A Modified Particle Swarm Optimizer

Yuhui Shi and Russell Eberhart
Department of Electrical Engineering
Indiana University Purdue University Indianapolis
Indianapolis, IN 46202-5160
Shi, eberhart@tech.iupui.edu

ABSTRACT

In this paper, we introduce a new parameter, called inertia weight, into the original particle swarm optimizer. Simulations have been done to illustrate the significant and effective impact of this new parameter on the particle swarm optimizer.

1. INTRODUCTION

Evolutionary computation techniques (evolutionary programming [4], genetic algorithms [5], evolutionary strategies [9] and genetic programming [8]) are motivated by the evolution of nature. A population of individuals, which encode the problem solutions, are manipulated according to the rule of survival of the fittest through "genetic" operations, such as mutation, crossover and reproduction. A best solution is evolved through the generations. These kinds of techniques have been successfully applied in many areas and a lot of new applications are expected to appear. In contrast to evolutionary computation techniques, Eberhart and Kennedy developed a different algorithm through simulating social behavior [2,3,6,7]. As in other algorithms, a population of individuals exists. This algorithm is called particle swarm optimization (PSO) since it resembles a school of flying birds. In a particle swarm optimizer, instead of using genetic operators, these individuals are "evolved" by cooperation and competition among the individuals themselves through generations. Each particle adjusts its flying according to its own flying experience and its companions' flying experience. Each individual is named as a "particle" which, in fact, represents a potential solution to a problem. Each particle is treated as a point in a D-dimensional space. The *i*th particle is represented as $X_i=(x_{i1},x_{i2}, \ldots , x_{iD})$. The best previous position (the position giving the best fitness value) of any particle is recorded and represented as $P_i=(p_{i1},p_{i2}, \ldots , p_{iD})$. The index of the best particle among all the particles in the population is represented by the symbol g. The rate of the position change (velocity) for particle i is represented as $V_i=(v_{i1},v_{i2}, \ldots , v_{iD})$. The particles are manipulated according to the following equation:

$$v_{id} = v_{id} + c_1 * \text{rand}( ) * (p_{id} - x_{id}) + c_2 * \text{Rand}( ) * (p_{gd} - x_{id}) \quad (1a)$$

$$x_{id} = x_{id} + v_{id} \quad (1b)$$

where c_1 and c_2 are two positive constants, rand() and Rand() are two random functions in the range [0,1]. The second part of the equation (1a) is the "cognition" part, which represents the private thinking of the particle itself. The third part is the "social" part, which represents the collaboration among the particles [7]. The equation (1a) is used to calculate the particle's new velocity according to its previous velocity and the distances of its current position from its own best experience (position) and the group's best experience. Then the particle flies toward a new position according to equation (1b). The performance of each particle is measured according to a pre-defined fitness function, which is related to the problem to be solved.

The particle swarm optimizer has been found to be robust and fast in solving nonlinear, non-differentiable, multi-modal problems, but it is still in its infancy. A lot of work and research are needed. In this paper, a new parameter is introduced into the equation, which can improve the performance of the particle swarm optimizer.

2. A MODIFIED PARTICLE SWARM OPTIMIZER

Refer to equation (1a), the right side of which consists of three parts: the first part is the previous velocity of the particle; the second and third parts are the ones

