

Random Asynchronous PSO

Juan Rada-Vilela

Mengjie Zhang

Winston Seah

School of Engineering and Computer Science

Victoria University of Wellington

PO Box 600, Wellington, New Zealand

{juan.rada-vilela, mengjie.zhang, winston.seah}@ecs.vuw.ac.nz

Abstract—In this work we propose the Random Asynchronous PSO (RAPSO) algorithm, a rather simple but intuitive variant of the Asynchronous PSO (APSO) that introduces a randomized order in which particles share their information. Our algorithm, while conceived as serial in terms of execution, is able to model the behavior of the Parallel Asynchronous PSO (PAPSO) but in a controlled environment where results from independent runs are reproducible and the implementation does not need additional computational requirements (e.g. computational grids or GPUs). We support our proposal by comparing it favorably with respect to the original APSO in ten well-known benchmark functions. Statistical tests show that RAPSO generally yields better results (or at least not worse) and generally faster than APSO, making it a more attractive choice to tackle optimization problems where asynchronous updates is a desirable property.

I. INTRODUCTION

Since the appearance of Particle Swarm Optimization (PSO) in 1995 [1], variants and modifications to the original algorithm have been proposed in order to improve on results, speed of convergence, or computational requirements (examples can be found in [2]).

The variants we are concerned with are the ones regarding synchronicity in updates. The original PSO algorithm was conceived implicitly synchronous, that is, all the particles have perfect information about the quality of their neighborhood before performing their position update. Conversely, in the asynchronous variant the particles have only partial (or imperfect) information before updating their position. These algorithms are referred to as Synchronous PSO (SPSO) and Asynchronous PSO (APSO).

The APSO algorithm was first mentioned in [3] (to the best of our knowledge), and was assumed ever since to yield results faster than SPSO in terms of iterations. However, just recently we showed that SPSO is generally better and has a faster (or at least a similar) speed of convergence than APSO [4], contrary to what previous studies have suggested [3], [5], [6], [7], [8], [9]. Nevertheless, APSO is still an interesting variant.

The main advantage of APSO over SPSO is precisely its main characteristic: the lack of synchronicity between particles. This is a highly desirable property when implementing PSO using parallel particles because it reduces the idle time spent waiting for the information from all the neighborhood given that particles are able to work with only partial information. Furthermore, concurrency control is no longer needed.

In Parallel Asynchronous PSO (PAPSO [5]) particles perform their operations independently and simultaneously to others, thus requiring additional computational resources such as grid computing or multi-core processors (e.g. GPUs). This approach, while making an optimal use of computational resources, causes the results from experiments to be hardly reproducible (if not impossible) given the vast number of variables that intervene in the process and that cannot be controlled (e.g. network delays, kernel policies, parallel processes, etc.). An application where reproducibility is desirable can be found in simulations of swarm robotics where behavior is controlled by PAPSO and robots are modelled as particles [10].

In order to provide a controlled environment where experiments from results are reproducible, yet maintaining the properties of PAPSO, we propose the Random Asynchronous PSO where we model the factors that intervene during the optimization process in PAPSO by randomly selecting particles to perform their operations in an asynchronous fashion. Such randomness will cause particles to operate having information that might be partially (or even totally) obsolete about their neighborhood. Moreover, particles may be chosen to operate more than once during one iteration, or not at all be chosen during several iterations, thus causing an *unbalance* in terms of number of updates per particle.

In this contribution, we provide a detailed insight on the performance of RAPSO in ten well-known benchmark functions and we compare its results with respect to APSO to assess their performance. Specifically, we focus on the following objectives:

- 1) Compare the performance of RAPSO and APSO on ten well-known benchmark functions using the quality of their results and speed of convergence as performance indicators.
- 2) Perform statistical significance tests to measure the importance of the differences between RAPSO and APSO according to the performance indicators.
- 3) Assess the effect of network topologies in RAPSO and APSO with regards to the performance indicators.

II. PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization was invented by Eberhart and Kennedy in 1995 [1] with inspiration on social models (e.g. bird flocking, fish schooling) and swarming theory. It is a population-based algorithm in which its individuals (known

as particles) encode potential solutions to n -dimensional optimization problems and explore the search space through cooperation with other particles. The cooperation takes place by communicating the best solutions found so far and moving towards them.

