Step-Optimized Particle Swarm Optimization

Thomas Schoene
University of Saskatchewan
Saskatoon, SK, Canada
schoene@cs.usask.ca

Simone A. Ludwig
North Dakota State University
Fargo, ND, USA
simone.ludwig@ndsu.edu

Raymond J. Spiteri
University of Saskatchewan
Saskatoon, SK, Canada
spiteri@cs.usask.ca

***Abstract*—Recent developments of Particle Swarm Optimization (PSO) have successfully trended towards Adaptive PSO (APSO). APSO changes its behavior during the optimization process based on information gathered at each iteration. It has been shown that APSO is able to solve a wide range of difficult optimization problems efficiently and effectively. In classical PSO, all parameters remain constant for the entire swarm during the iterations. In particular, all particles share the same settings for their velocity weights. We propose a Step-Optimized PSO (SOPSO) algorithm in which every particle has its own velocity weights and an inner PSO iteration is used to take a step towards optimizing the settings of the velocity weights of every particle at every iteration. We compare SOPSO to four known PSO variants (global best PSO, decreasing weight PSO, time-varying acceleration coefficients PSO, and guaranteed convergence PSO). Experiments are conducted to compare the performance of SOPSO to the known PSO variants on 22 benchmark problems. The results show that SOPSO outperforms the known PSO variants on difficult optimization problems that require large numbers of function evaluations for their solution. This suggests that the SOPSO strategy of optimizing the settings of the velocity weights of every particle improves the robustness and performance of the algorithm.**

I. Introduction

In order to tackle complex real-world optimization problems, scientists have been looking into techniques inspired by natural processes such as Darwinian evolution, social group behavior, and foraging strategies. Accordingly there has been a remarkable growth in the field of nature-inspired search and optimization algorithms over the past few decades. These algorithms are mainly categorized as being based on either evolutionary computation or swarm intelligence. Evolutionary computation makes use of a population that is selected in a guided random search (inspired by biological mechanisms of evolution) to achieve the optimization process. Swarm intelligence is characterized by the collective behavior of decentralized, self-organized systems that are typically made up of a population of simple agents interacting locally with one another and with their environment.

Particle Swarm Optimization (PSO) is a swarm intelligence method first introduced by Kennedy and Eberhart [5]. The behavior of PSO can be envisioned by comparing it to bird swarms searching for optimal food sources, where the direction in which a bird moves is influenced by its current movement, the best food source it ever experienced, and the best food source any bird in the swarm ever experienced. In other words, birds move according to their inertia, their personal knowledge, and the knowledge of the swarm. In terms of PSO, the movement of a particle is influenced by its inertia, its personal best position, and the global best position.

PSO has multiple particles, and every particle consists of its current objective value, its position, its velocity, its *personal best value*, that is the best objective value the particle ever experienced, and its *personal best position*, that is the position at which the personal best value has been found. In addition, PSO maintains a *global best value*, that is the best objective value any particle has ever experienced, and a *global best position*, that is the position at which the global best value has been found. Classical PSO [5] uses the following iteration to move the particles:

$$\mathbf{x}^{(i)}(n+1) = \mathbf{x}^{(i)}(n) + \mathbf{v}^{(i)}(n+1), \quad n = 0, 1, 2, \ldots, N-1, \quad \text{(1a)}$$

where $\mathbf{x}^{(i)}$ is the position of particle i, n is the iteration number with $n = 0$ referring to the initialization, N is the total number of iterations, and $\mathbf{v}^{(i)}$ is the velocity of particle i, $i = 1, 2, \ldots, n_p$, where n_p is the number of particles. Classical PSO uses the following iteration to update the particle velocities:

$$\mathbf{v}^{(i)}(n+1) = \mathbf{v}^{(i)}(n) + 2\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] + 2\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \quad n = 0, 1, 2, \ldots, N-1, \quad \text{(1b)}$$

where $\mathbf{x}_p$ is the personal best position and $\mathbf{x}_g$ is the global best position. The vector $\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)$ is directed towards the personal best position, and the vector $\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)$ is directed towards the global best position. Both $\mathbf{r}_1^{(i)}$ and $\mathbf{r}_2^{(i)}$ are random vectors; i.e., their components consist of random values uniformly distributed between 0 and 1. The notation $\mathbf{r}^{(i)}(n)$ is meant to denote that a new random vector is generated for every particle i and iteration n.

PSO can promote either diversity or convergence at any iteration. Diversity favors scattered particles, searching a large area coarsely, whereas convergence favors clustered particles, searching a small area intensively. Promising strategies include promoting diversity of the swarm in early iterations and convergence in later iterations [3], [11] or assigning attributes to individual particles to promote diversity or convergence [6].

II. Related PSO Variants

Suppose we are interested in finding the global minimum of an objective function $f(\mathbf{x})$ in a D-dimensional domain (or

search space) of the form $[x_{\min}, x_{\max}]^D$. For the purposes of assessing the performance of SOPSO, we utilize four related PSO variants in this work: global best PSO, decreasing weight PSO, time-varying acceleration coefficients PSO, and guaranteed convergence PSO. These four PSO variants form a baseline against which the proposed SOPSO variant is compared and are now described in more detail.

A. Global Best Particle Swarm Optimization (GBPSO)

GBPSO [3] is a standard PSO variant. GBPSO [5] uses the following iteration to determine the velocities:

$$\begin{aligned}\mathbf{v}^{(i)}(n+1) = w\mathbf{v}^{(i)}(n) + c_1\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] \\ + c_2\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \\ n = 0, 1, 2, \ldots N-1. \quad (2)\end{aligned}$$

The velocity weights are the inertia weight w, the personal best weight c_1, and the global best weight c_2. Compared to the classical PSO [3], GBPSO uses an inertia weight w and does not require $c_1 = c_2 = 2$ as in Equation (1b).

