4th Inverse Problems, Design and Optimization Symposium (IPDO-2013)
Albi, France, June 26-28, 2013

MODERN OPTIMIZATION ALGORITHMS AND PARTICLE SWARM VARIATIONS

Eric J. Inclan[a], George Dulikravich[a], Xin-She Yang[b]
[a]*MAIDROC Laboratory, Department of Mechanical and Materials Engineering, Florida International University, Miami, Florida, USA,* einc1001@fiu.edu
[b]*School of Science and Technology, Middlesex University, London NW4 4BT, UK,* xy227@cam.ac.uk

Abstract

Many designs and inverse problems can be formulated as optimization problems. Due to the cost of evaluating real-world objective functions, optimization algorithms must be both fast and robust. While Particle Swarm (PS) is one of the fastest and most robust optimization algorithms ever developed, it cannot solve all problems and is poorly suited in many cases. This paper proposes a modification to PS and investigates five algorithms similar to PS: Quantum Particle Swarm (QPS), Modified Quantum Particle Swarm (MQP), Firefly Algorithm (FA), Cuckoo Search (CKO), and Bat-Inspired Metaheuristic Algorithm (BAT). Their performance is compared to standard PS using a subset of the Schittkowski & Hock's test cases. The modified PS is observed to outperform standard PS in twenty-seven percent of the test cases. QPS and MQP similarly outperform PS in roughly twenty-nine percent of test cases, while BAT and FA outperform PS in twenty-six percent of test cases.

Nomenclature

$\vec{B}$	vector, random perturbation of global best design
$\vec{C}$	vector, mean best value
$\hat{e}$	vector, unit basis
L	scalar, width of search space (domain)
max_gen	scalar, maximum number of generations (iterations)
N	scalar, normally distributed random number
$\vec{p}$	vector, local attractor
pop	scalar, population size (number of elements in set of design vectors)
r	scalar, Euclidean distance between two points
R	scalar, a uniformly distributed random number
$\vec{X}$	vector, design (solution)
$\vec{V}$	vector, velocity
$\vec{u}$	vector, perturbation
U(x)	objective function evaluated at x
α, β, β_o, β_{min}, γ, ρ, p_a	scalars, user-defined constants
A, f, f_{min}, f_{max}, ϕ, σ	scalars, algorithm-specific
ϵ	is an element of
Γ	gamma function
Σ	summation
g	superscript, generation (iteration) counter
G	subscript, denotes global
i, j, k	subscript, miscellaneous

1. Introduction

Particle Swarm (PS) is one of the most effective nature-inspired optimization algorithms ever developed. However, over the past eight years new methods have been proposed that are capable of outperforming PS in speed and robustness. Several of the methods presented in this paper represent viable

alternatives to PS and are either directly derived from PS, or bear at least some semblance to PS. Furthermore, a modification to PS is proposed that can increase its speed, robustness, or both.

2. Optimization Algorithms

2.1 Particle Swarm (PS)

Particle Swarm has become very popular due to its simplicity and speed. It is based on the social behavior of various species and uses an update equation to generate new solutions [1]. The basic PS algorithm is given below.

1) Create initial set (population) of solutions.
2) Evaluate objective function(s) for each solution.
3) Store copy of initial population to serve as individual best vectors and store the global best.
4) Begin main loop:
   a. For each solution in the population,
      i. Apply update equation (1).
      ii. Evaluate objective function(s) using new solution.
      iii. Replace individual best with new solution if new solution is superior.
   b. Replace global best if best new solution is superior to previous global best.
5) End main loop once population converges or maximum number of iterations is reached.

The update equations are,

$$\vec{X}_i^{\,g+1} = \vec{X}_i^{\,g} + \vec{V}_i^{\,g} \tag{1}$$

$$\vec{V}_i^{\,g} = \alpha \vec{V}_i^{\,g-1} + \beta R_1 \left(\vec{X}_{best,i} - \vec{X}_i \right) + \beta R_2 \left(\vec{X}_{best,G} - \vec{X}_i \right) \tag{2}$$

where $\vec{V}$ is the so-called "velocity vector," α and β (0.5 and 2, respectively) are user defined scalars, g is the iteration number, and R_1 and R_2 are uniformly distributed random numbers $\in [0,1]$. The vector $\vec{X}_{best,i}$ corresponds to the best value ever held by the i^{th} solution (referred to here as the "individual best", and $\vec{X}_{best,G}$ is the best solution ever found (also known as the "global best"). The first term on the right of (2) is the "inertia," which is effectively a scalar multiple of the velocity from the previous iterations.