Particles have a position vector $\mathbf{x}(t)$ that encodes a potential solution to the problem, and a velocity vector $\mathbf{v}(t)$ that determines the change in the position according to

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

The velocity vector balances the trade-off between exploration and exploitation of the search: high velocities result in large changes in the position of the particles (exploration), whereas low velocities produce small changes (exploitation). The velocity vector is computed for each dimension i as follows

$$v_i(t+1) = wv_i(t) + c_1r_1(t)[y_i(t) - x_i(t)] + c_2r_2(t)[\hat{y}_i(t) - x_i(t)] \quad (2)$$

where w is the inertia of the particle [11], c_1 and c_2 are positive acceleration coefficients (a.k.a. cognitive and social components) that weigh the importance of the personal and neighborhood knowledge, $r_1(t)$ and $r_2(t)$ are random values sampled from a uniform distribution, $y_i(t)$ is the best position found in dimension i (from $t = [0, t]$) by the particle itself, and $\hat{y}_i(t)$ is the best one found (from $t = [0, t]$) by its neighborhood.

A. Network topology

The network topology of the swarm defines the neighbors of each particle and how the interactions between them takes place. Several topologies have been proposed in the literature (see [2] for a survey review), but we focus only in the **ring** and **star** network topology.

The **ring** topology defines the neighborhood of each particle p_i according to Equation 3,

$$\mathcal{N}_i = \bigcup_{j=i-m}^{i+m} p_j \text{ with } m = \left\lfloor \frac{n}{2} \right\rfloor \bmod |\mathcal{S}| \quad (3)$$

where *mod* refers to the modulo operator using the Euclidean definition (for a convenient handling of negative j -values), n is the number of neighbors, and \mathcal{S} refers to the swarm. Notice that each particle belongs to its own neighborhood and to those of the $\lfloor \frac{n}{2} \rfloor$ adjacent particles, making neighborhoods overlap. Furthermore, we consider neighborhoods to be symmetrical so each particle has the same number of neighbors in both sides.

The **star** topology is a particular case of the **ring** topology when the size of the neighborhood equals the size of the whole swarm ($n = |\mathcal{S}|$). The implications of such fully connected swarm is that the best particle of the swarm is shared amongst all neighborhoods, hence all particles move towards it.

PSO algorithms using the **star** topology are often referred to as *gbest* or Global Best PSO, and when the **ring** topology is used with $n = 2$, the PSO algorithms are then referred to as *lbest* or Local Best PSO [2].

B. Synchronous updates

Particles in the original PSO algorithm have perfect information about their neighborhood, that is, all particles know the current best solution of their neighborhood. Particles operate considering such information and move towards such solution. We refer to this algorithm as the Synchronous PSO (SPSO), described in Algorithm 1.

```

while not stopping condition do
  foreach Particle  $p$  in Swarm do
     $p$ .evaluate();
     $p$ .communicate();
  end
  foreach Particle  $p$  in Swarm do
     $p$ .update();
  end
end

```

Algorithm 1: Synchronous PSO (SPSO).

The action `evaluate` performed by each particle computes the value of the objective function of the current solution. Next, the particle communicates the solution and its quality to the particles in the neighborhood only if such solution is better than the previous one. By the end of the first loop, all particles know which one is the best solution in their respective neighborhoods. Finally, they all proceed to update their velocity and position according to Equations 2 and 1.

C. Asynchronous updates

1) *Asynchronous PSO (APSO)*: This variant differs with respect to SPSO in terms of the information available to the particles prior operation. In asynchronous updates, particles have partial information about the current best solution, that is, they know the current best on one side of the neighborhood and the previous best on the other side. Therefore, we say the operate considering partial or imperfect information about the neighborhood. We refer to this algorithm as the Asynchronous PSO (APSO), and is described in Algorithm 2.

```

while not stopping condition do
  foreach Particle  $p$  in Swarm do
     $p$ .evaluate();
     $p$ .communicate();
     $p$ .update();
  end
end

```

Algorithm 2: Asynchronous PSO (APSO).

