RESEARCH ARTICLE OPEN ACCESS

An Investigation on Basic Concepts of Particle Swarm Optimization algorithm for VLSI Design

Rajeswari.P [1], Afsana R Chadachana [2,] Dr.Theodore Chandra S[3]
1(Dayananda Sagar College of Engineering, Bangalore, and)
2(PG student, Dayananda Sagar College of Engineering, Bangalore)
3 (Dept of ECE, Dayananda Sagar University, Bangalore)

Abstract:

Particle Swarm Optimization algorithm is an evolutionary algorithm that has been applied to many different engineering and technological problems with considerable success. Since its first publication is in 1995, it has been continually modified trying to improve its convergence properties. Thus, many variants have been proposed in the literature. Some of these variants were related to a particular problem and had little application outside the field where they have been proposed. These PSO variants have been used to solve a wide range of optimization and inverse problems: continuous, discrete, dynamical, multioptimal, and combinatorial, with and without additional constraints. This paper insisting the importance of the stochastic stability analysis of the particle trajectories in order to achieve convergence and an algorithm for classical particle swarm optimization (PSO) is discussed. Also, it codes in MATLAB environment is included.

Keywords **— particle swarm optimization (PSO), MATLAB.**

I. INTRODUCTION

This paper presents an investigation of particle swarm optimization algorithm inspired by the flocking and schooling patterns of birds and fish, Particle Swarm Optimization (PSO) was invented by Russell Eberhart and James Kennedy in 1995. Originally, these two started out developing computer software simulations of birds flocking around food sources, and then later realized how well their algorithms worked on optimization problems.

Particle Swarm Optimization might sound complicated, but it's really a very simple algorithm. Over a number of iterations, a group of variables have their values adjusted closer to the member whose value is closest to the target at any given moment. Imagine a flock of birds circling over an area where they can smell a hidden source of food. The one who is closest to the food chirps the loudest and the other birds swing around in his direction. If any of the other circling birds comes closer to the target than the first, it chirps louder and the others veer over toward him.

This tightening pattern continues until one of the birds happens upon the food. It's an algorithm that's simple and easy to implement.

The algorithm keeps track of three global variables. Target value or condition, global best (gBest) value indicating which particle's data is currently closest to the target, Stopping value indicating when the algorithm should stop if the target isn't found

Each particle consists of: Data representing a possible solution, a Velocity value indicating how much the Data can be changed, a personal best (pBest) value indicating the closest the particle's Data has ever come to the Target.

The particle’s data could be anything. In the flocking bird’s example above, the data would be the X, Y, Z coordinates of each bird. The individual coordinates of each bird would try to move closer to the coordinates of the bird which is closer to the food's coordinates (gBest). If the data is a pattern or sequence, then individual pieces of the data would be manipulated until the pattern matches the target pattern.

The velocity value is calculated according to how far an individual's data is from the target. The further it is, the larger the velocity value. In the bird's example, the individuals furthest from the food would make an effort to keep up with the others by flying faster toward the gBest bird. If the data is a pattern or sequence, the velocity would describe how different the pattern is from the target, and thus, how much it needs to be changed to match the target.

Each particle's pBest value only indicates the closest the data has ever come to the target since the algorithm started.

The gBest value only changes when any particle's pBest value comes closer to the target than gBest. Through each iteration of the algorithm, gBest gradually moves closer and closer to the target until one of the particles reaches the target.

It's also common to see PSO algorithms using population topologies, or "neighbourhoods", which can be smaller, localized subsets of the global best value. These neighbourhoods can involve two or more particles which are predetermined to act together, or subsets of the search space that particles happen into during testing. The use of neighbourhoods often helps the algorithm to avoid getting stuck in local minima.

II. LITERATURE SURVEY

Kennedy and R. Eberhart, Particle swarm optimization, Proceedings IEEE International Conference on Neural Networks (ICNN '95), Perth, WA, Australia, November–December (1995), Vol. 4, pp. 1942–1948. Particle Swarm Optimization (PSO) was first proposed in 1995.28 It was originated in social modelling, being one of its fathers, James Kennedy, a social psychologist. The first PSO algorithm was closely related to the graphic animation of flocks.