The algorithm proceeds as follows. First, the swarm is initialized; i.e., for $n = 0$ the positions and the velocities of the particles are randomly initialized within the search space [11]. After that, the objective values of particles are calculated. For each particle, its first objective values and positions are automatically its personal best values and the personal best positions. The global best value and the global best position are set to the objective value and position of the particle with the best objective value in the entire swarm. After this initialization, velocities for all particles are calculated using Equation (1b), and then they are moved to their new positions using Equation (1a). The particle objective values are evaluated again. Personal best positions are updated for particles that have a new objective value that is better than their previous personal best value. The global best position is updated if there is any particle with an objective value that is better than the previous global best value. New velocities are calculated using Equation (1b), and all particles are moved to their new positions using Equation (1a). The algorithm continues evaluating the particle objective values and updating personal best values, personal best positions, the global best value, the global best position, particle velocities, and particle positions until a termination criterion, such as a limit on the number of iterations, is satisfied.

B. Decreasing Weight Particle Swarm Optimization (DWPSO)

DWPSO is similar to GBPSO, but the inertia weight $w(n)$ is decreased linearly over time [3]. The idea behind DWPSO is to promote diversity in early iterations and convergence in late iterations. DWPSO uses the following iteration to determine the velocities:

$$\begin{aligned}\mathbf{v}^{(i)}(n+1) = w(n)\mathbf{v}^{(i)}(n) + c_1\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] \\ + c_2\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \\ n = 0, 1, 2, \ldots, N-1, \quad (3)\end{aligned}$$

where the inertia weight w at iteration n is calculated using

$$w(n) = w_s - (w_s - w_e)\frac{n}{N-1}, \quad (4)$$

where w_s is the inertia weight for the first iteration and w_e is the inertia weight for the last iteration.

C. Time-Varying Acceleration Coefficients PSO (TVACPSO)

TVACPSO changes not only the inertia weight $w(n)$ but also the acceleration coefficients, i.e., the personal $c_1(n)$ and global best weight $c_2(n)$, over time [3], [11]. The idea is to have a high diversity for early iterations and a high convergence for late iterations. The inertia weight $w(n)$ is changed as in DWPSO using Equation (4). TVACPSO uses the following iteration to determine the velocities:

$$\begin{aligned}\mathbf{v}^{(i)}(n+1) = w(n)\mathbf{v}^{(i)}(n) + c_1(n)\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] \\ + c_2(n)\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \\ n = 0, 1, 2, \ldots, N-1, \quad (5)\end{aligned}$$

where the personal best weight $c_1(n)$ and the global best weight $c_2(n)$ at iteration n are calculated using

$$\begin{aligned}c_1(n) &= c_{1s} - (c_{1s} - c_{1e})\frac{n}{N-1}, \\ c_2(n) &= c_{2s} - (c_{2s} - c_{2e})\frac{n}{N-1},\end{aligned}$$

where c_{1s} is the personal best weight for the first iteration, c_{1e} is the personal best weight for the last iteration, c_{2s} is the global best weight for the first iteration, and c_{2e} is the global best weight for the last iteration.

D. Guaranteed Convergence Particle Swarm Optimization (GCPSO)

GCPSO guarantees that the global best particle searches within a dynamically adapted radius at all times [14][15]. This technique addresses the problem of stagnation and increases local convergence [14] by using the global best particle to randomly search in an adaptively changing radius at every iteration. GCPSO, as described in [14], uses Equation (3) to determine the velocities $\mathbf{v}^{(i)}(n)$ and Equation (4) to change the inertia weight $w(n)$ over time. The personal best weight c_1 and the global best weight c_2 are held constant. GCPSO uses the following iteration to update the velocity of the global best particle:

$$\begin{aligned}\mathbf{v}^{(i_g)}(n+1) = -\mathbf{x}^{(i_g)}(n) + \mathbf{x}_g(n) + w(n)\mathbf{v}^{(i_g)}(n) \\ + \rho(n)(1 - 2\mathbf{r}_3(n)), \\ n = 0, 1, 2, \ldots, N-1, \quad (6a)\end{aligned}$$

where i_g is the index of the particle that updated the global best value most recently. The expression $-\mathbf{x}^{(i_g)}(n) + \mathbf{x}_g(n)$ is used to reset the position of particle i_g to the global best position. The components of $\mathbf{r}_3(n)$ are random numbers uniformly distributed between 0 and 1. The search radius is controlled by

the search radius parameter $\rho(n)$. The search radius parameter $\rho(n)$ is calculated using:

$$\rho(n+1) = \begin{cases} 2\rho(n), & \text{if } \sigma(n+1) > \sigma_c, \\ \frac{1}{2}\rho(n), & \text{if } \varphi(n+1) > \varphi_c, \\ \rho(n), & \text{otherwise,} \end{cases} \tag{6b}$$

where σ_c is the consecutive success threshold and φ_c is the consecutive failure threshold defined below. Success means that using Equations (1a) and (6a) to update the particle positions results in an improved global best value and position, and failure means it does not. The numbers of consecutive successes $\sigma(n)$ and failures $\varphi(n)$ are calculated using:

$$\sigma(n+1) = \begin{cases} 0, & \text{if } \varphi(n+1) > \varphi(n), \\ \sigma(n)+1, & \text{otherwise,} \end{cases} \tag{6c}$$

$$\varphi(n+1) = \begin{cases} 0, & \text{if } \sigma(n+1) > \sigma(n), \\ \varphi(n)+1, & \text{otherwise.} \end{cases} \tag{6d}$$