Particles in this variant perform all the operations as soon as they are selected, causing the algorithm to be less computationally expensive than SPSO. However, SPSO can yield better results and at similar or faster speeds of convergence than APSO for many benchmark functions [4].

2) *Parallel APSO (PAPSO)*: PAPSO was originally modeled in a grid computing environment [5] where a master processor was in charge of updating the position and velocity of all particles using the information available at the time, and then proceeded to assign one slave processor per particle to

evaluate the objective function on the encoded solution. Once slave processors finished their task, they reported the computed value to the master processor. This variant is described in Algorithm 3, where the instruction **parallel** indicates that each particle has one processor assigned exclusively to perform all of its own operations, and the instruction **barrier** indicates that the algorithm finishes as soon as all particles meet the stopping condition.

```

parallel foreach Particle  $p \in \mathcal{S}$  do
  while not stopping condition do
    p.evaluate();
    p.communicate();
    p.update();
  end
end
barrier

```

Algorithm 3: Parallel APSO (PAPSO)

In absolutely idealistic environments, with no external factors causing delays on the operations and all particles communicating their solutions at the exact same time, PAPSO would behave as a Parallel Synchronous PSO (PSPSO) [12] since all particles would perform their operations simultaneously and they would all have perfect information before updating their velocities and positions. However, this is not the case in real world, where delays cause particles to have assigned computational resources more often than others, leading to unbalanced numbers of updates in particles across the swarm. Furthermore, since delays are not constant, reproducibility or results become an issue in this kind of environments.

3) *Random APSO*: This is the variant we propose to model the behavior of PAPSO previously described. In each iteration, particles are selected randomly (following a uniform distribution) to perform their operations. Thus, in every iteration the order in which information flows between particles and neighborhoods is different from the previous iterations. Moreover, a particle can be selected more than once within the same iteration, or not be selected at all, recreating the unbalanced numbers of updates that occurs in PAPSO. We call this variant Random Asynchronous PSO (RAPSO), and it is described in Algorithm 4.

```

while not stopping condition do
  for  $i \leftarrow 1$  to  $|\mathcal{S}|$  do
    Particle  $p \leftarrow \text{randomParticle}(\mathcal{S})$ ;
    p.evaluate();
    p.communicate();
    p.update();
  end
end

```

Algorithm 4: Random APSO (RAPSO).

III. EXPERIMENTAL DESIGN

A. Benchmark functions

We chose the same ten well-known benchmark functions (unimodal and multimodal) we used previously [4], but using the Schwefel function instead of the Rosenbrock because

it was found that the latter function has two minima for four or more dimensions [13], thus rendering it multimodal. All functions are minimization problems and are detailed below:

- 1) The **Quadratic** function (unimodal)

$$f_1(\mathbf{x}) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2 \text{ where } -100 \leq x_j \leq 100.$$

- 2) The **Quartic** function (unimodal)

$$f_2(\mathbf{x}) = \sum_{i=1}^n i x_i^4 \text{ where } -1.28 \leq x_i \leq 1.28.$$

- 3) The **Schwefel** function (unimodal)

$$f_3(\mathbf{x}) = \sum_{i=1}^n x_i + \prod_{i=1}^n x_i \text{ where } -5.12 \leq x_i \leq 5.12.$$

- 4) The **Spherical** function (unimodal)

$$f_4(\mathbf{x}) = \sum_{i=1}^n x_i^2 \text{ where } -5.12 \leq x_i \leq 5.12.$$

- 5) The **HyperEllipsoid** function (unimodal)

$$f_5(\mathbf{x}) = \sum_{i=1}^n i x_i^2 \text{ where } -5.12 \leq x_i \leq 5.12.$$

- 6) The **Ackley** function (multimodal)

$$f_6(\mathbf{x}) = 20 + e - 20 \exp \left[-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right] - \exp \left[\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i) \right]$$

where $-32.768 \leq x_i \leq 32.768$.

- 7) The **Griewank** function (multimodal)

$$f_7(\mathbf{x}) = 1 + \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos \left(\frac{x_i}{\sqrt{i}} \right)$$

where $-600 \leq x_i \leq 600$.

