or CPSO, employing cooperative behavior to significantly improve the performance of the original algorithm. This is achieved by using multiple swarms to optimize different components of the solution vector cooperatively. Application of the new PSO algorithm on several benchmark optimization problems shows a marked improvement in performance over the traditional PSO.**

***Index Terms*—Convergence behavior, cooperative coevolutionary genetic algorithm, cooperative learning, cooperative swarms, particle swarm optimization.**

I. Introduction

MOST stochastic optimization algorithms [including particle swarm optimizers (PSOs) and genetic algorithms (GAs)] suffer from the "curse of dimensionality," which simply put, implies that their performance deteriorates as the dimensionality of the search space increases. Consider a basic stochastic global search algorithm (as defined by Solis and Wets [1]) that generates a sample for a uniform distribution that covers the entire search space. The algorithm stops when it generates a solution that falls in the *optimality region*, a small volume of the search space surrounding the global optimum. The probability of generating a sample inside the optimality region is simply the volume of the optimality region divided by the volume of the search space. This probability will decrease exponentially as the dimensionality of the search space increases. Given this explanation, it is clear that it is typically significantly harder to find the global optimum of a high-dimensional problem, compared with a low-dimensional problem with similar topology. One way to overcome this exponential increase in difficulty is to partition the search space into lower dimensional subspaces, as long as the optimization algorithm can guarantee that it will be able to search every possible region of the search space.

GAs [2] are part of the larger family of evolutionary algorithms [3]. GAs maintain a population of potential solutions to some optimization problem, generating new solutions during each iteration using a variety of recombination, selection, and mutation operators. Due to their stochastic nature they are also sensitive to an exponential increase in the volume of the search space. Potter suggested that the search space should be partitioned by splitting the solution vectors into smaller vectors [4]. Each of these smaller search spaces is then searched by a separate GA; the fitness function is evaluated by combining solutions found by each of the GAs representing the smaller subspaces. Potter found that this decomposition lead to a significant improvement in performance over the basic GA. Potter did not, however, investigate in detail the possibility that the partitioning could lead to the introduction of pseudominima, that is, minima that were created as a side effect of the partitioning of the search space. It was also realized that the performance of the cooperative coevolutionary genetic algorithm (CCGA) of Potter deteriorates when there exists a dependence among parameters. Ong *et al.* extended Potter's CCGA to work with correlated parameters using surrogate models [5].

This paper applies Potter's technique to the PSO, resulting in two new cooperative PSO models, namely CPSO-S_K and CPSO-H_K. The CPSO-S_K model is a direct application of Potter's CCGA model to the standard PSO, while the CPSO-H_K model combines the standard PSO with the CPSO-S_K model. The performance of these new PSO variants is compared with that of Potter's CCGA, as well as the traditional PSO. A discussion of the existence of pseudominima is presented here, as well as a proposed algorithm for avoiding these pseudominima in a provably correct way.

Section II presents an overview of the PSO, as well as a discussion of previous attempts to improve its performance. This is followed in Sections III and IV by new cooperative implementations of the PSO algorithm. Section V describes the problems used to evaluate the new algorithms, of which the results can be found in Section VI. Finally, some directions for future research are discussed in Section VII.

II. Particle Swarm Optimizers (PSOs)

The PSO, first introduced by Kennedy and Eberhart [6], [7], is a stochastic optimization technique that can be likened to the behavior of a flock of birds or the sociological behavior of a group of people. They have been used to solve a range of optimization problems, including neural network training [8]–[10] and function minimization [11], [12]. Several attempts have been made to improve the performance of the original PSO, some of which are discussed in this section.

Manuscript received July 12, 2002; revised October 20, 2003.

F. van den Bergh was with the Department of Computer Science, School of Information Technology, University of Pretoria, Pretoria 0002, South Africa. He is now with Rapid Mobile, Pretoria 0020, South Africa (e-mail: frans@rapidm.com).

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

Digital Object Identifier 10.1109/TEVC.2004.826069

A. PSO Operation

The PSO is a population based optimization technique, where the population is called a *swarm*. A simple explanation of the PSO's operation is as follows. Each particle represents a possible solution to the optimization task at hand. For the remainder of this paper, reference will be made to unconstrained minimization problems. During each iteration each particle accelerates in the direction of its own personal best solution found so far, as well as in the direction of the global best position discovered so far by any of the particles in the swarm. This means that if a particle discovers a promising new solution, all the other particles will move closer to it, exploring the region more thoroughly in the process.

Let s denote the swarm size. Each individual $1 \leq i \leq s$ has the following attributes. A current position in the search space $\mathbf{x}_i$, a current velocity $\mathbf{v}_i$, and a personal best position in the search space $\mathbf{y}_i$. During each iteration, each particle in the swarm is updated using (1) and (2). Assuming that the function f is to be minimized, that the swarm consists of n particles, and that $r_1 \sim U(0,1), r_2 \sim U(0,1)$ are elements from two uniform random sequences in the range (0,1), then

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,i}(t)\left[y_{i,j}(t) - x_{i,j}(t)\right] + c_2 r_{2,i}(t)\left[\hat{y}_j(t) - x_{i,j}(t)\right] \quad (1)$$

for all $j \in 1 \ldots n$, thus, $v_{i,j}$ is the velocity of the jth dimension of the ith particle, and c_1 and c_2 denote the *acceleration coefficients*. The new position of a particle is calculated using

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1). \quad (2)$$

The personal best position of each particle is updated using

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t), & \text{if } f\left(\mathbf{x}_i(t+1)\right) \geq f\left(\mathbf{y}_i(t)\right) \\ \mathbf{x}_i(t+1), & \text{if } f\left(\mathbf{x}_i(t+1)\right) < f(\mathbf{y}_i)(t) \end{cases} \quad (3)$$

and the global best position found by any particle during all previous steps, $\hat{\mathbf{y}}$, is defined as

$$\hat{\mathbf{y}}(t+1) = \arg\min_{\mathbf{y}_i} f\left(\mathbf{y}_i(t+1)\right), \quad 1 \leq i \leq s. \quad (4)$$

The value of each component in every $\mathbf{v}_i$ vector can be clamped to the range $[-v_{\max}, v_{\max}]$ to reduce the likelihood of particles leaving the search space. The value of $v_{\max}$ is usually chosen to be $k \times x_{\max}$, with $0.1 \leq k \leq 1.0$ [7]. Note that this does not restrict the values of $\mathbf{x}_i$ to the range $[-v_{\max}, v_{\max}]$; it only limits the maximum distance that a particle will move during one iteration.

The variable w in (1) is called the *inertia weight*; this value is typically setup to vary linearly from 1 to near 0 during the course of a training run. Note that this is reminiscent of the temperature adjustment schedule found in Simulated Annealing algorithms. The inertia weight is also similar to the momentum term in a gradient descent neural network training algorithm.

Acceleration coefficients c_1 and c_2 also control how far a particle will move in a single iteration. Typically, these are both set to a value of 2.0 [7], although assigning different values to c_1 and c_2 sometimes leads to improved performance [13].

Recently, work by Clerc [14]–[16] indicated that a *constriction factor* may help to ensure convergence. Application of the constriction factor results in (5). Note that explicit reference to the time step t will be omitted from now on for notational convenience

$$v_{i,j} := \chi\left[v_{i,j} + c_1 r_1(y_{i,j} - x_{i,j}) + c_2 r_2(\hat{y}_j - x_{i,j})\right] \quad (5)$$

where

$$\chi = \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|} \quad (6)$$

and $\phi = c_1 + c_2$, $\phi > 4$.

B. Improved PSOs

Since the introduction of the PSO algorithm, several improvements have been suggested, many of which have been incorporated into the equations shown in Section II-A. The original PSO did not have an inertia weight; this improvement was introduced by Shi and Eberhart [12]. The addition of the inertia weight results in faster convergence.

Although it was originally suggested that the constriction factor, as shown in (5) and (6) above, should replace the $v_{\max}$ clamping, Eberhart and Shi [17] have shown that the constriction factor alone does not necessarily result in the best performance. Combining the two approaches results in the fastest convergence overall, according to Eberhart and Shi [17]. These improvements appear to be effective on a large collection of problems.

An entirely different approach to improving PSO performance was taken by Angeline [18]. The objective was to introduce a form of selection so that the properties that make some solutions superior are transferred directly to some of the less effective particles. Angeline used a tournament selection process based on the particles' current fitness, copying the current positions and velocities of the better half of the population onto the worse half, *without* changing the "personal best" values of any of the particles in this step. This technique improved the performance of the PSO in three of the four functions tested (all but the Griewank function, see Section V for a definition of this function).

There exists another general form of particle swarm, referred to as the LBEST method in [7]. This approach divides the swarm into multiple "neighborhoods," where each neighborhood maintains its own local best solution. This approach is less prone to becoming trapped in local minima, but typically has slower convergence. Kennedy has taken this LBEST version of the particle swarm and applied to it a technique referred to as "social stereotyping" [19]. A clustering algorithm is used to group individual particles into "stereotypical groups." The cluster center $\mathbf{g}_j$ is computed for each group j and then substituted into (1), yielding three strategies to calculate the new velocity

$$\mathbf{v}_i := w\mathbf{v}_i + c_1 r_1(\mathbf{g}_j - \mathbf{x}_i) + c_2 r_2(\hat{\mathbf{y}} - \mathbf{x}_i) \quad (7)$$

$$\mathbf{v}_i := w\mathbf{v}_i + c_1 r_1(\mathbf{y}_i - \mathbf{x}_i) + c_2 r_2(\mathbf{g}_j - \mathbf{x}_i) \quad (8)$$

$$\mathbf{v}_i := w\mathbf{v}_i + c_1 r_1(\mathbf{g}_j - \mathbf{x}_i) + c_2 r_2(\mathbf{g}_j - \mathbf{x}_i). \quad (9)$$

The results presented in [19] indicate that only the method in (7) performs better than the standard PSO. This improvement comes at increased processing cost as the clustering algorithm needs a nonnegligible amount of time to form the stereotypical groups.

More recently, Kennedy investigated other neighborhood topologies, finding that the von Neumann topology resulted in superior performance [20]. Suganthan investigated the use of spatial topologies, as opposed to topologies based on particle indices [13].

III. Cooperative Learning

In a GA [2], [21] population, each individual aims to produce the best solution by combining (hopefully) desirable genetic or behavioral properties from other individuals. There is competition among the individual members of the population, as the most fit individual is rewarded with more opportunities to reproduce. In this scenario each individual represents a complete solution vector, encoded in the appropriate format for the GA operations.