E. Correcting Iterations

In addition to the specific details of each of the PSO variants described, there is additional practical consideration of how to handle particles that attempt to move out of the search space. If a particle attempts to leave the search space, our strategy is to return it along its proposed path through a series of *correcting iterations*. In particular, we use:

$$\breve{\mathbf{x}}^{(i)}(\breve{n}+1) = \breve{\mathbf{x}}^{(i)}(\breve{n}) - \breve{\mathbf{v}}^{(i)}(\breve{n}+1), \quad \breve{n} = 0, 1, ..., \breve{N}-1, \tag{7a}$$

where $\breve{\mathbf{x}}^{(i)}(\breve{n}+1)$ is the corrected position, $\breve{\mathbf{v}}^{(i)}$ is the corrected velocity, $\breve{n}$ is the count for the correcting iterations, and $\breve{N}$ is the total number of correcting iterations. The initial corrected position $\breve{\mathbf{x}}^{(i)}(0)$ is set to the position $\mathbf{x}^{(i)}(n+1)$, which is outside the search space. The corrected velocities $\breve{\mathbf{v}}^{(i)}$ are calculated using:

$$\breve{\mathbf{v}}^{(i)}(\breve{n}+1) = \alpha \breve{\mathbf{v}}^{(i)}(\breve{n}), \quad \breve{n} = 0, 1, ..., \breve{N}-1, \tag{7b}$$

where α is the correction factor, and the initial corrected velocity $\breve{\mathbf{v}}^{(i)}(0)$ is set to the velocity $\mathbf{v}^{(i)}(n+1)$ that caused the particle to attempt to leave the search space. Equation (7a) is applied until the corrected position $\breve{\mathbf{x}}^{(i)}(\breve{n}+1)$ is in the search space or the limit on the total number of correcting iterations $\breve{N}$ is reached. In the latter case, the components of $\breve{\mathbf{x}}^{(i)}(\breve{N})$ still outside the search space are clamped to the boundary of the search space. Based on good performance in empirical tests, the values chosen are $\alpha = 0.54$ and $\breve{N} = 4$. All PSO variants considered in this study, including step-optimized PSO (SOPSO), use these values.

III. Proposed Variant: SOPSO

The idea for our SOPSO variant is based on PSO techniques that give every particle its own velocity weights [18], [20]. These techniques typically move the velocity weights of all particles toward the velocity weights of a certain particle that is selected based on a measure of superior performance [20]. Self-tuning APSO moves the velocity weights towards the settings of the particle that yielded the most updates of the global best position [18], [20]. Controlled APSO adaptively changes the personal best weights $c_1^{(i)}(n)$ and the global best weights $c_2^{(i)}(n)$ based on the distance between the positions and the global best position [4]. Inertia weight APSO [22] grants every particle its own inertia weight $w^{(i)}(n)$ that is changed using a function of the objective values and the global best value. Optimized PSO uses multiple PSO subswarms, each having their own parameter settings, in an inner iteration to solve the original optimization problem. The parameter settings are then optimized in an outer iteration of PSO for a fixed number of iterations. A detailed description of optimized PSO can be found in [8]. Inspired by optimized PSO [8], we treat the problem of finding good velocity weights as an optimization problem.

A. Step-Optimized Particle Swarm Optimization (SOPSO)

In SOPSO every particle has its own velocity weights. A particular setting of the velocity weights is referred to as the *position of the velocity weights*. An objective function for the velocity weights is used to quantify how well the positions of the velocity weights perform for solving the optimization problem. Using the calculated objective values of the velocity weights, SOPSO takes a step toward optimizing the velocity weights using a PSO variant such as GBPSO, TVACPSO, or DWPSO at every iteration. The velocity weights are optimized in a fixed auxiliary search space. Compared to optimized PSO [8], the SOPSO approach of optimizing after every (outer) PSO iteration is more efficient because only one additional PSO instance (for optimizing the velocity weights) is executed and only for one (inner) iteration.

SOPSO uses the following iteration, with the notation used in Equation (2), to update the velocities of particles:

$$\begin{aligned} \mathbf{v}^{(i)}(n+1) = {} & w^{(i)}(n)\mathbf{v}^{(i)}(n) \\ & + c_1^{(i)}(n)\mathbf{r}_1^{(i)}(n)[\mathbf{x}_p^{(i)}(n) - \mathbf{x}^{(i)}(n)] \\ & + c_2^{(i)}(n)\mathbf{r}_2^{(i)}(n)[\mathbf{x}_g(n) - \mathbf{x}^{(i)}(n)], \\ & n = 0, 1, 2, \ldots, N-1, \end{aligned} \tag{8}$$

subject to correcting iterations (7).