- 8) The **Rastrigin** function (multimodal)

$$f_8(\mathbf{x}) = 10n + \sum_{i=1}^n x_i^2 - 10 \cos(2\pi x_i)$$

where $-5.12 \leq x_i \leq 5.12$.

- 9) The **Salomon** function (multimodal)

$$f_9(\mathbf{x}) = 1 - \cos \left(2\pi \sqrt{\sum_{i=1}^n x_i^2} \right) + 0.1 \sqrt{\sum_{i=1}^n x_i^2}$$

where $-600 \leq x_i \leq 600$.

- 10) The **EggHolder** function (multimodal)

$$f_{10}(\mathbf{x}) = \sum_{i=1}^{n-1} - (x_{i+1} + 47) \sin \left(\sqrt{|x_{i+1} + \frac{x_i}{2} + 47|} \right) - x_i \sin \left(\sqrt{|x_i - (x_{i+1} + 47)|} \right)$$

where $-512 \leq x_i \leq 512$.

Figure 1 shows the complexity of these functions in two dimensions. These functions are plotted using the range previously defined in each for coordinates x and y of the image, and the result of $f(x, y)$ is represented in a grayscale where darker tones mean lower values and lighter tones mean higher values. The coordinate $(0, 0)$ is located in the middle of the image, and it represents the global minima in all functions except for EggHolder which global minimum is located approximately at $f(512, 404) = -959.57$ (near the upper-right corner).

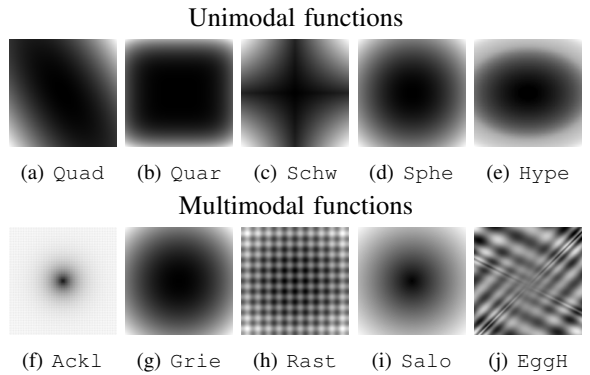


Figure 1: Benchmark functions in two dimensions.

B. Experimental setup

The comparison between RAPSO and APSO is performed on the benchmark functions previously described. The results of both variants in each function will determine which one yields better results. In order to assess the effect of the network topology, we change the neighborhood size from $n = 2$ to $n = 30$ (from *lbest* to *gbest*) for each benchmark function.

We consider a similar configuration for each swarm as in our previous work [4]. The swarms have 30 particles with 30 dimensions initialized randomly from a uniform distribution. The velocity of the particles is constrained using the hyperbolic tangent function [2]. The values regarding acceleration coefficients and inertia are chosen according to the guidelines in [14]. Table I summarizes the parameters used in all independent runs.

Table I: Algorithm parameters.

Parameter	Value
Independent runs	50×300 iterations
Number of particles	30 in \mathbb{R}^{30}
Topology	Ring with $n = \{2, 6, 14, 22, 30\}$
Acceleration	Static with $c_1 = c_2 = 1.49618$
Inertia	Static with $w = 0.729844$
Velocity clamping	hyperbolic tangent
Maximum velocity	$0.25 \cdot x_{\max} - x_{\min} $

We are going to use the quality of the results and the speed of convergence as performance indicators to assess which variant is better. The quality of results is measured by considering the best result obtained in each independent run, thus, having 50 results per variant on each benchmark function. The speed of convergence is measured as we did in [4], that is, considering the Area Under the Curve (AUC) of the evolutionary curves depicted by the quality of the best solution in each iteration during the whole optimization process. The AUC, also known as hypervolume indicator [15], is a unary metric that in this case equally balances the importance of having steep descents in the evolutionary curve (i.e. significant improvements in quality) and the quality of the final solution. Thus, we also have 50 values that indicate the speed per variant on each benchmark function. Finally, since we are dealing with minimization problems, the lower the results are, and the smaller the AUC is, the better quality and faster convergence a variant has.

The results will be shown by means of boxplots to visually assess which variant is better according to each performance indicator. We further perform statistical tests on these results in order to determine more accurately which variant is able to yield results that are significantly better.

