

Tracking Changing Extrema with Particle Swarm Optimizer

Anthony Carlisle

Department of Mathematical and Computer Sciences,
Huntingdon College
antho@huntingdon.edu

Gerry Dozier

Department of Computer Science and Software
Engineering, Auburn University
gvdozier@eng.auburn.edu

Abstract

The modification of the Particle Swarm Optimizer has been shown to be effective in locating a changing extrema. In this paper we investigate the effectiveness of the modified PSO in tracking changing extrema over time. We demonstrate that a modified PSO is reliable and accurate in tracking a continuously changing solution.

1. Introduction

The Particles Swarm Optimizer was introduced by Kennedy and Eberhart [4,9] in 1995 and has been compared favorably to genetic algorithms [5,10]. The basic algorithm involved casting a population of particles over the search space, each with an individual, initially random, location and velocity vector. The particles *fly* over the solution space, remembering the best (most fit) solution encountered. At each iteration, each particle adjusts its velocity vector, based on its momentum and the influence of its best solution and the best solution of its neighbors, then computes a new point to examine. The momentum of each particle tends to keep it from being trapped by a local, non-optimal, extrema, yet by each particle considering both its own memory and that of its neighbors, the entire swarm tends to converge on the global extrema.

The original PSO formulae, as described in [7], are:

$$v_{id} = v_{id} + \eta_1 * \text{rand}() * (p_{id} - x_{id}) + \eta_2 * \text{rand}() * (p_{gd} - x_{id}) \quad (1a)$$

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

where d is the number of channels (variables), i is an individual particle in the population, g is the particle in the neighborhood with the best fitness, v is the velocity vector, x is the location vector, and p is the location vector for the individual particle's best fitness yet encountered.

Parameters η_1 and η_2 are the *cognitive* and *social learning rates*, respectively [1]. These two rates control the relative influence of the memory of the neighborhood to the memory of the individual.

In addition to the η_1 and η_2 parameters, implementation of the original algorithm also requires placing limits on the search area (X_{max} and X_{min}) and limiting the velocities (V_{max}).

Kennedy described four models of the PSO in [7]. The first three implemented the extreme values for η_1 and η_2 : The *full model* gave equal weight to the two influences; the *social-only model* set the cognitive learning rate to zero; the *cognition-only model* set the social learning rate to zero; and the *selfless model* is the social-only model, but with the additional constraint that a particle's neighborhood best does not include consideration of itself.

Shi and Eberhart devised an inertia weight, ω , to improve the accuracy of PSO by damping the velocities over time, allowing the swarm to converge with greater precision [11,12]. Integrating ω into the algorithm, the formula for computing the new velocities is:

$$v_{id} = \omega * v_{id} + \eta_1 * \text{rand}() * (p_{id} - x_{id}) + \eta_2 * \text{rand}() * (p_{gd} - x_{id}) \quad (2)$$

PSO is effective in determining optimal solutions in static environments, but in [2] we demonstrated that the original algorithm was poor in locating a changing extrema. We determined that if the extrema was changing rapidly, the influence of the particle and neighborhood memory favored the original solution over new, revised fitness values. This caused the particles to circle previous solution points, rather than chase down the relocated extrema.

We addressed this problem by constructing a method to recognize when the environment has changed, and to reset the swarm memories by

replacing all the p vectors with their respective x vectors. In this way, stale memories did not influence finding the moving solution.

In our specific implementation of PSO, we made use of Shi and Eberhart's ω [11,12] to avoid having to set a specific V_{max} . A V_{max} value would guarantee that the swarm would be left behind by any solution that was changing faster than V_{max} . Maurice Clerc has also derived a *constraint coefficient*, a modification of the PSO that also operates without V_{max} [3]. James Kennedy provided the formulae for this modification in [8], and Eberhart and Shi compared this "constriction factor", K , to their inertia weight in [12]. The constriction factor is computed as:

$$K = \frac{1}{\left| 1 - \frac{\phi}{2} - \sqrt{\frac{\phi^2 - 4\phi}{2}} \right|} \quad (3)$$

where ψ is $\eta_1 + \eta_2$. Since this value is based on η_1 and η_2 , using the constriction factor does not require any additional parameters.

