Weightless Swarm Algorithm (WSA) for Dynamic Optimization Problems

T. Ting, Ka Man, Sheng-Uei Guan, Mohamed Nayel, Kaiyu Wan

Weightless Swarm Algorithm (WSA) for Dynamic Optimization Problems

T. O. Ting[1,*], K. L. Man[2], Sheng-Uei Guan[2], Mohamed Nayel[1] and Kaiyu Wan[2]

[1]Dept. Electrical and Electronic Eng and [2]Dept. Computer Science and Software Eng,
Xi'an Jiaotong-Liverpool University, Suzhou, China.
{toting, ka.man, steven.guan, mohamed.nayel,
kaiyu.wan}@xjtlu.edu.cn

Abstract. In this work the well-known Particle Swarm Optimization (PSO) algorithm is applied to some Dynamic Optimization Problems (DOPs). The PSO algorithm is improved by simplification instead of introducing additional strategies into the algorithm as done by many other researchers in the aim of improving an algorithm. Several parameters (w, V_{max}, V_{min} and c_2) are being excluded from the conventional PSO. This algorithm is called Weightless Swarm Algorithm (WSA) as the prominent parameter, inertia weight w does not exist in this proposed algorithm. Interestingly, WSA still works effectively via swapping strategy found from countless trials and errors. We then incorporate the proven clustering technique from literature into the framework of the algorithm to solve the six dynamic problems in literature. From the series of tabulated results, we proved that WSA is competitive as compared to PSO. As only one parameter exists in WSA, it is feasible to carry out parameter sensitivity to find the optimal acceleration coefficient, c_1 for each problem set.

Keywords: dynamic optimization, swapping, weightless swarm,

1 Introduction

Particle swarm optimization (PSO) [1] is one of the prominent algorithms in the category of nature-inspired algorithms and it has been one of the most successful numerical optimization algorithms applied in many fields [2]. One of the advantageous features of Particle Swarm Optimization is its ability to converge quickly to a potential solution. In other words, PSO is faster compared to many evolutionary algorithms such as Genetic Algorithm (GA) [3], Evolutionary Programming (EP) [4], Evolutionary Strategies (ES) [5], Differential Evolution (DE) [6] etc.

This is how PSO works. Firstly, candidate solutions (or commonly known as particles) are seeded onto the search space in a random manner. These particles will then move through the problem space in the aim of finding the global optimum. The movement is guided by the essentially important ingredient formulas:

$$V_i^t = w \square V_i^{t-1} + 2r_1\left(Pbest_i^t - X_i^t\right) + 2r_2\left(Gbest_i^t - X_i^t\right) \quad (1)$$

$$X_i^{t+1} = X_i^t + V_i^t \quad (2)$$

whereby:

V_i^t	is the velocity for ith dimension at time t
w	is the inertia weight , usually set to 0.5
X_i^t	is the current position of ith dimension at time t
$Pbest_i^t$	is the best position for ith dimension at time t of a particle, also known as personal best
$Gbest_i^t$	is the best solution among all participating particles for ith dimension at time t, also known as global best
r_1, r_2	These are independent uniform random numbers within [0, 1]

In each iteration, all the particles will be evaluated through a similar cost function. Then, the update of *Pbest* and *Gbest* values are performed instantly. In other words, the asynchronous update is adopted here. The reason for asynchronous update is that the information of *Pbest* and *Gbest* can be feedback into the whole population instantly without delay and this will accelerate the convergence rate. In literature, many works prefer asynchronous update [7, 8].

During the update of velocity, V through (1), the limit of Vmax and Vmin is imposed, usually within 10%, 50% or 100% of search space. The value chosen for Vmax and Vmin is not really crucial and does not affect the performance of PSO drastically. Also, after the update through (2), checking is done to ensure that particles only explore the predefined search space. There are many techniques to handle these boundary limits, which are beyond the scope of this paper. By simply set the value to boundary limit is one of the alternatives. Another alternative will be to impose re-initialization within the search space upon violation. The later alternative is preferred as this will increase the diversity of the entire population and hence assists in avoiding local optima. The similar boundary handling technique is adopted in this work. As the number of iteration increases, particles accelerate towards those with better fitness until maximum iteration is reached.

