

Controlling Particle Swarm Optimization with Learned Parameters

Kevin Winner, Don Miner and Marie desJardins
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, Maryland, U.S.A.
winnerk1@umbc.edu, don1@umbc.edu, mariedj@cs.umbc.edu

Abstract—Controlling particle swarm optimization is typically an unintuitive task, involving a process of adjusting low-level parameters of the system that often do not have obvious correlations with the emergent properties of the optimization process. We propose a method for controlling particle swarm optimization with *non-explicit control parameters*: parameters that describe self-organizing systems at an abstract level. Effectively, this process converts intuitive control parameter values into explicit configurations that particle swarm optimization can directly apply. In this paper, we introduce the motivation, methodology, and implementation of our approach.

I. INTRODUCTION

Particle swarm optimization (PSO) has been shown to be very effective in multiple optimization domains, particularly function approximation [1]. However, each instance of PSO will behave differently based on the values of the control parameters. Selecting ideal values for these control parameters is important, since they affect the success of the overall system.

Traditionally, to use PSO, a user must select a value for each of PSO's control parameters. We call these parameters *explicit control parameters* (ECPs), because they are explicitly specified in the PSO program. For each application of PSO, the user has to configure each of the following ECPs: the number of particles; the size of the neighborhoods; the number of neighborhoods in the swarm; the strength of the velocity vectors that are contributed to a particle's velocity by the personal, neighborhood, and global best values; and the magnitude of the particles' momentum. Generally, the relationships between the values of these parameters and the swarm-level behavior are not intuitive and not linear. This means that even experienced users can only do slightly better than trial and error when configuring PSO for a particular domain. Previously, analysis of these parameters has been performed by hand. For example, Shi and Eberhart discovered that using an inertial weight roughly between 0.9 and 1.2 gives the best overall performance of the algorithm [2]. They performed a number of experiments using hand-selected values for each of the parameters and used these results as the basis for providing insight on the general behavior of PSO.

When observing PSO, we can identify certain numerically measurable properties that can be used to describe the entire swarm. For example, each swarm will have a different average particle velocity. We call these types of properties *non-explicit control parameters* (NECPs). When observing a

self-organizing system, such as PSO, most of the properties users naturally notice are non-explicit, and thus traditionally not directly controllable. However, it would be more intuitive to control the behavior of PSO by adjusting NECPs, instead of ECPs. Our contribution is a proposed method for implementing NECPs that can be used to control PSO intuitively at the swarm level. Implementation involves first defining the NECPs, then developing mappings that correlate the values of ECPs to observed NECPs, and vice versa. Basically, these mappings translate control parameters that are intuitive for a human user into configurations that PSO can understand. We have implemented several NECPs for PSO systems (as discussed later), and are currently implementing and testing the mappings.

II. BACKGROUND

PSO was introduced in 1995 by Kennedy and Eberhart [1] as an algorithm for predicting human social behavior. The algorithm was based on observations of bird flocking and fish schooling behavior. However, PSO was found to be very useful for approximating complex mathematical functions, such as the Rosenbrock function, the generalized Rastigrin function, the sphere function, and the generalized Griewank function. PSO approximates the maxima of these functions well, even in domains with 30 or more dimensions [3].

In PSO, many agents (particles) explore areas of the domain's search space. For our purposes, the domain being explored is a function for which the system is trying to approximate a value for $f(\mathbf{x})$, where f is the function and \mathbf{x} is the input vector. Each particle in the swarm has an associated position \mathbf{x}_k , which corresponds to a point in the search space. The particles are organized into neighborhoods, which keep track of the maximum observed value for $f(\mathbf{x})$ found by one of their member particles. Three types of observed maxima of $f(\mathbf{x})$ are tracked: each agent's personal best ($local_k$), each neighborhood's best ($neighborhoodbest_j$), and the global best ($globalbest$). The velocity for agent k , v_k , which is a member of neighborhood j , is adjusted at every iteration with the following policy:

$$v_k \leftarrow c_0 v_k + c_1 (local_k - x_k) + c_2 (neighborhoodbest_j - x_k) + c_3 (globalbest - x_k).$$

The second, third, and fourth terms in this summation are velocity vectors pointing in the direction of the local, neighborhood, and global bests, respectively. The first term represents momentum, which preserves some of the agent's previous direction. The constants c_0, c_1, c_2, c_3 adjust the strength of each force and are some of the explicit control parameters of PSO.