F. Heppner and U. Grenander, A stochastic nonlinear model for coordinated bird flocks, The Ubiquity of Chaos, edited by E. Krasner, AAAS Publications (1990), pp. 233–238. 0 With time, PSO has been further developed, modified, and successfully applied for optimization in many engineering problems and technological fields. Although the PSO algorithm seems very simple to implement, one of the most interesting questions from the beginning was understanding its convergence properties and avoiding its numerical instabilities.

B.Brandstätter and U.Baumgartner, Particle swarm optimization—Mass-spring system analogon. IEEE Transactions on Magnetics 38, 997 (2002). All these modifications tried to improve the PSO performance; one of the most important achievements was the stochastic stability analysis of the PSO trajectories and the use of physical models to understand the PSO swarm dynamics. Both approaches served to dramatically improve the knowledge about how PSO works. In conclusion, nowadays PSO should be considered as a stochastic algorithm with a well established theoretical background. Thus, PSO should not be considered a heuristic algorithm anymore.

A. Banks, J. Vincent, and C. Anyakoha, A review of particle swarm optimization. part I: background and development. Natural Computing 6, 467 (2007). Natural and biologically inspired relates to the algorithms that are based on certain natural phenomena, arising from the connection between biology, computer science, artificial intelligence and applied mathematics. In the field of optimization, these algorithms try to provide a solution to the difficulties that appear in certain technological problems, such as ill-posed inverse problems with noisy data, non-convex and non differentiable optimization, etc. that cannot be easily tackled by traditional optimization methods.

Colorni, M. Dorigo, and V. Maniezzo, Distributed optimization by ant colonies, European Conference on Artificial Life (1991), pp. 134–142. A subgroup of bio-inspired algorithms is the evolutionary algorithms. These are based on populations that evolve with time and have a stochastic component aimed at escaping from entrapment in local optima. Examples of such algorithms are Genetic Algorithms, Differential Evolution, Ant Colony System and Particle Swarm Optimization.

J. L. Fernandez Martinez, Z. Fernández-Muñiz, and M. J. Tompkins, on the topography of the cost functional in linear and nonlinear inverse

problems. Geophysics 77, W1 (2012). Ant Colony System and Particle Swarm Optimization are in case of nonlinear inverse problems these algorithm lie in flat elongated valleys of the cost function topography. In this case, these algorithms can be used to explore the cost functions, if the fitness of the members of the population is fast to manage. In any case, it is a best universal algorithm over all the possible problems.

A. Carlisle and G. Dozier, An off-the-shelf PSO, Proceedings of the Particle Swarm Optimization Workshop, Indianapolis, April (2001), pp. 1–6. The behaviour of PSO was analyzed by doing numerical experiments. As noticed before, in the initial PSO version without the inertia weight, the particles were stable if the mean total acceleration was chosen in less range and trajectories were oscillatory. Mean trajectories of the particles were unstable for the values expect this interval. Therefore the initial PSO modifications were focused at maintaining the particles inside the search space, providing swarm stability and convergence for different benchmark functions, we can expect that these results are applicable to the real life problems.

A. ADVANTAGES

1. Insensitive to scaling of design variables.
2. Simple implementation.
3. Easily parallelized for concurrent processing.
4. Derivative free.
5. Very few algorithm parameters.
6. Very efficient global search algorithm.

B.DISADNANTAGES

1. Slow convergence in refined search stage (weak local search ability)

C.PSO APPLICATIONS

1. Training of neural networks
2. Identification of Parkinson's disease
3. Extraction of rules from fuzzy networks
4. Image recognition
5. Optimization of electric power distribution networks
6. Structural optimization
7. Optimal shape and sizing design
8. Topology optimization
9. Process biochemistry
10. System identification in biomechanics.