By careful inspection of (1) and (2), the following interpretations are valid in regards to PSO:

i. The velocity somehow acts as short-term memory retention and plays a crucial role in the update process.

ii. The update of a dimensional value is guided by Pbest and Gbest. Simply, this means that a particle moves between Pbest and Gbest.

iii. The independent random numbers r_1 and r_2 control the ratio of movement towards Pbest and Gbest.

In this work, instead of using conventional Particle Swarm Optimization (PSO), a much simpler yet robust variant is presented to solve Dynamic Optimization Problems (DOPs). This novel algorithm is given the name, Weightless Swarm Algorithm

(WSA) as the inertia weight introduced by Shi and Eberhart in the year 1998 [9] is not present in this algorithm. The work on WSA is novel and thorough work will be carried out in future to stabilize its performance. The exclusion of inertia weight reduces several other parameters such as *Velocity*, V_{max} and V_{min}. Due to this, WSA is faster compared to its original form. As the complexity of the algorithm is reduced, the tuning of the algorithm is much easier in this work.

The rest of the paper is organized as follows. Section 2 describes the essence of WSA; this includes the explanation of the strategies incorporated to enable PSO to work without inertia weight. Parameter settings and experimental results are given in Section 3 and 4 respectively. Lastly is the conclusion in Section 5.

2 The Essence of Weightless Swarm Algorithm

Weightless Swarm Algorithm (henceforth abbreviated as WSA) has the same form as the canonical PSO. Without the inertia weight, the updated equation is simplified from two-line equation to a single line:

$$X_{i,d}^{t+1} = X_{i,d}^{t} + r_1 c_1 \left(Gbest_{i,d} - X_{i,d}^{t} \right) + r_2 c_2 \left(Pbest_{i,d} - X_{i,d}^{t} \right) \tag{3}$$

whereby Xi,d is the position of dth dimension of ith particle. *Pbest* is the best position found in the search history of a particle whereas *Gbest* is the best solution found in the entire search history. r_1 and r_2 are two independent uniform random number generators within [0, 1]. The acceleration coefficient, c_1 and c_2 are both set to 1.7. Following the theoretical analysis by Clerc and Kennedy [10], a constriction factor K = 0.729 is introduced on the basis of $|c_1+c_2| \leq 4.1$. If we assume $c_1=c_2=2.05$ and $K=0.729$, the new coefficients will have the value of $0.729 \times 2.05=1.49445$. Furthermore, new results presented in [11] based on the theory of dynamic systems for analysis of a particle trajectory have been carried out with different parameter set ($w=0.6$ and $c_1=c_2=1.7$) which showed slightly superior performance. The WSA introduced here agreed to these parameter settings even without the present of inertia weight. By setting $c_1=c_2=1.7$, results are slightly improved as compared to 1.5. Interestingly, the default setting for c_1 and c_2 of PSO in EAlib [12] is also 1.7. From the results on static numerical problems, we found that equation (1) can be further simplified as:

$$X_{i,d}^{t+1} = X_{i,d}^{t} + r_1 c_1 \left(Gbest_{i,d} - X_{i,d}^{t} \right) \tag{4}$$

Using Equation (4) works effectively as when swapping is done during the update of *Pbest* and *Gbest* values; many *X* values are actually the previous *Pbest* values. Hence, it is not really necessary to learn from oneself.

Therefore, in WSA, several parameters prominent in PSO are omitted. The well-known inertia weight, w is now not present. Hence, it means that the velocity, V is also unnecessary. Without V, a user also discards the concern of the bound for this parameter, namely V_{max} and V_{min}. Also, by adopting (4) in WSA, one of the acceleration coefficients is automatically discarded. Thus, the proposed algorithm has a much simpler form compared to canonical PSO. By this form of algorithm, the complexity present is

greatly reduced and we only need to tune c_1 for optimal performance; this has been done successfully in this work.

The reduction in the complexity of the algorithm thereby results in a lower computational cost of the relevant computer program. By running both programs (PSO and WSA) concurrently, it is observed that at the point whereby WSA is completing 20 runs, the PSO is only at its 4th run. It means that the proposed method solves dynamic optimization problems five times faster compared to PSO. This is due to simpler code and more resource effective as memory allocation for both *w* and *V* are commented in the existing C++ EAlib program, available from [12].