Since the velocity magnitude can grow each iteration, it is possible for the inertia to become so large that PS oscillates erratically or diverges altogether. To help mitigate this risk the following condition was imposed (written in C-style pseudo-code),

$$\begin{aligned} &while\ (V_k > L/3) \\ &\quad V_k = V_k / 2 \end{aligned} \tag{3}$$

where V_k is the k^{th} dimension of the velocity vector. The condition above causes the velocity component to be halved until its magnitude is less than one-third the width of the search space, L, for that dimension. This restriction is applied before the velocity is added to the individual vector. A second restriction was also added to the velocity vector, as follows (again in pseudo-code)

$$while\ (X_{i,k} + V_{i,k}\ is\ outside\ \ domain) \qquad V_{i,k} = V_{i,k} \times 0.9 \tag{4}$$

This restriction causes the entire velocity vector to be scaled down further if it would cause the new solution to move outside of the search domain.

2.2 Quantum-Behaved Particle Swarm (QPS)

First proposed in 2004 and inspired by quantum mechanics, QPS treats the individual solutions of PSO as quantum particles [6]. The fundamental difference between this algorithm and PS is simply the update equation. QPS no longer utilizes an inertia term, and (1) is replaced with

$$\vec{X}_i^{g+1} = \vec{p}_i \pm \alpha \left| \vec{C}^g - \vec{X}_i^g \right| \ln\left(1/R_3 \right) \tag{5}$$

where $\vec{p}_i$ is the "local attractor" defined using an equation reminiscent of (2),

$$\vec{p}_i = \phi \vec{X}_{best,i} + (1-\phi)\vec{X}_{best,G} \tag{6a}$$

$$\phi = \frac{\beta R_1}{\beta R_1 + \gamma R_2} \tag{6b}$$

and $\vec{C}^g$ is the "mean best value" which is simply the arithmetic mean of the individual best vectors for each generation. The scalars R_1, R_2 and R_3 are random numbers $\in [0,1]$, and α is a user-defined parameter, set here to decrease linearly during each generation from 1 to 0.5, per the authors' recommendation in [6]. The scalars β and γ are also user-defined scalars (here $\beta = \gamma = 2$). Note that the $\pm$ operation in (5) is resolved simply by assigning a 50% probability that the term will be either positive or negative.

2.3 Modified Quantum-Behaved Particle Swarm (MQP)

There are a number of variations to QPS. One intriguing modification to this method, MQP, is to randomly modify the local attractor according to the scheme recommended in [6]:

a. Randomly select another individual best vector *j* different from $\vec{X}_{best,i}$.
b. If $U(\vec{X}_{best,j}) < U(\vec{X}_{best,i})$, utilize $\vec{X}_{best,j}$ in place of $\vec{X}_{best,G}$ in (6), otherwise, simply use (6) without modification.

In other words, this method causes (6) to be either the linear combination of vector i's individual best and the global best, or the linear combination of vector i's individual best and a randomly selected, superior individual best. The rest of the algorithm is identical to QPS. The authors suggest setting α to decrease linearly during each generation from 1 to 0.4 [6].

2.4 Firefly Algorithm (FA)

Proposed by Dr. Yang in 2009 [7], this algorithm is based on the mating behavior of fireflies. Under certain circumstances, Dr. Yang points out that FA reduces to a special case of PS [7]. Unlike any other method presented here, it uses two loops over the population with one loop nested inside the other. This makes FA a very slow algorithm for problems with large populations, which are often high-dimensional problems. The basic algorithm is as follows [3].