III. ALGORITHM FLOW CHART FOR PSO

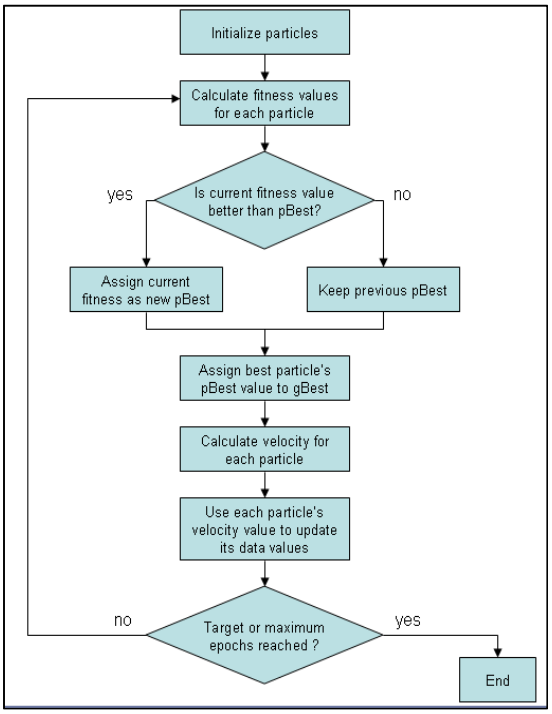

Figure 1.Flow diagram illustrating the particle swarm optimization algorithm.

IV.PSEUDO CODE OF FLOWCHART

```
For each particle
{
   Initialize particle
}

Do until maximum iterations or minimum error criteria
{
   For each particle
   {
      Calculate Data fitness value
      If the fitness value is better than pBest
      {
         Set pBest = current fitness value
      }
```

```
    If pBest is better than gBest
    {
       Set gBest = pBest
    }
}

For each particle
  {
    Calculate particle Velocity
    Use gBest and Velocity to update particle Data
  }
```

I. MAIN CODE OF THE PSO EXAMPLE

```
% ----------------------------------Start
clc;
clear all;
close all;

%% initialization

swarm_size = 64;                % number of the
swarm particles
maxIter = 50;                   % maximum
number of iterations
inertia = 1.0;
correction_factor = 2.0;

% set the position of the initial swarm

a = 1:8;
[X,Y] = meshgrid(a,a);
C = cat(2,X',Y');
D = reshape(C,[],2);
swarm(1:swarm_size,1,1:2) = D;      % set the
position of the particles in 2D
swarm(:,2,:) = 0;                   % set initial
velocity for particles
swarm(:,4,1) = 1000;                % set the
best value so far
plotObjFcn = 1;          % set to zero if you do not
need a final plot

%% define the objective funcion here (vectorized
form)

objfcn=  @(x)(x(:,1) - 20).^2 + (x(:,2) - 25).^2;
tic;
%% The main loop of PSO
for iter = 1:maxIter
   swarm(:, 1, 1) = swarm(:, 1, 1) + swarm(:, 2,
1)/1.3;      %update x position with the velocity
   swarm(:, 1, 2) = swarm(:, 1, 2) + swarm(:, 2,
2)/1.3;      %update y position with the velocity
   x = swarm(:, 1, 1);
                % get the updated position
   y = swarm(:, 1, 2);                  %
updated position
fval = objfcn([x y]);                   %
evaluate the function using the …..

% position  of the particle

   % compare the function values to find the best
ones
   for ii = 1:swarm_size
      if fval(ii,1) < swarm(ii,4,1)
swarm(ii, 3, 1) = swarm(ii, 1, 1);              %
update best x position,
swarm(ii, 3, 2) = swarm(ii, 1, 2);              %
update best y postions
swarm(ii, 4, 1) = fval(ii,1);                   %
update the best value so far
      end
   end

   [~, gbest] = min(swarm(:, 4, 1));
% find the best function value in total

   % update the velocity of the particles
   swarm(:, 2, 1) =
inertia*(rand(swarm_size,1).*swarm(:, 2, 1)) +
correction_factor*(rand(swarm_size,1).*(swarm(:,
3, 1) ...
      - swarm(:, 1, 1))) +
correction_factor*(rand(swarm_size,1).*(swarm(g
best, 3, 1) - swarm(:, 1, 1)));          %x velocity
component

   swarm(:, 2, 2) =
inertia*(rand(swarm_size,1).*swarm(:, 2, 2)) +
correction_factor*(rand(swarm_size,1).*(swarm(:,
3, 2) ...
      - swarm(:, 1, 2))) +
correction_factor*(rand(swarm_size,1).*(swarm(g
```

```
best, 3, 2) - swarm(:, 1, 2)));          %y velocity
component

    % plot the particles
clf;plot(swarm(:, 1, 1), swarm(:, 1, 2), 'bx');
% drawing swarm movements
axis([-2 40 -2 40]);
pause(.1);
            % un-comment this line to…

          % decrease the animation speed
disp(['iteration: ' num2str(iter)]);
end
toc
%% plot the function
if plotObjFcn
ub = 40;
    lb = 0;
npoints = 1000;
    x = (ub-lb) .* rand(npoints,2) + lb;
    for ii = 1:npoints
        f = objfcn([x(ii,1) x(ii,2)]);
        plot3(x(ii,1),x(ii,2),f,'.r');hold on
    end

% ----------------------------------END
```