It is also possible to view a GA as a cooperative learner [22]. Clearwater *et al.* [23] define cooperation as follows: "*Cooperation involves a collection of agents that interact by communicating information to each other while solving a problem*." They further state that "*The information exchanged between agents may be incorrect, and should sometimes alter the behavior of the agent receiving it*." Clearly, by viewing the population members of a GA as agents, and the crossover operation as information exchange, the GA can be considered to be a cooperative system.

Another form of cooperation, as used by Clearwater *et al.* [23], is the use of a "blackboard." This device is a shared memory where agents can post hints, or read hints from. An agent can combine the hints read from the blackboard with its own knowledge to produce a better partial solution, or hint, that may lead to the solution more quickly than the agent would have been able to discover on its own.

Although competition among individual humans usually improves their performance, much greater improvements can be obtained through *cooperation*. This idea has been implemented in the context of GAs by Potter and De Jong [4]. Instead of using a single GA to optimize the whole solution vector in one population, the vector is split into its constituent components and assigned to multiple GA populations. In this configuration, each population is then optimizing a single component (genetic or behavioral trait) of the solution vector—a one-dimensional (1-D) optimization problem.

To produce a solution vector for the function being minimized, all the populations have to cooperate, as a valid solution vector can only be formed by using information from all the populations. This means that on top of the inherent cooperation in the population itself, a new layer of cooperation between populations has been added.

A. Cooperative Swarms

The same concept can easily be applied to PSOs, creating a family of CPSOs. Instead of having one swarm (of s particles) trying to find the optimal n-dimensional vector, the vector is split into its components so that n swarms (of s particles each) are optimizing a 1-D vector. Keep in mind that the function being optimized still requires an n-dimensional vector to evaluate. This introduces the following problems.

```
Create and initialise an n-dimensional PSO : P
repeat:
   for each particle i ∈ [1..s] :
      if f(P.x_i) < f(P.y_i)
         then P.y_i = P.x_i
      if f(P.y_i) < f(P.ŷ)
         then P.ŷ = P.y_i
   endfor
   Perform PSO updates on P using eqns. (1–2)
until stopping criterion is met
```

Fig. 1. Pseudocode for the PSO algorithm.

- Selection: The solution vector is split into n parts, each part being optimized by a swarm with m particles. This allows for $m \times n$ combinations for constructing the composite n-component vector. The simplest approach is to select the best particle from each swarm (how to calculate which particle is best will be discussed later). Note that this might not be the optimal choice; it could lead to undersampling and "greedy" behavior.
- Credit assignment: The solution to the credit assignment problem is the answer to the question: "To what degree is each individual component responsible for the overall quality of the solution?" In terms of swarms, how much credit should each swarm be awarded when the combined vector (built from all the swarms) results in a better solution? One simple solution is to give all swarms an equal amount of credit. If this problem is not addressed properly by the optimization algorithm, then the algorithm could spend too much time optimizing variables that have little effect on the overall solution.

Possible solutions to these problems are presented in Section III-C.

The main difference between the CPSO and the cooperative GA of Potter and De Jong [4] is that the optimization process of a PSO is driven by the social interaction [effected through the use of both the cognitive and social terms in (1)] of the individuals within that swarm; no exchange of genetic information takes place. In contrast, the cooperative GA is driven by changes in genetic or behavioral traits within individuals of the populations.

B. Two Steps Forward, One Step Back

Before looking at cooperative swarms in depth, let us first consider the weakness of the standard PSO. Fig. 1 lists the pseudocode algorithm for the standard PSO. The following naming convention applies to Fig. 1. For a particle i in a swarm P : $P.\mathbf{x}_i$, $P.\mathbf{v}_i$, $P.\mathbf{y}_i$ corresponds to the position, velocity, and personal best position, respectively, as defined in (1)–(4). The global best particle of the swarm is represented by the symbol $P.\hat{\mathbf{y}}$. The objective function f remains unchanged. This algorithm will be referred to as the standard PSO in this article.

As can be seen from Fig. 1, each particle represents a complete vector that can be used as a potential solution. Each update step is also performed on a full n-dimensional vector. This allows for the possibility that some components in the vector have moved closer to the solution, while others actually moved *away* from the solution. As long as the effect of the improvement outweighs the effect of the components that deteriorated,

the standard PSO will consider the new vector an overall improvement, even though some components of the vector may have moved further from the solution.

A simple example to illustrate this concept follows. Consider a three-dimensional vector $\mathbf{x}$, and the error function $f(\mathbf{x}) = ||\mathbf{x} - \mathbf{a}||^2$, where $\mathbf{a} = (20, 20, 20)$. This implies that the global minimizer of the function $\mathbf{x}^*$, is equal to $\mathbf{a}$. Now, consider a particle swarm containing, among others, a vector $\mathbf{x}_2$, and the global best position $\hat{\mathbf{y}}$. If t represents the current time step, then, with a high probability

$$||\mathbf{x}_2(t+1) - \hat{\mathbf{y}}(t+1)|| < ||\mathbf{x}_2(t) - \hat{\mathbf{y}}(t)||$$

if it is assumed that $\hat{\mathbf{y}}$ does not change during this specific iteration. That is, in the next time step $t+1$, particle 2 (represented by $\mathbf{x}_2$) will be drawn closer to $\hat{\mathbf{y}}$, as stipulated by the PSO update equations.

Assume that the following holds:

$$\hat{\mathbf{y}}(t) = (17, 2, 17)$$
$$\mathbf{x}_2(t) = (5, 20, 5).$$

Application of the function f to these points shows that $f(\hat{\mathbf{y}}(t)) = 342$ and $f(\mathbf{x}_2(t)) = 450$. In the next epoch, the vector $\mathbf{x}_2$ will be drawn closer to $\hat{\mathbf{y}}$, so that the following configuration may result:

$$\hat{\mathbf{y}}(t+1) = (17, 2, 17)$$
$$\mathbf{x}_2(t+1) = (15, 5, 15).$$

Note that the actual values of the components of $\mathbf{x}_2(t+1)$ depend on the stochastic influence present in the PSO update equations. The configuration above is certainly one possibility. This implies that $f(\mathbf{x}_2(t+1)) = 275$, even better than the function value of the global best position, which means that $\hat{\mathbf{y}}$ will be updated now. Although the fitness of the particle improved considerably, note that the second component of the vector has changed from the correct value of 20, to the rather poor value of 5; valuable information has, thus, been lost unknowingly. This example can clearly be extended to a general case involving an arbitrary number of components.

This undesirable behavior is a case of taking two steps forward, and one step back. It is caused by the fact that the error function is computed only after all the components in the vector have been updated to their new values. This means an improvement in two components (two steps forward) will overrule a potentially good value for a single component (one step back).

One way to overcome this problem is to evaluate the error function more frequently, for example, once for every time a component in the vector has been updated, resulting in much quicker feedback. A problem still remains with this approach: evaluation of the error function is only possible using a complete n-dimensional vector. Thus, after updating a specific component, $n-1$ values for the other components of the vector still have to be chosen. A method for doing just this is presented in the following section.

In the next section, a new PSO algorithm will be described. This algorithm can be mislead by a particular class of deceptive function (as shown below), however, Section IV presents another algorithm that addresses this weakness.

define
$\mathbf{b}(j, z) \equiv (P_1.\hat{\mathbf{y}}, P_2.\hat{\mathbf{y}}, \ldots, P_{j-1}.\hat{\mathbf{y}}, z, P_{j+1}.\hat{\mathbf{y}}, \ldots, P_n.\hat{\mathbf{y}})$
Create and initialise n one-dimensional PSOs : $P_j,\ j \in [1..n]$
repeat:
  for each swarm $j \in [1..n]$:
    for each particle $i \in [1..s]$:
      if $f(\mathbf{b}(j, P_j.\mathbf{x}_i)) < f(\mathbf{b}(j, P_j.\mathbf{y}_i))$
        then $P_j.\mathbf{y}_i = P_j.\mathbf{x}_i$
      if $f(\mathbf{b}(j, P_j.\mathbf{y}_i)) < f(\mathbf{b}(j, P_j.\hat{\mathbf{y}}))$
        then $P_j.\hat{\mathbf{y}} = P_j.\mathbf{y}_i$
    endfor
    Perform PSO updates on P_j using equations (1–2)
  endfor
until stopping condition is true

Fig. 2. Pseudocode for the CPSO-S algorithm.

C. CPSO-S_k Algorithm

The original PSO uses a population of n-dimensional vectors. These vectors can be partitioned into n swarms of 1-D vectors, each swarm representing a dimension of the original problem. Each swarm attempts to optimize a single component of the solution vector, essentially a 1-D optimization problem. This decomposition is analogous to the decomposition used in the relaxation method [24], [25].

One complication to this configuration is the fact that the function to be minimized, f, requires an n-dimensional vector as input. If each swarm represents only a single dimension of the search space, it is clearly not possible to directly compute the fitness of the individuals of a single population considered in isolation. A *context vector* is required to provide a suitable context in which the individuals of a population can be evaluated. The simplest scheme for constructing such a context vector is to take the global best particle from each of the n swarms and concatenating them to form such an n-dimensional vector. To calculate the fitness for all particles in swarm j, the other $n-1$ components in the context vector are kept constant (with their values set to the global best particles from the other $n-1$ swarms), while the jth component of the context vector is replaced in turn by each particle from the jth swarm.

Fig. 2 presents the CPSO-S algorithm, first introduced by Van den Bergh and Engelbrecht in [9], a PSO that splits the search space into exactly n subspaces. Extending the convention introduced in Fig. 1, $P_j.\mathbf{x}_i$ now refers to the position of particle i of swarm j, which can therefore be substituted into the jth component of the context vector when needed. Each of the n swarms now has a global best particle $P_j.\hat{\mathbf{y}}$. The function $\mathbf{b}(j, z)$ returns an n-dimensional vector formed by concatenating all the global best vectors across all swarms, except for the jth component, which is replaced with z, where z represents the position of any particle from swarm P_j.

This algorithm has the advantage that the error function f is evaluated after each component in the vector is updated, resulting in much finer-grained credit assignment. The current "best" context vector will be denoted $\mathbf{b}(1, P_1.\hat{\mathbf{y}})$. Note that $f(\mathbf{b}(1, P_1.\hat{\mathbf{y}}))$ is a strictly nonincreasing function, since it is composed of the global best particles $P_j.\hat{\mathbf{y}}$ of each of the swarms, which themselves are only updated when their fitness improves.

```
define
b(j, z) ≡ (P_1.ŷ, ..., P_{j-1}.ŷ, z, P_{j+1}.ŷ, ..., P_K.ŷ)
K_1 = n mod K
K_2 = K − (n mod K)
Initialise K_1 ⌈n/K⌉-dimensional PSOs:
P_j, j ∈ [1..K_1]
Initialise K_2 ⌊n/K⌋-dimensional PSOs:
P_j, j ∈ [(K_1 + 1)..K]
repeat:
   for each swarm j ∈ [1..K] :
      for each particle i ∈ [1..s] :
         if f(b(j, P_j.x_i)) < f(b(j, P_j.y_i))
            then P_j.y_i = P_j.x_i
         if f(b(j, P_j.y_i)) < f(b(j, P_j.ŷ))
            then P_j.ŷ = P_j.y_i
      endfor
      Perform PSO updates on P_j using (1–2)
   endfor
until stopping condition is true
```

Fig. 3. Pseudocode for the generic CPSO-S_K algorithm.

