

Steady state particle swarm

Carlos M. Fernandes ^{Corresp., 1}, **Nuno Fachada** ^{1,2}, **Juan Juilán Merelo** ³, **Agostinho C. Rosa** ¹

¹ University of Lisbon, Institute for Systems and Robotics (ISR/IST), LARSyS, Instituto Superior Técnico, Lisbon, Portugal

² Universidade Lusófona de Humanidades e Tecnologias, HEI-LAB - Digital Human-Environment and Interactions Labs, Lisbon, Portugal

³ Department of Computer Architecture, University of Granada, Granada, Spain

Corresponding Author: Carlos M. Fernandes

Email address: cfernandes@laseeb.org

This paper investigates the performance and scalability of a new update strategy for the Particle Swarm Optimization (PSO) algorithm. The strategy is inspired by the Bak-Sneppen model of co-evolution between interacting species, which is basically a network of fitness values (representing species) that change over time according to a simple rule: the least fit species and its neighbors are iteratively replaced with random values. Following these guidelines, a steady state update strategy for PSO algorithms is proposed: only the least fit particle and its neighbors are updated and evaluated in each time-step; the remaining particles maintain the same position and fitness, unless they meet the update criterion. The steady state PSO (SS-PSO) was tested on a set of unimodal, multimodal, noisy and rotated benchmark functions, significantly improving the quality of results and convergence speed of the standard PSO. A sensitivity analysis of the parameters confirms the performance enhancement on different parameter settings and scalability tests show that the algorithm behavior is consistent throughout a substantial range of solution vector dimensions. The strategy has also been compared to state-of-the-art PSOs with promising results. Finally, a parallel SS-PSO has been implemented and compared to the standard PSO.

Steady State Particle Swarm

Carlos M. Fernandes¹, Nuno Fachada^{2,1}, J.J. Merelo³, Agostinho C. Rosa¹

¹ LARSyS: Laboratory for Robotics and Systems in Engineering and Science, University of Lisbon, Lisbon, Portugal

² HEI-LAB - Digital Human-Environment and Interactions Labs, Universidade Lusófona de Humanidades e Tecnologias, Lisbon, Portugal

³ Department of Computer Architecture, University of Granada, Granada, Spain

Corresponding Author:

Carlos M. Fernandes¹

Av. Rovisco Pais, 1, Torre Norte, 6.21, Lisboa, Portugal

Email address: cfernandes@laseeb.org

Abstract

This paper investigates the performance and scalability of a new update strategy for the Particle Swarm Optimization (PSO) algorithm. The strategy is inspired by the Bak-Sneppen model of co-evolution between interacting species, which is basically a network of fitness values (representing species) that change over time according to a simple rule: the least fit species and its neighbors are iteratively replaced with random values. Following these guidelines, a steady state update strategy for PSO algorithms is proposed: only the least fit particle and its neighbors are updated and evaluated in each time-step; the remaining particles maintain the same position and fitness, unless they meet the update criterion. The steady state PSO (SS-PSO) was tested on a set of unimodal, multimodal, noisy and rotated benchmark functions, significantly improving the quality of results and convergence speed of the standard PSO. A sensitivity analysis of the parameters confirms the performance enhancement on different parameter settings and scalability tests show that the algorithm behavior is consistent throughout a substantial range of solution vector dimensions. The strategy has also been compared to a state-of-the-art PSO with promising results. Finally, a parallel SS-PSO has been implemented and compared to the standard PSO.

Introduction

Particle Swarm Optimization (PSO) is a social intelligence model for optimization and learning (*Kennedy & Eberhart, 1995*) that uses a set of position vectors (or *particles*) to represent candidate solutions to a specific problem. Every particle is evaluated by computing its fitness, after its speed and position are updated according to local and global information about the search. During the search, the particles move through the fitness landscape of the problem, following a simple set of equations that define the velocity (equation 1) and position (equation 2) of each particle in each time step and drive them heuristically towards optimal regions of a D -dimensional search space. Here, equations 1 and 2 describe a variant proposed by *Shi and Eberhart (1999)* that is widely used in PSO implementations. The difference to the original PSO

- -

is the introduction of the inertia weight parameter ω in order to help (together with c_1 and c_2) fine-tuning the balance between local and global search. All PSO implementations in this paper use inertia weight. The velocity $v_{i,d}$ and position $x_{i,d}$ of the d -th dimension of the i -th particle are therefore updated as follows:

$$v_{i,d}(t) = \omega v_{i,d}(t-1) + c_1 r_{1,d}(p_{best_{i,d}} - x_{i,d}(t-1)) + c_2 r_{2,d}(g_{best_{i,d}} - x_{i,d}(t-1)) \quad (1)$$

$$x_{i,d}(t) = x_{i,d}(t-1) + v_{i,d}(t) \quad (2)$$

where $\vec{X}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,D})$ is the position vector of particle i ; $\vec{V}_i = (v_{i,1}, v_{i,2}, \dots, v_{i,D})$ is the velocity of particle i ; $\vec{p}_{best_i} = (p_{best_{i,1}}, p_{best_{i,2}}, \dots, p_{best_{i,D}})$ is the best solution found so far by particle i ; $\vec{g}_{best_i} = (g_{best_{i,1}}, g_{best_{i,2}}, \dots, g_{best_{i,D}})$ is the best solution found so far by the neighborhood of particle i . The neighborhood of a particle is defined by the network configuration that connects the population and structures the information flow. Parameters $r_{1,d}$ and $r_{2,d}$ are random numbers uniformly distributed within the range $[0,1]$ and c_1 and c_2 are the acceleration coefficients, which are used to tune the relative influence of each term of the formula.

Most of the PSOs use one of two simple sociometric principles for constructing the neighborhood network (which defines the \vec{g}_{best_i} values). *Gbest* (where g stands for global) connects all the members of the swarm to one another. The degree of connectivity of *gbest* is $k = n$, where n is the number of particles. *Lbest* (where l stands for local), creates a neighborhood with the particle itself and its k nearest neighbors. A particular case of the *lbest* topology is the *ring* structure, in which the particles are arranged in a ring, with a degree of connectivity $k = 3$, including the particle itself. Between the $k = 3$ connectivity of *lbest* ring and $k = n$ of *gbest*, there are several possibilities. Two of the most used are the 2-dimensional square lattices with von Neumann and Moore neighborhoods.

Usually, PSOs are synchronous, meaning that first, the fitness values of all vectors must be computed, and only then their velocity is updated. However, there is another possible approach, in which the velocity of the particles is updated immediately after computing the fitness. In this case, the particles move with incomplete knowledge about the global search: if, for instance, the underlying network connecting the particles is a regular graph, then, on average, each particle is updated knowing the current best position found by half of its neighbors and the previous best found by the other half. This variant, which is called asynchronous PSO (A-PSO), was tested by **Carlisle and Dozier (2001)**. In the paper, the authors claim that A-PSO yields better results than the synchronous version (i.e., S-PSO), but since then other authors reached different conclusions: **Engelbrecht (2013)** and **Rada-Vilela et al. (2013)**, for instance, reported that S-PSO is better than A-PSO in terms of the quality of the solutions and convergence speed.

The importance of investigating update strategies for PSO lies in the possibility of distributed computation (McNabb, 2014). Even though standard PSOs can be easily parallelized – a particle or a set of particles can be assigned to each processor, for instance –, load imbalances may cause an inefficient use of the computational resources if synchronous updates are used. Asynchronous strategies do not require that all particles in the population have perfect knowledge about the search before the update step (a requirement that may cause idle processor times in a synchronous implementation), and therefore are a valid approach for parallelizing particle swarms. In addition, asynchronism can also be useful in preventing premature convergence (Aziz et al., 2014), or to speed up convergence by skipping function evaluations (Majercik, 2013).

Here, we are mainly concerned with performance issues, in general, and convergence speed in particular. The goal is to design an asynchronous PSO that, unlike the standard A-PSO, significantly improves on the performance of S-PSO in a wide range of problems. We hypothesize that reducing the number of evaluations in each time step, while focusing only on harder cases (i.e., worst solutions), reduces the number of evaluations required to converge to a specific criterion, i.e., the computational effort to reach a solution. With that objective in mind, we have designed and implemented a novel strategy for one of the fundamental mechanisms of PSO: the velocity update strategy. Following the nature of the method, the algorithm has been entitled *steady state PSO* (SS-PSO).

In systems theory, a system is said to be in steady state when some of its parts do not change for a period of time (Baillieul & Samad, 2015). SS-PSO only updates and evaluates a fraction of the population in each time step: the worst particles and its neighbors. The other particles remain in the same position until they eventually fulfill the criterion (being the worst particle or one of its neighbors). The strategy was inspired by the Bak-Sneppen model of co-evolution between interacting species and by the theory of Self-Organized Criticality (SOC) (Bak & Sneppen, 1993).

SOC is a property of some systems that have a critical point as an attractor. However, unlike classical phase transitions, where a parameter needs to be tuned for the system to reach critical point, SOC systems spontaneously reach that critical state between order and randomness. In a SOC system near the critical point, small disturbances can cause changes of all magnitudes. These events, which are spatially or temporally spread through the system, are known as *avalanches*.

Avalanches occur independently of the initial state. Moreover, the same perturbation may cause small or large avalanches, depending on the current state of the system – i.e., its proximity to the critical point. The distribution of avalanches during a large period displays a power-law between their size and frequency: small avalanches occur very often while large events that reconfigure almost the entire system are scarcer. SOC complex systems balance between stability and creative destruction. In fact, power-law relationships between the size of events and their frequency, one of SOC's signatures, are widespread in Nature. Earthquake distribution, for instance, follows the Gutenberg-Richter law (Gutenberg & Richter, 1956), a power-law proportion

between the magnitude of the earthquakes that occurred in a specific area during a specific period of time, and the frequency of those earthquakes.

SOC was studied for the first time in the sandpile model (*Bak et al., 1987*). Since then, the concept has been extended to other complex systems: besides the aforementioned earthquakes, the proponents of the theory claim that SOC may be a link between a broad range of phenomena, like forest-fires, ecosystems, financial markets and the brain (*Bak, 1996*). One of such systems is the Bak-Sneppen model of co-evolution between interacting species (*Bak & Sneppen, 1993*).

The Bak-Sneppen model was developed with the main objective of trying to understand the mechanisms underlying mass extinctions in nature. Ecosystems are complex adaptive systems in which the agents (the natural species) are related through several features, like food chains or symbiosis, for instance. In such interconnected environments, the extinction of one species affects the species that are related to it, in a chain reaction that can be of any size: in fact, fossil records suggest that the size of extinction outbreaks is in power-law proportion to their frequency.

In order to model the extinction patterns in nature and search for SOC signatures in co-evolutionary systems, *Bak and Sneppen (1993)* structured a set of species in a ring network and assigned a fitness value to each. Then, in every time step, the least fit species and its neighbors are eliminated from the system and replaced by individuals with random fitness. To put it in mathematical terms, the system is defined by n^d fitness values f_i arranged on a d -dimensional lattice (ecosystem) with n cells. At each time step, the smallest f value and its $2 \times d$ neighbours are replaced by uncorrelated random values drawn from a uniform distribution. Operating with this set of rules, the system is driven to a critical state where most species have reached a fitness value above a certain threshold. Near the critical point, extinction events of all scales can be observed.

SOC theory has been a source of inspiration for metaheuristics and unconventional computing techniques. Extremal Optimization (EO) (*Boettcher & Percurs, 2003*), for example, is based in the Bak-Sneppen model. EO uses a single solution vector that is modified by local search. The algorithm removes the worst components of the vector and replaces them with randomly generated material. By plotting the fitness of the solution, it is possible to observe distinct stages of evolution, where improvement is disturbed by brief periods of dramatic decrease in the quality.

Løvbjerg and Krink (2002) applied modeled SOC in a PSO in order to control the convergence of the algorithm and maintain population diversity. The authors claim that their method is faster and attains better solutions than the standard PSO. However, the algorithm introduces parameters in the standard PSO and overall there are five parameters that must be tuned or set to constant *ad hoc* values.