An auxiliary objective function is used to quantify the success of particles as a function of their velocity weights. There are reliable and directly employable entities to measure the success of particles. In particular, we use the improvement in the objective value of the particle [1], the number of updates of the global best position that the particle yielded [18], [20], and the number of updates of the personal best position that the particle yielded. We propose the following objective function for the velocity weights, selected based on good performance in empirical tests:

$$\tilde{f}^{(i)}(n) = e^{(i)}(n)\big(1 + w_l u_l^{(i)}(n) + w_g u_g^{(i)}(n)\big), \quad n = 1, 2, \ldots, N-1, \tag{9}$$

where $\tilde{f}^{(i)}(n)$ is the objective value of the velocity weights for particle i at iteration n, $e^{(i)}(n)$ is the normalized improvement described below, $u_l^{(i)}$ is the number of times particle i updated its personal best position, $u_g^{(i)}$ is the number of times particle i updated the global best position, w_l is the local weight factor used to weigh the number of personal best updates $u_l^{(i)}$, and w_g is the global weight factor used to weigh the number of global best updates $u_g^{(i)}$. The value of w_g is usually set to a larger number than the value of w_l because updates to the global best position are relatively more important. Equation (9) is thus used to guide the evolution of the positions of the velocity weights towards optimal values. Alternative objective functions are possible, e.g., ones that use the normalized improvements $e^{(i)}(n)$ or the local and global best update counters individually. The normalized improvements $e^{(i)}(n)$ are calculated as follows, based on good performance in empirical tests:

$$e^{(i)}(n) = \frac{\delta^{(i)}(n)}{\sigma(n)}, \tag{10a}$$

where $\sigma(n)$ is the normalization sum described below and $\delta^{(i)}(n)$ is the difference in the objective values calculated using:

$$\delta^{(i)}(n) = f^{(i)}(n) - f^{(i)}(n-1), \tag{10b}$$

where $f^{(i)}$ is the objective value of particle i.

In practice, early iterations might yield large absolute values of $\delta^{(i)}$, whereas late iterations might only yield small absolute values of $\delta^{(i)}$. Therefore, we propose the following normalization to fairly account for the contribution of the velocity weights from late iterations:

$$\sigma(n) = \begin{cases} \sum_{i=1}^{n_p} -\delta^{(i)}(n), \text{ for } \delta^{(i)}(n) < 0, \\ 1, \text{ otherwise.} \end{cases} \tag{10c}$$

In other words, the normalization sum $\sigma(n)$ makes objective values of the velocity weights comparable for different n. This normalization is selected based on good performance in tests.

The velocity weights are optimized using one step of PSO in an inner iteration, resulting in the following overall iteration to update the positions of the velocity weights:

$$\tilde{\mathbf{x}}^{(i)}(n+1) = \tilde{\mathbf{x}}^{(i)}(n) + \tilde{\mathbf{v}}^{(i)}(n+1), \quad n = 1, 2, \ldots, N-1, \tag{11a}$$

$$\begin{aligned} \tilde{\mathbf{v}}^{(i)}(n+1) = \tilde{w}(n)\tilde{\mathbf{v}}^{(i)}(n) + \tilde{c}_1(n)\tilde{\mathbf{r}}_1^{(i)}(n)[\tilde{\mathbf{x}}_p^{(i)}(n) - \tilde{\mathbf{x}}^{(i)}(n)] \\ + \tilde{c}_2(n)\tilde{\mathbf{r}}_2^{(i)}(n)[\tilde{\mathbf{x}}_g^{(i)}(n) - \tilde{\mathbf{x}}^{(i)}(n)], \\ n = 1, 2, \ldots, N-1, \end{aligned} \tag{11b}$$

where $\tilde{\mathbf{x}}^{(i)}(n)$ is the position of the velocity weights, $\tilde{\mathbf{v}}^{(i)}(n)$ is the velocity of the velocity weights, $\tilde{\mathbf{x}}_p^{(i)}(n)$ is the personal best position of the velocity weights, $\tilde{\mathbf{x}}_g^{(i)}(n)$ is the global best position of the velocity weights, $\tilde{w}(n)$ is the inertia weight for optimizing the velocity weights, $\tilde{c}_1(n)$ is the personal best weight for optimizing the velocity weights, $\tilde{c}_2(n)$ is the global best weight for optimizing the velocity weights, and $\tilde{\mathbf{r}}_1^{(i)}$ and $\tilde{\mathbf{r}}_2^{(i)}(n)$ are random vectors with components that are uniformly distributed between 0 and 1 for every particle i and iteration n. Equations (11) are used after Equation (1a) and (8) have been used to update the positions of the particles and the new objective values have been calculated. The first component of $\tilde{\mathbf{x}}^{(i)}(n)$ is used as the inertia weight $w^{(i)}(n)$, the second component of $\tilde{\mathbf{x}}^{(i)}(n)$ is used as the personal best weight $c_1^{(i)}(n)$, and third component of $\tilde{\mathbf{x}}^{(i)}(n)$ is used as the global best weight $c_2^{(i)}(n)$ in Equation (8). These iterations are subject to correcting iterations (7) with a corresponding reduction factor denoted by $\tilde{\alpha}$.

As in other APSO variants, e.g., [13], SOPSO incorporates mutation. Specifically, a certain percentage of particles are selected for mutation. Additionally, the local and global best positions and values of the velocity weights are reset to the current positions and values of the velocity weights after a specified number of iterations. If a particle is selected for mutation, its velocity weights are reinitialized within a reinitialization space of the velocity weights. Mutation is used to maintain diverse behavior of particles and to treat the dynamic optimization problem of optimizing the velocity weights. The problem of optimizing the velocity weights is a dynamic optimization problem because the objective function for the velocity weights itself changes during the solution process; e.g., at times the objective function for the velocity weights may promote diversity whereas at others it may promote convergence, further enhancing the utility of the SOPSO approach.