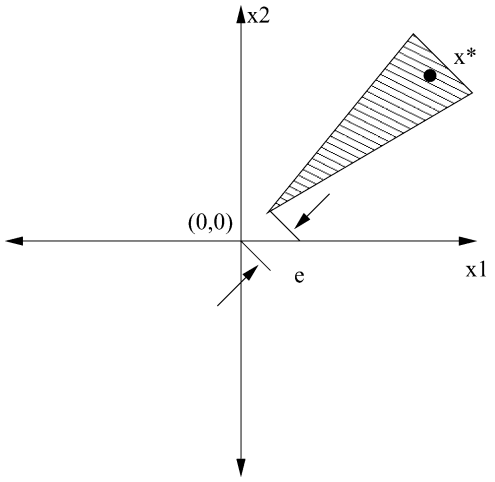

Fig. 4. Diagram illustrating the constrained suboptimality problem.

Each swarm in the group only has information regarding a specific component of the solution vector; the rest of the vector is provided by the other $n-1$ swarms. This promotes cooperation between the different swarms, since they all contribute to $\mathbf{b}$, the context vector. Another interpretation of the cooperative mechanism is possible. Each particle i of swarm j represents a different context in which the vector $\mathbf{b}(j,\cdot)$ is evaluated, so that the fitness of the context vector itself is measured in different contexts. The most successful context, corresponding to the particle i yielding the highest fitness, is retained for future use. For example, a 30-dimensional search space results in a CPSO-S algorithm with 30 1-D swarms. During one iteration of the algorithm, $30\times30 = 900$ different combinations are formed, compared with only 30 variations produced by the original PSO. The advantage of the CPSO-S approach is that only one component is modified at a time, yielding the desired fine-grained search, effectively preventing the "two steps forward, one step back" scenario. There is also a significant increase in the solution diversity in the CPSO-S algorithm, because of the many combinations that are formed using different members from different swarms.

Note that, should some of the components in the vector be correlated, they should be grouped in the same swarm (by using an arbitrarily configurable partitioning mechanism), since the independent changes made by the different swarms will have a detrimental effect on correlated variables. This results in some swarms having 1-D vectors and others having c-dimensional vectors $(c < n)$, something which is easily allowed in the framework presented above. Unfortunately, it is not always known in advance how the components will be related. A simple approximation would be to blindly take the variables c at a time, hoping that some correlated variables will end up in the same swarm. Fig. 3 presents the CPSO-S_K algorithm, where a vector is split into K parts. Note that the CPSO-S algorithm presented in Fig. 2 is really a special case of the CPSO-S_K algorithm with $K = n$. The number of parts K is also called the *split factor*.

There is no explicit restriction on the type of PSO algorithm that should be used in the CPSO-S_K algorithm. The guaranteed convergence PSO (GCPSO) [26] is a PSO variant that offers guaranteed convergence onto local minima. A discussion of this algorithm is outside of the scope of this article, but substituting the GCPSO for the PSO in the CPSO-S_K algorithm allows for the construction of a proof of guaranteed convergence for the CPSO-S_K algorithm too. This article will focus on the use of the standard PSO as preliminary approach to investigate the cooperative approach.

D. Convergence Behavior of the CPSO-S_K Algorithm

The CPSO-S_K algorithm is typically able to solve any problem that the standard PSO can solve. It is possible, however, for the algorithm to become trapped in a state where all the swarms are unable to discover better solutions, but the algorithm has not yet reached a local minimum. This is an example of *stagnation*, caused by the restriction that only one swarm is updated at a time, i.e., only one subspace is searched at a time.

An example function will now be presented to show a scenario in which the CPSO-S_K algorithm stagnates. The example will assume that a CPSO-S_2 algorithm is used to minimize the function. Fig. 4 illustrates in two dimensions the nature of the problem. The figure is a top-down view of the search space, with the shaded triangular area representing a region that contains f-values that are smaller than any other values in the search space. This region has a slope that runs downward from the point (0,0) to the point $\mathbf{x}^*$, the global minimizer. The symbol ε denotes the distance from the origin to the tip of the triangular region; ε can be made arbitrarily small so that the triangle touches the origin in the limit. To simplify the discussion, assume that the function has the form $f(\mathbf{x}) = \|\mathbf{x}\|^2$, except for the shaded triangular region, which contains points yielding negative f-values.

If the swarm P_1 (constrained to the subspace $x_1 \in \mathbb{R}$) reaches the state where $P_1.\hat{y} = 0$, the context vector $\mathbf{b}$ will be of the form $\mathbf{b}(2, P_2.x_i) = (0, P_2.x_i)$, so that $f(\mathbf{b}) = \|(0, P_2.x_i)\|^2 = (P_2.x_i)^2$. This function can easily be minimized by the second swarm P_2, which is constrained to the subspace $x_2 \in \mathbb{R}$. The second swarm will find the minimum located at $x_2 = 0$, so that the algorithm will terminate with a proposed solution of (0,0), which is clearly not the correct answer, since $\mathbf{x}^* \neq (0,0)$. Both

P_1 and P_2 have converged onto the local minimum of their respective subspaces. The problem is that the algorithm will find that 0 is in fact the local minimizer when only one dimension is considered at a time. The sequential nature of the algorithm, coupled with the property that $f(\mathbf{b}(1, P_1.\hat{y})$ is a strictly nonincreasing sequence, prevents the algorithm from temporarily taking an "uphill" step, which is required to solve this particular problem. Even if ε is made arbitrarily small, the algorithm will not be able to sample a point inside the shaded triangular area, since that would require the other swarm to have a global best position (i.e., $P_j.\hat{y}$) other than zero, which would require a step that would increase $f(\mathbf{b}(1, P_1.\hat{y}))$. What has happened here is that a local optimization problem in $\mathbb{R}^2$ has become a global optimization problem when considering the two subspaces x_1 and x_2 one at a time.

Note that the point (0,0) is not a local minimizer of the search space, although it is the concatenation of the individual minimizers of the subspaces x_1 and x_2. The fact that (0,0) is not a local minimizer can easily be verified by examining a small region around the point (0,0), which clearly contains points belonging to the shaded region as ε approaches zero. The term *pseudominimizer* will be used to describe a point in search space that is a local minimizer in all the predefined subspaces of $\mathbb{R}^n$, but not a local minimizer in $\mathbb{R}^n$ considered as a whole. This shows that the CPSO-S_K algorithm is not guaranteed to converge on the local minimizer, because there exists states from which it can become trapped in the pseudominimizer located at (0,0). Due to the stochastic components in the PSO algorithm, it is unlikely that the CPSO-S_K algorithm will become trapped in the pseudominimizer every time. The existence of a state that prevents the algorithm from reaching the minimizer destroys the guaranteed convergence property, though.

This type of function can be said to exhibit *deceptive* behavior [27], where good solutions, or even good directions of search, must be abandoned since they lead to suboptimal solutions. Deceptive functions have been studied extensively in the GA field, although it has been shown that many deceptive functions can be solved without difficulty with only minor changes to the basic GA [28].

In contrast to the CPSO-S_K algorithm, the normal PSO would not have the same problem. If the global best particle of the PSO algorithm is located at this pseudominimum position, i.e., $\hat{\mathbf{y}} = (0, 0)$, then the sample space from which the other particles could choose their next position could include a square with nonzero side lengths ρ, centred at (0,0). Since $\rho > 0$ per definition,[1] this square would always include points from the triangular shaded region in Fig. 4. This implies that the PSO will be able to move away from the point (0,0) toward the local minimizer in $\mathbb{R}^2$ located at $\mathbf{x}^*$.

There are several ways to augment the CPSO-S_K algorithm to prevent it from becoming trapped in such pseudominima. The original CCGA-1 algorithm, due to Potter [4], [29], suffers from the same problem, although Potter did not identify the problem as such. Potter suggested that each element of the population P_j should be evaluated in two contexts. He called this approach the CCGA-2 algorithm. One context is constructed using the best element from the other populations, similar to the CCGA-1 and CPSO-S_K algorithms. The second context is constructed using a randomly chosen element from each of the other populations. The individual under consideration receives the better of the two fitness values obtained in the two contexts. This approach is a compromise between the CCGA-1 approach and an exhaustive evaluation, where each element is evaluated against all other possible contexts that can be constructed from the current collection of populations. The exhaustive approach would require s^{K-1} function evaluations to determine the fitness of a single individual, where s is the population size, and K the number of populations. This rather large increase in the number of function evaluations would outweigh the advantage of using a cooperative approach.

The CCGA-2 approach has the disadvantage that the fitness of an individual is still only evaluated against a sample of possible values obtained from a search restricted to a subspace of the complete search space. In other words, it could still become trapped in a pseudominimizer, although this event is significantly less likely than for the CCGA-1 algorithm. The next section introduces a different solution that allows the CPSO-S_K algorithm to escape from pseudominima.

IV. Hybrid CPSOs

In the previous sectionn it was shown that the CPSO-S_K algorithm can become trapped in suboptimal locations in search space. This section introduces an algorithm that combines the CPSO-S_K algorithm with the PSO in an attempt to retain the best properties of both algorithms. The term "hybrid" has been used to describe at least three different PSO-based algorithms [9], [30], [31]. The algorithm presented here will be called the CPSO-H_K algorithm to resolve any ambiguities.

A. *CPSO*-H_K *Algorithm*

Given that the PSO has the ability to escape from pseudominimizers, and that the CPSO-S_K algorithm has faster convergence on certain functions (see Section VI), it would be ideal to have an algorithm that could exploit both of these properties. In principle, one could construct an algorithm that attempts to use a CPSO-S_K algorithm, but switches over to a PSO algorithm when it appears that the CPSO-S_K algorithm has become trapped. While this approach is a sound idea, it is difficult to design robust, general heuristics to decide when to switch between algorithms.

An alternative is to interleave the two algorithms, so that the CPSO-S_K algorithm is executed for one iteration, followed by one iteration of the PSO algorithm. Even more powerful algorithms can be constructed by exchanging information regarding the best solutions discovered so far by either component at the end of each iteration. This information exchange is then a form of cooperation between the CPSO-S_K component and the PSO component. Note that this is a form of "blackboard" cooperation, similar to the type described by Clearwater *et al.* [23].

