MTRN4010 - Particle Swarm Optimization

Contents

1 Swarm Intelligence

1.1 Properties of Swarm Intelligence

Swarm Intelligence (SI) refers to a class of algorithms that emulates systems constituted by many individuals, being those individuals coordinated by a decentralized control and by self-organization. The algorithm focuses on the collective behavior which results from the local interactions between individuals and from the interaction of those individuals with the environment in which they operate. Clear examples of systems involved in swarm intelligence are certain colonies of insects (ants, bees, termites), fish schools, among many other cases in nature.

A typical swarm intelligence system usually presents the following properties:

1. 1. It is composed by many individuals.
2. The individuals are relatively homogeneous.

3. 3. The interactions between the individuals are based on simple behavioral rules exploiting local information which the individuals directly exchange, or which is perceived from the environment.

4. 4. The global behavior of the system results from the local interactions of the individuals with each other and with their surrounding context.

The main property of a SI system is its ability to act coordinately not relying on a coordinator or on an external supervisor. Despite the lack any higher-level agent coordinating the group, the swarm, as a whole entity, does show intelligence in certain sense. This is the consequence of the multiple interactions of spatially spread neighboring individuals, all of them continuously applying pre-defined basic rules. The SI concept has been applied in many fields, in recent years, with success, for solving diverse problems, many of them difficult to be treated with traditional optimization approaches.

1.2 Particle Swarm Optimization

The PSO algorithm belongs to the family of SI algorithms, which were developed and introduced by Kennedy and Eberhart [1]. It is a population-based optimization approach. It presents similarities to evolutionary computation techniques, particularly with Genetic Algorithms (GA). The PSO optimization process is initialized with a population of particles, each of them corresponding to a hypothetical solution of the problem to be solved. Those initial particles are usually generated randomly (although not necessarily uniformly distributed). After the initialization stage, the process searches for optima through continuously updating properties of the individuals at each step or generation. However, PSO operation differs to that of the GA, as PSO does not involve typical evolution operations such as mutation or crossover. The potential solutions are known as particles having those particles the capability to navigate through the domain of the variables (or parameters) being optimized, with the goal of minimizing a defined cost function. This navigation is decided, by each individual particle, based on its individual best experience, and on those of the currently known best particles. This characteristic of using particles, for optimization purposes, may be comparable to that of particle filters in the context of Bayesian estimation, although the ways of updating populations are based on different procedures.

2 Particle Swarm Optimization

2.1 Formal Description of PSO

Suppose there are M individual agents, which we call particles, used in the PSO algorithm. Each agent is treated as a particle (having no volume, such as a point) and being associated to certain properties: current position vector, velocity vector and the personal best position vector. These vectors are N-dimensional vectors (whose dimensionality depends on the problem being treated). At the n-th iteration, the three vectors describing the properties of the particle $i(1 \leq i \leq M)$ are:

1. The current position vector: $X_{i,n} = (X_{i,n}^1, \cdots, X_{i,n}^j, \cdots, X_{i,n}^N)$.
2. The velocity vector: $V_{i,n} = (V_{i,n}^1, \cdots, V_{i,n}^j, \cdots, V_{i,n}^N)$, denoting the increment of the current position, in which $1 \leq j \leq N$.
3. The personal best (experienced) position vector: $P_{i,n} = (P_{i,n}^1, \cdots, P_{i,n}^{;} \cdots, P_{i,n}^N)$ for $1 \leq j \leq N$.

We remark that the personal best position vector, whose notation is p_{best}, contains the position at which the particle experienced its best objective function (aka fitness value) during the previous iterations of the recursive process. At the initialization step, the initial values $X_{i,0}$ may be randomly generated in the domain of the variable being adjusted in the optimization; being the values of the vectors' components limited in the ranges $[X_{min}^j, X_{max}^j](1 \leq j \leq N)$, in which X_{min}^j and X_{max}^j are, respectively, the lower and and upper limits of the particle positions in the j-th dimension. Usually, a uniform distribution, in the range $[X_{min}^j, X_{max}^j]$, in the j-th dimension, is used to deploy the initial position vector. Similarly, the initial velocity vector $V_{i,0}$ may be initialized by by choosing its j-th component randomly in the range $[-V_{max}^j, V_{max}^j]$, where V_{max}^j is a upper limit for the magnitude of the velocities in the j-th dimension. The initial best position, $P_{i,n}$, can assumed equal to the initial current position vector.