III. APPROACH

The first step of our approach is to define each NECP using a function $g : \mathbf{s} \rightarrow \text{NECP}$ that computes a property of the observed state \mathbf{s} of PSO. For example, the velocities of the particles are averaged to produce the value for the NECP \bar{v} . A user could determine the values for NECPs by selecting values for the ECPs, then running PSO. If PSO did not perform desirably, the user would have to adjust the ECPs and once again observe the resultant NECP values. This trial-and-error process is an inefficient use of both the computer's and the user's time. Instead, we propose to learn a mapping $T : \mathbf{p} \rightarrow \mathbf{y}$, where \mathbf{p} is the vector of ECPs and \mathbf{y} is the vector of NECPs. With T , we can predict the observed behavior of PSO without having to run it. Since particles will behave differently in different domains, T will vary from domain to domain. For this reason, it is not practical to prepare the mapping by hand, as the results will not always generalize to other domains.

Instead of developing the ECP to NECP mapping manually, we use standard regression techniques, such as k-nearest neighbor or local linear regression, to learn these mappings for any optimization problem we are using PSO on. Once we have produced this mapping, a user can query it to determine the behavior of the particles without having to run PSO. This will prevent users from having to run PSO only to realize that the swarm did not behave desirably.

However, this will still require users to try multiple configurations of ECPs with a user interface before producing NECPs that are within desired ranges. We can further improve the configuration process for PSO by allowing users to directly control the swarms in terms of the NECPs, rather than the ECPs. In order to do this, we need to find the mapping $T^{-1} : \mathbf{y} \rightarrow \mathbf{p}$, the inverse of T . Once we have computed T , we can approximate T^{-1} by using an optimization algorithm, such as gradient descent, simulated annealing, or even PSO. Alternatively, if we are able to "invert" the regression algorithm we used to determine T^{-1} , we can use it to calculate the inverted mapping T^{-1} directly. Once T^{-1} is defined, users can provide desired values of the NECPs, which are translated into a set of values for the ECPs. This provides users with a more intuitive means of controlling PSO.

IV. CONTROLLING PSO

We have applied our approach to implement several useful NECPs for use in controlling PSO. Using these NECPs, users can describe their desired PSO behavior in intuitive terms. These NECPs include: average change in density with respect to time ($\Delta\bar{\rho}$), average particle velocity (\bar{v}), and ratio of average

global density to average neighborhood density (*spread*). Our approach is general enough that new NECPs can be added to this set. We have experimented with over ten NECPs, but we only describe some of the more useful ones in this paper.

A. Average Change in Density Over Time

The average change in density over time, $\Delta\bar{\rho}$, measures the average rate of convergence of the optimizations. It is computed by using the density ρ_t at each time step to calculate the change in density $\Delta\rho_t$ at each time step. These values are then averaged to produce $\Delta\bar{\rho}$. This NECP can be used to control the searching behavior of PSO. If the rate of change is high, PSO will converge very quickly, but will then have a greater probability of finding only a local maximum. With a low rate of change, the swarm will tend to search over more available space before focusing on a maximum. In some cases, the rate of convergence is not linear, and could be represented by taking another set of differences to generate a similar NECP that represents the "acceleration" of the density.

B. Average Particle Velocity

The average particle velocity \bar{v} , which computed as:

$$\bar{v} = \frac{1}{t_f} \sum_{t=0}^{t_f} \frac{1}{N} \sum_{i=0}^N v_{i_t},$$

measures the average particle velocity over every time step of the system. In this equation, t_f is the total length of time that the optimization runs; N is the number of particles; and v_{i_t} is the velocity of particle i at time t . Modifying this NECP controls the breadth of the search, because particles that move faster cover more area, but in less detail.

C. Average Global Density vs. Average Neighborhood Density

The NECP *spread* is the ratio between the global density of the system and the average density of the neighborhoods. If neighborhoods are denser than the global density, the neighborhoods do not overlap very much with one another. Meanwhile, if the neighborhoods are less dense than the global density, the neighborhoods are intermingling in the search space. Therefore, to create a more clustered PSO, the value for *spread* should be increased.

V. CONCLUSION

By using NECPs, we can improve the ease of configuring PSO systems by allowing users to control these swarms with intuitive parameters. Also, we can provide users with the ability to predict the behavior of PSO without actually running the costly optimization process.

REFERENCES

- [1] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, 1995.
- [2] Y. Shi and R. C. Eberhart, *Parameter selection in particle swarm optimization*, ser. Lecture Notes in Computer Science. Springer Berlin, 1998, vol. 1447/1998, pp. 591–600.
- [3] Y. Shi and R. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the 1999 Congress on Evolutionary Computation*, 1999.