A simple mechanism for implementing this information exchange is to replace some of the particles in one half of the algorithm with the best solution discovered so far by the other half of

[1]This is only guaranteed for the GCPSO, as discussed at the end of Section III-C.

define
$\mathbf{b}(j,\mathbf{z}) \equiv (P_1.\hat{\mathbf{y}},\ldots,P_{j-1}.\hat{\mathbf{y}},\mathbf{z},P_{j+1}.\hat{\mathbf{y}},\ldots,P_K.\hat{\mathbf{y}})$
$K_1 = n \bmod K$
$K_2 = K - (n \bmod K)$
Initialise K_1 $\lceil n/K \rceil$-dimensional PSOs:
$P_j,\ j \in [1..K_1]$
Initialise K_2 $\lfloor n/K \rfloor$-dimensional PSOs:
$P_j,\ j \in [(K_1+1)..K]$
Initialise an n-dimensional PSO : Q
repeat:
  for each swarm $j \in [1..K]$ **:**
    for each particle $i \in [1..s]$ **:**
      if $f(\mathbf{b}(j,P_j.\mathbf{x}_i)) < f(\mathbf{b}(j,P_j.\mathbf{y}_i))$
        then $P_j.\mathbf{y}_i = P_j.\mathbf{x}_i$
      if $f(\mathbf{b}(j,P_j.\mathbf{y}_i)) < f(\mathbf{b}(j,P_j.\hat{\mathbf{y}}))$
        then $P_j.\hat{\mathbf{y}} = P_j.\mathbf{y}_i$
    endfor
    Perform PSO updates on P_j using (1–2)
  endfor
  Select random $k \sim U(1,s/2) \mid Q.\mathbf{y}_k \neq Q.\hat{\mathbf{y}}$
  $Q.\mathbf{x}_k = \mathbf{b}(1,P_1.\hat{\mathbf{y}})$
  for each particle $j \in [1..s]$ **:**
    if $f(Q.\mathbf{x}_j) < f(Q.\mathbf{y}_j)$
      then $Q.\mathbf{y}_j = Q.\mathbf{x}_j$
    if $f(Q.\mathbf{y}_j) < f(Q.\hat{\mathbf{y}})$
      then $Q.\hat{\mathbf{y}} = Q.\mathbf{y}_j$
  endfor
  Perform PSO updates on Q using (1–2)
  for swarm $j \in [1..K]$ **:**
    Select random $k \sim U(1,s/2) \mid P_j.\mathbf{y}_k \neq P_j.\hat{\mathbf{y}}$
    $P_j.\mathbf{x}_k = Q.\hat{\mathbf{y}}_j$
  endfor
until stopping condition is true

Fig. 5. Pseudocode for the generic CPSO-H_K algorithm.

the algorithm. Specifically, after one iteration of the CPSO-S_K half (the P_j swarms in Fig. 5) of the algorithm, the context vector $\mathbf{b}(1,P_1.\hat{\mathbf{y}})$ is used to overwrite a randomly chosen particle in the PSO half (the Q swarm in Fig. 5) of the algorithm. This is followed by one iteration of the Q swarm component of the algorithm, which yields a new global best particle, $Q.\hat{\mathbf{y}}$. This vector is then split into subvectors of the right dimensions and used to overwrite the positions of randomly chosen particles in the P_j swarms.

Although the particles that are overwritten during the information exchange process are randomly chosen, the algorithm does not overwrite the position of the global best position of any of the swarms, since this could potentially have a detrimental effect on the performance of the affected swarm. Empirical studies also indicated that too much information exchange using this mechanism can actually impede the progress of the algorithm. By selecting a particle (targeted for replacement) using a uniform random distribution it is highly likely that a swarm of s particles will have had all its particles overwritten in, say $2s$, information exchange events, except for the global best particle, which is explicitly protected. If the P_j swarms are lagging behind the Q swarm in terms of performance, this means that the P_j swarms could overwrite all the particles in the Q swarm with inferior solutions in only a few iterations. On the other hand, the Q swarm would overwrite particles in the P_j swarms at the same rate, so the overall best solution in the algorithm will always be preserved. The diversity of the particles will decrease significantly because of too-frequent information exchange, though. A simple mechanism to prevent the swarms from accidentally reducing the diversity is implemented by limiting the number of particles that can actively participate in the information exchange. For example, if only half of the particles are possible targets for being overwritten, then at most half of the diversity of the swarm can be jeopardised. This does not significantly affect the positive influence of the information exchange process. For example, if the Q swarm overwrites an inferior particle $P_j.\mathbf{x}_i$ with a superior value (from Q), then that particle i will become the global best particle of swarm j. During subsequent iterations more particles will be drawn to this new global best particle, possibly discovering better solutions along the way, thus, the normal operation of the swarm is not disturbed.

V. Experimental Setup

In order to compare the different algorithms, a fair time measure must be selected. The split and hybrid CPSO algorithms have lower overheads due to the fact that they deal with smaller vectors. Therefore, using processor time as a time measure would give them an unfair advantage. The number of iterations cannot be used as a time measure, as the algorithms do differing amounts of work in their inner loops. It was, therefore, decided to use the number of function evaluations (FEs) as a time measure. All the functions presented here have the value $\mathbf{0}$ in their global minima.

The advantage of measuring complexity by counting the function evaluations is that there is a strong relationship between this measure and processor time as the function complexity increases. This measure, thus, provides a good indication of the relative ranking of the algorithms when using PSOs to train neural networks [8], [9], where the cost of a single function evaluation is large with respect to the overhead of the PSO algorithm itself.

The following functions were selected for testing, largely based on their popularity in the PSO community, allowing for easier comparison.

The Rosenbrock (or banana-valley) function (unimodal)

$$f_0(\mathbf{x}) = \sum_{i=1}^{\frac{n}{2}} \left(100\left(x_{2i} - x_{2i-1}^2\right)^2 + (1 - x_{2i-1})^2\right). \quad (10)$$

The Quadric function (unimodal)

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2. \quad (11)$$

Ackley's function (multimodal)

$$f_2(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e. \quad (12)$$

The generalized Rastrigin function (multimodal)

$$f_3(\mathbf{x}) = \sum_{i=1}^{n}\left(x_i^2 - 10\cos(2\pi x_i) + 10\right). \quad (13)$$

TABLE I
PARAMETERS USED FOR EXPERIMENTS

Function	n	domain	threshold
f_0	30	2.048	100
f_1	30	100	0.01
f_2	30	30	5.00
f_3	30	5.12	100
f_4	30	600	0.1

The generalized Griewank function (multimodal)

$$f_4(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1. \qquad (14)$$

Table I lists the parameters used for the experiments. The values listed in the "domain" column are used to specify the magnitude to which the initial random particles are scaled. The "threshold" column lists the function value threshold which is used as a stopping criterion in some tests (as specified below).

Most of these functions, with the exception of f_2 and f_3, have some interaction between their variables. This should make them more difficult to solve using simple approaches like the relaxation method. Thus, these functions were specifically chosen because it was expected that they would be more difficult to solve using the CPSO algorithms. To make sure that there was sufficient correlation between the variables, making it even harder for the CPSO algorithms, all the functions were further tested under coordinate rotation using Salomon's algorithm [32]. Before each individual run a new rotation was computed, thus, no bias was introduced because of a specific rotation.

A. PSO Configuration

All experiments were run for 2×10^5 error function evaluations (in Section VI-A), or until the error dropped below a stopping threshold (in Section VI-B), depending on the type of experiment being performed. The number of iterations was chosen to correspond to 10^4 iterations of the plain PSO (with 20 particles), following [17]. All experiments were run 50 times; the results reported are the averages (of the best value in the swarm) calculated from all 50 runs. The experiments were repeated for each type of swarm using 10, 15, and 20 particles per swarm. The following types of PSO were tested:

- PSO: "plain" swarm using $c_1 = 1.49$, $c_2 = 1.49$, $w = 0.72$, and $v_{\max}$ is clamped to the domain, following Eberhart and Shi [17].
- CPSO-S: A maximally "split" swarm using $c_1 = 1.49$, $c_2 = 1.49$, w decreases linearly over time, and $v_{\max}$ is clamped to the domain (refer to Table I).
- CPSO-S_6: A "split" swarm using $c_1 = 1.49$, $c_2 = 1.49$, w decreases linearly over time, and $v_{\max}$ is clamped to the domain (refer to Table I). The difference between this swarm type and the split CPSO (above) is that the search-space vector for CPSO-S_6 is split into only six parts (of five components each), instead of 30 parts.
- CPSO-H: A hybrid swarm, consisting of a maximally split swarm, coupled with a plain swarm, described in Section III-A. Both components use the values $c_1 = 1.49$, $c_2 = 1.49$, w decreasing linearly over time, and $v_{\max}$ clamped to the domain (refer to Table I).
- CPSO-H_6: A hybrid swarm, consisting of a CPSO-S_6 swarm, coupled with a plain swarm, described in Section IV. Both components use the values $c_1 = 1.49$, $c_2 = 1.49$, w decreasing linearly over time, and $v_{\max}$ clamped to the domain (refer to Table I).

The above values for the parameters c_1, c_2, and w were selected based on suggestions in other literature where these values have been found, empirically, to provide good performance [17], [26]. For a more detailed study of convergence characteristics for different values of these parameters, please refer to [26].

B. GA Configuration

In order to put the PSO (and, thus, the CPSO) performance into perspective the experiments were repeated using a GA. Results obtained using an implementation of the cooperative GA, as introduced by Potter and De Jong [4], are also provided for comparison. The two GA algorithms have been labeled as follows.

- GA: A standard genetic algorithm, with parameters specified below.
- CCGA: A cooperative genetic algorithm [4], where the search-space vector is maximally split so that each component belongs to its own swarm. For the functions tested here, this implies that 30 populations were employed in a cooperative fashion.

The parameters for both types of GA are as follows.

- Chromosome type: binary coded.
- Chromosome length: 48 bits per function variable.
- Crossover probability: 0.6.
- Crossover strategy: Two-point.
- Mutation probability: $1/(48 \times 30)$, assuming 30 variables per function.
- Fitness scaling: Scaling window of length 5.
- Reproduction strategy: Fitness-proportionate with a 1-element elitist strategy.
- Population size: 100.

Note that the CCGA places each parameter of the function under consideration in its own population, corresponding to the split CPSO. The choice of 48 bits per variable is to make the comparison between the PSO and the GA more fair, as the PSO uses double-precision floating point variables with 52-bit mantissas.