With the constriction factor, the PSO formula for computing the new velocity is:

$$v_{id} = K * [v_{id} + \eta_1 * \text{rand}() * (p_{id} - x_{id}) + \eta_2 * \text{rand}() * (p_{gd} - x_{id})] \quad (3)$$

Note that K is applied to the entire expression, whereas the inertia weight ω was applied only to the cognition portion of the expression.

We found our modified PSO was very successful in finding changing optimal solutions, even at significant rate of change.

In this paper, we look back at our previous algorithm with a new objective: instead of simply locating the solution, we wish to *track* the solution over time. That is, we want to be able to query the swarm at any time for the current optimal solution in the search space.

2. Experiment Design

We wish to test if our previous modification to PSO is effective in tracking nonstationary solutions.

We also wish to test a possible improvement: instead of replacing every p vector with the corresponding x vector when a change in the environment is detected, we simply recompute the fitness of each p vector. The rationale for this approach is that the current p location may, in fact, be closer to the new goal than the current location of the particle (x vector), and simply recomputing the fitness of the p vector allows us to preserve that location information if it is better, or replace it with the current location if it is not.

Finally, we wish to test if Clerc's constraint coefficient is an improvement over Shi and Eberhart's inertia weight in tracking changing solutions.

As in our original investigation [2], we set up a scenario where a beacon (the goal) moves linearly across the search space. Particles attempt to locate and track the moving goal on the basis of signal strength (distance). Thus, the fitness is the distance from the goal, computed as:

$$f = \sqrt{\sum_{i=1}^D (g_i - x_i)^2}$$

where g_i is the position of the goal and x_i is the position of the particle on each i of the D dimensions. The velocity of the goal is constant for any given run of the experiment. We have the swarm report the its best solution at each iteration (time),

3. Experiment Implementation

We conducted three experiments, each on a different variation of the PSO algorithm. In each experiment, the swarm was consisted of twenty particles, randomly distributed within the range $[-50,50]$ on each of the two channels with random velocities on $[-50,50]$. Both η_1 and η_2 were set to 2.05, and no V_{max} or X_{max} is used. Each swarm is run for 20 passes on each of the four models, against a goal moving at each of ten velocities (including zero, as a baseline), and each run was

for 2,500 iterations.

For this experiment, a new *sentry* particle is chosen at random each iteration. A sentry serves a one-iteration tour, and its task is simply to detect any change in the environment in its local search area. The process is simple: the chosen sentry particle stores a copy of its most recent fitness value, then, at the start of the next iteration, it reevaluates its fitness and compares it to the previously stored value. If the two differ, then the environment, at least in the immediate area of the sentry particle, has changed since the last update of the sentry particle. Activation of a sentry adds a single fitness evaluation as overhead.

Note that the choice of a single sentry for each iteration in this experiment is arbitrary. For a population of N particles, this means we can generally expect to identify any changes in the swarm's search space within N iterations. (Changes occurring in the environment, but outside of the search area of the swarm, go undetected using the sentry method.) If it is necessary that we identify changes more quickly, we can activate more than one sentry in each iteration, with, of course, the corresponding additional overhead. For maximum resolution, we can make all N particles sentries, guaranteeing any changes in the search region will be detected.

Conversely, for problems where there is no need to identify changes rapidly, such as interfacing the swarm to real-world, mechanical systems, we could choose to activate a sentry every fifth, or tenth, or hundredth iteration. The sentry system provides flexibility to control the finesse, as well as the overhead costs, of recognizing changes in the search space.

If a sentry detects a change that requires the swarm seek a new goal, then it must sound an alarm. Detecting a change, in itself, is not sufficient grounds for adjusting the swarm, as a small change, within the error radius, will not cause the swarm to lose the goal. However, if the change is significant, the sentry particle sets a flag indicating that each particle should examine its p (best) value.

We compared three variations of the modified swarm. The first swarm variation was based on our original PSO modification, i.e., resetting each

particle's p vector to its x vector whenever a change in the environment is detected, and using the inertia weight ω . Resetting p to x allowed us to preserve some of the gains made by the particle in locating the moving goal (the current position), without retaining the influence of the now obsolete previous goal position. The inertia weight was computed as described in [13], as a linearly decreasing function, initially set to 0.65, decreasing to 0.15 over the first 1000 iterations, and remaining at 0.15 for the remainder of the run.

