An Overview on use of Particle Swarms for Tracking and Optimizing Dynamic Systems

K.A.Joshi
Electrical Engineering Department
Prof Ram Meghe College of Engineering & Management
Amravati-India
M.M.Janolkar
First Year Engineering Department
Prof Ram Meghe College of Engineering & Management
Amravati-India
K.L.Bondar
P.G. Department of Mathematics
N.E.S. Science College
Nanded-India

***Abstract*— Using particle swarms to track and optimize dynamic systems is described. Issues related to tracking and optimizing dynamic systems are briefly reviewed. Three kinds of dynamic systems are defined for the purposes of this paper. One of them is chosen for preliminary analysis using the particle swarm on the parabolic benchmark function. Successful tracking of a IO-dimensional parabolic function with a severity of up to 1.0 is demonstrated. A number of issues related to tracking and optimizing dynamic systems with particle swarms are identified. Directions for future research and applications are suggested.**

Keywords—Particle Swarm, Dynamic Systems, Tracking, algorithms

I. INTRODUCTION

Particle swarm optimization (PSO) is an evolutionary computation technique developed by Kennedy and Eberhart [4,6]. PSO is similar to a genetic algorithm (GA) in that the system is initialized with a population of random solutions. It is unlike a GA, however, in that each potential solution is also assigned a randomized velocity, and the potential solutions, called *particles*, are then "flown" through the problem space.

Equations (1) and (2) describe the velocity and position update equations with an inertia weight included. Equation (1) calculates a new velocity for each particle (potential solution) based on its previous velocity, the particle's location at which the best fitness so far has been achieved, and the population global (or local neighborhood, in the neighborhood version of the algorithm) location at which the best fitness so far has been achieved. Equation (2) updates each particle's position in solution hyperspace. The two random numbers are independently generated. The use of the inertia weight *w* has provided improved performance in a number of applications. As originally developed, *w* often is decreased linearly from about 0.9 to 0.4 during a run. A different form of *w*, explained later, is currently being used by one of the authors **(**RE**),** and was used in this paper.

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

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

There are several ways in which systems can change over time. First, the location in problem space of the optimum value can change. Second, the location can remain constant, but the optimum value can vary. Third, both the location and value of the optimum can vary. Fourth, in a multidimensional system, these variations can occur on one or more dimensions, either independently or simultaneously.

Thus, there are many ways of looking at dynamic systems. In this paper, we chose the relatively simple approach of varying the location in problem space where the optimum value occurs. Further, we vary it simultaneously and equally on each dimension. This choice was made primarily to facilitate comparison with prior work.

II. LITERATURE REVIEW

Most papers reporting applications of evolutionary algorithms discuss only the solution of static problems. Examples are comparisons of various approaches to finding the optimum solutions for benchmark problems. Since its inception in 1995, particle swarm Optimization has been proven to be very effective for this type of application. Many real-world systems, however, change state frequently; some change almost continuously. Most of the computational time in scheduling systems is spent in *rescheduling*, caused by changes in customer priorities, unexpected equipment maintenance, etc.

In real-world applications, these system state changes result in a requirement for frequent re-optimization. When a ship-to-shore crane in an automated container terminal malfunctions, a new equipment schedule for the entire port facility must be available within five to ten minutes. When a helicopter ferrying supplies ashore to a Marine combat unit is lost, a new optimized schedule for the remaining helicopters and boats is needed within a minute or two. When medical supplies already on their way to a location are suddenly needed more urgently, the change in priority must be reflected within minutes in the revised delivery schedule. These types of events can be characterized as changes in objectives, priorities. and/or resources.

Note that the situations described above often involve large machines or systems operated by humans. The inertia in the systems (including the human operators) often makes changes more frequent than once per minute or so inefficient, perhaps even

meaningless. As a first guess, then, we speculate that computational optimization processes can often be allowed something on the order of 10-100 seconds to run between outputs for these human-machine systems. Keep in mind the law of sufficiency: if the solution is good enough, fast enough, and cheap enough, then it is sufficient. By good enough, we mean that it meets requirements reasonably well. By fast enough, we mean that it finishes within an acceptable time. By cheap enough, we mean that it uses available resources in an acceptable way. We may sometimes need to evolve a sufficient solution in 10 seconds, but if we are given more time we can probably do better.