VI. RESULTS

A. Fixed-Iteration Results

This section presents results gathered by allowing all of the methods tested to run for a fixed number of function evaluations, i.e., 2×10^5. The following format applies to Tables II–IV. The second column lists the number of particles per swarm s, or the population size for the GAs. The third and fourth columns list the mean error and 95% confidence interval after the 2×10^5 function evaluations, for the unrotated and rotated versions of

TABLE II
ROSENBROCK (f_0) AFTER 2×10^5 FUNCTION EVALUATIONS

Algorithm	s	Mean(Unrotated)	Mean(Rotated)
PSO	10	1.30e−01 ± 1.45e−01	3.32e−01 ± 9.50e−02
	15	5.53e−03 ± 6.19e−03	2.84e−01 ± 5.17e−02
	20	9.65e−03 ± 7.28e−03	3.16e−01 ± 3.41e−02
CPSO-S	10	7.58e−01 ± 1.16e−01	3.23e+00 ± 7.78e−01
	15	7.36e−01 ± 3.04e−02	2.58e+00 ± 5.36e−01
	20	9.06e−01 ± 3.56e−02	4.37e+00 ± 8.51e−01
CPSO-H	10	2.92e−01 ± 2.19e−02	4.26e−01 ± 3.83e−02
	15	3.14e−01 ± 1.74e−02	4.96e−01 ± 4.53e−02
	20	4.35e−01 ± 2.48e−02	1.06e+00 ± 2.96e−01
CPSO-S_6	10	1.41e+00 ± 4.73e−01	2.65e+00 ± 6.69e−01
	15	2.47e+00 ± 7.00e−01	3.84e+00 ± 9.81e−01
	20	1.59e+00 ± 5.03e−01	4.27e+00 ± 7.73e−01
CPSO-H_6	10	1.94e−01 ± 2.63e−01	1.77e−01 ± 3.62e−02
	15	2.59e−01 ± 2.47e−01	3.73e−01 ± 2.07e−01
	20	4.21e−01 ± 3.21e−01	4.73e−01 ± 1.35e−01
GA	100	6.32e+01 ± 1.19e+01	6.15e+01 ± 1.42e+01
CCGA	100	3.80e+00 ± 1.93e−01	1.32e+01 ± 2.19e+00

TABLE III
QUADRIC (f_1) AFTER 2×10^5 FUNCTION EVALUATIONS

Algorithm	s	Mean(Unrotated)	Mean(Rotated)
PSO	10	1.08e+00 ± 1.41e+00	6.02e+03 ± 2.17e+03
	15	2.85e−72 ± 5.41e−72	3.35e+02 ± 1.35e+02
	20	2.17e−98 ± 4.20e−98	1.12e+02 ± 4.91e+01
CPSO-S	10	2.55e−128 ±4.98e−128	1.47e+03 ± 4.77e+02
	15	7.26e−89 ± 1.14e−88	1.28e+03 ± 3.88e+02
	20	3.17e−67 ± 2.21e−67	1.72e+03 ± 5.91e+02
CPSO-H	10	5.41e−95 ± 1.05e−94	2.15e+02 ± 8.75e+01
	15	6.74e−81 ± 8.92e−81	3.45e+02 ± 9.92e+01
	20	1.45e−63 ± 1.98e−63	4.10e+02 ± 1.32e+02
CPSO-S_6	10	4.63e−07 ± 6.14e−07	2.89e+03 ± 1.07e+03
	15	1.36e−05 ± 1.76e−05	2.99e+03 ± 1.07e+03
	20	1.20e−04 ± 8.99e−05	4.64e+03 ± 1.55e+03
CPSO-H_6	10	2.63e−66 ± 5.08e−66	2.40e+02 ± 1.04e+02
	15	9.00e−46 ± 1.09e−45	7.06e+02 ± 3.24e+02
	20	1.40e−29 ± 1.15e−29	1.03e+03 ± 5.24e+02
GA	100	1.68e+06 ± 2.56e+05	1.07e+06 ± 2.09e+05
CCGA	100	1.38e+02 ± 9.20e+01	6.53e+03 ± 2.38e+03

the functions, respectively. Keep in mind that all the functions used here have a minimum function value of 0.

Table II shows that the Rosenbrock function in its unrotated form is easily optimized by the standard PSO, with the CPSO-H_6 performing better (relative to the others) on the rotated version. Fig. 6 shows a plot of the performance of the various algorithms over time. Note that in the rotated case, there is little difference between the performance of the PSO, CPSO-H, and CPSO-H_6 algorithms.

The Quadric function presents some interesting results, as can be seen in Table III. There is a very large difference in performance between the rotated and unrotated cases. The PSO, CPSO-S, CPSO-H, and CPSO-H_6 algorithms all perform well on the unrotated case, as can be seen in Fig. 7. When the search space is rotated, however, only the PSO, CPSO-H, and CPSO-H_6 algorithms belong to the cluster of performance leaders.

Ackley's function is a multimodal function with many local minima positioned on a regular grid. In the unrotated case, the CPSO-S, CPSO-H and CPSO-H_6 algorithms take the lead, as can be seen in Table IV. In the rotated case, the standard PSO

TABLE IV
ACKLEY (f_2) AFTER 2×10^5 FUNCTION EVALUATIONS

Algorithm	s	Mean(Unrotated)	Mean(Rotated)
PSO	10	7.33e+00 ± 6.23e−01	7.54e+00 ± 5.82e−01
	15	4.92e+00 ± 5.81e−01	5.09e+00 ± 5.11e−01
	20	3.57e+00 ± 4.58e−01	3.42e+00 ± 3.74e−01
CPSO-S	10	2.90e−14 ± 1.60e−15	1.73e+01 ± 1.45e+00
	15	3.01e−14 ± 1.42e−15	1.81e+01 ± 1.09e+00
	20	3.05e−14 ± 1.84e−15	1.85e+01 ± 7.76e−01
CPSO-H	10	2.78e−14 ± 1.71e−15	1.43e+01 ± 1.57e+00
	15	2.92e−14 ± 1.67e−15	1.43e+01 ± 1.48e+00
	20	2.98e−14 ± 1.56e−15	1.60e+01 ± 1.42e+00
CPSO-S_6	10	1.12e−06 ± 4.01e−07	7.98e−01 ± 1.06e+00
	15	1.11e−05 ± 4.35e−06	1.14e+00 ± 1.26e+00
	20	5.42e−05 ± 1.66e−05	1.54e+00 ± 1.46e+00
CPSO-H_6	10	9.42e−11 ± 7.58e−11	8.23e−01 ± 1.04e+00
	15	9.57e−12 ± 7.96e−12	8.12e−01 ± 1.05e+00
	20	2.73e−12 ± 2.03e−12	1.51e−12 ± 6.83e−13
GA	100	1.38e+01 ± 4.04e−01	1.27e+01 ± 1.55e+00
CCGA	100	9.51e−02 ± 3.39e−02	1.57e+01 ± 1.87e+00

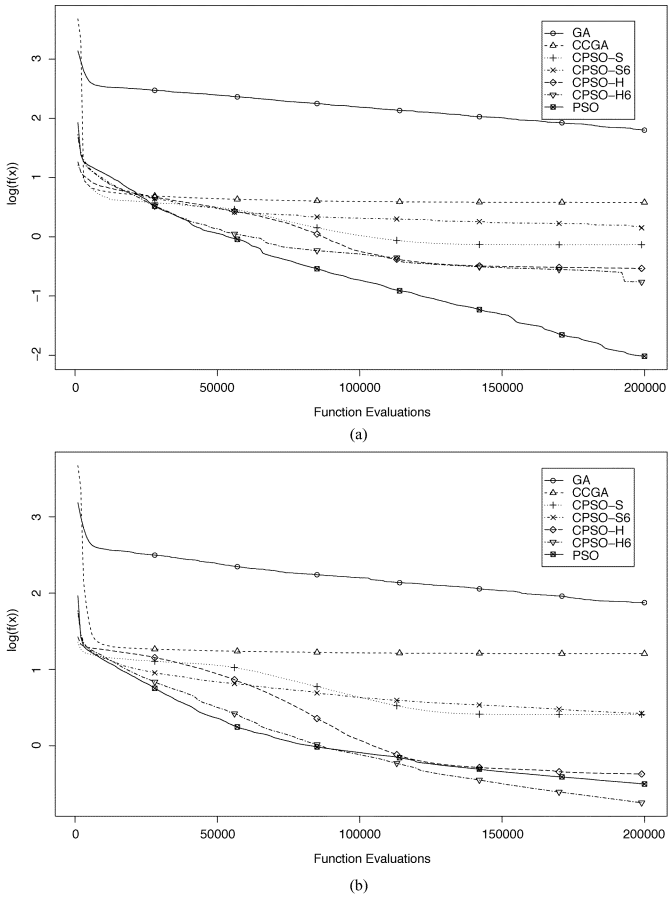

Fig. 6. Rosenbrock (f_0) mean best function value profile. (a) Rosenbrock mean best function value profile. (b) Rotated Rosenbrock mean best function value profile.

algorithm becomes trapped in a local minimum early on, as can be seen from the flat line in Fig. 8. The CPSO-H_6 algorithm is able to continue improving its solution, regardless of rotation. A comment on the performance of the CPSO-S and CPSO-H algorithms in the rotated case is in order. Ackley's function is covered by sinusoidal minima arranged in a regular grid. If the function is unrotated, these "dents" are uncorrelated,

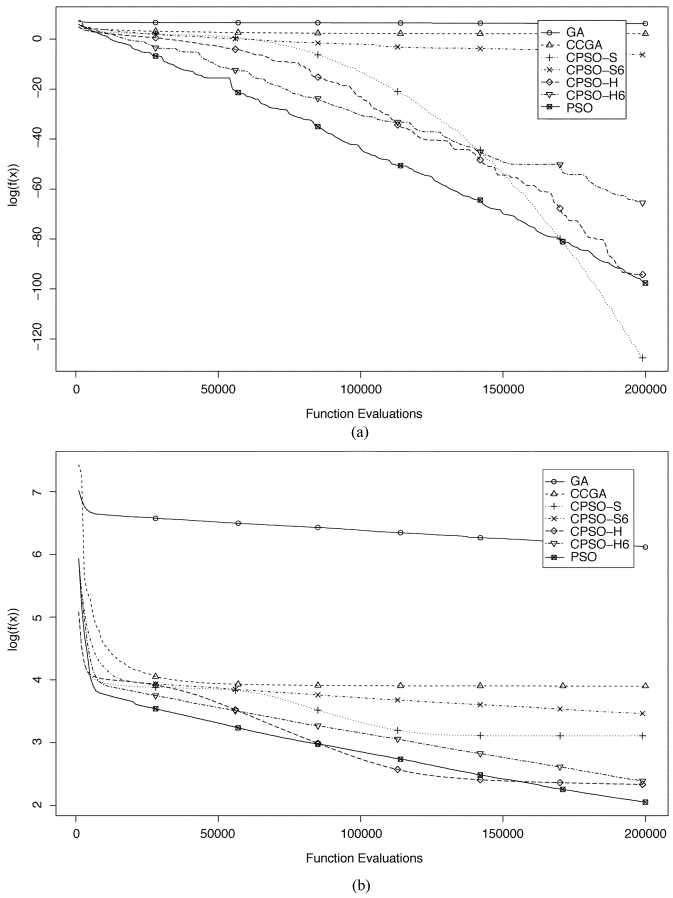

Fig. 7. Quadric (f_1) mean best function value profile. (a) Quadric mean best function value profile. (b) Rotated Quadric mean best function value profile.