1) Create initial population.
2) Calculate fitness for each individual.
3) Begin main loop:
   a. Calculate alpha.
   b. For each individual, $\vec{X}_i$, in the original population, loop again over the population.

If $\vec{X}_i$ is superior to $\vec{X}_j$ (that is, if $U(\vec{X}_i) < U(\vec{X}_j)$), then

1. Calculate the Euclidean distance, r, between $\vec{X}_i$ and $\vec{X}_j$.
2. Calculate the attractiveness.
3. Generate a small random perturbation vector.
4. Apply update equation.

Evaluate fitness of new population.

4) End main loop once population converges or maximum number of iterations is reached.

The attractiveness equation used in this paper is a replica of the equation provided in Dr. Yang's MATLAB code, which is,

$$\beta = \left(\beta_0 - \beta_{\min}\right) e^{-\gamma r^2} + \beta_{\min} \tag{7}$$

where γ is a user-defined parameter (here set to 1). The scalars β_0 and β_{min} are also user-defined parameters (here set to 1 and 0.2, respectively). Finally, r is the previously mentioned Euclidean distance. The small random perturbation vector is generated in the MATLAB code according to

$$\vec{u} = \sum_{i=1}^{\dim} \left(\alpha(R-0.5)\times L\right)\hat{e}_i \tag{8}$$

where α is a user-defined parameter that decreases linearly with the generation number from 0.25 to 0, R is a uniformly distributed random number ϵ [0,1], and L is the width of the search domain in the $\hat{e}_i$ direction. Finally, the update equation is given as

$$\vec{X}_i^{\,g+1} = (1-\beta)\vec{X}_i^{\,g} + \beta\vec{X}_j^{\,g} + \vec{u} \tag{9}$$

The form and implementation of (9) are very important. One might conclude from Dr. Yang's paper that each firefly should be attracted to the single brightest firefly from its perspective. However, when implemented, (algorithm step 3.b.i) the firefly is actually compared to every firefly that is superior to it and moved in that direction without re-evaluating the objective function. Therefore, a single firefly may have its position updated several times before reaching its final location and, since the previous position is included in each update, this equation effectively compounds the previous steps.

2.5 Bat-Inspired Algorithm (BAT)

Also proposed by Dr. Yang [8], this algorithm resembles DE in that it uses comparisons to update each new generation of candidate solutions. Reminiscent of best/2/bin it partially incorporates perturbations of the best solution into its formulation, and like sociability-only-PS, it searches the function space based on a multiple of the difference between the global best and a particular individual. However, each of these steps are performed separately, and the Differential Evolution (DE) style comparison is used at the end to determine which candidate solutions are retained and which are discarded. The algorithm can be expressed as follows [8],

1) Create initial population.
2) Calculate fitness for each individual.
3) Begin main loop:
   a. Calculate *A*.
   b. Copy the original population to a temporary population.
   c. For each individual in the temporary population,
      i. Calculate the frequency and velocity.
      ii. Update the temporary population using the velocity.
      iii. If ($R_1 > \rho$),
         1. Replace the given member of the temporary population with a random perturbation of the global best individual
   d. Evaluate fitness of temporary population.
   e. If new temporary population member is better than original and if (R_2 < A), replace original.
4) End main loop once population converges or maximum number of iterations is reached.

The scalar ρ in step 3.c.iii is a user-defined parameter. In this paper ρ = 0.5, but the comments in Dr. Yang's MATLAB code suggests that it can be an increasing function of the generation number. The scalars R_1 and R_2 are uniformly distributed random numbers ϵ [0,1]. The scalar A is defined by the recursive formula,

$$A^g = A^{g-1}\left(\frac{1}{9000}\right)^{\frac{1}{\max\_gen}} \tag{10}$$

with A initialized to 0.25. The variable max_*gen* is the maximum number of generations allowed for a given optimization run. Recall that *g* is the generation number, and not an exponent. For step 3.c.i, the frequency, *f*, is given by,

$$f = f_{\min} - (f_{\max} - f_{\min}) \times R \tag{11}$$

where *R* is a uniformly distributed random number ∈ [0,1], and $f_{\min}$ and $f_{\max}$ are user-defined parameters set to 0 and 2, respectively. Similar to PS, the velocity is given by,