In this work, the Bak-Sneppen model is used to design an alternative update strategy for the PSO. The strategy has been previously tested on a set of benchmark functions and compared to a standard S-PSO (*Fernandes et al. 2016*). The results show that SS-PSO significantly improves the performance of a S-PSO structured in a 2-dimensional square lattice with Moore

neighborhood. This paper is an extension of the aforementioned work. The main contributions here are: a) a complete statistical analysis of the performance, comparing the algorithm with standard PSOs and variations of the proposed strategy; b) a parameter sensitivity analysis and scalability tests showing that the performance enhancement introduced by the steady-state strategy is maintained throughout a reasonable range of parameter values and search space dimension ranging from 10 to 50; c) a comparison with two state-of-the-art PSOs; and, d) an analysis of a parallel implementation of the algorithm.

Materials & Methods

SS-PSO algorithm

SS-PSO was inspired by a similarity between PSO and the Bak-Sneppen model: both are population models in which the individuals are structured by a network and evolve towards better fitness values. With this likeness in mind, we have devised an asynchronous and steady state update strategy for PSO in which only the least fit particle and its neighbors are updated and evaluated in each time step. Please note that SS-PSO is not an extinction model like the Bak-Sneppen system: the worst particle and its neighbors are not replaced by random values; they are updated according to Equations 1 and 2. As for the other particles, they remain steady – hence the name of the algorithm: Steady-State Particle Swarm Optimization.

Algorithm 1: Steady State Particle Swarm Optimization

```

for all particles  $i \in \{1, 2, \dots, \mu\}$  do
    initialize velocity and position of particle  $i$ 
    compute fitness of particle  $i$ 
end
for all particles  $i \in \{1, 2, \dots, \mu\}$  do
    compute  $pbest$  and  $gbest$  of particle  $i$ 
end
repeat
    update velocity (equation 1) of particle with worst fitness and its neighbors
    update position (equation 2) of particle with worst fitness and its neighbors
    compute fitness of particle with worst fitness and its neighbors
    for all particles  $i \in \{1, 2, \dots, \mu\}$  do
        compute  $pbest$  and  $gbest$  of particle  $i$ 
    end
until termination criterion is met

```

The neighborhood is defined by the social structure. For instance, if the particles are connected by a $lbest$ topology with $k = 3$, then only the worst particle and its two nearest neighbors are updated and evaluated. Please note that local synchronicity is used here: the fitness values of the worst and its neighbors are first computed and only then the particles update their velocity. For

the remaining mechanisms and parameters, the algorithm is exactly as a standard PSO. For a detailed description of SS-PSO, please refer to Algorithm 1.

The PSOs discussed in this paper, including the proposed SS-PSO, are available in the OpenPSO package, which offers an efficient, modular and multicore-aware framework for experimenting with different approaches. The package is implemented in C99, and transparently parallelized with OpenMP (Dagum & Menon, 1998). OpenPSO is composed of three modules:

1. A PSO algorithm library, with the capability of performing parallel function evaluations.
2. A library of benchmarking functions.
3. A command-line tool for directly experimenting with the different PSO algorithms and benchmarking functions.

The library components can be interfaced with other programs and programming languages, making OpenPSO a flexible and adaptable framework for PSO research. Its source code is available at <https://github.com/laseeb/openpso>.

Experimental Setup

For testing the algorithm, ten benchmark problems (Table 1) are used. Functions f_1 - f_3 are unimodal; f_4 - f_8 are multimodal; f_9 is the shifted f_2 with noise and f_{10} is the rotated f_5 (f_9 global optimum and f_{10} matrix were taken from the CEC2005 benchmark). Population size μ is set to 49. This particular value, which lies within the typical range (Kennedy & Eberhart, 1995), was set in order to construct square lattices with von Neumann and Moore neighborhood. Following (Rada-Vilela et al., 2013), c_1 and c_2 were set to 1.494 and ω to 0.7298. X_{max} , the maximum position value, and V_{max} , the maximum velocity value, are defined by the domain's upper limit. Asymmetrical initialization is used, with the initialization ranges in Table 1.

This work reports an extensive study of the proposed methodology. Different kinds of experiments have been performed, each one to investigate different aspects of the steady-state update strategy. The first experiment attempts at a proof-of-concept: SS-PSO is compared with standard (and synchronous) update strategies. The objective of the second experiment is to check if the convergence speed-up is caused in fact by the selective strategy or instead by the restricted evaluation pool, which is a consequence of the proposed method. The third test aims at studying the parameter sensitivity and the scalability with problem size. For that purpose, several tests have been conducted in a wide range of parameter values and problem dimension. The fourth experiment investigates SS-PSO under time-varying parameters and the experiment number five compares SS-PSO with a dynamically structured PSO. Finally, the parallelization potential of SS-PSO is investigated.

Results

Proof-of-concept

The first experiment intends to determine if SS-PSO is able to improve the performance of a standard S-PSO. For that purpose, three S-PSOs with different topologies have been implemented: $lbest$ with $k = 3$ and 2-dimensional square lattices with von Neumann ($k = 5$) and

Moore neighborhood ($k = 9$). $Gbest$ ($k = n$) is not included in the comparisons because SS-PSO uses the neighborhood structure to decide how many and which particles to update: for instance, in the von Neumann topology ($k = 5$), five particles are updated. Since $gbest$ has $k = n$, the proposed strategy would update the entire population, i.e., it would be equivalent to a S-PSO. Therefore, we have restricted the study to $lbest$, von Neumann and Moore structures, labeling the algorithms, respectively, S-PSO_{lbest}, S-PSO_{VN} and S-PSO_{Moore}.

Two sets of experiments were conducted. First, the algorithms were run for a specific amount of function evaluations (49000 for f_1 , f_3 and f_6 , 980000 for the remaining). After each run, the best solution was recorded. Each algorithm was executed 50 times in each function and statistical measures were taken over those 50 runs. In the second set of experiments the algorithms were all run for 980000 function evaluations or until reaching a function-specific stop criterion (given in **Table 1**). A success measure was defined as the number of runs in which an algorithm attains the stop criterion. Again, each algorithm was executed 50 times in each function. This experimental setup is similar to those in (Kennedy & Mendes, 2002) and (Rada-Vilela et al., 2013). The dimension of the functions search space is $D = 30$ (except f_6 , with $D = 2$). The results are in **Error! Reference source not found.** (fitness), **Error! Reference source not found.** (evaluations) and **Table 4** (success rates). The best results among the three algorithms are shown in bold.

When compared to S-PSO_{lbest}, S-PSO_{Moore} attains better solutions (considering median values of fitness distributions over 50 runs) in most of the functions and is faster (considering median values of evaluations required to meet the criteria) in every function. When compared to S-PSO_{VN}, S-PSO_{Moore} is faster in every function and yields better median fitness values in unimodal functions.

In terms of success rates, S-PSO_{Moore} clearly outperforms the other topologies in function f_9 and is much more efficient than S-PSO_{lbest} in function f_4 . These results are consistent with (Kennedy & Mendes, 2002).

The algorithms were ranked by the Friedman test for each function. **Error! Reference source not found.** shows the ranks according to the quality of solutions, while **Error! Reference source not found.** shows the ranks according to the convergence speed (only the functions on which the three algorithms attained the same success rates were considered in the ranking by convergence speed). Overall, S-PSO_{Moore} ranks first in terms of solutions quality and convergence speed – see **Error! Reference source not found.**. Therefore, we conclude that the Moore structure is well suited for assessing the validity and relevance the SS-PSO.

Once the best network has been found for this particular set of problems, the next step was to compare synchronous and asynchronous PSOs on the most efficient topology. For that purpose, we have implemented a SS-PSO_{Moore} and tested it on the 10-function set under the same conditions described above. The results are found in **Error! Reference source not found.**.

Error! Reference source not found. gives a comparison between the performance of S-PSO_{Moore} and SS-PSO_{Moore} based on the numerical results and statistical analysis of those same results. Mann-Whitney tests were performed to compare the distribution of fitness values and

number of evaluations to meet criteria of each algorithm in each function. The ranking of fitness distributions are significant at $p \leq 0.05$ for $f_1, f_2, f_3, f_6, f_7, f_9$, i.e., in these functions, the null hypothesis that the two samples come from the same population is rejected. For the remaining functions (f_5, f_8, f_{10}), the null hypothesis is not rejected: the differences are not significant.

In terms of function evaluations, SS-PSO_{Moore} is faster in the entire set of unimodal problems. In multimodal problems, SS-PSO_{Moore} needs less evaluations in f_5, f_6, f_7 and f_8 . Results of Mann-Whitney tests are significant at $p \leq 0.05$ for functions $f_1, f_2, f_3, f_5, f_7, f_8, f_9$ and f_{10} – see **Error! Reference source not found.**

The success rates are similar, except for f_7 (in which SS-PSO clearly outperforms the standard algorithm) and f_9 . In conclusion: empirical results, together with statistical tests, show that according to accuracy, speed and reliability, SS-PSO_{Moore} outperforms S-PSO_{Moore} in most of the benchmark functions selected for this test, while not being outperformed in any case.

Update Strategy

The preceding tests show that the steady state update strategy when implemented in a PSO structured in a lattice with Moore neighborhood improves its performance. The following experiment aims at answering an important question: what is the major factor in the performance enhancement? Is it the steady state update, or instead the particles that are updated?

In order to investigate this issue, two variants of SS-PSO were implemented: one that updates the best particle and its neighbors (*replace-best*); and another that updates a randomly selected particle and its neighbors (*replace-random*). The algorithms were tested on the same set of benchmark functions and compared the proposed SS-PSO_{Moore} (or *replace-worst*). Results are in **Error! Reference source not found.**

Replace-best update strategy is outperformed by *replace-worst* SS-PSO. With the exception of f_1 and f_3 , the quality of solutions is degraded when compared to the proposed SS-PSO. Success rates are considerably lower in most functions. As for *replace-random*, it improves S-PSO in some functions, but in general is not better than *replace-worst*: *replace-random* SS-PSO is less accurate and slower in most of the functions. The Friedman test shows that SS-PSO with *replace-worst* strategy ranks first in terms of solutions quality – see **Fig. 2**.

Error! Reference source not found. compares *replace-random* and *replace-worst* with the assistance of Mann-Whitney statistical tests. Except for f_4 , *replace-worst* is significantly more efficient than *replace-random*. The experiment demonstrates that selective pressure imposed on the least fit individuals is the major factor in the performance of SS-PSO.

Scalability

The proof-of-concept showed that SS-PSO outperforms S-PSO in most of the functions in the test set, and the previous experiment demonstrate that the major factor in the performance enhancement is the pressure on the least fit particles. However, only instances of the problems with $D = 30$ have been tested; therefore, another question arises at this point: does the

improvement shown by SS-PSO hold for a wide range of problem sizes? In order to answer that question, we have conducted a scalability study: the algorithms were tested on the same set of functions but with D ranging from 10 to 50 (except f_6 , which is a 2-dimensional function and for that reason was excluded from this test).

As in previous experiments, the algorithms were first run for a limited amount of function evaluations and the best fitness values were recorded. Then, the algorithms were all run for 980000 evaluations or until reaching a function-specific stop criterion. The number of iterations required to meet the criterion was recorded and statistical measures were taken over 50 runs. (Function f_{10} has not been tested for dimensions 20 and 40 because the CEC2005 benchmark, from where the orthogonal rotational matrices M have been taken, does not provide the matrices for those dimensions.)

Error! Reference source not found. shows the median best fitness values attained by each algorithm on each instance of the problems and **Error! Reference source not found.** shows the success rates. In terms of quality of solutions, the performance patterns observed with $D = 30$ are maintained: the strategy does not introduce scalability difficulties. As for the success rates, except for a few instances, SS-PSO attains better or equal success rates.

The convergence speed has been graphically represented for better assessment of the effects of growing problem size – see **Error! Reference source not found.**. The graphs show that the proposed strategy does not introduce scalability difficulties (other than the ones intrinsic to standard PSOs). It also shows that, in general, SS-PSO is faster than S-PSO.

Parameter Sensitivity

PSO performance can be severely affected by the parameter values. The inertia weight and acceleration coefficients must be tuned in order to balance exploration and exploitation: if far from the optimal values, convergence speed and/or solution quality can be significantly reduced. Population size also influences the performance of population-based metaheuristics: larger populations help to maintain diversity, but they slow down convergence speed; on the other hand, smaller populations are faster but they are more likely to converge to local optima.