The Wilcoxon test (with a significance level of $\alpha = 0.95$) is the statistical test we chose to measure the significance of the difference in quality between both variants on each benchmark function. The reasons for such choice are that 1) it does not assume the normality of the samples, and 2) it has already shown to be helpful analyzing the behavior of metaheuristics such as evolutionary algorithms [16].

IV. RESULTS AND DISCUSSION

A. Quality of the results

The quality of the results yielded by both PSO variants is shown in Figure 2, where boxplots display the distribution of the 50 best results obtained by each configuration in the different benchmark functions.

The configurations are denoted by letters from *a* to *e* represent the different neighborhood sizes ($n_a = 2$, $n_b = 6$, $n_c = 14$, $n_d = 22$, $n_e = 30$) for the APSO variant, and when the letter is starred (*) represents the results from the RAPSO variant. The boxplots do not include outliers to allow a better visualization of the results, and those boxplots with gray background were scaled down to improve the visualization of other boxplots within the figure. The scaling factors were applied on: Quartic in $\dot{a} = \frac{1}{20}$, Schwefel in $\dot{a} = \frac{1}{2}$, Spherical in $\dot{a} = \frac{1}{10}$, and HyperEllipsoid in $\dot{a} = \frac{1}{8}$.

We observe that the RAPSO variant yields better results than APSO in all unimodal functions regardless the neighborhood size, except for *Quadric* in which the differences are not clear enough. If we observe the results from the statistical tests (Table II), we notice that RAPSO yielded significant better results than APSO in most cases. Moreover, we easily observe that in all these functions larger neighborhoods yield better results, which is expected as the best solution propagates faster to the whole swarm and the problem does not present difficulties with local minima.

In multimodal problems we observe similar patterns in *Griewank* and *Salomon* to those in unimodal functions: larger neighborhoods yield better results. This behavior was observed in our previous research [4] as well, and it is not a surprise since in Figure 1 both functions resemble unimodal functions. However, they are not, and this can be checked in their respective equations where cosine functions ensure multimodality is a property of the function. The results from the other functions show that a large neighborhood leads to a premature high exploitation which results in solutions around local minima. Again, this is expected to happen in multimodal functions according to the difficulty involved.

Regarding both variants, we observe that RAPSO is better in all configurations only on *Griewank*. In the other benchmark functions, the differences are not that clear, suggesting that their results are quite similar. In fact, we can confirm our observations checking Table III, where both variants had differences in general not statistically significant. Nevertheless, RAPSO had seven configurations which were significantly better.

In summary, the worst case scenario for RAPSO is to produce similar results as APSO. Nevertheless, remark that in unimodal functions results are significantly better. Therefore, based on these results, we have strong evidence to claim that so far RAPSO is a more attractive alternative than APSO in terms of quality of results.

B. Speed of convergence

The speed of convergence computed for both PSO variants in each benchmark function is shown in Figure 3. The boxplots