The second swarm changed the reset operation to only recompute the best fitness of each particle, based on the current p vector. We then set p to x only if p is inferior to x . The same inertia weight computation was used.

In the third swarm we replaced the inertia weight of the second swarm with Clerc's constriction factor.

All experiments reported the average of the best solution for each of the 20 passes at each iteration.

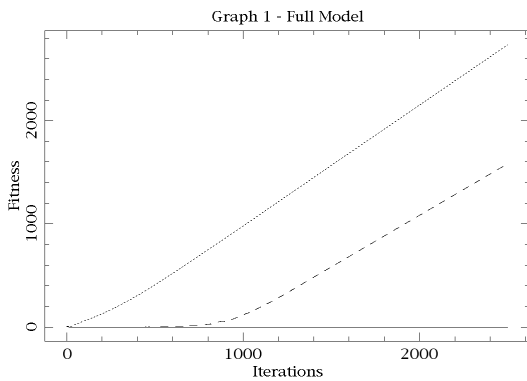
4. Results

We found the change in the method of resetting the swarm's p vectors resulted in a small improvement. The revised method retains the previous best values for a particle if those values reflect a solution that is superior to the particle's current position, or replace p with x in the normal course of events otherwise. We also found that the change to the constriction factor from the inertia weight resulted in a significant improvement, due to the change in objectives for this set of experiments. The inertia weight tends to accelerate the convergence of the swarm, which was desirable when attempting to simply *locate* an moving extrema. However, when attempting to *track* such a solution, we need to avoid limiting the velocities, as that reduces our ability to follow the moving goal.

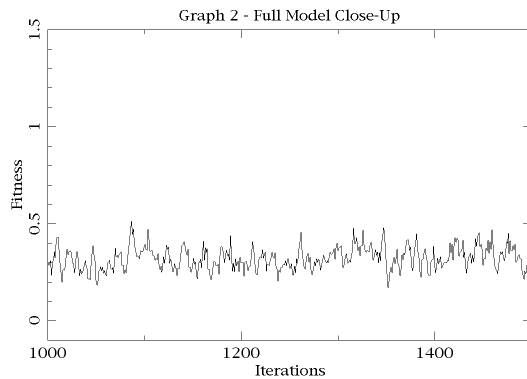
The following graphs compare the performance of the various swarm in tracking a fast moving goal. In each graph, the dotted line represents the first swarm, the dashed line represents the second swarm (improved memory reset), and the solid line represents the third swarm (constriction

factor). Fitness for these experiments is measured in term of closeness to the goal, i.e., smaller values are better.

Graph 1 compares each swarm’s implementation of the full model. It is obvious that the first swarm quickly loses goal. The second swarm manages to track the goal longer (about 600 iterations) before falling behind. However, the third swarm tracks the goal for the full 2,500 iterations. At lower velocities of the goal, the lines for swarms one and two shift to the right, as each manages to track the goal longer before being left behind.

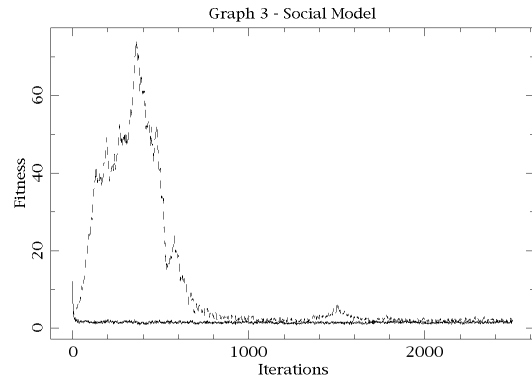


Graph 2 is a magnification a section of *Graph 1*, showing how closely the third swarm tracks the goal.