$$V_{i,k}^g = V_{i,k}^{g-1} + (X_{i,k}^g - X_{best\,G,k}^g) \times f \tag{12}$$

where a new frequency is generated for each velocity vector component (denoted by *k*). As in PS, the bat update equation is simply,

$$\vec{X}^g = \vec{X}^{g-1} + \vec{V}^g \tag{13}$$

The equation defining the random perturbation of the global best individual (B) is,

$$B_{i,k}^g = X_{best\,G,k}^g + 0.01\left(a + (b-a) \times R\right) \tag{14}$$

where R is a uniformly distributed random number ∈ [0,1], and *b* and *a* are the upper and lower limits of the domain for the k^{th} coordinate, respectively.

2.6 Cuckoo Search (CKO)

The last of Dr. Yang's methods to be reviewed in this paper is CKO [9]. Based on the strange nesting habits of the Cuckoo bird, this method also incorporates Lévy flights (random walks) for the purpose of optimization. Like BAT, it also uses DE style comparisons. The CKO algorithm proceeds as follows [9],

1) Create initial population.
2) Calculate fitness for each individual.
3) Begin main loop:
   a. Copy original population into temporary population.
   b. Update the temporary population positions using random Lévy flights.
   c. Evaluate fitness of temporary population.
   d. Further modify the temporary population using the "empty nest" procedure.
   e. Evaluate fitness of temporary population.
   f. If new temporary population member is better than original, replace original.
4) End main loop once population converges or maximum number of iterations is reached.

The main drawback of this algorithm is that it requires two objective function evaluations per generation. In many real-world problems, evaluating the objective function is very time consuming, which makes this scheme more computationally expensive than any other algorithm. The Lévy flight update is executed according to the following equation,

$$X_{i,k}^{g+1} = X_{i,k}^{g} + \left(0.01 \frac{\sigma N_1}{|N_2|^{1/\beta}} \left(X_{i,k}^{g} - X_{best\,G,k}^{g} \right) \right) \times N_3 \tag{15}$$

where N_1, N_2 and N_3 are normally distributed numbers centered at zero, β is a user-defined parameter (here, $\beta = 1.5$), k is the dimension number, and σ is given by,

$$\sigma = \left(\frac{\Gamma(1+\beta)\sin\left(\frac{\pi\beta}{2}\right)}{\beta \times 2^{\frac{\beta-1}{2}} \Gamma\left(\frac{1+\beta}{2}\right)} \right)^{1/\beta} \approx 0.69657 \tag{16}$$

In this paper, the gamma function was evaluated based on the algorithm in [2]. The empty nest population update is executed as follows,

$$\begin{aligned} &if\ (R_1 > p_a) \\ &\quad \vec{X}_i^{g+1} = \vec{X}_i^{g} + R_2 \left(\vec{X}_j^{g} - \vec{X}_k^{g} \right) \end{aligned} \tag{17}$$

where R_1 and R_2 are uniformly distributed random numbers $\in [0,1]$, and p_a is a user-defined parameter (here $p_a = 0.25$). The two population members denoted by j and k are randomly selected from the population.

2.7 Particle Swarm With Random Differences (RD)

Inspired by the concept of random walks and the modified local attractor of MQP, this proposed modification adds versatility to PS without dramatically changing its basic procedure. Traditionally, each solution in PS is subtracted from its own individual best vector. This method introduces a probability that a solution will be subtracted from an individual best other than its own. The revised PS algorithm becomes,

1) Create initial population of solutions.
2) Evaluate objective function(s) for each solution.
3) Store copy of initial population to serve as individual best vectors and store the global best.
4) Begin main loop:
   a. For each solution in the population,
      i. If $R < R_p$
         1. Randomly shuffle the individual best vectors so that each solution is subtracted from the individual best of another solution rather than its own in step 4.a.ii.
      ii. Apply update equation (see (18)).
      iii. Evaluate objective function(s) using new solution.
      iv. Replace individual best with new solution if new solution is superior.
   b. Replace global best if best new solution is superior to previous global best.
5) End main loop once population converges or maximum number of iterations is reached.