V.RESULTS

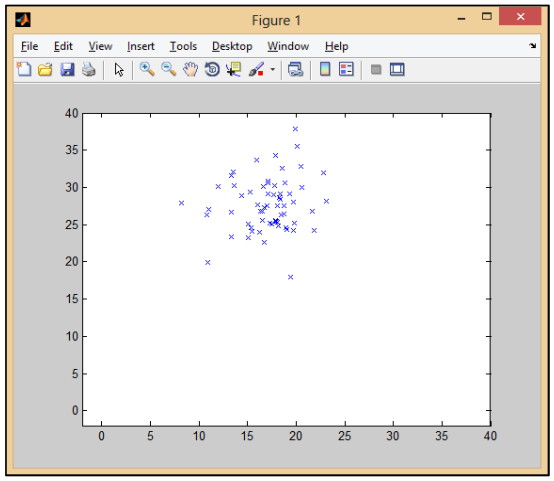

Fig2:2D view of the output (At starting point)

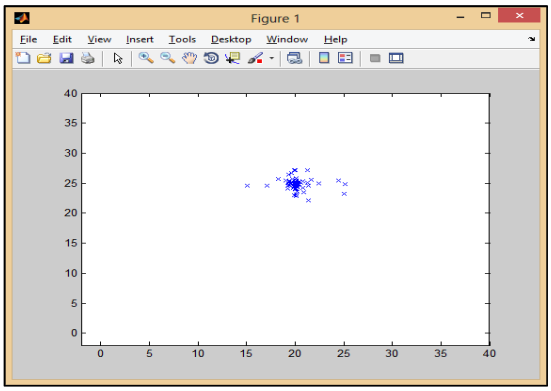

Fig3:2D view of the output

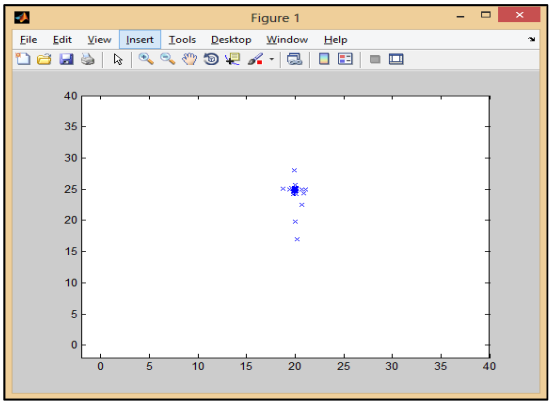

Fig4:2D view of the output (close to the destination)