Given the speed of the particle swarm optimization code, we can often achieve on the order of 100-300 iterations (generations) per second for a reasonably complex problem being optimized on a personal computer. Many of the benchmark functions about which we have published results can be optimized in about 1,000-2,000 iterations or less [5]. So we can see that for at least some problems we can already achieve optimization well within in the 10-100 second window described above for human-machine systems. What, then, do we mean by "tracking and optimizing dynamic systems?" If we can already achieve "optimization" within a 10-second window, aren't we back to a static optimization situation? We assert that the answer is "no" for the following reasons.

First, the "optimal" values achieved for benchmark functions are often values generally agreed upon in the literature. For example, if we want to achieve an error of **.**000001 for Schaffer's f6 function instead of the value of **.**00001 often quoted in the literature, significantly more iterations may be required. Second, we need to investigate the performance of particle swarms in dynamic environments from the perspective of the swarms' dynamics. Whenever the system changes such that we ask the swarm to look for a new optimum location and/or value, each particle in the swarm has a position and a velocity. The positions and velocities are presumably converging on the old optimum. So it would seem to make sense to continue on from where we left off, i.e., the last swarm configuration we had while looking for the old optimum becomes the initial configuration as we look for the new optimum. Another approach, however, is to completely re-initialize (randomize) the swarm.

Which approach is better depends on the situation. If the system change is relatively small, using the old swarm may be the better approach. A large system perturbation, however, may make starting over with a randomized swarm a more efficient approach. Another approach, of course, is a combination of the two: initialize the new population with part (say one-half) of its members from the old population, and the other part randomized within the problem space. Perhaps only GBEST (or the LBESTs) should be retained. Other issues also deserve attention. For example, should maximum velocities and/or constriction factors [3] be modified. What about the momentum of the particles? Early versions of the swarm included a USEBETTER parameter that kept a particle going in the direction it was going if its fitness improved during the current iteration. This parameter wasn't found to be particularly useful for static problems, but its usefulness needs to be re-examined for dynamic systems. Previous work has been done in this area for evolutionary programming [1] and evolution strategies [2]. Both researchers used as their dynamic base function the parabolic function in three dimensions: $f(x, y, z) = x^2 + y^2 + z^2$ This is functionally the same as what is often referred to as the "spherical function" in three dimensions ***(n = 3*** in equation ***3):***

$$f_0(x) = \sum_{i=1}^{n} x_i^2 \qquad (3)$$

Authors in [1] compare the ability of an evolutionary program that does not evolve mutation variance parameters to one that does over several "simple but non-trivial dynamic environments." His results indicate that for some dynamic functions, self-adaptation is effective while for others it is detrimental. He uses a dynamic range of -50 to *50* on each of the three axes, and a population size of *200.* Each run is for 500 generations. He uses three types of dynamics: linear, circular, and random. The linear dynamics version changes the location of the optimum by an amount he calls the severity factor s. The severity factor can take on values of 0.01, 0.1, and *0.5.* A complete set of experiments is run for each value. In addition, the interval between movements of the optimum can be 1, 5, or 10 generations. In his circular dynamics version, the severity factor controls the radius of a circular dynamic in three dimensions. The period is always *25* intervals for a complete circle. The random dynamics version varies the optimum location randomly, with increments of sN(0,1), or the severity factor times a Gaussian mutation with a mean of zero and a variance of one. In this paper, we focus on linear dynamics; the circular and random versions are not considered further. Angeline ran fifty experiments for each value of **s,** at each interval. He recorded the mean of the best of each generation for each of the *50* experiments. So for the linear dynamic, which we now focus on, when updating was done each 10 generations and the severity factor was 0.01, the optimum was moved 50 times, and ended up at (0.5,0.5,0.5). When updating was done each generation with s = 0.5, the optimum was moved 500 times, and ended up at (250,250,250). Note that this is well outside of the dynamic range (-50 to 50) on each axis that was defined earlier.