0-7803-4869-9/98 $10.00©1998 IEEE

contributing to the change of the velocity of a particle. Without these two parts, the particles will keep on "flying" at the current speed in the same direction until they hit the boundary. PSO will not find a acceptable solution unless there are acceptable solutions on their "flying" trajectories. But that is a rare case. On the other hand, refer to equation (1a) without the first part. Then the "flying" particles' velocities are only determined by their current positions and their best positions in history. The velocity itself is memoryless. Assume at the beginning, the particle *i* has the best global position, then the particle *i* will be "flying" at the velocity 0, that is, it will keep still until another particle takes over the global best position. At the same time, each other particle will be "flying" toward its weighted centroid of its own best position and the global best position of the population. As mentioned in [6], a recommended choice for constant c_1 and c_2 is integer 2 since it on average makes the weights for "social" and "cognition" parts to be 1. Under this condition, the particles statistically contract swarm to the current global best position until another particle takes over from which time all the particles statistically contract to the new global best position. Therefore, it can be imagined that the search process for PSO without the first part is a process where the search space statistically shrinks through the generations. It resembles a local search algorithm. This can be illuminated more clearly by displaying the "flying" process on a screen. From the screen, it can be easily seen that without the first part of equation (1a), all the particles will tend to move toward the same position, that is, the search area is contracting through the generations. Only when the global optimum is within the initial search space, then there is a chance for PSO to find the solution. The final solution is heavily dependent on the initial seeds (population). So it is more likely to exhibit a local search ability without the first part. On the other hand, by adding the first part, the particles have a tendency to expand the search space, that is, they have the ability to explore the new area. So they more likely have a global search ability by adding the first part. Both the local search and global search will benefit solving some kinds of problems. There is a tradeoff between the global and local search. For different problems, there should be different balances between the local search ability and global search ability. Considering of this, a inertia weight *w* is brought into the equation (1) as shown in equation (2). This *w* plays the role of balancing the global search and local search. It can be a positive constant or even a positive linear or nonlinear function of time.

$$v_{id} = w * v_{id} + c_1 * rand( ) * (p_{id} - x_{id}) + c_2 * Rand( ) * (p_{gd} - x_{id}) \quad (2a)$$

$$x_{id} = x_{id} + v_{id} \quad (2b)$$

3. EXPERIMENTS AND DISCUSSION

In order to see the influence which the inertia weight has on the PSO performance, the benchmark problem of Schaffer's f6 function [1] was adopted as a testing problem since it is a well known problem and its global optimum is known. The implementation of the PSO was written in C and compiled using the Borland C++ 4.5 compiler. For the purpose of comparison, all the simulations deploy the same parameter settings for the PSO except the inertia weight *w*. The population size (number of particles) is 20; the maximum velocity is set as 2; the dynamic range for all elements of a particle is defined as (-100, 100), that is, the particle cannot move out of this range in each dimension. For the Schaffer's f6 function , the dimension is 2. So we display particles' "flying" process on the computer screen to get a visual understanding of the PSO performance; the maximum number of iterations allowed is 4000. If PSO cannot find a acceptable solution within 4000 iterations, it is claimed that the PSO fails to find the global optimum in this run.

Different inertia weights *w* have been chosen for simulation. For each selected *w*, thirty runs are performed and the iterations required for finding the global optimum are recorded. The results are listed in Table 1. In Table 1, the empty cells indicate that the PSO failed to find the global optimum within 4000 iterations in that running. For each *w*, the average number of iterations required to find the global optimum is calculated and only the runs which find the global optimum are used for calculating the average. For example, when *w=0.95*, one run out of 30 failed to find the global optimum, so only 29 run results are used to calculate the average. The average number of iterations for each *w* is listed at the bottom of the table 1. From Table 1, it's easy to see that when *w* is small (< 0.8), if the PSO finds the global optimum, then it finds it fast. Also from the display on the screen, we can see that all the particles tend to move together quickly. This confirms our discussion in the previous section that when *w* is small, the PSO is more like a local search algorithm. If there is acceptable solution within the initial search space, then the PSO will find the global optimum quickly, otherwise it will not find the global optimum. When *w* is large (>1.2), the PSO is more like a global search method and even more it always tries to exploit the new areas. This is also illustrated by the "flying" display on the screen. It is natural that the PSO will take more iterations to find the global optimum and have more chances of failing to find the global optimum. When *w* is medium (0.8<w<1.2), the PSO will have the best chance to find the global optimum but also takes a moderate number of iterations. For clearness, the

number of failures to find the global optimum out of 30 runs is listed for each inertia weight w in Table 2 and shown in Figure 1. From Table 2 and Figure 1, it can be seen that the range [0.9, 1.2] is a good area to choose w from.. The PSO with w in this range will have less chance to fail to find the global optimum within a reasonable number of iterations. And when $w=1.05$, the PSO finds the global optimum in all 30 running. This may be due to the use of the maximum velocity. When $w=0.9$, the PSO takes the least average number of iterations to find the global optimum for w within the range [0.9,1.2].