2.1 The Trick in WSA

The secret of WSA is extremely simple and this is indeed the core in this proposed methodology. In the canonical PSO, setting *w* to zero value resulted in stagnant search; the algorithm does not seem to work. During the update of *Pbest* and *Gbest*, it has been a traditional practice that these values are replaced by better particles. However, in our proposed WSA, instead of doing replacement, swapping of values is adopted. By swapping, we increase the diversity of the population and accelerate the convergence rate. This is due to the reason that when swapping is imposed, the probability of particle *X* equal to *Pbest* or *Gbest* is significantly reduced. For instance, in the case of replacement scenario in PSO, for a given iteration, if *Pbest* is updated for 5 times, there are 5 ineffective positional updates as the term $\left(Pbest_{i,d} - X^t_{i,d} \right) = 0$. These ineffective updates are avoided in WSA, resulting in better accuracy and faster convergence given the same number of function evaluations.

2.2 Implementation of WSA

The implementation of WSA is pretty simple and can be implemented into any existing PSO algorithm with the following steps:

i. *Set inertia weight = 0,*
ii. *Discard the Pbest term by setting c_1 in equation (1) to zero.*
iii. *Swapping is done during Pbest update. The swapping for Gbest may not be necessary as in many algorithm implementations; one of the Pbest values is actually the Gbest.*

The above three steps are simple yet they improve the performance of the algorithm drastically without the need of inertia weight. This simple strategy can be implemented easily into any existing PSO algorithm.

3 Parameter Settings

In this work, as PSO is the best algorithm in undetectable dynamic environments from the results presented in [13, 14], it is adopted for comparison in this work. The settings for both algorithms are as follows:

3.1 PSO

The inertia weight, w is linearly decreased from 0.6 to 0.3 and acceleration coefficients, $c_1 = c_2 = 1.7$

3.2 WSA

Inertia weight is not present; therefore it is in fact zero. The acceleration coefficient, c_1 is found through parameter sensitivity analysis and $c_2 = 0$

4 Experimental Results

Six dynamic problems from [15] are adopted as test bed in this work. Problems such as Sphere, Rastrigin, Griewank and Ackley are well-known problems in the area of numerical optimization. The descriptions of the different change strategies T1, T2, T3… are available from [15]. Simulation results are presented in Tables 1-4 with Tables 5 recording the mean and standard deviation for results in Table 4. From Table 2, it is observed that the performance of WSA is close to PSO in Table 1. As from Table 1, the PSO recorded total overall performance of 39.98494 whereas WSA recorded a figure of 38.74686. It means that the solution of WSA is competitive compared to its predecessor. As the nature of WSA may not be the same as PSO, the parameter sensitivity analysis is carried out to obtain the optimal settings for both diversity and overlapping ratio. From the analysis, the diversity: α=0.6 and overlapping ratio: β=0.9 are suggested for optimal performance. Results of parameter sensitivity analysis are not included to avoid extended paper. Results using these settings are recorded in Table 3, now with overall performance of 40.10769 (even closer to PSO's). It is interesting to note that with the optimal parameter setting, the performance of each problem set is slightly improved.

As WSA can perform effectively even without learning from *Pbest*, equation (2) is used for the results depicted in Table 4, now with independent values of acceleration coefficient. These values are depicted in the second column of the table, The total overall performance is now improved up to 41.14739. From our analysis, it is found that c1 ranges from 1.9 to 3.5. With different changing ratio for the case of F1, it is interesting to note that the value of c1 increases in a similar pattern; the greater changing ratio favors greater value of c1. From this behavior, the feasibility of adaptive c1 is observed. This will be one of the promising directions for future work. The composition of Rastrigin problem set seems to be most challenging in this study as this is a multimodal problem with many local optima residing close to one and other. For such a case, long jump or step is favored by setting c1=3.5 and this helps in reducing chances of falling into a local optimum in the light of dynamic environment. Again, it is interesting to note that the performance of each problem set is slightly improved compared to the one in Table 4. Mean values and standard deviation (STD) are tabulated in Table 5.