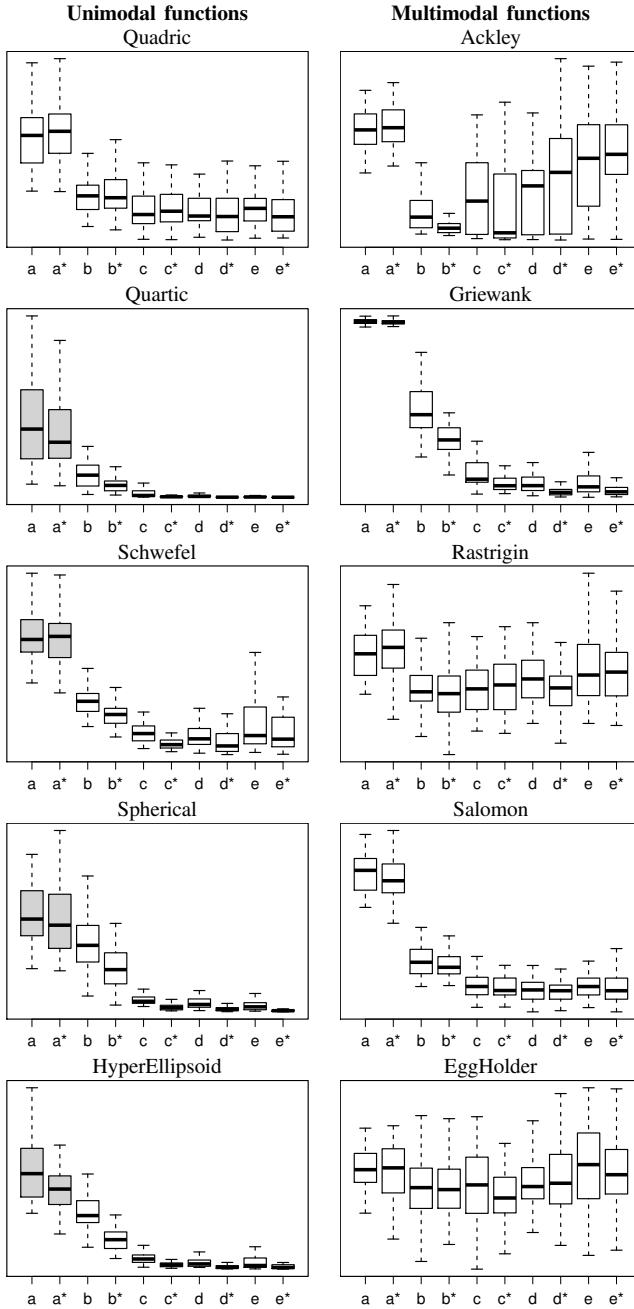


Figure 2: Quality of results in benchmark functions.

Table II: Quality in unimodal functions.

	Quad	Quar	Schw	Sphe	Hype	n
a/a^*	\neq	\neq	\neq	\neq	$+/-$	2
b/b^*	\neq	$+/-$	$+/-$	$+/-$	$+/-$	6
c/c^*	\neq	$+/-$	$+/-$	$+/-$	$+/-$	14
d/d^*	\neq	$+/-$	$+/-$	$+/-$	$+/-$	22
e/e^*	$+/-$	$+/-$	\neq	$+/-$	$+/-$	30

Table III: Quality in multimodal functions.

	Ackl	Grie	Rast	Salo	EggH	n
a/a^*	\neq	\neq	\neq	\neq	\neq	2
b/b^*	$+/-$	$+/-$	\neq	\neq	\neq	6
c/c^*	$+/-$	$+/-$	\neq	\neq	\neq	14
d/d^*	\neq	$+/-$	$+/-$	\neq	\neq	22
e/e^*	\neq	$+/-$	\neq	\neq	\neq	30

display the distribution of the values computed by the AUC.

In unimodal functions, the speed of convergence followed a similar pattern as the quality of results: larger neighborhoods had faster speeds of convergence. This behavior is expected since the information about the best solution is propagated faster within the whole swarm, contrary to the slower propagation in smaller neighborhoods. More importantly, we observe that RAPSO achieves faster speeds of convergence than APSO in most functions. This can be further detailed in Table IV, where statistical tests show that RAPSO is significantly faster in all functions with a few exceptions.

In multimodal functions, it can be seen that RAPSO is faster in many configurations, especially in Griewank and Salomon for larger neighborhoods. However, there are some cases like in Rastrigin where RAPSO seems similar or even worse. Our observations are supported by the statistical tests, where RAPSO had significantly faster speeds of convergence than APSO in more than half of the cases (14 out of 25). Regarding the rest, there were just five cases (out of 25) in which RAPSO was slower, and similar in the remaining six.

In summary, we have seen that the speed of convergence for RAPSO is generally faster than APSO in unimodal and most multimodal functions, especially for neighborhoods larger than $n = 2$.

V. CONCLUSIONS AND FUTURE WORK

The initial studies on parallel implementations of asynchronous PSO have been focused mostly on computational efficiency and not so much on addressing the issue of reproducibility. In this contribution we have presented a serial approach for modeling the behavior of the parallel asynchronous PSO while preserving reproducibility given the same initial conditions, and not requiring additional computational resources such as grid computing or multi-core processors. Moreover, its implementation does not require the use of third party libraries to deal with messaging across computers in a network, handling multiple threads, or writing particular code for GPU processors or alike.