Figure 1 shows the flowchart of SOPSO. Steps shaded in light yellow are common for standard PSO, and steps shaded in yellow are introduced for SOPSO. First, the swarm is initialized; i.e., the particle positions and velocities are randomly initialized within the search space and the positions and velocities of the velocity weights are randomly initialized in the initialization space of the velocity weights. The objective values of the particles are calculated. All particles are moved to their new positions using velocities calculated in Equation (8) subject to correcting iterations (7). All objective values are evaluated again. If there are particles with objective values better than their personal best value, their personal best positions and values are updated and their local best update counter u_l is increased by 1. The global best update counter u_g is increased for any particles that updated their global best position and value. After that, the objective values of the velocity weights are calculated for all particles using Equation (9). The first objective values and positions of the velocity weights are automatically the personal best values and positions of the velocity weights. The global best position and value of the velocity weights are selected by finding the particle with the best objective value of the velocity weights. The positions of the velocity weights are updated using Equations (11) subject to the appropriate correcting iterations (7). The positions of velocity weights used by particles that were selected for mutation are randomly reinitialized in a pre-

scribed reinitialization space. If the termination criterion is not satisfied, the algorithm continues with updating the positions, evaluating the objective values, and updating the personal best values and positions, the local best update counters, the global best position and value, and the global best update counter. Personal best positions and values of the velocity weights are only updated if the new objective values of the velocity weights are improved. The global best position and value of the velocity weights are updated if the particle with the best objective value of the velocity weights has an improved value for the global best value of the velocity weights. The positions of the velocity weights used by particles that were not selected for mutation are updated using Equations (11), and the positions of the velocity weights used by particles that were selected for mutation are randomly reinitialized in a prescribed reinitialization space. If the prescribed number of iterations after which to reset the local and global best velocity weights is reached, the local and global best positions and values of the velocity weights are reset to the current positions and objective values of the velocity weights. The algorithm stops when a termination criterion is satisfied.

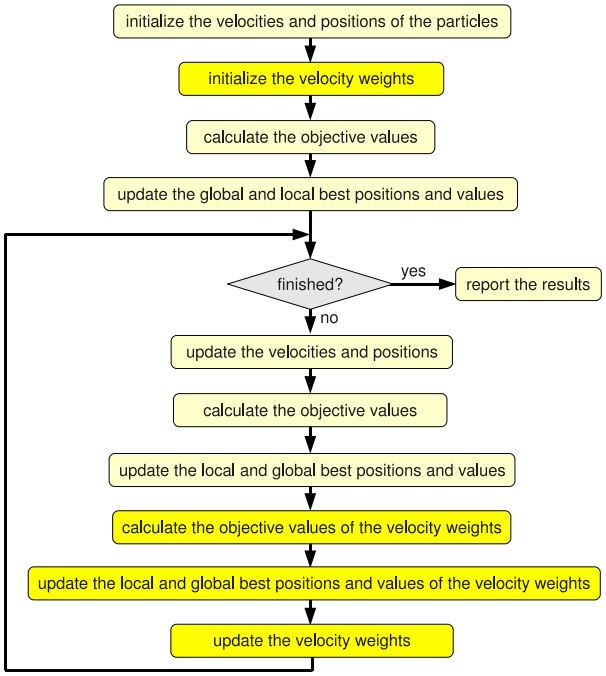

Fig. 1. Flowchart of SOPSO.

IV. Experiments and Results

A. Test Problems

Twenty-two optimization test problems are used to compare our SOPSO with the four known PSO variants. The problems are chosen as follows. All the test problems from the semi-continuous challenge [2] are used, including the Ackley test problem, the Alpine test problem, the Griewank test problem, the Parabola test problem, the Rosenbrock test problem, and the Tripod test problem. Some of the optimization test problems from [9] have been selected based on their shapes to guarantee a diverse set of problems, including the Six-hump Camel Back test problem, the De Jong 5 test problem, the Deceptive test problem, the Drop Wave test problem, the Easom test problem, the Goldstein–Price test problem, the Axis Parallel Hyper-ellipsoid test problem, the Michalewicz test problem, and the Shubert test problem [2]. We also use optimization test problems from [21], where we selected diverse test problems to expand our test set. These include the Generalized Penalized test problem, the Non-continuous Rastrigin test problem, the Rastrigin test problem, the Schwefel's P2.22 test problem, the Sphere test problem, and the Step test problem [21]. We also use Schaffer's F6 test problem from [8]. For ease of comparison, we normalize the test problems to have a global optimum of 0.

Table I lists the test functions, brief descriptions of their properties, bounds on the components of $\mathbf{x}$, and the search space dimension.

B. Parameter Settings

Unless stated otherwise, the parameters are set to the values described in Table II.

C. Experimental Setup

We compare all PSO variants on four test cases using four different fixed numbers of function evaluations. The first test uses $n_p = 100$ particles and $N = 75,000$ iterations for a total of $7,500,000$ function evaluations (including initialization). This test is meant to represent a relatively large number of function evaluations, as is often required to solve the most difficult problems. The second test uses $n_p = 100$ particles and $N = 15,000$ iterations for a total of $1,500,000$ function evaluations (including initialization). This test is meant to represent a relatively moderate number of function evaluations. The third test uses $n_p = 100$ particles and $N = 2,500$ iterations for a total of 250,000 function evaluations (including initialization). This test is meant to represent a relatively small number of function evaluations. The fourth test uses $n_p = 30$ particles and $N = 100$ iterations for a total of $3,000$ function evaluations (including initialization). This test is meant to represent an extremely small number of function evaluations, as in the case where function evaluations are expensive. All calculations are performed in double precision.