Furthermore, PSOs empirical studies usually depend on a single set of parameters for several functions with different characteristics. This is the case of this paper, in which a typical parameter setting has been used for evaluating the performance of the PSOs. That set of parameter is not expected to be the optimal tuning for every function, but instead a compromised solution to avoid the exponential growth of experimental procedures.

For these reasons, when testing a new PSO, it is important to investigate its sensitivity to the parameter values. With that purpose in mind, the following experimental procedure has been designed.

S-PSO and SS-PSO have been tested on function f_1 (unimodal), f_2 (multimodal), f_9 (shifted and noisy) and f_{10} (rotated) with the following parameter values: inertia weight was set to 0.6798, 0.7048, 0.7298, 0.7548 and 0.7798, while acceleration coefficients and population size remained fixed at 1.494 and 49; then, c_1 and c_2 were set to 1.294, 1.394, 1.494, 1.594 and 1.694

while ω and μ remained fixed at 0.7298 and 49, respectively; finally, population size was set to 36, 49 and 64, while ω and the acceleration coefficients were set to 0.7298 and 1.4962. The results are depicted in **Figures 4-7**.

The graphics show that the performance indeed varies with the parameter values, as expected. In the case of function f_1 , other parameter settings attain better results than the ones used in previous section. However, the relative performance of S-PSO and SS-PSO maintains throughout the parameters ranges. In functions f_8 , f_9 and f_{10} , the quality of solutions is in general maximized by ω and c values around the ones used in previous sections. Convergence speed, in general, improves with lower ω , c and μ values.

Please remember that we are not trying to find the best set of parameters for each function. The most important conclusions here is that SS-PSO does not seem to be more sensitive to the parameters than S-PSO, displaying similar patterns when varying ω , c_1 and c_2 and μ , and that the performance enhancement brought by SS-PSO is observed on a reasonably wide range of parameter values.

Time-varying parameters

An alternative approach to parameter tuning is to let the parameters values change during the run, according to deterministic or adaptive rules. In order to avoid tuning effort and adapt the balance between local and global search to the search stage, *Shi and Eberhart (1999)* proposed a linearly time-varying inertia weight: starting with an initial and pre-defined value, the parameter value decreases linearly with time, until it reaches the minimum value. The variation rule is given by Equation (3):

$$\omega(t) = (\omega_1 - \omega_2) \times \frac{(\max_t - t)}{\max_t} + \omega_2 \quad (3)$$

where t is the current iteration, \max_t is the maximum number of iterations, ω_1 the inertia weight initial value and ω_2 its final value.

Later, *Ratnaweera et al. (2004)* tried to improve Shi and Eberhart's PSO with time-varying inertia weight (PSO-TVIW) using a similar concept applied to the acceleration coefficients. In the PSO with time-varying acceleration coefficients PSO (PSO-TVAC) the parameters c_1 and c_2 change during the run according to the following equations:

$$c_1 = (c_{1f} - c_{1i}) \times \frac{t}{\max_t} + c_{1i} \quad (4)$$

$$c_2 = (c_{2f} - c_{2i}) \times \frac{t}{\max_t} + c_{2i} \quad (5)$$

where c_{1i} , c_{1f} , c_{2i} , c_{2f} are the acceleration coefficients initial and final values.

The experiments in this section compare PSO-TVAC with SS-PSO-TVAC (i.e., PSO-TVAC with the steady-state update strategy). Parameters ω_1 and ω_2 were set to 0.75 and 0.5. The acceleration coefficient c_1 initial and final values were set to 2.5 and 0.5 and c_2 ranges from 0.5 to 2.5, as suggested by *Ratnaweera et al. (2004)*. The results are in **Error! Reference source not found.** (PSO-TVAC) and **Error! Reference source not found.** (SS-PSO-TVAC).

Error! Reference source not found. compares the algorithms using Mann-Whitney tests. SS-PSO-TVAC improves PSO-TVAC in every unimodal function in terms of accuracy and convergence speed and it is significantly faster in functions f_6 , f_7 , f_8 and f_{10} while attaining similar results. PSO-TVAC only outperforms SS-PSO-TVAC in the noisy f_9 function. These results show that the steady state version of PSO-TVAC is able to improve the convergence speed of the original algorithm in several types of fitness landscapes. Furthermore, SS-PSO-TVAC achieves more accurate solutions in the unimodal problems.

Comprehensive Learning PSO

The following experiment aims at comparing the proposed SS-PSO with the *comprehensive learning* PSO (CLPSO) (*Liang et al., 2006; Lynn & Suganthan, 2015*). CLPSO uses an alternative velocity updating equation:

$$v_{i,d}(t) = \omega \times v_{i,d}(t-1) + c \times r \times (p_{fi(d),d} - x_{i,d}(t-1)) \quad (6)$$

where $\vec{f}_i = (f_i(1), f_i(2), \dots, f_i(D))$ defines which particle's best solutions particle i should follow. Hence, the term $p_{fi(d),d}$ can refer to the corresponding dimension of any particle's best found solution so far. The decision depends on a probability p_c , different for each particle and computed *a priori*. Following the guidelines and parameters in (*Liang et al., 2006*), CLPSO and SS-CLPSO have been implemented and tested in the set of 10 benchmark functions.

CLPSO performance is strongly dependent on the refreshing gap parameter m , which defines the number of generations during which the particles are allowed to learn from f_i without improving their fitness. After m generations without fitness improvement, f_i is reassigned. In order to make fair comparisons, parameter m was first optimized for each function. The other parameters were set as in (*Liang et al., 2006*). Then, SS-CLPSO was tuned using the same parameter setting as the corresponding CLPSO.

The results are in **Table 16 and 17** and statistical analysis is in **Table 18**. On the one hand, the results show that, in general, a steady-state strategy applied to CLPSO does not improve the performance of the algorithm. On the other hand, SS-CLPSO does not degrade the general behavior of CLPSO. Please note that CLPSO does not use a traditional topology. In this case, to construct SS-CLPSO, we use a Moore neighborhood to decide which particles to update along with the least fit individuals, but, unlike SS-PSO or SS-PSO-TVAC, the structure does not define the information flow within the swarm. Since neighboring particles communicate and use each other's information, they tend to travel through similar regions of the landscape, but in CLPOS

there is not necessarily a relationship between the particles in the set and this clustering behavior is not present. For a steady-state strategy to take full advantage of the CLPSO dynamic network, maybe it is necessary to define a dynamic update strategy takes into account the current set of particles from which an individual is learning at a specific period of the run. Steady-state updates strategies for PSO in dynamic networks is planned as future work.

Parallelization Potential

A final set of experiments was performed with the goal of assessing the parallelization potential of the proposed SS-PSO. Contrary to S-PSO, only a limited set of particles are evaluated per iteration with SS-PSO – a conceivably limiting factor in its parallelization potential. In order to gain some insight on this issue, we compared the run times of S-PSO_{Moore} and SS-PSO_{Moore} (replace-worst) with 1 and 8 threads on an Intel Core i7-4770 (four cores, eight virtual threads). Population size was set at 49 (or 7×7 , since a Moore neighborhood is being used), and the problem dimension was set to $D = 30$, except for f_6 , which is a two-dimensional function. The PerfAndPubTools software (*Fachada et al., 2016*) was used to analyze these experiments.

In the first experiment, the two algorithms were tested on every problem, while measuring the total run time of 50 runs under two different scenarios: 1) runs finish only when the function-specific maximum number of evaluations is reached; and, 2) runs finish immediately after the function-specific stop criterion is met.

The first case is a worst-case scenario for SS-PSO, since it requires considerably more iterations in order to perform the same number of function evaluations as S-PSO. For example, during 20000 iterations, S-PSO performs 980000 function evaluations – evaluating 49 particles per iteration. However, to perform the same number of evaluations, SS-PSO requires 108889 iterations – as it only evaluates nine particles per iteration. Since it is the evaluation stage that is parallelized, SS-PSO is more limited in the parallel work it can perform per iteration. **Figure 8** shows the speedups obtained when parallelizing S-PSO and SS-PSO respectively. While the former can be between 3.5x and 5.5x faster when parallelized, the latter is only improved by 2x to 3x. However, as shown in **Table 8**, it is important to keep in mind that SS-PSO typically yields better fitness for the same number of evaluations.

In the second scenario runs finish immediately after the function-specific stop criterion is met. Unfortunately, large run time variations were observed, making it difficult to perform a direct comparison. This was to be expected, since a single unsuccessful run is enough to skew the results towards either algorithm. Repeating the test with different seeds often lead to opposing results. Nonetheless, some conclusions can be drawn from a qualitative perspective. When both algorithms execute serially, SS-PSO was typically faster (since it commonly requires less evaluations to meet the stop criteria). Under a parallel scenario, S-PSO becomes more competitive, beating SS-PSO in absolute run time in some occasions due to its higher parallelization potential. However, since SS-PSO usually requires fewer evaluations to achieve the same result, S-PSO's parallelization advantage does not appear so strong, especially on computationally expensive problems.

With the insight provided by the previous experiment, we setup a second experiment to analyze the scalability of both algorithms ($D = 10$ to $D = 50$) when optimizing problem f_7 , the Weierstrass function. This is the most computationally expensive problem of the ones benchmarked in this paper, and thus a good candidate for highlighting the benefits provided by SS-PSO. The algorithms were run 50 times per problem dimension, with runs finishing immediately after the stop criterion is met (0.01 in the case of f_7). **Figure 9** shows the total run times of the four setups: Serial S-PSO, Parallel S-PSO, Serial SS-PSO and Parallel SS-PSO. These results show that, with a computationally costly function and when aiming at a specific stop criterion, SS-PSO is consistently faster than S-PSO, in both serial and parallel contexts.

Discussion

The experiments in the previous sections demonstrate that SS-PSO is able to significantly improve the performance of the standard PSO, at least on the set of benchmark functions. The differences are particularly noticeable in the convergence speed of the algorithms, but SS-PSO is also able to improve the solution quality in several functions (see **Table 8**). An experiment comparing three different steady-state strategies show that replacing the worst particle and its neighbors is the best strategy. Our initial hypothesis (reducing the number of evaluations in each time step, while focusing only on the worst solutions, reduces the computational effort to reach a solution) is confirmed.

The relative performance of SS-PSO and standard PSO has also been verified for a wide range of parameter values (see **Figures 4-7**) as well as for different problem dimensions (see **Figure 3**). These results are important since they demonstrate that the proposed strategy has not been fine-tuned and that its validity is not restricted to a particular region of the parameter space or problem dimension. The algorithm was also compared to a PSO with time-varying acceleration, again attaining good results, thus reinforcing the idea that the steady-state strategy is consistent and robust. Finally, SS-PSO was compared to CLPSO, and while being outperformed in terms of solution quality in four functions, it yields better solutions in two problems, and is faster in other two functions. Since CLPSO is considered to be a very efficient algorithm, these results are promising. It deserves further examination whether variants of SS-PSO could clearly outperform CLPSO.

The parallelization potential of S-PSO and SS-PSO was also investigated. Although the latter showed some limitations in this regard - mainly due to performing fewer evaluations per iteration - it was still faster than S-PSO when aiming for a specific fitness criterion on more computationally intensive problems.

Conclusions

This paper investigates the performance of a new and unconventional updated strategy for the PSO. The steady state PSO (SS-PSO) is inspired by the Bak-Sneppen model of coevolution. However, while in the Bak-Sneppen model the worst individual and its neighbors are replaced by random values, in SS-PSO the worst particle and its neighbors are updated and evaluated in each

time step. The remaining particles are kept in a steady state until they eventually satisfy the update criterion. Due to its strategy, SS-PSO may be classified within the asynchronous PSOs category. However, its working mechanisms are radically different from standard asynchronous PSOs.