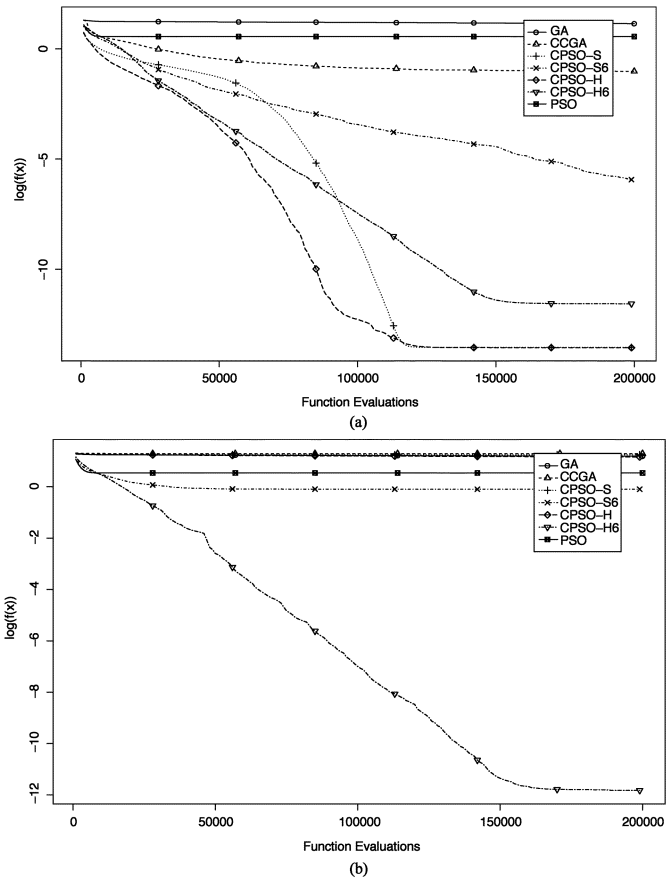

Fig. 8. Ackley (f_2) mean best function value profile. (a) Ackley mean best function value profile. (b) Rotated Ackley mean best function value profile.

so that each dimension can be searched independently. After rotation the dents no longer form a grid aligned with the coordinate axes. This makes the problem significantly harder for the cooperative swarms; however, the CPSO-H_6 algorithm manages to overcome this difficulty. Note that the CCGA algorithm is also negatively affected by the search space rotation.

Rastrigin's function exhibits a pattern similar to that observed with Ackley's function. In the unrotated experiment, the CPSO-S and CPSO-H algorithms perform very well, but their performance rapidly deteriorates when the search space is rotated. The best performer in the rotated case is the CPSO-S_6 algorithm, followed closely by the CPSO-H_6 algorithm, as can be seen in Table V. Given that the CPSO-H_6 algorithm has to devote some of its function evaluations to the standard PSO component it contains, it is conceivable that it may converge more slowly than the CPSO-S_6 algorithm on some problems on which the CPSO-S_6 excels, since the CPSO-S_6 does not have that overhead. Fig. 9 shows a familiar pattern: the standard PSO quickly becomes trapped in a local minimum, while some of the cooperative swarms manage to continue improving.

Table VI shows that the cooperative PSO algorithms performed better than the standard PSO algorithm in all the experiments on Griewank's function. Fig. 10 shows the same trend, however, note how all the algorithms, even the cooperative ones, tend to stagnate after the first 10^5 function evaluations.

TABLE V
RASTRIGIN (f_3) AFTER 2×10^5 FUNCTION EVALUATIONS

Algorithm	s	Mean(Unrotated)	Mean(Rotated)
PSO	10	8.27e+01 ± 5.64e+00	9.76e+01 ± 5.90e+00
	15	7.44e+01 ± 5.66e+00	8.48e+01 ± 5.42e+00
	20	6.79e+01 ± 4.84e+00	7.87e+01 ± 6.79e+00
CPSO-S	10	0.00e+00 ± 0.00e+00	7.55e+01 ± 7.53e+00
	15	0.00e+00 ± 0.00e+00	8.15e+01 ± 6.26e+00
	20	0.00e+00 ± 0.00e+00	7.89e+01 ± 6.41e+00
CPSO-H	10	0.00e+00 ± 0.00e+00	7.91e+01 ± 6.97e+00
	15	0.00e+00 ± 0.00e+00	8.21e+01 ± 6.49e+00
	20	0.00e+00 ± 0.00e+00	8.12e+01 ± 5.92e+00
CPSO-S_6	10	1.39e−01 ± 1.12e−01	5.41e+01 ± 5.18e+00
	15	6.00e−02 ± 6.62e−02	4.66e+01 ± 3.84e+00
	20	1.46e−01 ± 1.03e−01	5.04e+01 ± 5.50e+00
CPSO-H_6	10	1.47e+00 ± 3.16e−01	6.16e+01 ± 5.08e+00
	15	8.77e−01 ± 2.20e−01	5.94e+01 ± 5.04e+00
	20	7.78e−01 ± 1.87e−01	5.41e+01 ± 4.63e+00
GA	100	1.29e+02 ± 7.00e+00	1.37e+02 ± 1.78e+01
CCGA	100	1.22e+00 ± 2.35e−01	6.93e+01 ± 1.02e+01

The results show that the PSO-based algorithms performed better than the GA algorithms in general. The cooperative algorithms collectively performed better than the standard PSO in 80% of the test cases. In particular, the CPSO-H_6 algorithm was able improve on the performance offered by the standard PSO on the rotated multimodal problems, which were the hardest problems to solve among those tested.

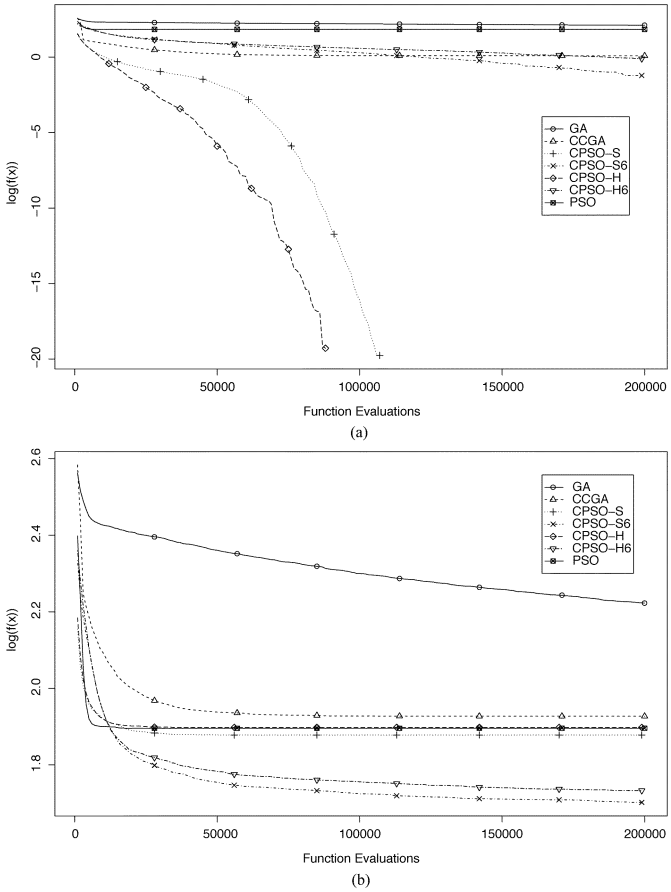

Fig. 9. Rastrigin (f_3) mean best function value profile. (a) Rastrigin mean best function value profile. (b) Rotated Rastrigen mean best function value profile.

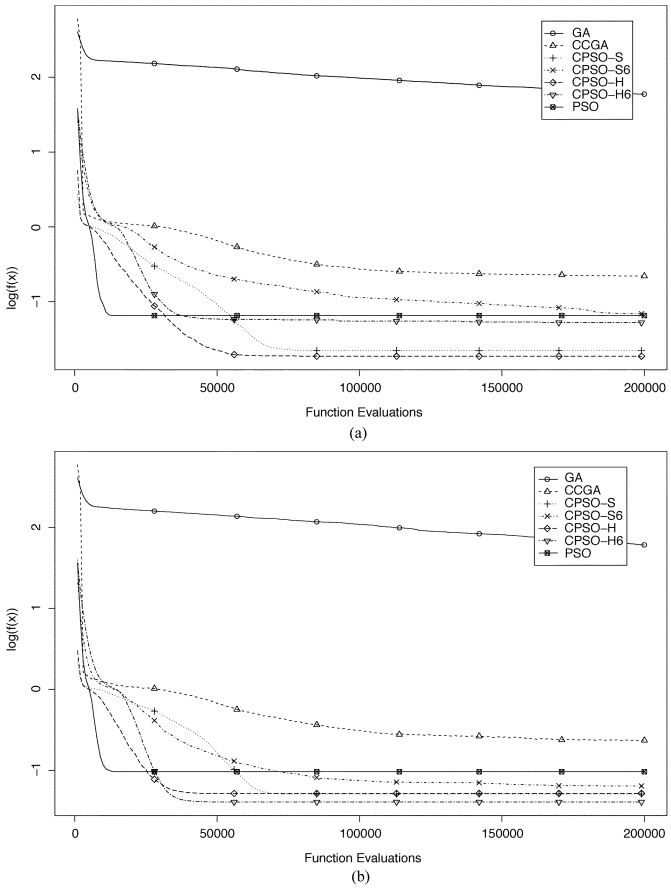

Fig. 10. Griewank (f_4) mean best function value profile. (a) Griewank mean best function value profile. (b) Rotated Griewank mean best function value profile.