At the n-th iteration, the particle whose personal best position is better than those of the rest of the particles, is considered to be the (current) global best particle. This personal best position is called $P_{g,n}$, which is recorded in a position vector $G_n = (G_n^1, \cdots, G_n^N)$ known as the global best position. In the expression $P_{g,n}$, g is the index of the global best particle.

Although an optimization process may involve a minimization or a maximization process, we will assume, in the rest of this discussion, that we are trying to minimize certain cost function (which is equivalent to maximizing certain reward function). We consider the minimization of a cost function $f(x)$. The update of $P_{i,n}$ is performed as follows:

$$P_{i,n} = \begin{cases} X_{i,n}, & f(X_{i,n}) < f(P_{i,n-1}) \\ P_{i,n-1}, & f(X_{i,n}) \geq f(P_{i,n-1}), \end{cases} \tag{1}$$

The global best particle is also updated, G_n is then $G_n = P_{g,n}$, in which $g = \text{argmax}_{1 \leq j \leq M}\{f(P_j, n)\}$.

At the subsequent iteration, $(n+1)$, the properties of each particle are updated. For each particle, the components of its velocity vector are modified according to:

$$V_{i,n+1}^j = V_{i,n}^j + c_1 \cdot r_{i,n}^j \cdot (P_{i,n}^j - X_{i,n}^j) + c_2 \cdot R_{i,n}^j \cdot (G_n^j - X_{i,n}^j), \tag{2}$$

and based on its new velocity vector, the particle's position is then updated,

$$X_{i,n+1}^j = X_{i,n}^j + V_{i,n+1}^j, \tag{3}$$

for $i = 1, \cdots, M$, $j = 1, \cdots, N$, in which c_1 and c_2 are constant acceleration coefficients. The parameters $r^j_{i,n}$, and $R^j_{i,n}$, are independent random numbers, uniformly distributed over the range $(0, 1)$ ($\{r^j_{i,n} : j = 1, \cdots, N\} \sim U(0, 1)$ and $\{R^j_{i,n} : j = 1, \cdots, N\} \sim U(0, 1)$.) Usually, the vector $V_{i,n}$ has each of its components bounded the interval $[-V^j_{max}, V^j_{max}]$.

It is interesting to see the role of the terms $(G^j_n - X^j_{i,n})$ and $(P^j_{i,n} - X^j_{i,n})$; both acting as attractors, influencing the acceleration of the particle.

The PSO in which the velocities of the particles are updated according to the expression in (2) is called the original PSO.

2.2 Qualitative Analysis of the Velocity Updating Equation

The parameters c_1 and c_2, called the acceleration coefficients, are positive constants; the factors $r^j_{i,n}$ and $R^j_{i,n}$ are random numbers, independent and uniformly distributed in the range $(0, 1)$. $P_{i,n}$ is the personal best position of the i-th particle, and G_n is the global best known position at the n-th iteration. The global best position is selected from the personal best positions of all particles in the swarm, at each iteration. From Equation 2, it is we see that the updating of each individual velocity is defined by three contributions as:

1. A previous velocity, $V^j_{i,n}$, known as the 'inertia' part, usually securing sufficient momentum for the particles, to navigate through the search space.

2. The 'cognition' part, $c_1 r^j_{i,n}(P^j_{i,n} - X^j_{i,n})$, which constitutes the particle's own experience.

3. The 'social' part, $c_2 R^j_{i,n}(G^j_{i,n} - X^j_{i,n})$, representing the information shared by the whole population.

In those cases in which $c_1 = 0, c_2 = 0$, there are no cognition nor social parts contributing to the velocity update. In such case there is no acceleration, so that the particle moves at constant velocity, for finally reaching the boundary of the search domain. No optimal solution would be found except in the unlikely cases in which a solution lies on that trajectory. When $c_1 = 0$, there is no cognition part in Equation 2. The resulting velocity update equation is known as the social-only model. Due to this interaction among particles, the PSO algorithm has the ability to reach new search areas. The particle swarm converges at a relatively fast speed; however, the algorithm can frequently converge and get trapped in local optimal points, when applied in non-convex complex problems. However, it may have good convergence and speed for treating some specific problems.