After preliminary tests that determined the best topology for a set of ten unimodal, multimodal, shifted, noisy and rotated benchmark problems, the strategy was implemented on the winning structure: 2-dimensional lattice with Moore neighborhood. Quality of solutions, convergence speed and success rates were compared and statistical analyses were conducted on the results. SS-PSO significantly improved the performance of a standard synchronous PSO (S-PSO) in every function, at least in one of the two criteria (quality of final solutions and convergence speed). A parameter sensitivity analysis showed that SS-PSO is not more sensitive to the variation of parameter values than S-PSO. A scalability test showed that the proposed strategy does not introduce scalability difficulties. The algorithm was compared to the PSO with time-varying acceleration coefficients (PSO-TVA) and to the comprehensive learning PSO (CLPSO) with good results. The parallelization of the algorithm was also studied.

The first step in future works is to increase the size of the test with more functions, hoping that an extended test set can improve our insight into the behavior of the algorithm. Also, the implementation of the strategy in other state-of-the-art PSO could confirm the validity of the method as an alternative and general-purpose velocity update strategy for the particle swarm. The emergent properties of the algorithm (size of events, duration of stasis, critical values) will be also studied and compared to those of the Bak-Sneppen model. Finally, steady-state update strategies in dynamic topologies will be investigated.

References

- Aziz NAB, Mubin M, Mohamad MS, Aziz KA. 2014. A Synchronous-Asynchronous Particle Swarm Optimisation Algorithm, *The Scientific World Journal*, Volume 2014, Article ID 123019, 17 pages.
- Carlisle A, Dozier G. 2001. An off-the-shelf PSO. *Workshop on Particle Swarm Optimization*.
- Baillieul J, Samad T. 2015. Encyclopedia of Systems and Control, London: Springer-Verlag.
- Bak P, Tang C, Wiesenfeld K. 1987. Self-organized Criticality: an Explanation of 1/f Noise, *Physical Review Letters*, 59(4), 381-384.
- Bak P, Chen K, Tang C. 1990. A forest-fire model and some thoughts on turbulence, *Physics Letters, A* 147, 197-300.
- Bak P, Sneppen K. 1993. Punctuated equilibrium and criticality in a simple model of evolution, *Physical Review Letters*, 71 (24), 4083-4086.
- Bak P. 1996. *How Nature Works*. New York: Springer-Verlag, 1996.
- Boettcher S, Percus AG. 2003. Optimization with Extremal Dynamics, *Complexity*, 8(2), 57-62.
- Dagum L, Menon R. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering*, 5(1), 46-55.

- Engelbrecht AP. 2013. Particle Swarm Optimization: Iteration Strategies Revisited, 2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence, 119-123.
- Fachada N, Lopes V.V, Martins RC, Rosa AC. 2016. PerfAndPubTools – Tools for Software Performance Analysis and Publishing of Results. *Journal of Open Research Software*, 4(1), e18.
- Fernandes CM, Merelo JJ, Rosa AC. 2016. An Asynchronous and Steady State Update Strategy for the Particle Swarm Optimization Algorithm, *Proceedings of Parallel Problem Solving from Nature – PPSN XIV*, Springer, pp. 166-177.
- Gutenberg B, Richter, C. F. 1956. Magnitude and Energy of Earthquakes. *Annali di Geofisica*, 9, 1–15.
- Kennedy J, Eberhart R. 1995. Particle Swarm Optimization, *Proceedings of IEEE International Conference on Neural Networks*, Vol.4, 1942–1948.
- Kennedy J, Mendes R. 2002. Population structure and particle swarm performance, *Proceedings of the IEEE World Congress on Evolutionary Computation*, 1671–1676.
- Løvbjerg M, Krink T. 2002. Extending particle swarm optimizers with self-organized criticality, *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, Vol. 2, IEEE Computer Society, 1588–1593.
- Liang JJ, Qin AK, Suganthan PN. 2006. Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions, *IEEE Transactions on Evolutionary Computation*, 10(3), 281-294.
- Lynn N, Suganthan PN. 2015. Heterogeneous comprehensive learning particle swarm optimization with enhanced exploration and exploitation, *Swarm and Evolutionary Computation*, 24, pp. 11-24.
- Majercik S. 2013. GREEN-PSO: Conserving Function Evaluations in Particle Swarm Optimization, *Proceedings of the IJCCI 2013*, 160-167.
- McNabb A. 2014. Serial PSO results are irrelevant in a multi-core parallel world, *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, 3143-3150.
- Rada-Vilela J, Zhang M., Seah W. 2013. A Performance Study on Synchronous and Asynchronous Updates in Particle Swarm, *Soft Computing*, 17(6), 1019-1030.
- Ratnaweera, A., Halgamuge, S. & Watson, H. (2004). Self-organizing hierarchical particle swarm optimizer with time varying accelerating coefficients, *IEEE Transactions on Evolutionary Computation*, 8(3), 240–255.
- Shi Y, Eberhart R.C. 1999. Empirical study of particle swarm optimization, *Proceedings of IEEE International Congress on Evolutionary Computation*, vol. 3,101–106.

Figure 1(on next page)

S-PSO_{lbest}, S-PSO_{VN} and S-PSO_{moore}: Solutions quality and convergence speed rank by the Friedman test.

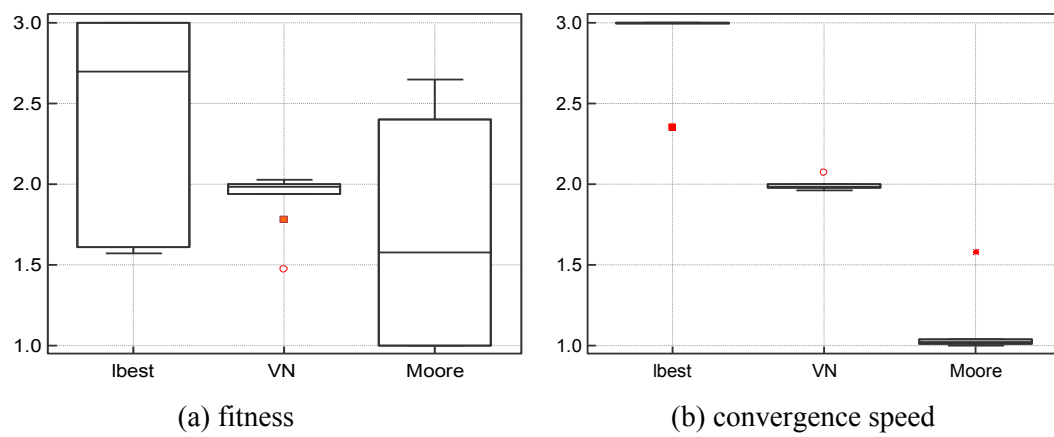


Figure 2 (on next page)

Fitness rank by Friedman test.

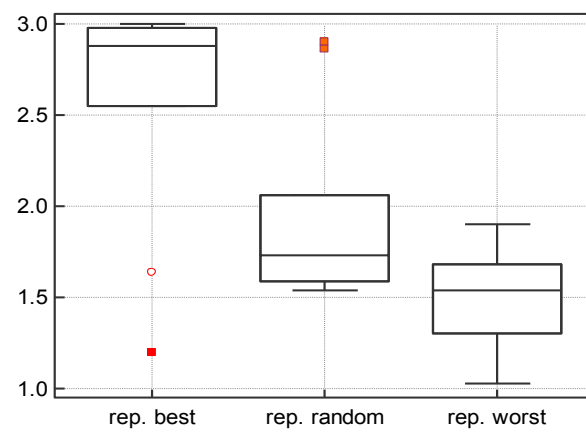


Figure 3(on next page)

Convergence speed and problem dimension.

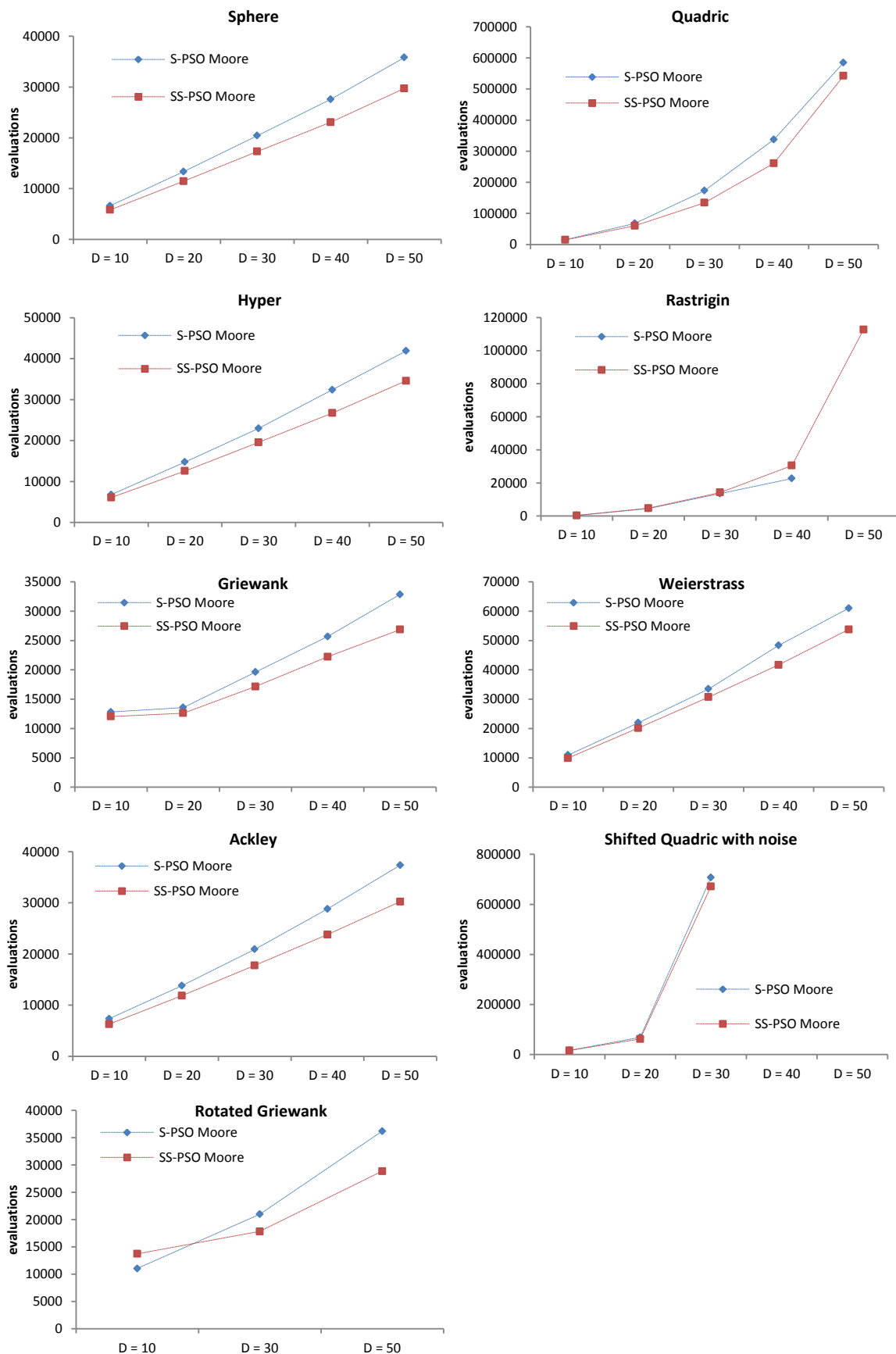


Figure 4(on next page)

Parameters sensitivity: Sphere function (f1).