More importantly, our RAPSO algorithm significantly outperforms the classical APSO in the unimodal functions we tackled. In more difficult functions such as those with multimodal properties, our algorithm is as good as the APSO. Furthermore, the speed of convergence of RAPSO is generally faster than that of APSO regardless the modality of the function, as long as the neighborhood sizes are greater than $n = 2$. Therefore, based on these results, RAPSO becomes a more attractive alternative to classical APSO with not additional complexity added.

A final word on the importance of reproducibility goes to applications in swarm robotics where robots are modeled as particles and the swarm behavior as an objective function (examples of these works can be found in [17], [18], [19] and many others). In these applications, the reproducibility given by RAPSO is desirable in order to be able to study with greater detail the behavior of robotic swarms driven by PSO in simulations prior to deployment in real world robots.

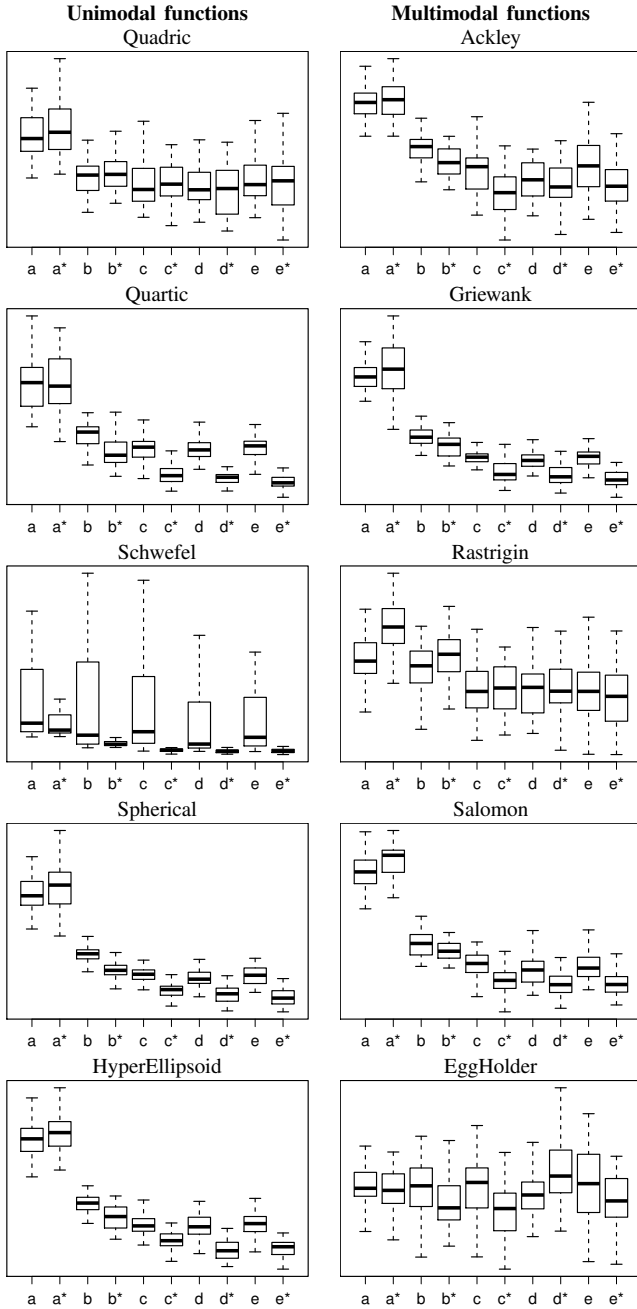


Figure 3: Speed of convergence in benchmark functions.

Table IV: Speed of convergence in unimodal functions.

	Quad	Quar	Schw	Sphe	Hype	n
a/a^*	=/=	=/=	+/-	=/=	=/=	2
b/b^*	=/=	+/-	+/-	+/-	+/-	6
c/c^*	=/=	+/-	+/-	+/-	+/-	14
d/d^*	=/=	+/-	+/-	+/-	+/-	22
e/e^*	=/=	+/-	+/-	+/-	+/-	30

Table V: Speed of convergence in multimodal functions.