In the above algorithm, R_p is the "probability of random differencing," a user-defined parameter $\in [0,1]$ and R is a uniformly distributed random number $\in [0,1]$. In this research, R_p decreases linearly with generation number from 0.5 initially to 0.0 upon reaching the maximum number of generations. The above algorithm is roughly equivalent to rewriting (2) as follows,

$$\vec{V}_i^{\,g} = \alpha \vec{V}_i^{\,g-1} + \beta R_1\left(\vec{X}_{best,j} - \vec{X}_i\right) + \beta R_2\left(\vec{X}_{best,G} - \vec{X}_i\right)$$
$$where\ i, j \in [1, pop]\ and\ i \neq j \tag{18}$$

where *pop* is the number of candidate solutions in the population. The requirement that *i* not equal *j* is not strictly enforced when the vectors are shuffled.

3. Results

The algorithms were applied to a subset of the Schittkowski and Hock analytical test cases [4][5], listed in Table 1. Constraints were enforced using a penalty function.

Table 1. Dimension and Population of Various Test Problems

Dimension	Population	Test Problem Number
2	20	1, 4, 8, 13, 14, 22, 201, 212, 221, 234, 324
3	28	27, 32, 35, 61, 65, 241, 245, 249, 254, 335, 342
4	34	39, 71, 76, 256, 258, 261, 265, 275, 352, 353, 354
5	39	45, 52, 53, 266, 268, 270, 358
6	43	95, 96, 97, 98, 271, 273
10	57	110, 283, 291, 374, 375
15	70	384, 385, 386, 387, 388, 389
20	81	300, 394
30	99	292, 297, 391
50	128	298, 395
100	182	299, 302

Each test problem was optimized 50 times, for 200 generations using the population sizes shown in Table 1. The algorithm used to generate these population values has been withheld, because it is still undergoing research.

Table 2 compares the aggregate accuracy of each algorithm. The first row for each method, labeled "Mean," lists the percentage of test problems for which a given method obtained the lowest average minimum (i.e. the best average accuracy). In some test problems, multiple methods converged to the global minimum, therefore, the sum of these rows exceeds 100%. The second row is the percentage of test problems for which a given method obtained the lowest standard deviation of minima. As indicated in Table 2, PS had the highest accuracy (mean and standard deviation) of any method, followed by RD. As one might expect, the performance of the RD method is sensitive to the selection of R_p. If R_p is held constant at 0.25, for example, the accuracy rating of RD drops to 16.7%.

Table 2. Comparison of Accuracy of Different Optimization Algorithms

Accuracy Comparison							
	Algorithm						
Category	QPS	MQP	RD	PS	FA	BAT	CKO
Mean	15.2%	10.6%	28.8%	**31.8%**	19.7%	6.1%	10.6%
Std. Dev.	19.7%	10.6%	27.3%	**30.3%**	10.6%	7.6%	16.7%
Best	24.2%	21.2%	12.1%	**60.6%**	27.3%	4.5%	4.5%
Worst	4.5%	0.0%	0.0%	9.1%	9.1%	**56.1%**	21.2%

In Table 2, the third row lists the percentage of test problems for which a given method found the best single result, while the fourth row relates to the worst single result.

Table 3 shows the aggregate speed of the iterative process for each algorithm. Here, the "Mean" lists the percentage of test problems for which a given method obtained the highest average rate of convergence (including premature convergence to local minima). The second row is the percentage of test

problems for which a given method obtained the lowest standard deviation of convergence rates. The third and fourth rows list the percentage of test problems for which a given method converged the fastest and the slowest, respectively (including premature convergence to local minima). The convergence rate was based on the change in value of the global best for each generation of the optimization run. The convergence rate is the slope of the secant line, subtracting the final objective function value of the global best from the initial objective function of the global best and dividing by the number of generations.

Each convergence history (Fig. 1-4) contains two curves for each method. These curves represent the best convergence history and worst convergence history for that method.