D. Results

Optimization techniques are often compared in terms of the number of function evaluations required to find a given optimum or the best objective value found for a specified number of function evaluations [16]. We follow the latter approach of reporting the best objective function value given a certain number of function evaluations. We focus on results given for relatively large numbers of function evaluations (FE) because the proposed SOPSO variant was developed with a focus on efficiently and effectively solving difficult problems that require high numbers of function evaluations for their

TABLE I
DESCRIPTION OF TEST PROBLEMS.

Name	**Feature**	$[x_{\min}, x_{\max}]$	D
Ackley	global optimum has narrow neighborhood	[-30,30]	30
Alpine	multiple local optima	[-10,10]	10
Six-hump Camel Back	six local minima are global minima	[-2,2]	2
De Jong 5	multimodal	[-65.536,65.536]	2
Deceptive Type 3	$3^{30} - 1$ local optima	[0,1]	30
Drop Wave	multimodal	[-5.12,5.12]	2
Easom	one minima	[-100,100]	2
Generalized Penalized	multimodal	[-50,50]	30
Griewank	many local minima with small neighborhoods	[-300,300]	30
Goldstein–Price	unimodal	[-2,2]	2
Axis Parallel Hyper-ellipsoid	local optima	[-5.12,5.12]	100
Michalewicz	$D!$ local optima	$[0, \pi]$	10
Non-continous Rastrigin	multimodal	[-5.12,5.12]	30
Parabola	convex	[-20,20]	200
Rastrigin	many local minima	[-10,10]	30
Rosenbrock	misleadingly flat towards global optimum	[-10,10]	30
Schaffer's F6	multimodal	[-100,100]	2
Schwefel's P2.22	unimodal	[-10,10]	30
Shubert	multimodal	[-10,10]	2
Sphere	unimodal	[-100,100]	100
Step	unimodal	[-100,100]	30
Tripod	discontinuous	[-100,100]	2

solution. Specifically we look at the minimum solutions, i.e., the solutions with the lowest value of the objective function, from three runs using different random seeds.

Tables III–VI compare the minimum solutions found by the implemented known PSO and proposed SOPSO variants in terms of wins, draws, and losses. A given variant earns a *win* if it is the only variant to find the best solution, a *draw* if it and at least one other variant find the best solution, and a *loss* if another variant finds the best solution. These tables compare five different solvers at the same time. We note that wins, draws, and losses do not provide information on the values of the objective function values that yielded them. Whether the global optimum is achieved and the margin of victory for winning methods can be found in Tables VII–X.

Table III compares performance of the implemented known PSO and proposed SOPSO variants for $7,500,000$ FE. The proposed SOPSO variant wins this comparison by scoring 2 wins and suffering no losses. Table IV compares the performance of the variants for $1,500,000$ FE. SOPSO performs best in this comparison by scoring 3 wins and suffering only 1 loss. Table V compares the performance of the variants for $250,000$ FE. GBPSO wins this comparison by scoring 3 wins and only 4 losses. Table VI compares performance of the variants for $3,000$ FE. TVACPSO wins this comparison by scoring 12 wins and only 5 losses.

Tables VII–X give more details on problems for which not all of the variants gave the same answer; i.e., at least one solver did not score a draw. In practice, if all solvers score a draw on a problem, they have all achieved the global optimum.

Table VII compares the performance of the variants for $7,500,000$ FE. In this case, SOPSO is the only variant to find the global optimum for all the problems considered in this study. Other variants are found to not perform as well on a consistent basis. Based on these results, we recommend the use of SOPSO for solving problems that require a relatively large number of function evaluations for their solution.

Table VIII compares the performance of the variants for $1,500,000$ FE. In this case, two of the problems (Rastigrin and Rosenbrock) were not solved by any of the variants for this number of FE. The SOPSO variant shows the best consistent performance, finding the global minimum to within 4 decimal places even in its one loss.

Table IX compares the performance of the variants for $250,000$ FE. In this case, GBPSO and TVACPSO show the best results. However, the data are somewhat mixed: every method underperforms on at least one problem shown. Nonetheless, we generally recommend the use of GBPSO or TVACPSO for problems requiring a moderate number of function evaluations.

Table X compares the performance of the variants for $3,000$ FE. In this case, TVACPSO consistently shows the best results, generally performing relatively well even in its losses. Based on these results, we recommend TVACPSO for solving problems when relatively few FE are available.

Overall we see that the proposed SOPSO variant is most effective when large numbers of FE are available or as the difficulty level, as measured by the number of FE required to find the global optimum, of the problem grows.

V. CONCLUSION

We proposed a Step-Optimized PSO (SOPSO) algorithm in which every particle has its own velocity weights; the settings of these velocity weights are themselves optimized by taking one iteration of PSO for every particle at every iteration. We compared the performance of SOPSO to four known PSO variants (global best PSO, decreasing weight PSO, time-varying acceleration coefficients PSO, and guaranteed convergence PSO). The optimization of the velocity weights in the

TABLE VII
COMPARISON FOR 7,500,000 FE.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO	SOPSO
Griewank	**0.0**	0.0073960403	**0.0**	0.0073960403	**0.0**
Michalewicz	0.2525725259	**0.0**	**0.0**	**0.0**	**0.0**
Non-continuous Rastrigin	6.0	**0.0**	3.0	0.0000000369	**0.0**
Rastrigin	25.8688951364	1.9899181142	3.9798362284	0.9949590571	**0.0**
Rosenbrock	0.0000114199	7.3443347199	13.9586009312	16.0244141123	**0.0**