	Ackl	Grie	Rast	Salo	EggH	n
a/a^*	=/=	-/+	-/+	-/+	=/=	2
b/b^*	+/-	+/-	-/+	+/-	+/-	6
c/c^*	+/-	+/-	=/=	+/-	+/-	14
d/d^*	=/=	+/-	=/=	+/-	-/+	22
e/e^*	+/-	+/-	=/=	+/-	+/-	30

In the future, we intend to extend this work by:

- Assessing the effect of unbalanced updates in RAPSO.
- Performing studies on the dynamics of APSO and RAPSO to understand the difference in quality of results.
- Comparing RAPSO and APSO on other benchmark functions, especially in more difficult multimodal functions since Griewank and Salomon do not pose much of a difficulty for the PSO variants considered.
- Experimenting with different parameters such as swarm size, dynamic neighborhoods, and randomly selecting particles in RAPSO using distributions that are not uniform.

REFERENCES

- [1] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.
- [2] A. Engelbrecht, *Computational Intelligence: An introduction*, 2nd ed. John Wiley & Sons Ltd, 2007.
- [3] A. Carlisle and G. Dozier, "An off-the-shelf PSO," in *Workshop on Particle Swarm Optimization*, 2001, pp. 1–6.
- [4] J. Rada-Vilela, M. Zhang, and W. Seah, "A performance study on synchronous and asynchronous updates in particle swarm optimization," in *GECCO'11*, Dublin, Ireland 2011, pp. 21–28.
- [5] B.-I. Koh, A. George, R. Haftka, and B. Fregly, "Parallel asynchronous particle swarm optimization," *International Journal for Numerical Methods in Engineering*, vol. 67, pp. 578–595, January 2006.
- [6] J. Luo and Z. Zhang, "Research on the Parallel Simulation of Asynchronous Pattern of Particle Swarm Optimization," *Computer Simulation*, vol. 22, no. 6, pp. 78–70 (in chinese), 2006.
- [7] J. R. Perez and J. Basterrechea, "Particle swarm optimization and its application to antenna farfield-pattern prediction from planar scanning," *Microwave and optical technology letters*, vol. 44, no. 5, pp. 398–403, 2005.
- [8] J. Schutte, "Particle Swarms in Sizing and Global Optimization," Master's thesis, University of Pretoria, South Africa, 2001.
- [9] C. Sun, C. Chiu, and C. Li, "Time-Domain inverse scattering of a two-dimensional metallic cylinder in slab medium using asynchronous particle swarm optimization," *Progress In Electromagnetics Research M*, vol. 14, pp. 85–100, 2010.
- [10] S. Xue, J. Zhang, and J. Zeng, "Parallel asynchronous control strategy for target search with swarm robots," *International Journal of Bio-Inspired Computation*, vol. 1, pp. 151–163, March 2009.
- [11] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *IEEE World Congress on Computational Intelligence*, 1998, pp. 69–73.
- [12] J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka, and A. D. George, "Parallel global optimization with the particle swarm algorithm," *Journal of Numerical Methods in Engineering*, vol. 61, pp. 2296–2315, 2003.
- [13] Y.-W. Shang and Y.-H. Qiu, "A note on the extended rosenbrock function," *Evolutionary Computation*, vol. 14, no. 1, pp. 119–126, 2006.
- [14] F. van den Bergh, "An analysis of particle swarm optimizers," Ph.D. dissertation, University of Pretoria, South Africa, 2002.
- [15] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler, "Theory of the hypervolume indicator: optimal μ -distributions and the choice of the reference point," in *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*. ACM, 2009, pp. 87–102.
- [16] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization," *Journal of Heuristics*, vol. 15, no. 6, pp. 617–644, 2008.
- [17] J. Pugh and A. Martinoli, "Inspiring and modeling multi-robot search with particle swarm optimization," in *IEEE Swarm Intelligence Symposium, 2007. SIS 2007*, apr. 2007, pp. 332–339.
- [18] J. Hereford, "A Distributed Particle Swarm Optimization Algorithm for Swarm Robotic Applications," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 2006, pp. 1678–1685.
- [19] R. Kulkarni and G. Venayagamoorthy, "Particle swarm optimization in wireless-sensor networks: A brief survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 41, no. 2, pp. 262–267, march 2011.