Table 3. Comparison of Convergence Rates for Different Optimization Algorithms

Convergence Rate Comparison							
	Algorithm						
Category	QPS	MQP	RD	PS	FA	BAT	CKO
Mean	9.1%	10.6%	0.0%	25.8%	6.1%	**45.5%**	3.0%
Std. Dev.	10.6%	16.7%	18.2%	4.5%	13.6%	0.0%	**36.4%**
Best	1.5%	0.0%	0.0%	4.5%	0.0%	**92.4%**	1.5%
Worst	4.5%	0.0%	1.5%	10.6%	7.6%	**62.1%**	18.2%

From Table 3 it is apparent that BAT dominates all other algorithms in terms of speed, but it is very inconsistent. As shown in Table 2, this algorithm has the highest probability of producing the single worst answer compared to the other methods. This suggests that, in its current form, it is a very greedy method. Figures 2-4 provide examples of BAT converging rapidly to a local minimum.

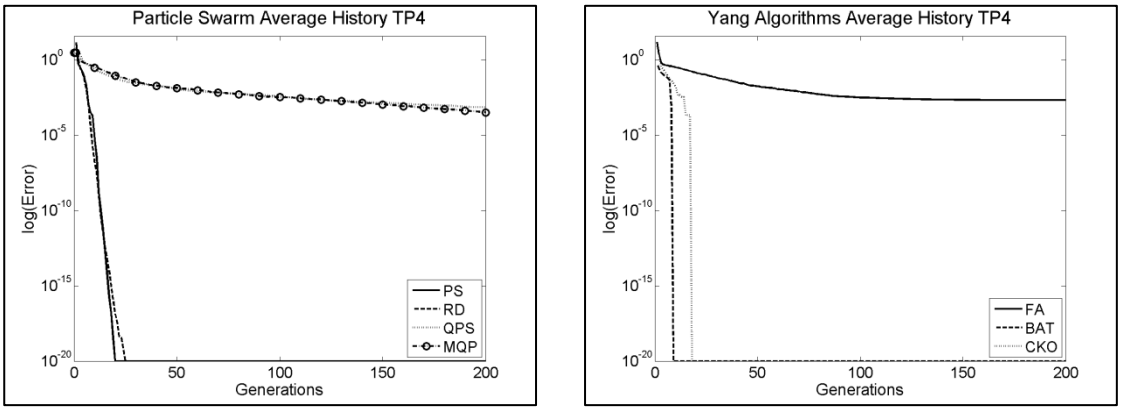
Since BAT skews the values in Table 3, Table 4 excludes BAT. Although PS is the fastest method overall and has the highest best-rate value. By comparison, FA has the highest worst-rate value. Nevertheless, its rank as third in terms of accuracy makes unmodified FA a reliable algorithm. While CKO has the highest standard deviation, making it fairly consistent in terms of speed.

Table 4. Comparison of Method Convergence Rates (excluding BAT)

Convergence Rate Comparison						
	Method					
Category	QPS	MQP	RD	PS	FA	CKO
Mean	18.2%	18.2%	3.0%	**43.9%**	10.6%	6.1%
Std. Dev.	10.6%	16.7%	18.2%	4.5%	13.6%	**36.4%**
Best	12.1%	13.6%	1.5%	**40.9%**	21.2%	10.6%
Worst	12.1%	6.1%	1.5%	13.6%	**36.4%**	31.8%