TABLE VI
GRIEWANK (f_4) AFTER 2×10^5 FUNCTION EVALUATIONS

Algorithm	s	Mean(Unrotated)	Mean(Rotated)
PSO	10	9.65e−01 ± 7.58e−01	3.45e−01 ± 1.64e−01
	15	2.62e−01 ± 1.61e−01	1.17e−01 ± 4.62e−02
	20	6.51e−02 ± 2.17e−02	9.64e−02 ± 4.95e−02
CPSO-S	10	2.79e−02 ± 8.36e−03	5.10e−02 ± 9.77e−03
	15	2.21e−02 ± 6.28e−03	5.77e−02 ± 1.26e−02
	20	2.25e−02 ± 6.10e−03	6.11e−02 ± 1.17e−02
CPSO-H	10	2.45e−02 ± 5.38e−03	5.19e−02 ± 1.34e−02
	15	2.38e−02 ± 9.81e−03	5.40e−02 ± 1.51e−02
	20	1.86e−02 ± 5.46e−03	4.42e−02 ± 1.08e−02
CPSO-S_6	10	7.29e−02 ± 1.49e−02	6.41e−02 ± 1.18e−02
	15	6.90e−02 ± 1.56e−02	7.40e−02 ± 1.39e−02
	20	8.95e−02 ± 1.68e−02	5.51e−02 ± 1.38e−02
CPSO-H_6	10	6.75e−02 ± 1.40e−02	4.67e−02 ± 1.32e−02
	15	5.54e−02 ± 1.27e−02	3.86e−02 ± 1.05e−02
	20	5.24e−02 ± 1.19e−02	4.06e−02 ± 1.03e−02
GA	100	5.94e+01 ± 6.92e+00	4.98e+01 ± 8.06e+00
CCGA	100	2.20e−01 ± 6.57e−02	1.93e−01 ± 4.82e−02

TABLE VII
ROSENBROCK (f_0) ROBUSTNESS ANALYSIS

		Unrotated		Rotated	
Algorithm	s	Succeeded	Fn Evals.	Succeeded	Fn Evals.
PSO	10	50	609	50	661
	15	50	820	50	790
	20	50	861	50	855
CPSO-S	10	50	320	50	420
	15	50	424	50	532
	20	50	562	50	672
CPSO-H	10	50	332	50	411
	15	50	426	50	525
	20	50	556	50	653
CPSO-S_6	10	50	436	50	516
	15	50	453	50	581
	20	50	521	50	660
CPSO-H_6	10	50	582	50	617
	15	50	655	50	721
	20	50	716	50	845
GA	100	49	16643	48	21234
CCGA	100	50	2652	50	2679

B. Robustness

This section compares the various algorithms to determine their relative rankings using both robustness and convergence speed as criteria. The term "robustness" is used here to mean that the algorithm succeeded in reducing the function value below a specified threshold using fewer than the maximum allocated number of function evaluations. A "robust" algorithm is one that manages to reach the threshold consistently (during all runs) in the experiments performed here. Robustness should not be confused here with sensitivity analysis, which is a study of the influence of parameter changes on performance.

Tables VII–XI present the following information: The "succeeded" column lists the number of runs (out of 50) that managed to attain a function value below the threshold in less

TABLE VIII
QUADRIC (f_1) ROBUSTNESS ANALYSIS

Algorithm	s	Unrotated		Rotated	
		Succeeded	Fn Evals.	Succeeded	Fn Evals.
PSO	10	38	34838	0	N/A
	15	50	16735	1	26161
	20	50	14574	2	175788
CPSO-S	10	50	70215	0	N/A
	15	50	77265	0	N/A
	20	50	83168	0	N/A
CPSO-H	10	50	40056	0	N/A
	15	50	53341	0	N/A
	20	50	61430	0	N/A
CPSO-S_6	10	50	77818	0	N/A
	15	50	101565	0	N/A
	20	50	115687	0	N/A
CPSO-H_6	10	50	22200	1	126271
	15	50	31503	0	N/A
	20	50	43918	0	N/A
GA	100	0	N/A	0	N/A
CCGA	100	0	N/A	0	N/A

TABLE X
RASTRIGIN (f_3) ROBUSTNESS ANALYSIS

Algorithm	s	Unrotated		Rotated	
		Succeeded	Fn Evals.	Succeeded	Fn Evals.
PSO	10	45	2112	41	2403
	15	43	2525	39	2912
	20	49	3341	41	3142
CPSO-S	10	50	375	40	3516
	15	50	436	37	5187
	20	50	546	41	4817
CPSO-H	10	50	388	40	4484
	15	50	430	39	5366
	20	50	545	37	5658
CPSO-S_6	10	50	2226	48	7562
	15	50	2750	50	7517
	20	50	3029	50	9874
CPSO-H_6	10	50	2386	48	16212
	15	50	2748	50	12133
	20	50	3499	50	11964
GA	100	27	75341	1	59100
CCGA	100	50	2339	50	2659

TABLE IX
ACKLEY (f_2) ROBUSTNESS ANALYSIS

Algorithm	s	Unrotated		Rotated	
		Succeeded	Fn Evals.	Succeeded	Fn Evals.
PSO	10	11	2099	6	1988
	15	32	3019	32	3385
	20	37	2986	41	3200
CPSO-S	10	50	935	5	6240
	15	50	1053	2	11644
	20	50	1227	2	43314
CPSO-H	10	50	1068	4	24420
	15	50	1154	2	5836
	20	50	1245	2	2401
CPSO-S_6	10	50	3264	50	6670
	15	50	4136	47	4533
	20	50	4994	46	5686
CPSO-H_6	10	50	3105	49	3494
	15	50	3924	46	5355
	20	50	4947	49	5657
GA	100	50	100	50	100
CCGA	100	50	100	50	100

TABLE XI
GRIEWANK (f_4) ROBUSTNESS ANALYSIS

Algorithm	s	Unrotated		Rotated	
		Succeeded	Fn Evals.	Succeeded	Fn Evals.
PSO	10	19	17521	18	24081
	15	30	8066	35	9095
	20	34	8405	34	8620
CPSO-S	10	50	46963	45	55532
	15	50	47174	40	59911
	20	50	46679	42	59389
CPSO-H	10	47	20170	40	24374
	15	49	24183	46	30257
	20	50	27121	43	35715
CPSO-S_6	10	40	85580	44	64311
	15	33	98075	40	72844
	20	34	105770	40	77259
CPSO-H_6	10	40	24445	44	19478
	15	44	21063	39	21282
	20	40	28577	43	28099
GA	100	0	N/A	0	N/A
CCGA	100	26	134056	5	128545

than 2×10^5 FEs, while the "Fn Evals." column presents the number of function evaluations needed on average to reach the threshold, calculated only for the runs that "succeeded." Note that no confidence intervals or standard deviations are reported for the number of function evaluations required to reach the threshold. One reason for this omission is that the number of times that the algorithm succeeded in reaching the threshold already provides information regarding the variability of the result, meaning that a robust algorithm will typically have a small standard deviation. Keep in mind that the less robust algorithms sometimes have as few as four runs that succeeded in reaching the threshold, so that the sample standard deviation would be quite inaccurate. The distributions of the results were also tested for normality (a requirement for sensible interpretation of the standard deviation). Although not reported individually here, most of these results had highly nonnormal distributions, usually a distribution that appeared one-sided, with the reported mean being close to the minimum value.

None of the algorithms, with the exception of the standard GA, had any difficulty reaching the threshold of the Rosenbrock function during any of the runs. Table VII further shows that all the PSO-based algorithm solved the problem in fewer than 1000 function evaluations, with the CPOS-S algorithm requiring the fewest function evaluations overall.

The Quadric function shows how much more difficult it can become to minimize the rotated version of a function. The cooperative algorithms reached the threshold during all the runs in the unrotated case, but failed completely on the rotated problem. The standard PSO and the GAs had some difficulty solving the unrotated case, with the GAs consistently failing on all the runs. Looking at the number of function evaluations, the standard PSO was in the lead, followed by the CPSO-H_6 algorithm, as shown in Table VIII.

The standard PSO had some difficulty with Ackley's function, as can be seen in Table IX. Note that both the CPOS-S and CPOS-H algorithms failed almost completely on the rotated

function, but that the CPSO-S_6 and CPSO-H_6 algorithms managed to solve the rotated problem consistently. This function represents a very important result regarding the nature of the cooperative algorithms: on uncorrelated functions, the CPSO-S and CPSO-H algorithms have the speed advantage, but they fail on highly correlated multimodal functions. The CPSO-S_K and CPSO-H_K algorithms may have somewhat slower rates of convergence compared with the CPSO-S and CPSO-H algorithms, but they are significantly more robust—in many cases, more robust than the original PSO algorithm. Note that the GAs were very consistent in solving this problem.

Table X shows a similar, but less pronounced scenario. The cooperative algorithms again perform admirably on the unrotated Rastrigin function, but the CPSO-S and CPSO-H algorithms are less robust on the rotated problem. Note that the CCGA algorithm is doing very well on this problem, delivering the best overall performance for the rotated case.

Griewank's function proves to be hard to solve for all the algorithms, as can be seen in Table XI. Only the CPSO-S and CPSO-H algorithms consistently reached the threshold during some runs on the unrotated problem. No algorithm could achieve a perfect score on the rotated problem, but the cooperative algorithms appear to have performed better than the standard PSO and the GAs.

Overall, as far as robustness is concerned, the CPSO-H_6 algorithm appears to be the winner, since it achieved a perfect score in seven of the ten test cases. The CPSO-S, CPSO-H, and CPSO-S_6 algorithms were slightly less robust, followed closely by the CCGA. The standard PSO and the GA were fairly unreliable on this set of problems.

When looking at the number of function evaluations, the CPSO-S algorithm was usually the fastest, followed by the standard PSO and the CCGA. These results indicate that there is a tradeoff between the convergence speed and the robustness of the algorithm.

C. Discussion of Results

The results presented in Sections VI-A and VI-B can be summarized as follows.

- On unimodal functions, the standard PSO and CPSOs performed very well in the unrotated case.
- On functions containing lattice-based local minima, the CPSOs perform very well when the lattice is aligned with the coordinate axes. When the coordinate axes are rotated, CPSO-S and CPSO-H performance degrades (to a degree depending on the specific function), while the CPSO-S_K and CPSO-H_K algorithms handle these cases better. The standard PSO quickly becomes trapped in local minima on some of these problems.
- All the PSO-based algorithms are highly competitive with the GA-based algorithms on all of the problems, usually surpassing their performance.
- The CPSO-H_K algorithm is very robust, even when dealing with multimodal rotated functions.
- The standard PSO performs best when using 20 particles per swarm.
- The CPSO-S and CPSO-H algorithms perform better when ten particles per swarm are used.
- The CPSO-S_K and CPSO-H_K algorithms are somewhat faster when using 10 particles per swarm, but more robust using 20 particles per swarm. The speed improvement using 10 particles is sufficient to warrant the small loss in robustness.

From this summary, it can be hypothesized that the PSO performs best when the size of the search space is constrained. Consider that the initialization step of the PSO scatters the particles uniformly through the search space. If the number of particles is finite, the probability of having a particle's position initialized close to a minimum (or any specific small volume in the search space) tends to zero as the dimensionality of the search space approaches infinity. In fact, the probability of finding a particle in a specific region (of small, specified volume) decreases exponentially as the number of dimensions increases. Each iteration of the PSO algorithm takes another random sample from a subspace specified by the relative positions of the particles at that time, so the probability of a particle landing in a specific region is again influenced exponentially by the dimensionality of the search space. This argument illustrates that the PSO (like most other stochastic algorithms) is expected to perform better in low-dimensional search spaces.