From the display of the "flying" process on the screen and the simulation results in Table 1 and 2, we can see that the bigger the inertia weight w is, the less dependent on initial population the solution is, and the more capable to exploit new areas the PSO is.
For any optimization search algorithm, generally, it is a good idea for the algorithm to possess more exploitation ability at the beginning to find a good seed, then have the algorithm to have more exploration ability to fine search the local area around the seed. Accordingly, we defined the inertia weight w as a decreasing function of time instead of a fixed constant. It started with a large value 1.4 and linearly decreased to 0 when the iteration number reached 4000. Thirty experiments were performed and the results are listed in the furthest right column in Table 1. Compared with the previous results, we can see it has the best performance. All the 30 runs find the global optimum and the average number of iterations required is less than that when w is larger than 0.9.

4. CONCLUSION

In this paper, we have introduced a parameter *inertia weight* into the original particle swarm optimizer. Simulations have been performed to illustrate the impact of this parameter on the performance of PSO. It is concluded that the PSO with the inertia weight in the range [0.9,1.2] on average will have a better performance; that is, it has a bigger chance to find the global optimum within a reasonable number of iterations. Furthermore, a time decreasing inertia weight is introduced which brings in a significant improvement on the PSO performance. Simulations have been done to support it.

Even though good results have been obtained through introducing a time varying inertia weight, many more studies are still required to be done. Different time decreasing functions need be tested to seek a better function of time. For example, from Figure 1, it is easy to see that the inertia weight does not need to decrease from 1.4 to 0. A decrease from 1.4 to 0.5 will maybe work better. Nonlinear decreasing functions of time also need to be tested. In this paper, only a small benchmark problem has been tested; to fully claim the benefits of the inertia weight, more problems need to be tested. By doing all this testing, a better understanding of the inertia weight's impact on PSO performance can be expected. After gaining experience, a fuzzy system [10] could be built to tune the inertia weight on line. We are doing these things now.

References

1. Davis, L., Ed. (1991), Handbook of Genetic Algorithms, New York, NY: Van Nostrand Reinhold.
2. Eberhart, R. C., Dobbins, R. C., and Simpson, P. (1996), Computational Intelligence PC Tools, Boston: Academic Press.
3. Eberhart, R. C., and Kennedy, J. (1995). A New Optimizer Using Particles Swarm Theory, Proc. Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan), IEEE Service Center, Piscataway, NJ, 39-43.
4. Fogel, L. J. (1994), Evolutionary Programming in Perspective: the Top-down View, In Computational Intelligence: Imitating Life, J.M. Zurada, R. J. Marks II, and C. J. Robinson, Eds., IEEE Press, Piscataway, NJ.
5. Goldberg, D. E. (1989), Genetic Algorithms in Search, Optimization, and Machine Learning, Reading MA: Addison-Welsey.
6. Kennedy, J., and Eberhart, R. C. (1995). Particle Swarm Optimization, Proc. IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV: 1942-1948.
7. Kennedy, J. (1997), The Particle Swarm: Social Adaptation of Knowledge, Proc. IEEE International Conference on Evolutionary Computation (Indianapolis, Indiana), IEEE Service Center, Piscataway, NJ, 303-308.
8. Koza, J. R. (1992), Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA.
9. Rechenberg, I. (1994), Evolution Strategy, In Computational Intelligence: Imitating Life, J. M. Zurada, R. J. Marks II, and C. Robinson, Eds., IEEE Press, Piscataway, NJ.
10. Shi, Y. H., Eberhart, R. C., and Chen, Y. B. (1997), Design of Evolutionary Fuzzy Expert system, Proc. of 1997 Artificial Neural Networks in Engineering Conference, St. Louis, November, 1997.