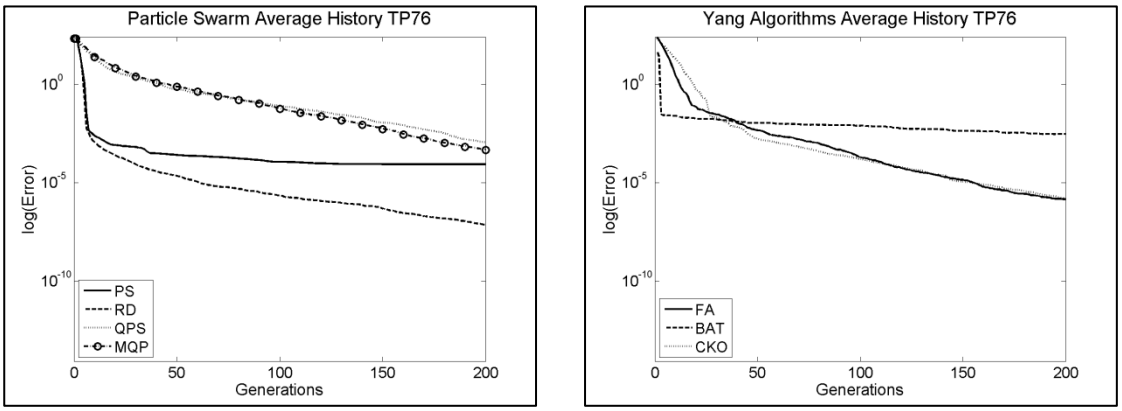
Although QPS and MQP are generally less accurate than FA, they are also faster. In many problems, such as TP110 (Fig. 3), they can outperform every other method entirely. It is worth noting that QPS, MQP, and RD each have much lower single-worst values in convergence speed and accuracy than PS. Therefore, while they do not outperform PS overall, they are not subject to the same drawbacks as PS. Although PS typically outperforms other methods in TP 4, Fig.1 demonstrates that BAT can in fact converge faster. This is quite remarkable because in all fifty trials, BAT converged to the global minimum in less than six iterations (less than 120 objective function evaluations). Although it appears that CKO also outperformed PS, recall that CKO requires two objective function evaluations per generation effectively making it slower than PS. Similar rapid convergence can be seen in TP 45, 234, and 265. QPS and MQP perform equally poorly in TP 4, and it is with noting that the two algorithms often exhibit similar performance as shown in Fig. 1 - 4.

It was frequently observed that RD converges to an answer within the range of solutions found by PS, as though its performance were bounded by that of PS. This results in RD obtaining a slightly better average result as indicated in Fig. 2. Similar behavior can be seen in one quarter of the problems listed in Table 1. The figures that follow (Fig. 3 – 4), however, highlight the strengths of the RD variation, as well as other methods.



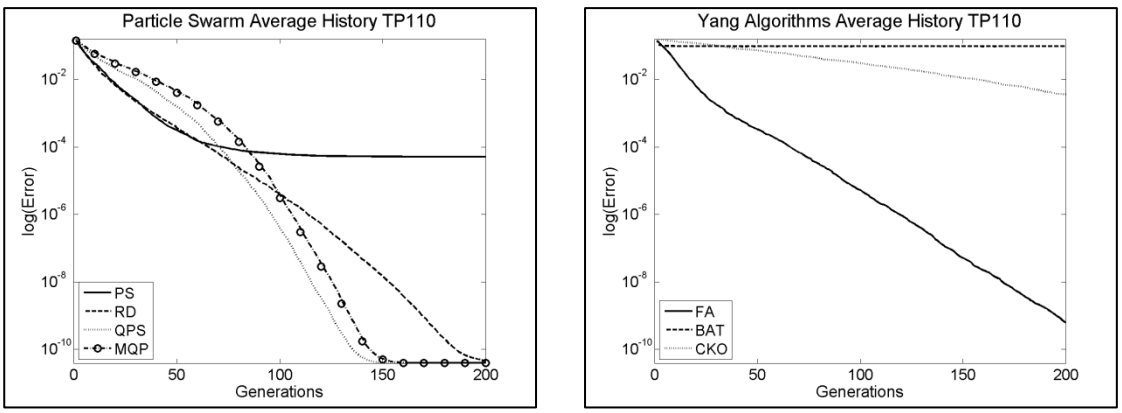
(a) (b)

Fig. 1. Convergence Histories for TP 4



(a) (b)