To summarize, authors in [1] found that some combinations of evolutionary programming parameters were helpful when tracking the dynamic function, others were detrimental. Authors in [2] found that one particular combination of evolution strategies parameters (discrete recombination for object variables, etc.) resulted in an effective tracking strategy for a (μ, λ)-evolution strategy. Other combinations of parameters and other strategies were not reported.

III. Experimental Design

One of the reasons that particle swarm optimization is attractive is that there are very few parameters to adjust. One version, with very slight variations (or none at all) works well in a wide variety of applications. For the comparisons in this paper, we used the version of the particle swarm we are currently using for our other applications. It utilizes an inertia weight factor, *w* in equation (1), that is [0.5 + (Rnd/2.0)]. This produces a number randomly varying between 0.5 and 1.0, with a mean value of 0.75. This was selected in the spirit of Clerc's constriction factor [3], which sets *w* to a value of 0.729. Constants c_1 and c_2 in equation (1) were set to 1.494, also in accordance with Clerc's constriction factor.

Previously, we have used a linearly decreasing inertia weight, often decreasing from 0.9 to 0.4 during a run. When tracking a nonlinear dynamic system, however, it cannot be predicted whether exploration (a larger inertia weight value) or exploitation (a smaller inertia weight) will be better at any given time. An inertia weight that randomly varies roughly within our previous range addresses this. The dynamic range was set to [-50, 50] on each dimension. A smaller range of [-10, 10] is often used for the

parabolic (or spherical) function; see [5,7]. In accordance with the way we typically implement particle swarm optimization, we set the maximum velocity *Vmax* equal to the dynamic range on each dimension. We made 20 runs for each combination of parameters.

Angeline used a population size of 200; back replicated his conditions. It is usual, however, to set the population size of a particle swarm to smaller values, often between 10 and 40. We chose a population of 20 particles for this paper, and in order to keep the number of evaluations per dynamic change identical to Angeline, we chose a dynamic update interval of 100 generations as analogous to his interval of 10 generations. We did not investigate more frequent updates (analogous to Angeline and Back's intervals of one and five generations), because 100 generations (2,000 evaluations) is already at the lower end of the number we believe we will have available when tracking most dynamic systems. As stated previously, it is likely that we will have at least 10 seconds between updates, and at 100 generations per second (a conservative number with most particle swarm systems on most computers) we will have at least 1,000 generations between dynamic updates. Whenever a dynamic update occurs, we retained the position of each particle, and reset the value of each pbest to the error value with respect to the dynamically updated optimum location.

We thus chose to replicate Angeline's and Back's results for 2,000 evaluations between dynamic updates, and we used severity factors of 0.1 and 0.5, the two highest values used by each researcher. For these factors, results are in Figure 2 (e) and (f) for adaptive and self-adaptive, respectively, of [1], and on the right side of Figure 1 in [2].

IV. Experimental Results

Angeline's adaptive version settled out to oscillations of error values of approximately 1.0 ± 0.5 for severity factor 0.5, and of approximately 0.1 ± 0.0 1 for severity factor 0.1. For severity factor of 0.5, his self-adaptive version oscillated between approximately 0.01 and 0.1 early in the run, with the oscillations damping out and the error increasing to about ten at the end of the run. At 100 iterations (analogous to our system at 1,000 iterations), the error is oscillating between about 0.1 and 0.3. For severity factor 0.1, the oscillations are around the value 0.001, ranging approximately from 0.0005 to 0.007. Back's results for severity of 0.1 oscillate between approximately 10^{-2} and $2*10^{-5}$. For severity of 0.5, the values oscillate between approximately $7*10^{-1}$ and 10^{-3}.