TABLE VIII
COMPARISON FOR 1,500,000 FE.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO	SOPSO
Griewank	**0.0**	**0.0**	0.0073960403	**0.0**	**0.0**
Michalewicz	0.2525725259	0.0049111478	**0.0**	**0.0**	**0.0**
Non-continuous Rastrigin	6.0	4.0	11.0	1.0	**0.0**
Parabola	**0.0**	0.0000002112	0.0030505369	0.0000000870	0.0000215830
Rastrigin	25.8688951364	8.9546315138	10.9445445902	9.9495855331	**4.9747952855**
Rosenbrock	0.0954049541	19.6918183861	3.9869199112	4.1464319620	**0.0000071796**

TABLE IX
COMPARISON FOR 250,000 FE.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO	SOPSO
Hyper-ellipsoid	**0.0000001135**	0.0061106210	0.0034030448	0.0024181041	0.0008089862
Michalewicz	0.2574836737	**0.0**	**0.0**	**0.0**	**0.0**
Non-continuous Rastrigin	6.0	16.0000003717	22.0	21.0017979253	**1.0**
Parabola	**9.3296371480**	16.6223866133	17.7093667245	9.4273873954	41.8289099310
Rastrigin	25.8688951364	15.9193398757	**11.9395086851**	24.8739663523	15.9193401243
Rosenbrock	15.6856726242	18.4717862358	**8.2346336506**	22.6233668383	18.6604566082
Sphere	**0.0000000647**	0.2551991869	0.1273730986	0.0402738382	0.0025875183

proposed SOPSO variant is based on the objective function in Equation (9), the proposed normalization of the improvements in the objective values in Equations (10), and the concept of taking one (inner) PSO iteration toward optimizing the velocity weights after every (outer) PSO iteration.

Based on the observed results, we conclude that the proposed SOPSO variant performs best for relatively high numbers of function evaluations such as are required for the solution of difficult optimization problems. For such problems, the concept of stepping toward optimizing the velocity weights improves the performance of PSO. When relatively few function evaluations are available, the known TVACPSO variant generally outperforms the proposed SOPSO variant, indicating that the optimization of velocity weights is most useful for objective functions with extremely difficult topographies.

REFERENCES

[1] X. Cai, Z. Cui, J. Zeng, and Y. Tan. Self-learning particle swarm optimization based on environmental feedback. *Innovative Computing, Information and Control, 2007. ICICIC '07*, page 570, 2007.

[2] M. Clerc. Semi-continuous challenge, http://clerc.maurice.free.fr/pso/semi-continuous_challenge/, April 2004.

[3] A. Engelbrecht. *Computational Intelligence — An Introduction 2nd Edition*. Wiley, 2007.

[4] A. Ide and K. Yasuda. A basic study of adaptive particle swarm optimization. *Denki Gakkai Ronbunshi / Electrical Engineering in Japan*, 151(3):41–49, 2005.

[5] J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceedings, IEEE International Conference on Neural Networks*, 4:1942–1948, 1995.

[6] X. Li, H. Fu, and C. Zhang. A self-adaptive particle swarm optimization algorithm. *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, pages 186–189, 2008.

[7] H. Liu. Fuzzy adaptive turbulent particle swarm optimization. *Proceedings IEEE Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, 2005.

[8] M. Meissner, M. Schmuker, and G. Schneider. Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, 7(1):125, 2006.

[9] M. Molga and C. Smutnicki. Test functions for optimization needs, http://zsd.iiar.pwr.wroc.pl/, 2005.

[10] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1:33–57, 2007.

[11] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transaction on Evolutionary Computation*, 8(3):240–255, 2004.

[12] Y. Shi and R. Eberhart. A modified particle swarm optimizer. *Proceedings of the IEEE World Congress on Computation Intelligence*, pages 69–73, 1998.

[13] J. Tang and X. Zhao. Particle swarm optimization with adaptive mutation. *2009 WASE International Conference on Information Engineering (ICIE)*, 2:234–237, 2009.

[14] F. Van den Bergh and A.P. Engelbrecht. A new locally convergent particle swarm optimiser. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 3:94–99, 2002.

[15] C. K. Monson, K. D. Seppi. Exposing Origin-Seeking Bias in PSO. *Proceedings of GECCO'05*, pp. 241-248, 2005.

[16] Kumar, Nagesh D and Reddy, Janga M Multipurpose Reservoir Operation Using Particle Swarm Optimization. *Journal of Water Resources Planning and Management*, 133 (3). pp. 192-201.

[17] D. Wong. Fortran data types and specification statements, http://www-classes.usc.edu/engr/ce/108/text/fbk01.htm, 2011.

[18] T. Yamaguchi, N. Iwasaki, and K. Yasuda. Adaptive particle swarm optimization using information about global best. *IEEE Transactions on Electronics, Information and Systems*, 126:270–276, 2006.

[19] X. Yang, J. Yuan, J. Yuan, and H. Mao. A modified particle swarm optimizer with dynamic adaptation. *Applied Mathematics and Computation*, 189(2):1205–1213, 2007.

[20] K. Yasuda, K. Yazawa, and M. Motoki. Particle swarm optimization with

TABLE X
COMPARISON FOR 3, 000 FE.