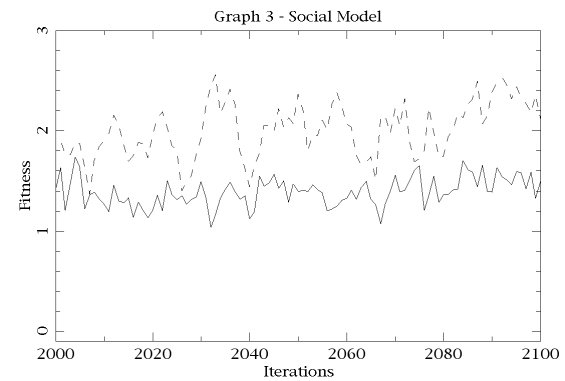


Graph 3 shows swarms two and three implementing the social-only model. Swarm two, using the inertia weight, has difficulty tracking the goal while the weight is stabilizing, but performs reasonably well after that 1,000 iterations. However, swarm three performs consistently well.

Swarm one is not shown, as its performance was very poor.



Graph 4 zooms in on a section of *Graph 3*, showing the difference in the stabilized performance of swarms two and three. Once past the initial adjustments, swarm two perform nearly as well as swarm three. However, three still performs consistently better.



All three swarms performed similarly, although slightly poorer, when implementing the selfless model, and the congition-only model performed badly in all cases.

5. Discussion

There are additional costs when using the tracking algorithm. If the algorithm is applied to a stationary environment, that cost is one additional evaluation per sentry activation, when the reset trigger is tested. In an environment under constant change, where a recalculation of the population’s best values is performed every iteration, the cost can be as high as N additional

evaluations per iteration. (N is the population size.) If the environment is changing, the costs can be much higher. One method of controlling costs in continuously changing environments is to place a threshold on the trigger, that is, to only trigger a population-wide recomputation if the change exceeds a certain magnitude. The threshold value then controls the accuracy of the solution returned by the swarm.

6. Conclusions

The two improvements to our original PSO modification have resulted in an algorithm that can detect when the environment changes, and react quickly to those changes. This algorithm has fewer parameters to tune, thus is a more universal tool, and can be used to locate or track stationary as well as nonstationary extrema.

7. Future Work

The scenario implemented in this set of experiments is simplistic - a single minima, moving linearly at a constant rate. We intend to first extend the scenario to include circular and random motion, then to encompass multiple extrema with time-dependent magnitudes.

References

- [1] Angeline, P. (1998). *Evolutionary Optimization versus Particle Swarm Optimization: Philosophy and Performance Difference*, The 7th Annual Conference on Evolutionary Programming, San Diego, USA.
- [2] Carlisle, A. and Dozier, G. (2000), *Adapting Particle Swarm Optimization to Dynamic Environments*, Proceedings, 2000 ICAI, Las Vegas, NV, Vol. I, pp 429-434.
- [3] Clerc, M. (1999) *The swarm and the queen: towards a deterministic and adaptive particle swarm optimization*. Proceedings, 1999 ICEC, Washinton, DC, pp 1951-1957.
- [4] Eberhart, R. 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.
- [5] Eberhart, R. and Shi, Y. (1998). *Comparison between Genetic Algorithms and Particle Swarm Optimization*, The 7th Annual Conference on Evolutionary Programming, San Diego, USA.
- [6] Eberhart, R.C., and Shi, Y. (2000), *Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization*, 2000 Congress on Evolutionary Computation
- [7] Kennedy, J. (1997), *The Particle Swarm: Social Adaptation of Knowledge*, IEEE International Conference on Evolutionary Computation (Indianapolis, Indiana), IEEE Service Center, Piscataway, NJ, 303-308.
- [8] Kennedy, J. (1999), *Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance*. 1999 Congress on Evolutionary Computation, Vol. III, pp 1931-1938.
- [9] Kennedy, J. and Eberhart, R. (1995). *Particle Swarm Optimization*, IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV: 1942-1948.
- [10] Kennedy, J. and Spears, W. (1998). *Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and some Genetic Algorithms on the Multimodal Problem Generator*, IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, USA.
- [11] Shi, Y. H., Eberhart, R. C., (1998), *A Modified Particle Swarm Optimizer*, IEEE International Conference on Evolutionary Computation, Anchorage, Alaska.
- [12] Shi, Y. H., Eberhart, R. C., (1998). *Parameter Selection in Particle Swarm Optimization*, The 7th Annual Conference on Evolutionary Programming, San Diego, USA.