When $c_2 = 0$, there is no social part in the velocity update. This means that there is no information being shared between particles. The PSO algorithm is said to be running a cognition-only model. By ignoring the interaction between individuals,the swarm of M particles operates as M isolated individuals running independently. In practice such PSO algorithm implements a multi-start random search algorithm, and usually has slower convergence rate than the algorithm based on social-only model.

When there is no inertia part, the particle velocity is determined by its personal best position and the global best position. In this mode of operation, the velocity of the particle is said to be memory-less, having the particle a better local search ability. Due to that, the algorithm does usually converge more rapidly than the original PSO. However, not having the inertia part, particles can only sample positions in the proximity of their personal best, and in the proximity of global best position, decreasing the probability of exploring other areas, in which the global optimal solution may actually be located.

3 Variants of PSO

3.1 Inertia Weight

For tuning the balance between the local search and global search, during an optimization process in the original PSO, the concept of an inertia weight, w, was introduced. The question was about preferring a swarm inclined to explore far away, or to explore in more detail locally. The addition of the inertia weight resulted in a modified velocity update:

$$V_{i,n+1}^{j} = w \cdot V_{i,n}^{j} + c_1 \cdot r_{i,n}^{j} \cdot (P_{i,n}^{j} - X_{i,n}^{j}) + c_2 \cdot R_{i,n}^{j} \cdot (G_{n}^{j} - X_{i,n}^{j}). \tag{4}$$

The PSO algorithm based on (4) is known as PSO with Inertia Weight (PSO-In). The inertia factor w (which must be a positive value) can be chosen according to experience, or can be dynamically defined by a function of the iteration number. When $w = 1$, the PSO-In is equivalent to the original PSO. The values of c_1 and c_2 in Equation 2 are usually set to be 2, which means that the social and the cognition parts have identical influence on the velocity update. For values of w in the range [0.8 , 1.2], the PSO algorithm has better chances of finding the global minimum in a reasonable number of iterations. For settings $w > 1$, the system may diverge rapidly (it is an unstable model, only compensated by the social and cognitive parts.) However, for values of w in the range of $0.9 - 1.2$, Shi and Eberhart [2] introduced a significant improvement in the performance of the PSO method, by defining and applying an iteration-varying $w(n)$, which linearly decreases through the generations. This variation of w is expressed as follows:

$$w_n = \frac{(w_{initial} - w_{final}) \cdot (n_{max} - n)}{n_{max}} + w_{final}, \tag{5}$$

in which w_n is the value of the inertial weight at the n-th iteration, $w_{initial}$ and w_{final} are the initial and final values of the inertia weight, n is the current iteration number, and where n_{max} is the expected maximum number of iterations.

Due to that, during the initial iterations of the optimization process, the PSO has, temporarily, a large w_n, which makes the swarm to achieve high diversity, tending to spread and explore the full domain. However, w_n keeps gradually decreasing, so that at the end of the search it turns to be small; which usually helps the PSO to converge to the optimal solution, nearby, performing a fine-tuning. In commonly used configurations, the value of w varies from $w_{initial} = 0.9$ at the beginning, reaching $w_{final} = 0.4$ at the end of the run.

3.2 Random Inertia Weight

For a more effective treatment of the dynamic nature of many applications, a random inertia weight was proposed in [3]:

$$w_n = \frac{0.5 + urand}{2}, \tag{6}$$

in which urand is generated following an uniform distribution, in the range $(0, 1)$. Due to that, w_n ranges between 0.5 and 1.5, having a mean value of 0.75. Diverse benchmark cost functions have been treated applying this way; the results indicated that it produces a fast convergence in the early stages of the optimization process, and that it can obtain a satisfactory solution for most of the functions treated.

In [4], Pant et al. proposed a novel way ot dynamic inertia weight, by using a Gaussian distribution. This kind of random inertia weight is defined by the following equation:

$$w_n = \frac{|randn|}{2}, \tag{7}$$

in which *randn* is a random number generated from a Gaussian distribution, of zero means, and defined variance. The absolute value secures that the generated w_n are always positive.