Table 1. Performance of PSO on F1-F6, Overlapping ratio = 0.1, Diversity = 0.3 and Popsize = 40 / 10

Problem	Changing ratio	T1	T2	T3	T4	T5	T6
F1	0.3	0.987776	0.982685	0.993593	0.970057	0.993230	0.959536
	0.7	0.916581	0.957208	0.985420	0.975387	0.991905	0.976794
	1.0	0.949945	0.926093	0.953148	0.971681	0.994805	0.957577
F2	1.0	0.892211	0.878057	0.854241	0.872302	0.845164	0.866242
F3	1.0	0.757464	0.500827	0.556597	0.591180	0.629322	0.565224
F4	1.0	0.729952	0.698798	0.688129	0.693392	0.684277	0.688200
F5	1.0	0.840236	0.825325	0.826304	0.802516	0.831110	0.811447
F6	1.0	0.830845	0.755024	0.737995	0.747123	0.809007	0.733006
Total overall performance = 39.98494							

Table 2. Performance of WSA on F1-F6 (c_1=c_2 =1.7) Overlapping ratio = 0.1, Diversity = 0.3 and Popsize = 40 / 10

Problem	Changing ratio	T1	T2	T3	T4	T5	T6
F1	0.3	0.985672	0.976931	0.980326	0.922440	0.992259	0.912831
	0.7	0.899224	0.954392	0.980958	0.933082	0.992136	0.933062
	1.0	0.934711	0.904565	0.948396	0.946560	0.994972	0.941083
F2	1.0	0.877429	0.850753	0.831603	0.837051	0.843487	0.801454
F3	1.0	0.551327	0.480017	0.565144	0.666168	0.600981	0.568956
F4	1.0	0.591770	0.676461	0.631459	0.673681	0.658180	0.676794
F5	1.0	0.833339	0.782035	0.798453	0.767672	0.823851	0.738357
F6	1.0	0.814561	0.738476	0.719556	0.708658	0.814662	0.690929
The overall performance = 38.74686							

Table 3. Performance of WSA on F1-F6 (c_1=c_2 =1.7) Overlapping ratio = 0.9, Diversity = 0.6 and Popsize = 40 / 10

Problem	Changing ratio	T1	T2	T3	T4	T5	T6
F1	0.3	0.988957	0.983702	0.995235	0.963875	0.996286	0.959345
	0.7	0.928869	0.965581	0.987264	0.975389	0.994323	0.970561
	1.0	0.953625	0.936263	0.957113	0.962115	0.996393	0.959473
F2	1.0	0.886155	0.859200	0.838945	0.840452	0.831960	0.824130
F3	1.0	0.776315	0.551812	0.604169	0.617472	0.706699	0.592349
F4	1.0	0.730942	0.690995	0.718009	0.685624	0.708417	0.662063
F5	1.0	0.845622	0.807329	0.816988	0.784206	0.848746	0.774031
F6	1.0	0.820322	0.763701	0.751292	0.749416	0.822872	0.723090
The overall performance = 40.10769							

Table 4. Performance of WSA on F1-F6 (c_1 varies) Overlapping ratio = 0.9, Diversity = 0.6 and Popsize = 40 / 10

Problem	c_1	Changing ratio	T1	T2	T3	T4	T5	T6
F1	1.9	0.3	0.989761	0.983868	0.995160	0.978730	0.997635	0.967037
	2.2	0.7	0.925635	0.967005	0.988625	0.977918	0.996809	0.975176
	2.5	1.0	0.961981	0.943873	0.960278	0.974210	0.996547	0.961759
F2	2.5	1.0	0.925925	0.896458	0.882685	0.899841	0.867598	0.864477
F3	3.5	1.0	0.814633	0.541270	0.605347	0.638468	0.723133	0.604509
F4	2.5	1.0	0.748691	0.729934	0.721208	0.718273	0.718745	0.698000
F5	2.3	1.0	0.892377	0.862897	0.856936	0.843768	0.880029	0.826867
F6	2.5	1.0	0.838275	0.808686	0.780733	0.796042	0.852256	0.767322
The overall performance = 41.14739								