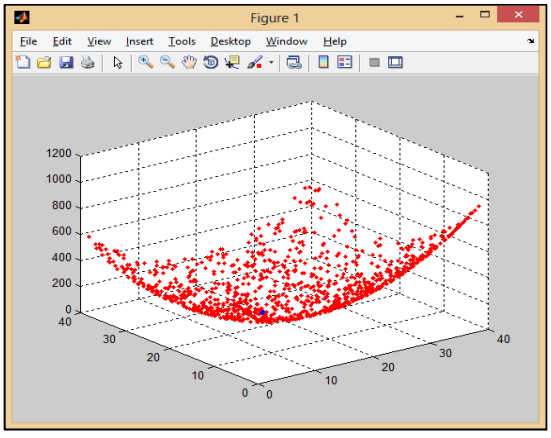

Fig5:3D view of the output

In the particle swarm algorithm, the trajectory of each individual in the search space is adjusted by dynamically altering the velocity of each particle, according to its own moving experience and the moving experience of the other particles in the search space. The position vector and the velocity vector of the i^{th} particle in the d-dimensional search space can be represented as $Xi = (x_{i1},x_{i2},x_{i3},...,x_{id})$ and $Vi = (v_{i1},v_{i2},v_{i3},...,v_{id})$ respectively. According to a user defined fitness function, let us say the best position of each particle (which corresponds to the best fitness value obtained by that particle at time t) is $Pi = (P_{i1},P_{i2},P_{i3},...,P_{id})$, and the fittest particle found so far at time t is $Pg=(P_{g1},P_{g2},P_{g3},...,P_{gd})$. Then, the new velocities and the positions of the particles for the next fitness evaluation are calculated using the following two equations:

$$V_i\ (t+1) = w\ *V_i\ (t) + C_1\ (\ P_i\ (t)\ - X_i\ (t)\ ) + C_2\ (\ g(t)\ -\ X_i\ (t)\ )$$

$$X_i\ (t+1) = X_i\ (t) + V_i\ (t)$$

Now the standard PSO is as follows, the equations for updating the new velocities and the positions of the partiicles for every next fitness value, Pbest values, Gbest values and inertia term are as follows;

$$V_{ij}\ (t+1) = w\ *\ V_{ij}\ (t) + r_1\ *\ c_1\ (P_{ij}\ (t)\ -\ X_{ij}\ (t)\ ) + r_2\ *\ c_2\ (g_i(t) - x_{ij}\ (t))$$

$$X_{ij}\ (t+1) = X_{ij}\ (t) + V_{ij}\ (t+1)$$

VI.CONCLUSION

In this work, an algorithm for particle swarm optimization (PSO) is discussed. Also, its codes in MATLAB environment are included. The effectiveness of the algorithm was analyzed with the help of an example of variable optimization problem. Also, the convergence characteristic of the algorithm was discussed successfully.

REFERENCES

1 *J. Kennedy and R. Eberhart, Particle swarm optimization, Proceedings IEEE International Conference on Neural Networks (ICNN '95), Perth, WA, Australia, November–December (1995), Vol. 4, pp. 1942–1948. 29*

2 *F. Heppner and U. Grenander, A stochastic nonlinear model for coordinated bird flocks, The Ubiquity of Chaos, edited by E. Krasner, AAAS Publications (1990), pp. 233–238.*

3 *B. Brandstätter and U. Baumgartner, Particle swarm optimization— Mass-spring system analogon. IEEE Transactions on Magnetics 38, 997 (2002).*

4 *Banks, J. Vincent, and C. Anyakoha, A review of particle swarm optimization. part I: background and development. Natural Computing 6, 467 (2007).*

5 *Colorni, M. Dorigo, and V. Maniezzo, Distributed optimization by ant colonies, European Conference on Artificial Life (1991), pp. 134–142.*

6 *J. L. Fernández-Martínez, Z. Fernández-Muñiz, and M. J. Tompkins, On the topography of the cost functional in linear and nonlinear inverse problems. Geophysics 77, W1 (2012).*

7 *Carlisle and G. Dozier, An off-the-shelf PSO, Proceedings of the Particle Swarm Optimization Workshop, Indianapolis, April (2001), pp. 1–6.*