Test Problem	GBPSO	DWPSO	TVACPSO	GCPSO	SOPSO
Ackley	2.7864164356	2.7872954929	**1.7535935539**	3.4478779286	4.1442049449
Alpine	0.0053181546	0.0285053980	0.0007749829	0.0243886490	**0.0003859068**
Drop Wave	**0.0**	0.0000000008	**0.0**	0.0000000001	**0.0**
Easom	0.0000000957	0.0000000091	**0.0**	0.0000000184	0.0000000003
Gen. . . Penalized	15.5799816557	23.2912583962	**5.7588533203**	19.6150447267	11.9013451744
Griewank	1.3657400968	1.9707144262	**1.0743647829**	2.3992904992	1.4105078803
Hyper-ellipsoid	1838.0238219300	2132.7192963100	**1466.6333948600**	1706.0779538200	2650.1932225500
Michalewicz	0.9988765596	**0.5441955682**	0.8847682876	0.9090513559	1.3477754937
Non. . . Rastrigin	79.9054137431	96.2457432624	74.0262751250	**72.5516666727**	81.1015672036
Parabola	3127.9040880900	3576.9163285500	**2530.0733023200**	2977.8936030400	33390.4652516200
Rastrigin	141.0597885160	182.3730264280	**84.1419055611**	123.8922789690	175.8517446580
Rosenbrock	310.9464581860	1455.3539688200	**262.9586718660**	975.3149869380	1029.2307493200
Schaffer F6	**0.0000001510**	0.0097159308	0.0000003462	0.0097159099	0.0003885798
Schwefel P2.22	4.1215510202	6.1703772446	**2.7045438343**	7.6256666346	9.5389890792
Shubert	0.0000000242	0.0000000110	0.0000000006	0.0000000306	**0.0**
Sphere	16637.4774400000	16456.0655813000	**14235.7231372000**	21493.2244294000	16294.3398457000
Step	212.0	878.0	**48.0**	461.0	527.0
Tripod	0.0000762325	1.0000070836	**0.0000008724**	0.0000162968	0.0000152553

parameter self-adjusting mechanism. *IEEE Transactions on Electrical and Electronic Engineering*, 5(2):256–257, 2010.

[21] Z.H. Zhan, J. Zhang, Y. Li, and H.S.H. Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(6):1362–1381, 2009.

[22] J. Zhu, J. Zhao, and X. Li. A new adaptive particle swarm optimization algorithm. *International Workshop on Modelling, Simulation and Optimization*, pages 456–458, 2008.

TABLE II
PARAMETER SETTINGS.

Algorithm	Parameter and value
GBPSO	inertia weight $w = 0.7298$ [10] personal best weight $c_1 = 1.49618$ [10] global best weight $c_2 = 1.49618$ [10]
DWPSO	inertia weight for the first iteration $w_s = 0.9$ [3] personal best weight $c_1 = 2$ [12] global best weight $c_2 = 2$ [12]
TVACPSO	inertia weight for first iteration $w_s = 0.9$ [3] inertia weight for last iteration $w_e = 0.4$ [3] personal best weight for first iteration $c_{1s} = 2.5$ [11] personal best weight for last iteration $c_{1e} = 0.5$ [11] global best weight for first iteration $c_{2s} = 0.5$ [11] global best weight for last iteration $c_{2e} = 2.5$ [11]
GCPSO	inertia weight for the first iteration $w_s = 0.9$ [3] inertia weight for the last iteration $w_e = 0.4$ [3] personal best weight $c_1 = 2$ [14] global best weight $c_2 = 2$ [14] initial search radius argument $\rho(0) = 1$ [14] failure threshold $\varphi_c = 5$ [14] success threshold $\sigma_c = 15$ [14]
SOPSO	search space for velocity weights: $[-0.5, 2.0]$ search space for personal best weights: $[-1.0, 4.2]$ search space for global best weights: $[-1.0, 4.2]$ weight factor for local updates $w_l = 1$ weight factor for global updates $w_g = 6$ inertia weight for optimizing velocity weights for first iteration $w_s = 0.9$ inertia weight for optimizing velocity weights for last iteration $w_e = 0.4$ personal best weight for optimizing velocity weights for first iteration $c_{1s} = 2.5$ personal best weight for optimizing velocity weights for last iteration $c_{1e} = 0.5$ global best weight for optimizing velocity weights for first iteration $c_{2s} = 0.5$ global best weight for optimizing velocity weights for last iteration $c_{2e} = 2.5$ percent of particles selected for mutation of their velocity weights: 33 iterations before resetting best positions and velocity weights: 50 initialization space for inertia weights: $[0.4, 0.9]$ initialization space for personal best weights: $[0.5, 2.5]$ initialization space for global best weights: $[0.5, 2.5]$ reinitialization space for inertia weights: $[0.5, 0.8]$ reinitialization space for personal best weights: $[0.6, 2.4]$ reinitialization space for global best weights: $[0.6, 2.4]$ correction factor for velocity weights $\tilde{\alpha} = 0.5$

TABLE III
COUNT OF WINS, DRAWS, AND LOSSES FOR $7,500,000$ FE.

Solver	win	draw	loss
GBPSO	0	18	4
DWPSO	0	19	3
TVACPSO	0	19	3
GCPSO	0	18	4
SOPSO	2	20	0

TABLE IV
COUNT OF WINS, DRAWS, AND LOSSES FOR $1,500,000$ FE.

Solver	win	draw	loss
GBPSO	1	17	4
DWPSO	0	17	5
TVACPSO	0	17	5
GCPSO	0	18	4
SOPSO	3	18	1

TABLE V
COUNT OF WINS, DRAWS, AND LOSSES FOR $250,000$ FE.

Solver	win	draw	loss
GBPSO	3	15	4
DWPSO	0	16	6
TVACPSO	2	16	4
GCPSO	0	16	6
SOPSO	1	16	5

TABLE VI
COUNT OF WINS, DRAWS, AND LOSSES FOR $3,000$ FE.

Solver	win	draw	loss
GBPSO	1	5	16
DWPSO	1	4	17
TVACPSO	12	5	5
GCPSO	1	4	17
SOPSO	2	5	15