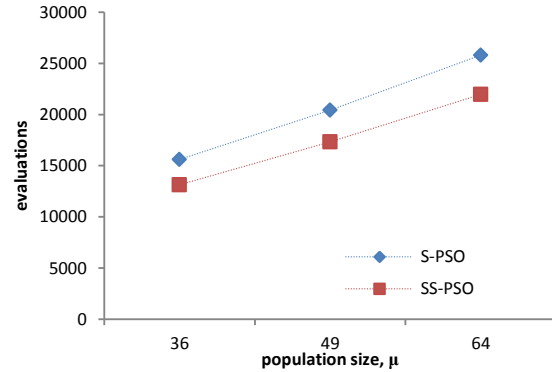
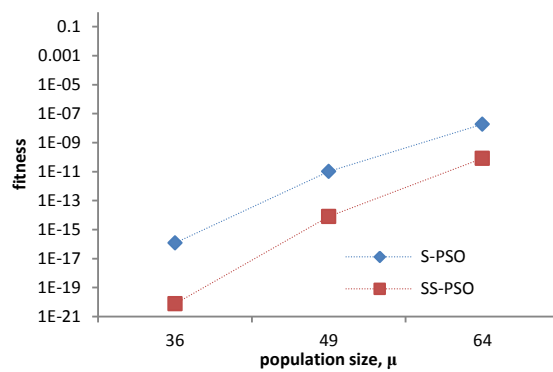
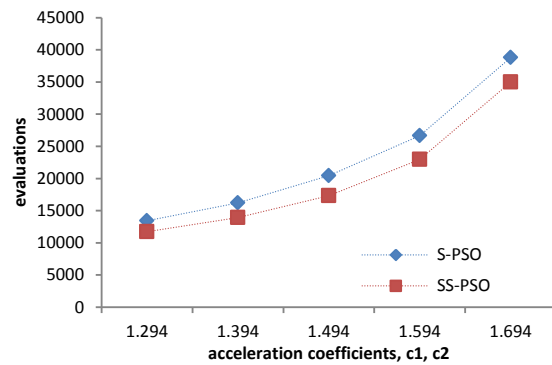
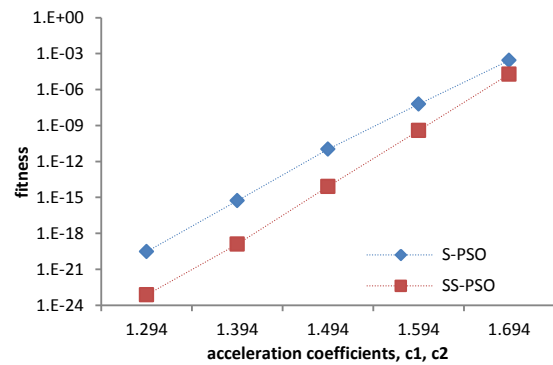
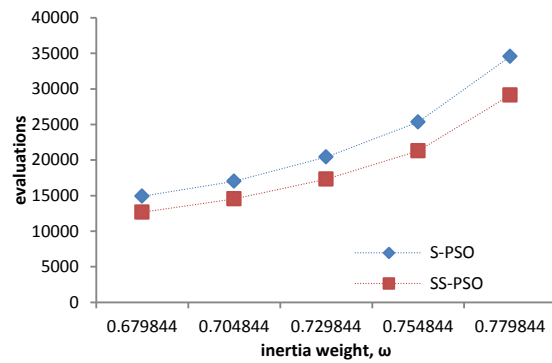
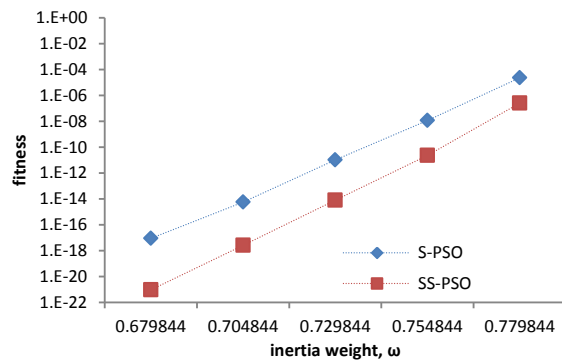


Figure 5(on next page)

Parameters sensitivity: Ackley function (f8)

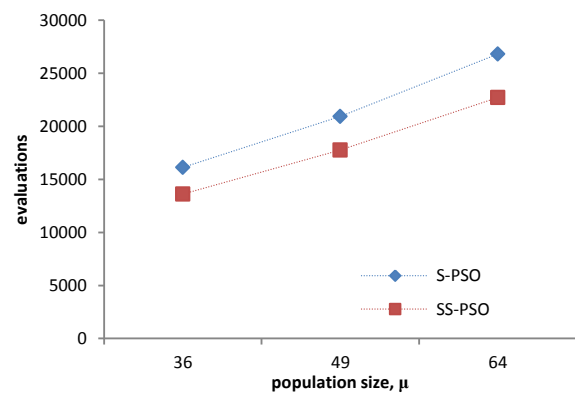
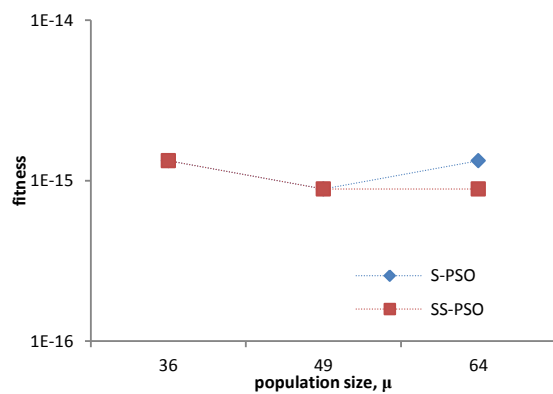
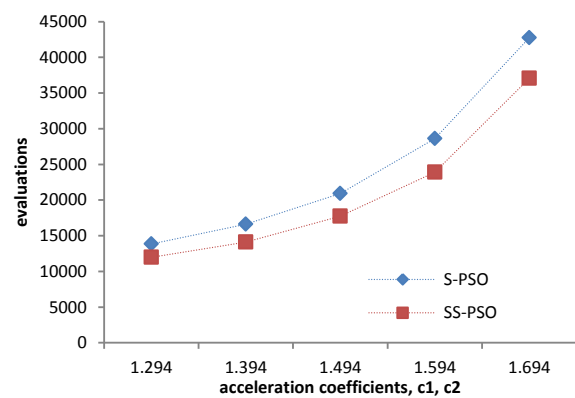
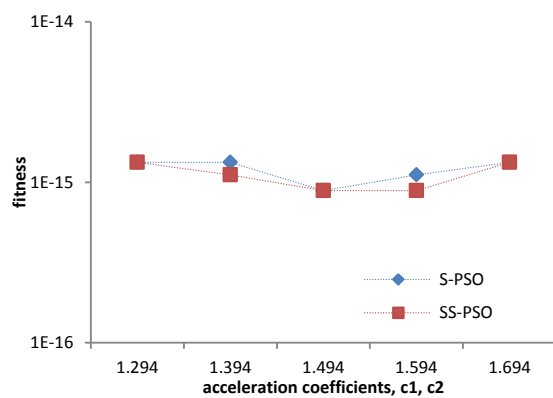
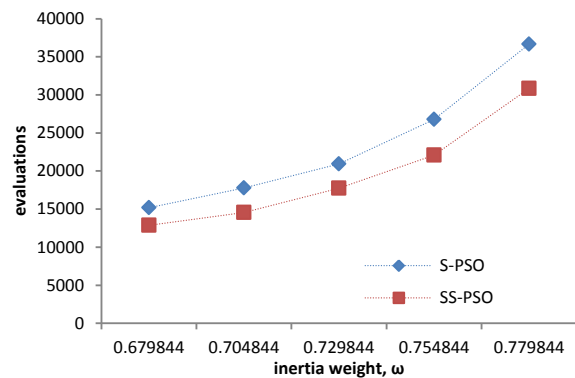
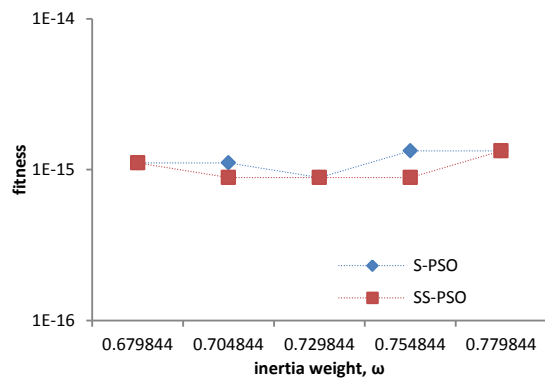


Figure 6(on next page)

Parameters sensitivity: Shifted Quadric with noise.

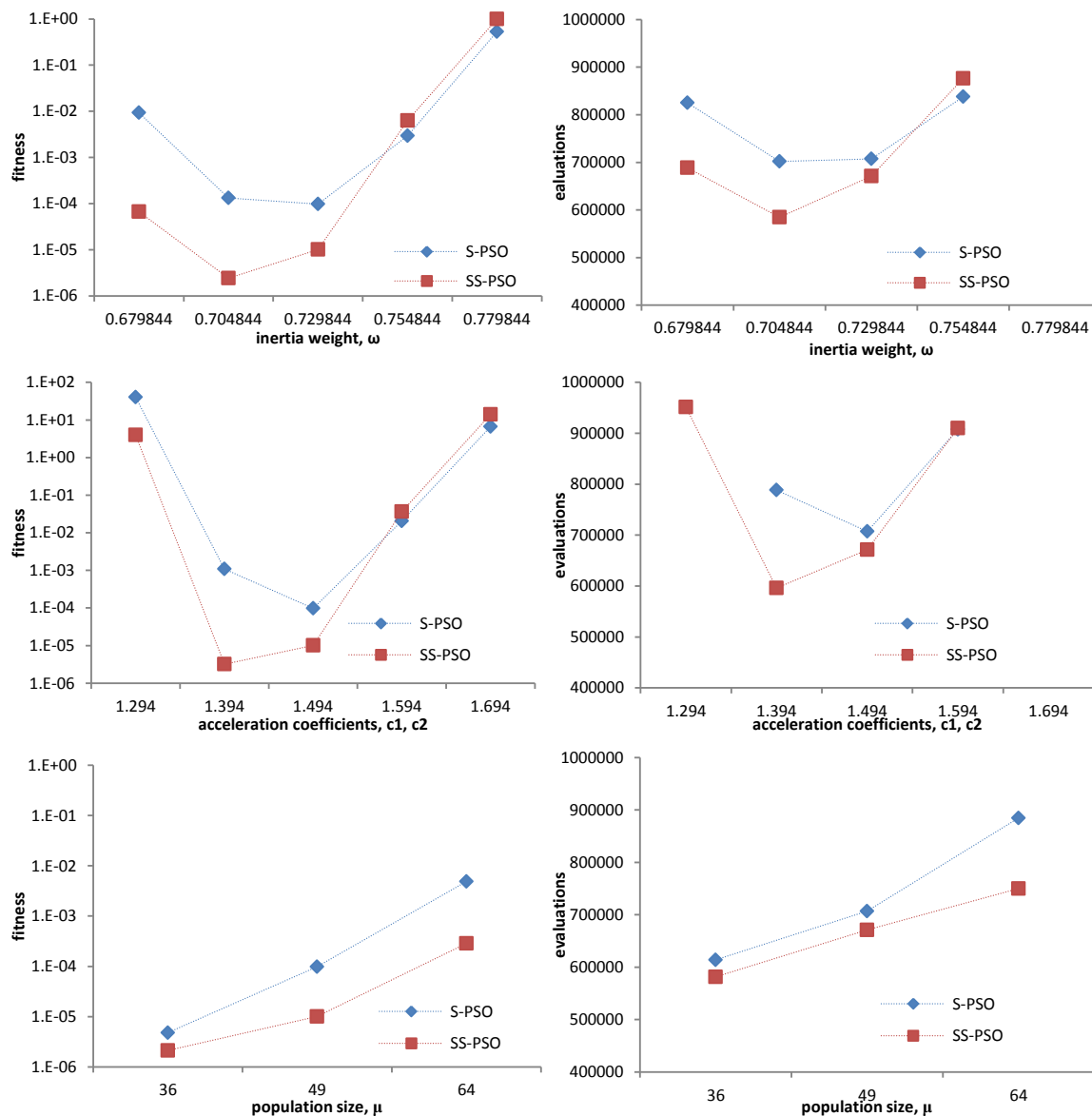


Figure 7(on next page)

Parameters sensitivity: Rotated Griewank (f10).