Summarizing, for a severity of 0.1, the best results (lowest errors) obtained by Angeline were approximately $5*10^{-4}$ to 10^{-3} using the self-adaptive version. For severity of 0.5, his best error results were approximately .01 to 0.1 using the adaptive version. Back obtained minimum error values of approximately $2*10^{-5}$ for severity of 0.1, and approximately 10^{-3} for severity of 0.5.

Our results are shown in figures 1 and 2. For severity of 0.5, figure 1 shows that the error value oscillates between approximately 1 and 10^{-8}, with minimum error values being generally about 10^{-8} to 10^{-9} For severity of 0.1, figure 2 shows the error oscillating between approximately 0.05 and 10^{-9}, with minimum error values ranging approximately from to 10^{-9} to 10^{-10}. These errors are several orders of magnitude less that obtained by either Angeline or Back. Also, there are somewhat smaller changes in minimum error values with changes in severity.

Note that all of these results are for the parabolic function in three dimensions. Few systems in the real world are this simple. Furthermore, we believe that severity factors greater than 0.5 (which represents a change of one percent of the dynamic range of *50)* are encountered routinely.

We therefore investigated the behavior of the particle swarm on the parabolic function in IO dimensions. Further, we used severity factors of 0.1, 0.5, and 1.0. Results appear in figures 3, 4, and 5 for severities of 0.1, 0.5, and 1.0, respectively. The swarm seems to track well in all cases, achieving minimum errors of approximately 10^{-4} to 10^{-5} for severity of 0.1, 10^{-3} to 10^{-4} for severity of 0.5, and 10^{-2} to 10^{-3} for severity of 1.0.

V. Concusion

Particle swarm optimization is able to track dynamically varying parabolic functions. For the limited testing done here, the performance of particle swarm optimization compares favorably with other evolutionary algorithms at all three severities tested. The ability to track a 10-dimensional function was demonstrated, including tracking at severities of 0.1, 0.5, and 1.0. This paper represents only the first step in the investigation of tracking dynamic systems using particle swarm optimization. To be useful in practical applications, the ability of particle swarms to track and optimize highly nonlinear systems with multimodal error surfaces will need to be proven. These systems often change states chaotically, rather than linearly or randomly. Furthermore, the magnitudes of these system state changes can be significant with respect to the dynamic range of the variables, not limited to only one or two percent.

Included in the tracking of nonlinear systems, then, must be a strategy to effectively respond to a wide variety of changes. In the case of particle swarm, we are investigating the re-randomization of a portion of the population when a change is detected. For tracking relatively small changes that occur frequently, the USEBETTER parameter that is essentially a momentum factor may be useful. It is not clear whether including this factor would be generally useful, however.

References

[1] Angeline, P. J. (1997). Tracking extrema in dynamic environments. Proc. Evolutionaty Progrumming VI, Indianapolis, IN. Berlin: Springer-Verlag, pp. 335-345.

[2] Back, T. (1998). On the behavior of evolutionary algorithms in dynamic environments. Proc. Int. Conf on Evolutionary Computation, Anchorage, AK. Piscataway, NJ: IEEE Press, pp. 446-45 1 .

[3] Clerc, M. (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. Proc. CEC 1999, Washington, DC, pp 195 1 - I 9.57.

[4] Eberhart, R., Simpson, P., and Dobbins, R. (1996). Computational Intelligence PC Tools. Boston: Academic Press Professional.

[5] Eberhart, R., and Shi, Y. (2000). Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization. Proc. CEC 2000, San Diego, CA, pp 84-88.

[6] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. Proc. 1995 ICEC, Perth, Australia.

[7] Kennedy, J., Eberhart, R., and Shi, Y. (2001). Svarm Intelligence, San Francisco: Morgan Kaufmann Pub1ishers.

[8] Kuhfeld, W.F., Tobias, R.D., Garrat, M., "Efficient Experimental Design with Marketing Research Applications", Journal of Marketing Research, 32 (1994) 545-557.

Shi, Y. and Eberhart, R. (1998) Parameter selection in particle swarm optimization. In Evolutionary Programming VII: Proc. EP98, New York: Springer-Verlag, pp. 591 -600.