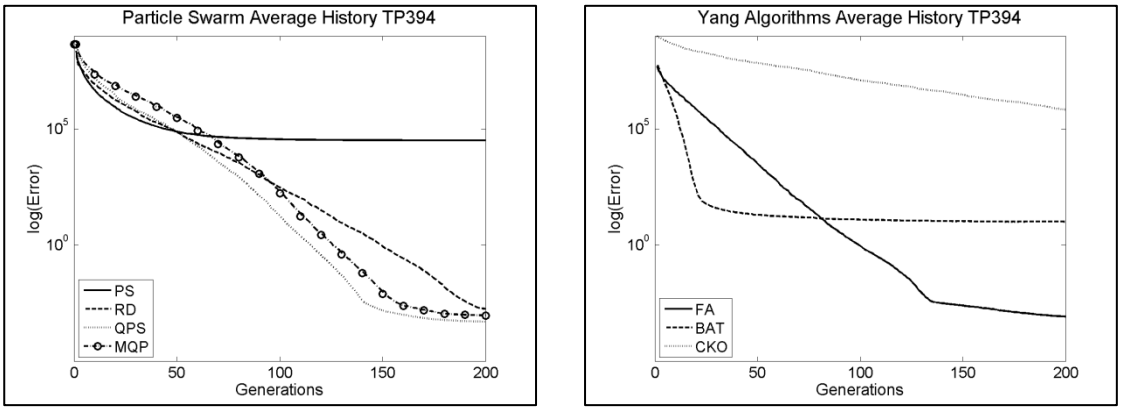
Fig. 2. Convergence Histories for TP 76



(a) (b)

Fig. 3. Convergence Histories for TP 110

The RD variation surpasses PS in convergence rate and accuracy in 27% of the test cases, while MQP and QPS surpass PS in nearly 29% of the test cases. The BAT and FA algorithms surpass PS in roughly 26% of the test cases. CKO, in its standard form, rarely surpasses PS.



(a) (b)

Fig. 4. Convergence Histories for TP 394

4. Conclusions

Among the algorithms considered here, PS obtained the highest average accuracy of any optimization algorithm when tested on a large suite of analytical test cases. However, the newer algorithms have demonstrated the ability to locate the global minimum in a variety of problems that PS cannot solve, making them viable alternatives to PS. Furthermore, RD, the proposed modification to PS, has been shown to surpass standard PS in 25% of the test cases, and scored the second-highest average accuracy of all methods considered here. Further research in modifying the formula for the probability of random differencing may lead to additional improvements in this method's performance.

Acknowledgements

This work was partially funded by the US Air Force Office of Scientific Research under grant FA9550-12-1-0440 monitored by Dr. Ali Sayir. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US Air Force Office of Scientific Research or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon.

References

[1] Kennedy, J., and Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks*, pp. 1942-1948.

[2] Lau, H. T. (1995). *A Numerical Library in C for Scientists and Engineers.* Boca Raton: CRC Press.

[3] Lukasik, S., and Zak, S. (2009). Firefly Algorithm for Continuous Constrained Optimization Tasks. In *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems* (pp. 97-206). Springer.

[4] Schittkowski, K. (1987). *More Test Examples for Nonlinear Programming Codes.* Berlin: Springer-Verlag.

[5] Schittkowski, K., and Hock, W. (1981). *Test Examples for Nonlinear Programming Codes - All Problems from the Hock-Schittkowski-Collection.* Bayeruth: Springer.

[6] Sun, J., Lai, C. H., Xu, W., and Chai, Z. (2007). A Novel and More Efficient Search Strategy of Quantum-Behaved Particle Swarm Optimization. *ICANNGA '07 Proceedings of the 8th international conference on Adaptive and Natural Computing Algorithms*, pp. 394 - 403.

[7] Yang, X.-S. (2009). Firefly Algorithms for Multimodal Optimization. *Stochastic Algorithms: Foundations and Applications, SAGA 2009, 5792*, pp. 169-178.

[8] Yang, X.-S. (2010). A New Metaheuristic Bat-Inspired Algorithm. (J. R. *et al.*, Ed.) *Nature Inspired Cooperative Strategies for Optimization (NISCO 2010), 284*, 65-74.

[9] Yang, X.-S., and Deb, S. (2010). Engineering Optimisation by Cuckoo Search. *Int. J. Mathematical Modelling and Numerical Optimisation, 1*(4), pp. 330-343.