Table 1. Numbers of iterations for finding the global optimum using different inertia weights. The blank cells mean these runs didn't find the global optimum within the maximum number of iterations

No.	Weights										Time
	1.4	1.2	1.1	1.05	1	0.95	0.9	0.85	0.8	0	varying
1	684	819	989	994	864	680	633	1639	138	115	433
2		2521	2571	1117	885	2278	985	281			1382
3	1534	1632	2645	1636	2086	1264	3131	2480	158		982
4		480		2415	2250	908	252	1131	122		1482
5	577	3222	174	1564	1638	540	342	1562	120		898
6	854	984	1808	2040	984	799	1253	414			1384
7	2733	3371	1578	1982	1143	659	1076	116	262	96	1354
8	1498	3886	1351	826	796		878	353			1558
9	965	2085	272	1843	654	1158	1806	260			1268
10	3423	847	149	2097	748	2079	344	1587	147		1341
11		1660	2024	3153	341	1597	480	2416	147	158	1435
12		1361	960	481		1056	1298	206	174		534
13		610	1565	3724	1678	3283	1122	1733	163		621
14	3599	372	1260	833	1168	917	756	2644			524
15			1158	2767	1449	423	777	279	251		1462
16	3221	813	587	2954	1002	1746	474	1474		84	1508
17		1740	1477	2323	3789	651	1262	44		402	1362
18	718		1544	3065	2278	1458	1615	162		151	1510
19	1174	735	377	2179	342	712	933	337	3274		956
20		925	1744	673	1234	3458		700	1965	95	1571
21	2766	319	463	1672	274	2425	787	203			680
22		621	2071	2934	2153	2165	3383	555	192	283	1431
23		1071	1074	2243	316	3639	948	1030		150	1361
24	3313	2038	1535	1994	1264	619	1660		319	102	1125
25	266	2137	1437	2336	2055	468	3159		167		1516
26	837	952	563	1455	628	2038	431	312		139	1450
27	745	962	1596	262	1878	508		697			834
28		3245	657	786	1159	919	1969				479
29	3354	2702	1673	3800	2772	1955	470	640			1567
30	1753	1476	3768	1206	691	328	2681		170	340	1591
Avg.	1790.21	1556.64	1347.24	1911.8	1328.24	1404.48	1246.60	894.423	485.56	176.25	1186.6

Table 2. The number of runs which failed to find global optimum using different inertia weights

Weight	0.0	0.8	0.85	0.9	0.95	1.0	1.05	1.1	1.2	1.4
No.	18	14	4	2	1	1	0	1	2	11

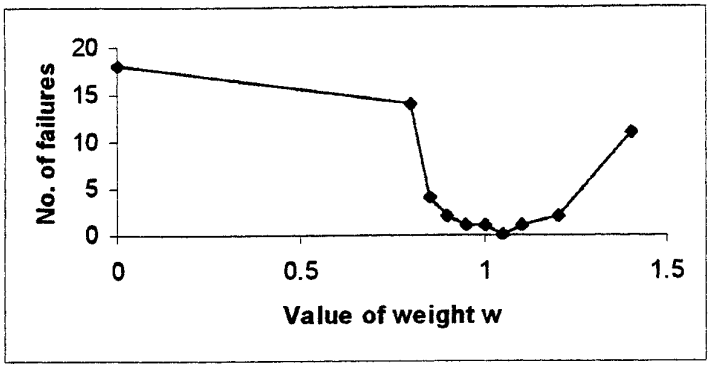

Figure 1. Number of failures for different weights