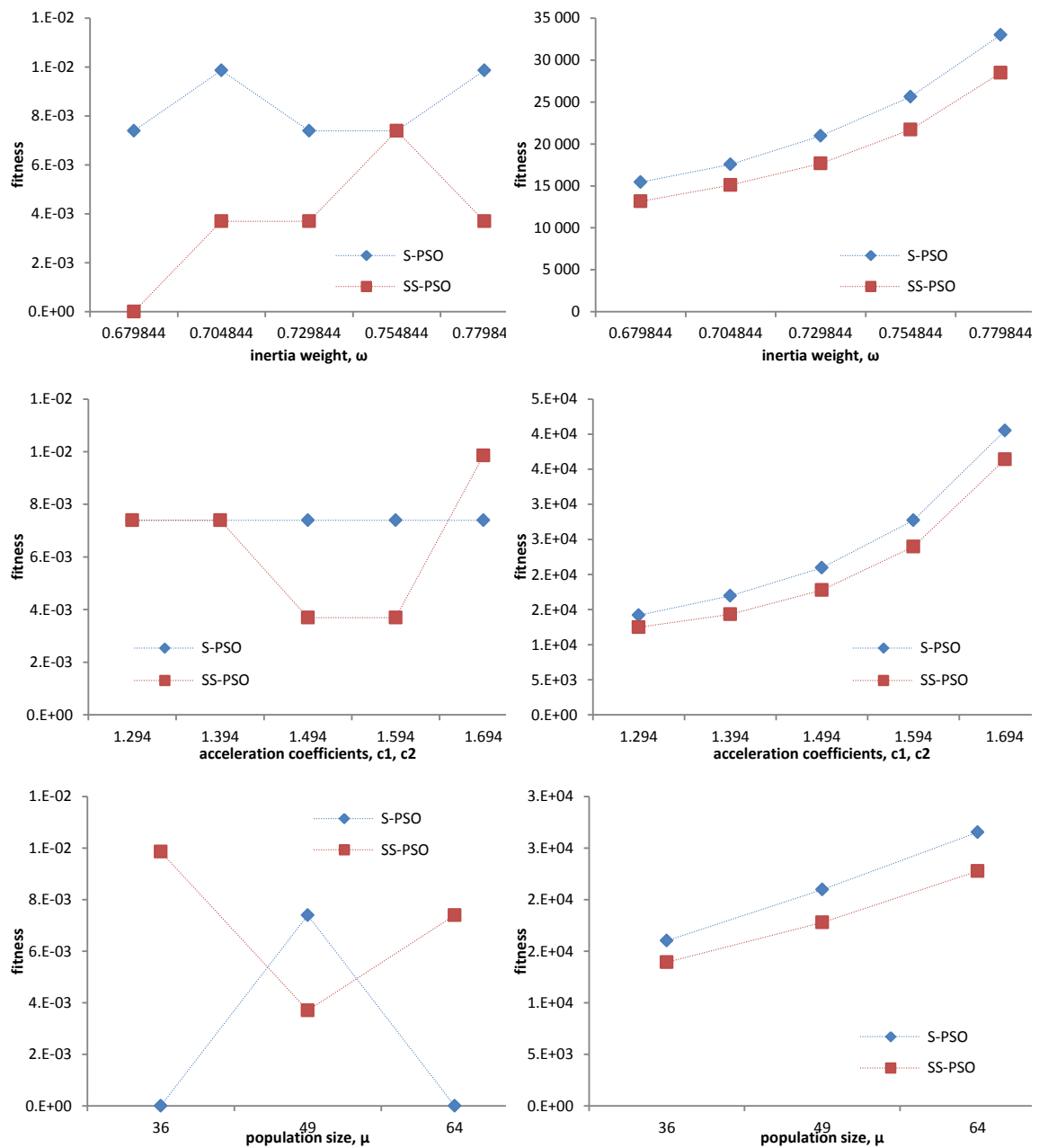


Figure 8(on next page)

Speedups obtained when parallelizing S-PSO and SS-PSO with 8 threads. A total of 50 runs were performed for each algorithm, terminating only when the function-specific maximum number of evaluations was reached.

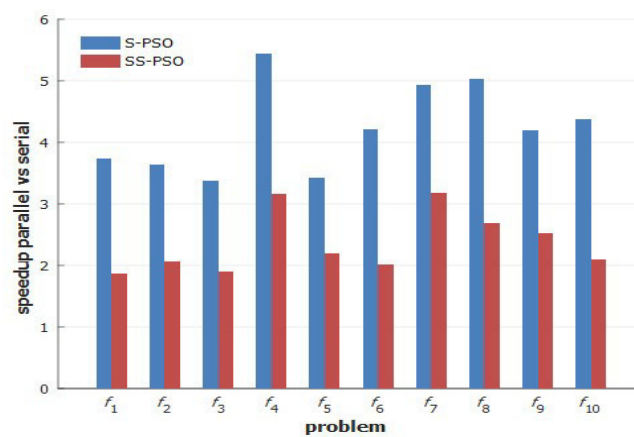


Figure 9(on next page)

Scalability of the S-PSO and SS-PSO algorithms for problem $\text{maximize } f(x) = \sum_{i=1}^n x_i^2$ with increasing number of variables. The serial versions were executed with 1 thread, while the parallel versions were executed with 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608, 16777216, 33554432, 67108864, 134217728, 268435456, 536870912, 1073741824, 2147483648, 4294967296, 8589934592, 17179869184, 34359738368, 68719476736, 137438953472, 274877906944, 549755813888, 1099511627776, 2199023255552, 4398046511104, 8796093022208, 17592186044416, 35184372088832, 70368744177664, 140737488355328, 281474976710656, 562949953421312, 1125899906842624, 2251799813685248, 4503599627370496, 9007199254740992, 18014398509481984, 36028797018963968, 72057594037927936, 144115188075855872, 288230376151711744, 576460752303423488, 1152921504606846976, 2305843009213693952, 4611686018427387904, 9223372036854775808, 18446744073709551616, 36893488147419103232, 73786976294838206464, 147573952589676412928, 295147905179352825856, 590295810358705651712, 1180591620717411303424, 2361183241434822606848, 4722366482869645213696, 9444732965739290427392, 18889465931478580854784, 37778931862957161709568, 75557863725914323419136, 151115727451828646838272, 302231454903657293676544, 604462909807314587353088, 1208925819614629174706176, 2417851639229258349412352, 4835703278458516698824704, 9671406556917033397649408, 19342813113834066795298816, 38685626227668133590597632, 77371252455336267181195264, 154742504910672534362390528, 309485009821345068724781056, 618970019642690137449562112, 1237940039285380274899124224, 2475880078570760549798248448, 4951760157141521099596496896, 9903520314283042199192993792, 19807040628566084398385987584, 39614081257132168796771975168, 79228162514264337593543950336, 158456325028528675187087900672, 316912650057057350374175801344, 633825300114114700748351602688, 1267650600228229401496703205376, 2535301200456458802993406410752, 5070602400912917605986812821504, 10141204801825835211973625643008, 20282409603651670423947251286016, 40564819207303340847894502572032, 81129638414606681695789005144064, 162259276829213363391578010288128, 324518553658426726783156020576256, 649037107316853453566312041152512, 1298074214633706907132624082305024, 2596148429267413814265248164610048, 5192296858534827628530496329220096, 10384593717069655257060992658440192, 20769187434139310514121985316880384, 41538374868278621028243970633760768, 83076749736557242056487941267521536, 166153499473114484112975882535043072, 332306998946228968225951765070086144, 664613997892457936451903530140172288, 1329227995784915872903807060280344576, 2658455991569831745807614120560689152, 5316911983139663491615228241121378304, 10633823966279326983230456482242756608, 21267647932558653966460912964485513216, 42535295865117307932921825928971026432, 85070591730234615865843651857942052864, 170141183460469231731687303715884105728, 340282366920938463463374607431768211456, 680564733841876926926749214863536422912, 1361129467683753853853498429727072845824, 2722258935367507707706996859454145691648, 5444517870735015415413993718908291383296, 10889035741470030830827987437816582766592, 21778071482940061661655974875633165533184, 43556142965880123323311949751266331066368, 87112285931760246646623899502532662132736, 174224571863520493293247799005065324265472, 348449143727040986586495598010130648530944, 696898287454081973172991196020261297061888, 1393796574908163946345982392040522594123776, 2787593149816327892691964784081045188247552, 5575186299632655785383929568162090376495104, 11150372599265311570767859136324180752990208, 22300745198530623141535718272648361505980416, 44601490397061246283071436545296723011960832, 89202980794122492566142873090593446023921664, 178405961588244985132285746181186892047843328, 356811923176489970264571492362373784095686656, 713623846352979940529142984724747568191373312, 1427247692705959881058285969449495136382746624, 2854495385411919762116571938898990272765493248, 5708990770823839524233143877797980545530986496, 11417981541647679048466287755595961091061972992, 22835963083295358096932575511191922182123945984, 45671926166590716193865151022383844364247891968, 91343852333181432387730302044767688728495783936, 182687704666362864775460604089535377456991567872, 365375409332725729550921208179070754913983135744, 730750818665451459101842416358141509827966271488, 1461501637330902918203684832716283019655932542976, 2923003274661805836407369665432566039311865085952, 5846006549323611672814739330865132078623730171904, 11692013098647223345629478661730264157247460343808, 23384026197294446691258957323460528314494920687616, 46768052394588893382517914646921056628989841375232, 93536104789177786765035829293842113257979682750464, 187072209578355573530071658587684226515959365500928, 374144419156711147060143317175368453031918731001856, 748288838313422294120286634350736906063837462003712, 1496577676626844588240573268701473812127674924007424, 2993155353253689176481146537402947624255349848014848, 5986310706507378352962293074805895248510699696029696, 11972621413014756705924586149611790497021399392059392, 23945242826029513411849172299223580994042798784118784, 47890485652059026823698344598447161988085597568237568, 95780971304118053647396689196894323976171195136475136, 191561942608236107294793378393788647952342390272950272, 383123885216472214589586756787577295904684780545900544, 766247770432944429179173513575154591809369561091801088, 1532495540865888858358347027150309183618739122183602176, 3064991081731777716716694054300618367237478244367204352, 6129982163463555433433388108601236734474956488734408704, 12259964326927110866866776217202473468949912977468817408, 24519928653854221733733552434404946937899825954937634816, 49039857307708443467467104868809893875799651909875269632, 98079714615416886934934209737619787751599303819750539264, 196159429230833773869868419475239575503198607639501078528, 392318858461667547739736838950479151006397215279002157056, 784637716923335095479473677900958302012794430558004314112, 1569275433846670190958947355801916604025588861116008628224, 3138550867693340381917894711603833208051177722232017256448, 6277101735386680763835789423207666416102355444464034512896, 12554203470773361527671578846415332832204710888928069025792, 25108406941546723055343157692830665664409421777856138051584, 50216813883093446110686315385661331328818843555712276103168, 100433627766186892221372630771322662657637687111424552206336, 200867255532373784442745261542645325315275374222849104412672, 401734511064747568885490523085290650630550748445698208825344, 803469022129495137770981046170581301261101496891396417650688, 1606938044258990275541962092341162602522202993782792835301376, 3213876088517980551083924184682325205044405987565585670602752, 6427752177035961102167848369364650410088811975131171341205504, 12855504354071922204335696738729300820177623950262342682411008, 25711008708143844408671393477458601640355247900524685364822016, 51422017416287688817342786954917203280710495801049370729644032, 102844034832575377634685573909834406561420991602098741459288064, 205688069665150755269371147819668813122841983204197482918576128, 411376139330301510538742295639337626245683966408394965837152256, 822752278660603021077484591278675252491367932816789931674304512, 1645504557321206042154969182557350504982735865633579863348609024, 3291009114642412084309938365114701009965471731267159726697218048, 6582018229284824168619876730229402019930943462534319453394436096, 13164036458569648337239753460458804039861886925068638906788872192, 26328072917139296674479506920917608079723773850137277813577744384, 52656145834278593348959013841835216159447547700274555627155488768, 105312291668557186697918027683670432318895095400549111254310977536, 210624583337114373395836055367340864637790190801098222508621955072, 421249166674228746791672110734681729275580381602196445017243910144, 842498333348457493583344221469363458551160763204392890034487820288, 1684996666696914987166688442938726917102321526408785780068975640576, 3369993333393829974333376885877453834204643052817571560137951281152, 6739986666787659948666753771754907668409286105635143120275902562304, 13479973333575319897333507543509815336818572211270286240551805124608, 26959946667150639794667015087019630673637144422540572481103610249216, 53919893334301279589334030174039261347274288845081144962207220498432, 107839786668602559178668060348078522694548577690162289924414440996864, 215679573337205118357336120696157045389097155380324579848828881993728, 431359146674410236714672241392314090778194310760649159697657763987456, 862718293348820473429344482784628181556388621521298319395315527974912, 1725436586697640946858688965569256363112777243042596638790631055949824, 3450873173395281893717377931138512726225554486085193277581262111899648, 6901746346790563787434755862277025452451108972170386555162524223799296, 13803492693581127574869511724554050904902217944340773110325048447598592, 27606985387162255149739023449108101809804435888681546220650096895197184, 55213970774324510299478046898216203619608871777363092441300193790394368, 110427941548649020598956093796432407239217743554726184882600387580788736, 220855883097298041197912187592864814478435487109452369765200775161577472, 441711766194596082395824375185729628956870974218904739530401550323154944, 883423532389192164791648750371459257913741948437809479060803100646309888, 1766847064778384329583297500742918515827483896875618958121606201292619776, 3533694129556768659166595001485837031654967793751237916243212402585239552, 7067388259113537318333190002971674063309935587502475832486424805170479104, 14134776518227074636666380005943348126619871175004951664972849610340958208, 28269553036454149273332760011886696253239742350009903329945699220681916416, 56539106072908298546665520023773392506479484700019806659891398441363832832, 113078212145816597093331040047546785012958969400039613319782796882727665664, 226156424291633194186662080095093570025917938800079226639565593765455331328, 452312848583266388373324160190187140051835877600158453279131187530910662656, 904625697166532776746648320380374280103671755200316906558262375061821325312, 1809251394333065553493296640760748560207343510400633813116524750123642650624, 3618502788666131106986593281521497120414687020801267626233049500247285301248, 7237005577332262213973186563042994240829374041602535252466099000494570602496, 14474011154664524427946373126085988481658748083205070504932198000989141204992, 28948022309329048855892746252171976963317496166410141009864396001978282409984, 57896044618658097711785492504343953926634992332820282019728792003956564819968, 115792089237316195423570985008687907853269984665640564039457584007913129639936, 231584178474632390847141970017375815706539969331281128078915168015826259279872, 463168356949264781694283940034751631413079938662562256157830336031652518559744, 926336713898529563388567880069503262826159877325124512315660672063305037119488, 1852673427797059126777135760139006525652319754650249024631321344126610074238976, 3705346855594118253554271520278013051304639509300498049262642688253220148477952, 7410693711188236507108543040556026102609279018600996098525285376506440296955904, 14821387422376473014217086081112052205218558037201992197050570753012880593911808, 29642774844752946028434172162224104410437116074403984394101141506025761187823616, 59285549689505892056868344324448208820874232148807968788202283012051522375647232, 118571099379011784113736688648896417641748464297615937576404566024103044751294464, 237142198758023568227473377297792835283496928595231875152809132048206089502588928, 474284397516047136454946754595585670566993857190463750305618264096412179005177856, 948568795032094272909893509191171341133987714380927500611236528192824358010355712, 1897137590064188545819787018382342682267975428761855001222473056385648716020711424, 3794275180128377091639574036764685364535950857523710002444946112771297432041422848, 7588550360256754183279148073529370729071901715047420004889892225542594864082845696, 15177100720513508366558296147058741458143803430094840009779784451085189728165691392, 30354201441027016733116592294117482916287606860189680019559568902170379456331382784, 60708402882054033466233184588234965832575213720379360039119137804340758912662765568, 121416805764108066932466369176469931665150427440758720078