Table 5. Mean Values and STD for Results in Table 4

Problem	Changing ratio	T1	T2	T3	T4	T5	T6
F1	0.3	0.58±0.52	1.35±0.88	0.29±0.32	0.00±1.38	0.00±0.01	1.43±2.06
	0.7	4.85±5.27	2.74±2.02	0.36±0.37	1.12±1.40	0.00±0.02	0.88±1.39
	1.0	1.74±2.25	2.47±2.50	2.25±0.54	0.23±0.50	0.00±0.00	0.30±0.71
F2	1.0	0.83±0.92	1.25±1.81	2.22±2.97	0.67±1.49	1.20±0.81	1.44±2.34
F3	1.0	4.02±5.06	89.8±172	45.9±91.8	54.0±96.0	11.3±35.7	64.2±108
F4	1.0	6.40±11.2	6.59±9.84	8.14±8.54	5.34±8.81	14.0±21.3	7.40±12.4
F5	1.0	0.71±1.46	1.09±2.36	1.30±2.57	1.08±1.87	0.47±0.71	1.40±2.55
F6	1.0	3.50±4.11	4.26±4.04	6.89±7.45	2.58±3.68	1.42±1.22	3.71±4.17

5 Conclusions

In this work, it is proven that PSO works effectively even without the present of the prominent inertia weight on Dynamic Optimization Problems. Thus, the proposed algorithm is called Weightless Swarm Algorithm (WSA). The strategy can be incorporated into any existing PSO algorithm by discarding the inertia weight and changing the update strategy to swapping instead of replacement. From the series of results, it is shown that the nature of WSA is different from PSO and therefore parameter sensitivity analysis is done to obtain the optimal parameters (overlapping ratio and diversity). The performance of WSA is only slightly less compared to PSO without inertia weight. The simplicity of WSA allows the tuning of acceleration constant, c1 independently in order to obtain better results. Thus, it is evident that WSA has superior properties alongside extremely simple strategy and cheaper computational costs. Future work will be done to find the underlying properties yet to be discovered.

6 References

1. J. Kennedy and R. Eberhart, "Particle Swarm Optimization," Proceedings of 1995 *IEEE International Conference on Neural Networks*, vol. 4, pp. 1942-1948.
2. Robinson, J., & Rahmat-Samii, Y.: Particle Swarm Optimization in Electromagnetics. Antennas and Propagation, IEEE Transactions on, 52 (2004) 397-407
3. Deb, K., Pratap, A., Agarwal, S. et al.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. Evolutionary Computation, IEEE Transactions on, 6 (2002) 182-197
4. Xin Yao, Yong Liu, Guangming Lin: Evolutionary Programming made Faster. Evolutionary Computation, IEEE Transactions on, 3 (1999) 82-102
5. Francois, O.: An Evolutionary Strategy for Global Minimization and its Markov Chain Analysis. Evolutionary Computation, IEEE Transactions on, 2 (1998) 77-90
6. Brest, J., Greiner, S., Boskovic, B. et al.: Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. Evolutionary Computation, IEEE Transactions on, 10 (2006) 646-657
7. Gazi, V.: Asynchronous Particle Swarm Optimization. Signal Processing and Communications Applications, 2007. SIU 2007. IEEE 15th, (2007) 1-4
8. Rada-Vilela, J., Mengjie Zhang, Seah, W.: Random Asynchronous PSO. Automation, Robotics and Applications (ICARA), 2011 5th International Conference on, (2011) 220-225
9. Y. Shi and R. Eberhart, "A Modified Particle Swarm Optimizer," in Proceedings of 1998 *IEEE International Conference on Evolutionary Computation*, pp. 69-73.
10. M. Clerc and J. Kennedy, "The Particle Swarm - Explosion, Stability and Convergence in a Multidimensional Complex Space," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 58-73, 2002.
11. I. C. Trelea, "The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection," *Information Processing Letters*, vol. 85, pp. 317-325, 2003.
12. EAlib (open platform to test and compare the performances of EAs). Available at:
13. http://people.brunel.ac.uk/~csstssy/ECiDUE/ECDOP-Competition12-TestPlatform.tar.gz
14. Shengxiang Yang and Changhe Li, "A Clustering Particle Swarm Optimizer for Locating and Tracking Multiple Optima in Dynamic Environments," *IEEE Transactions on Evolutionary Computation*, vol. 14, pp. 959-974, 2010.
15. Changhe Li and Shengxiang Yang, "A Clustering Particle Swarm Optimizer for Dynamic Optimization," in *IEEE Congress on Evolutionary Computation*, 2009 (CEC '09). pp. 439-446.
16. Technical Benchmark Generator for the IEEE WCCI-2012 Competition on Evolutionary Computation for Dynamic Optimization Problems. Available at:
17. http://people.brunel.ac.uk/~csstssy/ECiDUE/TR-ECDOP-Competition12.pdf