The various CPSO algorithms aim to exploit this property by utilising multiple PSOs in an attempt to keep the dimensionality of the search space assigned to each PSO small, at the same time providing a mechanism for these swarms to cooperate toward the goal of solving the original high-dimensional problem. This offers some explanation as to the better performance of the CPSO algorithms on the unrotated problems, since the dimensions of the unrotated problems are relatively independent (for many of the functions tested). The rotated problems increase the correlation between the subspaces assigned to the different PSO subalgorithms used in the CPSO, thus reducing the effectiveness of the decomposition. For some problems, however, this reduction in efficacy is less significant than the performance gained by reducing the dimensionality of the problem through the decomposition.

Another benefit of the decomposition is that the overall diversity of solutions generated by the CPSO exceeds that of the standard PSO. This ensures that the search space is sampled more thoroughly, thus improving the algorithm's chances of finding a good solution.

A CPSO-S_K variant has been used to train product unit neural networks with promising results in [33]. There it was determined that around five function variables per swarm (corresponding to the CPSO-S_6 architecture presented here) offered the best performance. This would suggest that the error function of the network represents a problem like rotated Ackley, Griewank, or Rastrigin, that is, a function with local minima and interdependency between the variables.

Overall, the cooperative PSO algorithms offer improved performance over the standard PSO, especially in terms of robustness.

VII. Conclusion

This paper presented a method of casting particle swarm optimization into a cooperative framework. This resulted in a significant improvement in performance, especially in terms of

solution quality and robustness. One hypothesis is that the increased diversity of the cooperative swarms is responsible for the improved robustness on multimodal problems.

The cooperative approach introduced here performs better and better as the dimensionality of the problem increases (borne out by the results presented in [33]), compared with the traditional PSO. A likely explanation for this effect is that the PSO (like most other stochastic search algorithms) performs better in lower dimensional search spaces. This is mostly due to the exponential increase in the volume of the search space as the dimensionality increases, while the number of particles has to be kept fixed (and small) to keep the algorithm efficient. Large swarms tend to have numerous particles that do not contribute to the solution, especially during later iterations, so it would be impractical to increase the number of particles to match the increase in volume. Since the CPSO algorithms decompose the larger search space into several smaller spaces, the rate at which each of these subswarms converge onto solutions contained in their subspaces is significantly faster than the rate of convergence of the standard PSO on the original, n-dimensional search space.

The price paid for the increased performance is the chance that the CPSO algorithm may converge onto pseudominima that were called into existence by the decomposition of the search space. The efficacy of the decomposition is also affected by the degree of correlation between the subproblems created by the decomposition. It was found that in spite of these potential difficulties, the CPSO algorithms exhibited significantly better performance on many of the problems tested. The hybrid CPSO variants were found to exhibit emergent behavior, that is, they performed differently from their constituent parts, usually better. This phenomenon warrants more study.

The new algorithms presented here also lend themselves to distributed architectures, as the swarms can be processed on different machines concurrently. The CPSO-S_K and CPSO-H_K techniques require some form of shared memory to build the context vector, but it is hypothesized that this vector does not have to be updated during every cycle (to reduce bandwidth usage) for the algorithm to work well. This will be investigated at a later stage.

Several important properties of the split swarm technique still remain to be investigated. It is not yet clear whether the same parameters that work well for the plain swarm are optimal for the CPSOs. Although the cooperative swarms typically outperformed the traditional PSO on the functions evaluated in this paper, it should not be taken as proof that these new approaches will be better for all problems, especially in the light of the no free lunch theorem [34]. A theoretical analysis of the new technique is currently under development to further investigate the type of function for which the cooperative algorithms offer better performance. A study is also currently being done to investigate the performance of using the GCPSO instead of standard PSO within the cooperative version of the PSO.

REFERENCES

[1] F. Solis and R. Wets, "Minimization by random search techniques," *Math. Oper. Res.*, vol. 6, pp. 19–30, 1981.

[2] K. A. De Jong, "An Analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, Univ. Michigan, Ann Arbor, MI, 1975.

[3] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. London, U.K.: Oxford Univ. Press, 1996.

[4] M. A. Potter and K. A. de Jong, "A cooperative coevolutionary approach to function optimization," in *The Third Parallel Problem Solving From Nature*. Berlin, Germany: Springer-Verlag, 1994, pp. 249–257.

[5] Y. Ong, A. Keane, and P. Nair, "Surrogate-assisted coevolutionary search," in *Proc. 9th Int. Conf. Neural Information Processing*, Nov. 2002, pp. 1140–1145.

[6] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. 6th Int. Symp. Micro Machine and Human Science*, Nagoya, Japan, 1995, pp. 39–43.

[7] R. C. Eberhart, P. Simpson, and R. Dobbins, *Computational Intelligence PC Tools*: Academic, 1996, ch. 6, pp. 212–226.

[8] A. P. Engelbrecht and A. Ismail, "Training product unit neural networks," *Stability Control: Theory Appl.*, vol. 2, no. 1–2, pp. 59–74, 1999.

[9] F. van den Bergh and A. P. Engelbrecht, "Cooperative learning in neural networks using particle swarm optimizers," *South African Comput. J.*, vol. 26, pp. 84–90, Nov. 2000.

[10] R. C. Eberhart and X. Hu, "Human tremor analysis using particle swarm optimization," in *Proc. Congr. Evolutionary Computation*, Washington, DC, July 1999, pp. 1927–1930.

[11] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proc. Congr. Evolutionary Computation*, Washington, DC, July 1999, pp. 1945–1949.

[12] ——, "A modified particle swarm optimizer," in *Proc. IEEE Int. Conf. Evolutionary Computation*, Anchorage, AK, May 1998.

[13] P. N. Suganthan, "Particle swarm optimizer with neighborhood operator," in *Proc. Congr. Evolutionary Computation*, Washington, DC, July 1999, pp. 1958–1961.

[14] M. Clerc, "The swarm and the queen: toward a deterministic and adaptive particle swarm optimization," in *Proc. ICEC'99*, Washington, DC, 1999, pp. 1951–1957.

[15] D. Corne, M. Dorigo, and F. Glover, Eds., *New Ideas in Optimizaton*. New York: McGraw-Hill, 1999, ch. 25, pp. 379–387.

[16] M. Clerc and J. Kennedy, "The particle swarm: explosion, stability, and convergence in a multi-dimensional complex space," *IEEE Trans. Evol. Comput.*, no. 6, pp. 58–73, 2002.

[17] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proc. 2000 Congr. Evolutionary Computing*, 2000, pp. 84–89.

[18] P. Angeline, "Using selection to improve particle swarm optimization," in *Proc. IJCNN'99*, Washington, DC, July 1999, pp. 84–89.

[19] J. Kennedy, "Stereotyping: Improving particle swarm performance with cluster analysis," in *Proc. 2000 Congr. Evolutionary Computing*, 2000, pp. 1507–1512.

[20] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Proc. 2002 World Congr. Computational Intelligence*, Honolulu, HI, May 2002, pp. 1671–1676.

[21] J. Holland, *Adaption in Natural and Artificial Systems*. Ann Arbor, MI: Univ. of Michigan Press, 1975.

[22] H. G. Cobb, "Is the genetic algorithm a cooperative learner?," in *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann, 1992, pp. 277–296.

[23] S. H. Clearwater, T. Hogg, and B. A. Huberman, "Cooperative problem solving," in *Computation: The Micro and Macro View*, Singapore: World Scientific, 1992, pp. 33–70.

[24] R. V. Southwell, *Relaxation Methods in Theoretical Physics*. Oxford, U.K.: Clarendon Press, 1946.

[25] M. Friedman and L. S. Savage, "Planning experiments seeking minima," in *Selected Techniques of Statistical Analysis for Scientific and Industrial Research, and Production and Management Engineering*, C. Eisenhart, M. W. Hastay, and W. A. Wallis, Eds. New York: McGraw-Hill, 1947, pp. 363–372.

[26] F. van den Bergh, "An analysis of particle swarm optimizers," Ph.D. dissertation, Dept. Comput. Sci., Univ. Pretoria, Pretoria, South Africa, 2002.

[27] D. Goldberg, K. Deb, and J. Horn, "Massive multimodality, deception, and genetic algorithms," in *Parallel Problem Solving From Nature*. Amsterdam, The Netherlands: North-Holland, 1992, vol. 2, pp. 37–46.

[28] J. J. Grefenstette, "Deception considered harmful," in *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann, 1992, pp. 75–91.

[29] M. A. Potter, "The design and analysis of a computational model of cooperative coevolution," Ph.D. dissertation, George Mason Univ., Fairfax, VA, 1997.

[30] P. J. Angeline, "Using selection to improve particle swarm optimization," in *Proc. IJCNN'99*, Washington, DC, July 1999, pp. 84–89.
[31] M. Løvbjerg, T. K. Rasmussen, and T. Krink, "Hybrid particle swarm optimizer with breeding and subpopulations," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, San Francisco, CA, July 2001.
[32] R. Salomon, "Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions," *BioSystems*, vol. 39, pp. 263–278, 1996.
[33] F. van den Bergh and A. P. Engelbrecht, "Training product unit networks using cooperative particle swarm optimizers," in *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, Washington, D.C., July 2001, pp. 126–131.
[34] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, no. 4, pp. 67–82, 1997.

Frans van den Bergh received the M.Sc. degree in computer science (computer vision) and the Ph.D. degree in computer science (particle swarm optimization) from the University of Pretoria, Pretoria, South Africa, in 2000 and 2002, respectively.

He is currently with Rapid Mobile, Pretoria, South Africa. He maintains an active interest in the field of numerical optimization, specifically, in the area of particle swarm optimization. Further research interests include pattern recognition, photorealistic rendering, and computer vision.

Andries P. Engelbrecht (M'00) received the M.Sc. and Ph.D. degrees from the University of Stellenbosch, Stellenbosch, South Africa, in 1994 and 1999, respectively.

He is a Full Professor with the Department of Computer Science, University of Pretoria, Pretoria, South Africa. He is the Head of the Computational Intelligence Research Group, University of Pretoria, with a group of 40 postgraduate students. He is the author of *Computational Intelligence: An Introduction* (New York: Wiley, 2002). His research interests include aspects of swarm intelligence, evolutionary computation, artificial immune systems, and neural networks, with several publications in those fields.

Prof. Engelbrecht is a Member of the INNS and the IEEE Neural Network Society (NNS) Task Forces on Evolutionary Computation and Games, Swarm Intelligence, and Coevolution.