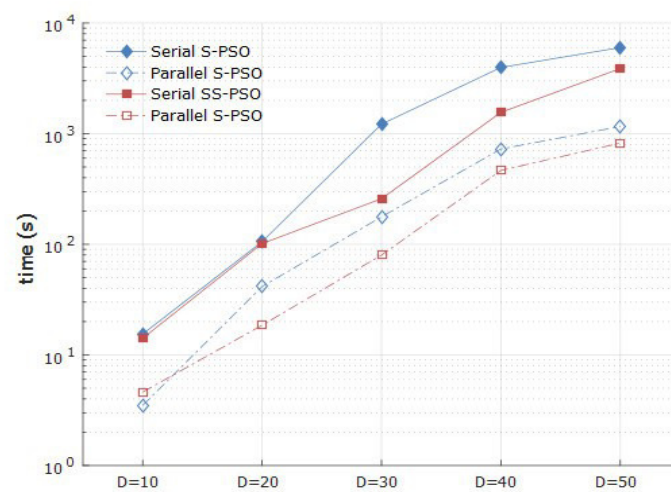


Table 1(on next page)

Benchmark functions.

	<i>mathematical representation</i>	<i>Range of search/ initialization</i>	<i>stop criterion</i>
<i>Sphere</i> f_1	$f_1(\vec{x}) = \sum_{i=1}^D x_i^2$	$(-100, 100)^D$ $(50, 100)^D$	0.01
<i>Quadric</i> f_2	$f_2(\vec{x}) = \sum_{i=1}^D \left(\sum_{j=1}^i x_j \right)^2$	$(-100, 100)^D$ $(50, 100)^D$	0.01
<i>Hyper Ellipsoid</i> f_3	$f_1(\vec{x}) = \sum_{i=1}^D i x_i^2$	$(-100, 100)^D$ $(50, 100)^D$	0.01
<i>Rastrigin</i> f_4	$f_4(\vec{x}) = \sum_{i=1}^D (x_i^2 - 10 \cos(2\pi x_i) + 10)$	$(-10, 10)^D$ $(2.56, 5.12)^D$	100
<i>Griewank</i> f_5	$f_5(\vec{x}) = 1 + \frac{1}{4000} \sum_{i=1}^D x_i^2 - \prod_{i=1}^D \cos\left(\frac{x_i}{\sqrt{i}}\right)$	$(-600, 600)^D$ $(300, 600)^D$	0.05
<i>Schaffer</i> f_6	$f_6(\vec{x}) = 0.5 + \frac{(\sin \sqrt{x^2 + y^2})^2 - 0.5}{(1.0 + 0.001(x^2 + y^2))^2}$	$(-100, 100)^2$ $(15, 30)^2$	0.00001
<i>Weierstrass</i> f_7	$f_7(\vec{x}) = \sum_{i=1}^D \left(\sum_{k=0}^{kmax} [a^k \cos(2\pi b^k(x_i + 0.5))] \right) - D \sum_{k=0}^{kmax} [a^k \cos(2\pi b^k \cdot 0.5)],$ $a = 0.5, b = 3, kmax = 20$	$(-0.5, 0.5)^D$ $(-0.5, 0.2)^D$	0.01
<i>Ackley</i> f_8	$f_8(\vec{x}) = -20 \exp \left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2} \right) - \exp \left(\frac{1}{D} \sum_{i=1}^D \cos(2\pi x_i) \right) + 20 + e$	$(-32.768, 32.768)^D$ $(2.56, 5.12)^D$	0.01
<i>Shifted Quadric with noise</i> f_9	$f_9(\vec{z}) = \sum_{i=1}^D \left(\sum_{j=1}^i z_j \right)^2 * (1 + 0.4 N(0,1)),$ $\vec{z} = \vec{x} - \vec{o}, \vec{o} = [o_1, \dots, o_D]: \text{shifted global optimum}$	$(-100, 100)^D$ $(50, 100)^D$	0.01
<i>Rotated Griewank</i> f_{10}	$f_{10}(\vec{z}) = 1 + \frac{1}{4000} \sum_{i=1}^D z_i^2 - \prod_{i=1}^D \cos\left(\frac{z_i}{\sqrt{i}}\right), \quad \vec{z} = \mathbf{M}\vec{x},$ \mathbf{M} : orthogonal matrix	$(-600, 600)^D$ $(300, 600)^D$	0.05

Table 2(on next page)

Median, minimum and maximum best fitness (50 runs).

	S-PSO _{lbest}			S-PSO _{vn}			S-PSO _{Moore}		
	median	min	max	median	min	max	median	min	max
f_1	4.57e-06	9.44e-07	2.83e-05	9.13e-10	1.68e-10	6.70e-09	5.05e-12	8.81e-13	4.43e-11
f_2	5.39e-13	3.09e-15	1.57e-11	4.52e-23	3.06e-25	2.81e-21	1.18e-30	1.01e-33	9.41e-28
f_3	3.01e-05	8.44e-06	1.65e-04	5.58e-09	1.16e-09	4.60e-08	2.53e-11	3.08e-12	1.94e-10
f_4	1.09e+02	6.57e+01	1.53e+02	6.02e+01	3.38e+01	1.09e+02	5.17e+01	3.78e+01	1.13e+02
f_5	0.00e00	0.00e00	7.40e-03	0.00e00	0.00e00	5.38e-02	0.00e00	0.00e00	4.92e-02
f_6	0.00e00	0.00e00	9.72e-03	0.00e00	0.00e00	0.00e00	0.00e00	0.00e00	9.72e-03
f_7	0.00e00	0.00e00	0.00e00	0.00e00	0.00e00	3.29e-02	9.03e-04	0.00e00	1.12e00
f_8	1.33e-15	8.88e-16	1.33e-15	1.33e-15	8.88e-16	1.33e-15	8.88e-16	8.88e-16	1.33e-15
f_9	1.74e+02	3.41e+01	1.07e+03	4.76e-02	4.87e-04	2.05e+02	9.80e-05	6.44e-07	1.64e+03
f_{10}	0.00e00	0.00e00	9.86e-03	0.00e00	0.00e00	3.19e-02	7.40e-03	0.00e00	5.19e-01

Table 3(on next page)

Median, minimum and maximum evaluations required to meet the criteria (50 runs).

	S-PSO _{lbest}			S-PSO _{vn}			S-PSO _{Moore}		
	median	min	max	median	min	max	median	min	max
f_1	32511.5	30135	34937	23544.5	21952	24990	20212	18669	22050
f_2	365270	313551	403858	217854	188111	242893	173117	142688	194530
f_3	36799	34496	40425	26827	25029	29253	23104	21462	24353
f_4	77518	21462	866173	15582	9604	74872	13524.0	7448	49392
f_5	31213	27244	34594	22736	20188	25333	19379.5	17248	23765
f_6	18865	5243	145334	12323.5	3626	80213	7105.0	3822	39788
f_7	62377	56399	69776	41356	37191	45766	33492	31801	42973
f_8	35206.5	31556	39249	24206	22834	28928	20923.0	19012	24794
f_9	-	.	.	883911	758961	976962	706972	453201	922327
f_{10}	33001.5	30331	37926	24157	21805	26460	21021	18865	29939

Table 4(on next page)

Success rates.

	S-PSO _{lbest}	S-PSO _{vn}	S-PSO _{Moore}
f_1	50	50	50
f_2	50	50	50
f_3	50	50	50
f_4	17	49	49
f_5	50	50	50
f_6	50	50	50
f_7	50	47	34
f_8	50	50	50
f_9	6	9	47
f_{10}	50	50	47

Table 5 (on next page)

Fitness rank by Friedman test (0.05 significance level). The table gives the rank of each algorithm and in parenthesis the algorithms to which the differences are significant according to the Friedman test.

	S-PSO _{lbest} (1)	S-PSO _{vn} (2)	S-PSO _{Moore} (3)	P value
f_1	3.0 (2) (3)	2.0 (1) (3)	1.0 (1) (2)	< 0.0001
f_2	3.0 (2) (3)	2.0 (1) (3)	1.0 (1) (2)	< 0.0001
f_3	3.0 (2) (3)	2.0 (1) (3)	1.0 (1) (2)	< 0.0001
f_4	2.98 (2) (3)	1.47 (1)	1.55 (1)	< 0.0001
f_5	1.57 (2) (3)	2.03 (1) (2)	2.40 (1) (3)	< 0.0001
f_6	2.24 (2) (3)	1.94 (1)	1.82 (1)	0.00025
f_7	1.57 (3)	1.78 (3)	2.65 (1) (3)	< 0.0001
f_8	2.44 (2) (3)	1.96 (1) (3)	1.60 (1) (2)	< 0.0001
f_9	2.96 (2) (3)	1.98 (1) (3)	1.06 (1) (2)	< 0.0001
f_{10}	1.61 (2) (3)	1.99 (2) (3)	2.40 (1) (2)	< 0.0001

Table 6 (on next page)

Convergence speed rank by Friedman test (0.05 significance level). The table gives the rank of each algorithm and in parenthesis the algorithms to which the differences are significant according to the Friedman test.