Different probability distributions, such as Gaussian distribution and other exponential probability density functions, may be used to initialize the velocities and positions of the particles. The distribution used in their study has the probability density function:

$$f(x) = \frac{1}{2b} \exp\left\{ -\frac{(x-a)}{b} \right\}, \tag{8}$$

in which $a, b > 0$ are parameters that can be changed to control the expected value and the variance of the PDF. The values of a and b were chosen to be $a = 0.3$ and $b = 1$ (which results in a mean value 0.78 and a standard deviation 0.93 for the randoms numbers generated by the exponential distribution.)

3.3 Nonlinear-Decreasing Inertia Weight

For adjusting the behavior of the minimization process, during the initial iterations and for the subsequent iterations, Chatterjee and Siarry [5] proposed a nonlinear adaptive inertia weight for PSO. The nonlinear variation of the velocity inertia weight can improve the speed of convergence and to be able to fine-tune the search. This nonlinear inertia weight is implemented as follows:

$$w_n = \frac{(n_{max} - n)^r}{n_{max}^r} \cdot (w_{initial} - w_{final}) + w_{final}, \tag{9}$$

in which $w_{initial}$ and w_{final} are the initial and final values of the inertia weight, n is the current iteration number, n_{max} is the maximum number of iterations, r is the modulation index.

3.4 Adaptive Inertia Weight

The adaptive inertia weight considers the influence of a particle on the others according to the effect of attraction towards the global best position, that is, the distance from the particles to the global best position, was proposed by Suresh et al. [6]. At the n-th iteration, the value of inertia weight for particle i is defined by:

$$w_{i,n} = w_0 \cdot \left(1 - \frac{d_{i,n}}{D_n}\right), \tag{10}$$

in which w_0 is a random number uniformly distributed in the range of $[0.5, 1]$, $d_{i,n}$ is the current Euclidean distance from particle i to the global best position, D_n is the maximum distance from a particle to the global best position at iteration n. That is:

$$D_n = \text{argmax}_i\{d_{i,n}\}. \tag{11}$$

The Euclidean distance $d_{i,n}$ is calculated as follows:

$$d_{i,n} = \sqrt{\sum_{j=1}^{N} \left(G_n^j - X_{i,n}^j\right)^2}. \tag{12}$$

In order to avoid premature convergence in the final stages of the search, Suresh et al. also changed the position update equation to the one given as follows:

$$X_{i,n+1}^j = V_{i,n+1}^j + (1-\rho) \cdot X_{i,n}^j, \tag{13}$$

where ρ is a random number uniformly distributed on the range of $(-0.25, 0.25)$.

3.5 Conclusion: when to use PSO

We end here this brief introduction about PSO, in which we also mentioned known variants of the algorithm. This is a large area of research. From our perspective, as users of this technique, we will use PSO as a tool for solving our particular optimization problems. PSO is usually preferred to solve problems whose characteristics make them difficult to treat by other approaches, in particular those which involve minimizing non convex cost functions (or reward functions, in maximization cases). PSO can also deal with discontinuous functions (which would be an issue for those methods based on gradients.) In addition to its good performance for dealing with complicated cost functions, the very nature of having particles, which evolve in a parallel fashion, makes PSO very well suited to gets the benefit of parallel processing (e.g. by GPUs). This is well relevant in cases in which the dimensionality of X is high, which requires to increase the number of particles. All those matters are usually transparent to us, as currently we can use PSO solvers, already implemented and refined such as the one offered by Matlab. We will use the PSO solver, included in the optimization toolbox, in Matlab, for solving a problem.

These lecture notes are a modified version of the lecture notes produced by a former lecturer, who used to teach PSO in MTRN4010.

References

[1] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pp. 39–43, IEEE, 1995.

[2] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," in *International conference on evolutionary programming*, pp. 591–600, Springer, 1998.

[3] R. C. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, pp. 94–100, IEEE, 2001.

[4] M. Pant, T. Radha, and V. Singh, "Particle swarm optimization using gaussian inertia weight," in *Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on*, vol. 1, pp. 97–102, IEEE, 2007.

[5] A. Chatterjee and P. Siarry, "Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization," *Computers & operations research*, vol. 33, no. 3, pp. 859–871, 2006.

[6] K. Suresh, S. Ghosh, D. Kundu, A. Sen, S. Das, and A. Abraham, "Inertia-adaptive particle swarm optimizer for improved global search," in *Intelligent Systems Design and Applications, 2008. ISDA'08. Eighth International Conference on*, vol. 2, pp. 253–258, IEEE, 2008.