	S-PSO _{lbest} (1)	S-PSO _{vn} (2)	S-PSO _{Moore} (3)	P value
f_1	3.0 (2) (3)	1.99 (1) (3)	1.01 (1) (2)	< 0.0001
f_2	3.0 (2) (3)	1.98 (1) (3)	1.02 (1) (2)	< 0.0001
f_3	3.0 (2) (3)	1.99 (1) (3)	1.01 (1) (2)	< 0.0001
f_5	3.0 (2) (3)	1.96 (1) (3)	1.04 (1) (2)	< 0.0001
f_6	2.35 (3)	2.07 (3)	1.58 (1) (2)	0.00039
f_8	3.0 (2) (3)	2.0 (1) (3)	1.0 (1) (2)	< 0.0001

Table 7 (on next page)

SS-PSO_{Moore} results: solutions quality, convergence speed and success rates.

	fitness			evaluations			
	median	min	max	median	min	max	SR
f_1	5.42e-15	3.45e-16	6.49e-14	17019	15327	18819	50
f_2	7.18e-54	8.41e-60	4.87e-49	133191	102258	163251	50
f_3	2.99e-14	1.15e-15	2.97e-13	19768.5	17460	21069	50
f_4	5.12e+01	2.19e+01	1.04e+02	14256	7659	58248	49
f_5	7.40e-03	0.00e00	3.69e-02	16884	14814	24291	50
f_6	0.00e00	0.00e00	0.00e00	6381	2727	21744	50
f_7	0.00e00	0.00e00	1.32e-01	30717	28089	34254	48
f_8	8.88e-16	8.88e-16	1.33e-15	17752.5	15750	19809	50
f_9	1.01e-05	1.73e-08	7.11e-04	671175	425655	852786	50
f_{10}	3.70e-03	0.00e00	5.24e-01	17662.5	15669	27252	48

Table 8(on next page)

Comparing S-PSO_{Moore} and SS-PSO_{Moore}: showing + if SS-PSO_{Moore} ranks first in the Mann-Whitney test and the result is significant, – if S-PSO_{Moore} ranks first and the results is significant and \approx if the differences

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
fitness	+	+	+	\approx	\approx	+	+	\approx	+	\approx
evaluations	+	+	+	\approx	+	\approx	+	+	+	+

Table 9(on next page)

Results of SS-PSO variants: median, min, max and success rates (SR)

	SS-PSO _{Moore} (replace-best)								SS-PSO _{Moore} (replace-random)						
	fitness			evaluations					fitness			evaluations			
	median	min	max	median	min	max	SR		median	min	max	median	min	max	SR
f_1	4.09e-29	2.50e-33	2.00e+04	9468	6714	24669	45		6.04e-14	7.86e-14	6.59e-12	18972	16425	20781	50
f_2	1.50e+04	4.12e-89	3.50e+04	66307	64251	68364	2		8.33e-32	4.59e-34	5.00e+03	170091	136062	195498	47
f_3	3.01e-27	9.54e-34	1.00e+05	11718	8208	36000	35		1.66e-12	1.30e-13	2.25e-11	21118	19548	23283	50
f_4	1.30e+02	7.46e+01	2.00e+02	15192	8964	108495	9		5.62e+01	2.39e+01	8.76e+01	11052	5679	23571	50
f_5	3.08e-02	0.00e00	1.81e+02	10287	8694	26838	12		0.00e00	0.00e00	8.33e-02	19849.5	17748	26739	36
f_6	3.59e-04	0.00e00	9.72e-03	39811.5	1242	140247	38		0.00e00	0.00e00	9.72e-03	8460	3276	62091	50
f_7	7.52e00	2.64e00	1.57e+01	-	-	-	0		1.58e-03	0.00e00	2.48e00	33912	31239	41211	30
f_8	2.28e00	8.86e-16	3.84e00	20898	13158	28764	6		1.11e-15	8.86e-16	1.33e-15	19822.5	18252	25416	50
f_9	1.06e-01	1.98e-03	1.53e+04	902407	812736	949590	12		1.64e-04	1.44e-06	6.01e+01	736713	546858	891432	49
f_{10}	1.04e+01	0.00e00	4.04e+02	16065	8388	23742	2		3.70e-03	0.00e00	5.09e-01	21915	18567	50607	39

Table 10(on next page)

Comparing *replace-worst* and *replace-random*: showing + if *replace-worst* ranks first in the Mann-Whitney test and the result is significant; – if *replace-random* ranks first and the results is significant; \approx if the differences are not

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
fitness	+	+	+	\approx	\approx	\approx	+	+	+	\approx
evaluations	+	+	+	—	+	+	+	+	+	+

Table 11(on next page)

Solutions quality with different problem dimension.

	D = 10		D = 20		D = 30		D = 40		D = 50	
	S-PSO	SS-PSO	S-PSO	SS-PSO	S-PSO	SS-PSO	S-PSO	SS-PSO	S-PSO	SS-PSO
f_1	1.06e-37	2.71e-47	1.87e-19	5.72e-24	1.04e-11	7.83e-15	7.15e-08	2.96e-10	3.69e-05	2.01e-10
f_2	0.00e00	0.00e00	4.63e-82	1.37e-89	1.17e-30	9.52e-54	1.18e-13	1.10e-20	1.36e-06	2.36e-06
f_3	1.57e-40	0.00e00	2.08e-19	3.37e-24	2.76e-11	1.58e-14	8.77e-07	2.58e-09	4.59e-04	3.19e-06
f_4	1.99e00	1.99e00	2.09e+01	2.04e+01	6.17e+01	5.12e+01	1.01e+02	1.06e+02	1.70e+02	1.37e+02
f_5	2.83e-02	3.60e-02	8.63e-03	1.11e-02	0.00e00	7.40e-03	0.00e00	7.40e-03	0.00e00	0.00e00
f_7	0.00e00	0.00e00	0.00e00	0.00e00	9.03e-04	0.00e00	9.03e-04	3.39e-04	1.34e-01	2.15e-02
f_8	4.44e-16	4.44e-16	8.88e-16	8.88e-16	8.88e-16	8.88e-16	1.33e-15	1.33e-15	1.33e-15	1.33e-15
f_9	0.00e00	0.00e00	1.92e-10	1.32e-10	9.8e-05	1.01e-05	6.18e+01	3.40e+01	1.34e+03	1.70e+03
f_{10}	3.20e-02	3.20e-02	-	-	7.40e-03	7.40e-03	-	-	0.00e00	0.00e00

Table 12(on next page)

Success rates with different problem dimension.

	D=10		D = 20		D = 30		D = 40		D = 50	
	S-PSO	SS-PSO	S-PSO	SS-PSO	S-PSO	SS-PSO	S-PSO	SS-PSO	S-PSO	SS-PSO
f_1	50	50	50	50	50	50	50	50	50	50
f_2	50	50	50	50	50	50	43	50	32	48
f_3	50	50	50	50	50	50	50	50	50	50
f_4	50	50	50	50	49	49	25	21	0	3
f_5	40	37	49	50	50	47	50	49	50	50
f_7	50	50	49	49	34	48	8	35	4	19
f_8	50	50	50	50	50	50	50	50	46	50
f_9	50	50	50	50	47	50	0	0	0	0
f_{10}	44	35	-	-	46	48	-	-	48	49

Table 13(on next page)

PSO-TVA results.

	fitness			evaluations			SR
	median	min	max	median	min	max	
f_1	2.85e-21	2.55e-22	1.84e-20	11956	11221	13181	50
f_2	4.47e-51	1.23e-54	5.00e03	208740	185514	238532	49
f_3	3.87e-21	3.01e-22	1.57e-19	13769	12740	16121	50
f_4	3.08e+01	1.11e+01	5.8e+01	31114	16661	59388	50
f_5	0.00e00	0.00e00	4.91e-02	15141	12642	91238	50
f_6	0.00e00	0.00e00	0.00e00	11956	5145	38612	49
f_7	0.00e00	0.00e00	1.64e-01	35280	31017	42336	49
f_8	7.55e-15	4.00e-15	7.55e-15	21070	17346	29988	50
f_9	6.14e-09	1.74e-09	6.28e-06	227066	199528	287042	47
f_{10}	7.40e-03	0.00e00	5.24e-01	18620	14602	87220	42

Table 14(on next page)

SS-PSO-TVAC results.

	fitness			evaluations			SR
	median	min	max	median	min	max	
f_1	7.85e-26	4.82e-27	2.35e-24	10417	9126	11322	50
f_2	5.18e-63	2.30e-67	5.77e-60	190458	168282	226062	50
f_3	1.66e-25	7.76e-27	9.14e-24	11925	10422	13923	50
f_4	3.48e+01	1.89e01	7.46e+01	38043	22032	108927	50
f_5	0.00e00	0.00e00	4.42e-02	13662	9963	56421	49
f_6	0.00e00	0.00e00	0.00e00	8421	2547	26325	49
f_7	0.00e00	0.00e00	2.62e-01	31752	28323	41193	43
f_8	7.55e-15	4.00e-15	7.55e-15	18756	14958	23904	49
f_9	5.41e-09	6.37e-10	5.80e-03	315792	192906	476532	48
f_{10}	0.00e00	0.00e00	3.93e-02	15948	12762	75510	40

Table 15(on next page)

Comparing SS-PSO-TVAC and PSO-TVAC: showing + if SS-PSO-TVAC ranks first in the Mann-Whitney test and the result is significant; – if PSO-TVAC ranks first and the results is significant; \approx if the differences are not significant.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
fitness	+	+	+	\approx	\approx	\approx	\approx	\approx	\approx	\approx
evaluations	+	+	+	\approx	\approx	+	+	+	—	+

Table 16(on next page)

CLPSO results.

	fitness			evaluations			
	median	min	max	median	min	max	SR
f_1	9.59e-07	4.00e-07	2.23e-06	34848	33355	35909	50
f_2	2.16e-01	8.66e-02	5.46e-01	-	-	-	-
f_3	2.31e-06	1.18e-06	4.96e-06	36777	35665	37972	50
f_4	4.97e00	1.99e00	1.20e+01	115701	94674	129493	50
f_5	0.00e00	0.00e00	9.74e-13	199537	164774	243806	50
f_6	6.49e-06	7.69e-09	1.13e-04	81149	37710	96320	31
f_7	8.67e-13	3.48e-13	1.61e-12	430069	418700	440035	50
f_8	7.55e-15	4.00e-15	7.55e-15	282613	275897	285290	50
f_9	4.36e-01	1.51e-01	1.11e00	-	-	-	-
f_{10}	0.00e00	0.00e00	2.26e-14	173346	151269	229975	50

Table 17(on next page)

SS-CLPSO results.

	fitness			evaluations			
	median	min	max	median	min	max	SR
f_1	1.33e-06	2.99e-07	4.98e-06	35998	34063	37956	50
f_2	5.14e-01	1.71e-01	1.44e00	-	-	-	-
f_3	1.46e-06	4.82e-07	7.44e-06	36079	33177	37961	50
f_4	4.09e+00	1.04e00	1.05e+01	190310	147544	217855	50
f_5	0.00e00	0.00e00	2.16e-14	181779	137821	225172	50
f_6	6.64e-06	3.10e-07	6.90e-05	86058	45530	97936	28
f_7	6.26e-11	3.27e-12	1.69e-08	409351	393553	423387	50
f_8	7.55e-15	4.00e-15	7.55e-15	358407	344448	374581	50
f_9	7.70e-01	3.26e-01	6.16e01	-	-	-	-
f_{10}	0.00e00	0.00e00	1.04e-13	152818	122165	207094	50

~

Table 18(on next page)

Comparing SS-PSO and CLPSO: showing + if SS-PSO ranks first in the Mann-Whitney test and the result is significant; – if CLPSO ranks first and the results is significant; \approx if the differences are not significant.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
fitness	\approx	\approx	\approx	\approx	\approx	\approx	—	\approx	\approx	\approx
eval.	\approx	.	\approx	—	+	\approx	+	—	.	+