Particle Swarm Optimization (PSO) Technique and its Variant Binary PSO (BPSO)

Presented by: Samia Abid
Student: MS(CS)

Supervised by: Dr. Nadeem Javaid
Associate Professor, Department of Computer Science,
COMSATS Institute of Information Technology, 44000, Islamabad, Pakistan.
Outline

- Bio-inspired algorithms
- Why bio-inspired techniques.
- Taxonomy of bio-inspired techniques.
- What is swarm intelligence?
  - Applications
- What is Particle Swarm Optimization (PSO)?
  - How does it work?
  - PSO algorithm.
  - Algorithm example.
  - Characteristics.
- BPSO
  - How does it work?
  - Energy optimization by BPSO.
Bio-inspired algorithms are search methods that simulate the natural biological evolution or the behaviour of biological entities [1].

Bio inspired algorithms has a wide range of applications covering almost all areas including:
- Computer networks
- Security
- Robotics
- Bio medical engineering
- Control systems
- Parallel processing
- Data mining
- Power systems
- Production engineering and many more.

Why Bio-inspired techniques?

- Optimization is a commonly encountered mathematical problem in all engineering disciplines. It literally means finding the best possible/desirable solution [1].
- Optimization problems are wide ranging and numerous.
- Optimization algorithms:
  - Deterministic
  - Stochastic in nature
- Former methods to solve optimization problems require enormous computational efforts, which tend to fail as the problem size increases.
- This is the motivation for employing bio inspired stochastic optimization algorithms as computationally efficient alternatives to deterministic approach.
Bio-inspired techniques

- Evolutionary
  - GP
  - ES
  - DE
  - PFA

- Swarm Intelligence
  - PSO
    - ACO
    - ABC
    - BFA
    - GSO
    - FA
    - SFLA
    - FSA

- Ecological
  - BBO
  - AWC
  - PS2O

Taxonomy of Bio-inspired techniques
Algorithms inspired by the collective behavior of social insect colonies and other animal societies are called swarm intelligence algorithms.

The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems [2].

SI systems are typically made up of a population of simple agents interacting locally with one another and with their environment.

Natural examples:

- Ant colonies
- Bird flocking
- Animal herding
- Bacteria growth
- Fish schooling.

SI Applications

- U.S. Military is investigating swarm techniques for controlling unmanned vehicles.

- Home energy management in Smart grid.

- NASA is investigating the use of swarm technology for planetary mapping.
PSO (1/2)

- First described in 1995 [2].
- By James Kennedy and Russel C. Eberhart.
- Inspired by social behavior of birds and fishes.
- Combines self-experience with social experience.
- Population-based optimization.
- Find approximate solutions of problems.
- Easy to implement.
- Few parameters to adjust.

Similarly to genetic algorithm (GA), it is a population-based method. It represents the state of the algorithm by a population, which is iteratively modified until a termination criterion is satisfied. Uses a number of **particles** that make a swarm moving around in the search space looking for the best solution. Each particle in search space adjusts its “**flying**” according to:
- its own flying experience
- the flying experience of other particles.
In PSO, each single solution is a "bird" in the search space. We call it "particle".

All the particles have fitness values which are evaluated by the fitness function.

All particles have velocities, which direct the flying of the particles.

The particles fly through the problem space by following the current optimum particles.
How does it Work? (1/2)

- Initialized with a group of random particles [4].
- Searches for optimal by updating generations.
- Particles move through the solution space, and are evaluated according to some fitness criterion.
- In every iteration, each particle is updated by following “best” values.
  - Pbest
  - Gbest

How does it Work? (2/2)

Each particle tries to modify its current position and velocity according to the distance between its current position and \( \text{Pbest} \), and the distance between its current position and \( \text{gbest} \) [3].

– Update particles’ velocities:

\[
V_{n+1} = V_n + c_1 \cdot \text{rand1} \cdot (P_{\text{best},n} - P_n) + c_2 \cdot \text{rand1} \cdot (P_{\text{gbest},n} - P_n)
\]

– Move particles to their new positions:

\[
P_{n+1} = P_n + V_{n+1}
\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{n+1} )</td>
<td>Particle velocity at ( n+1 )th iteration</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>Acceleration factor related to ( \text{gbest} )</td>
</tr>
<tr>
<td>( C_2 )</td>
<td>Acceleration factor related to ( \text{lbest} )</td>
</tr>
<tr>
<td>( \text{Rand1} )</td>
<td>Random number between 0 and 1</td>
</tr>
<tr>
<td>( \text{Pbest} )</td>
<td>Pbest position of swarm</td>
</tr>
<tr>
<td>( \text{Pn} )</td>
<td>Current position of particle</td>
</tr>
<tr>
<td>( \text{gbest} )</td>
<td>gbest position of swarm</td>
</tr>
</tbody>
</table>
• For each particle
  – Initialize particle with random number
End
DO
  – For each particle
    • Calculate the fitness value
    • If fitness values at time t is better than the its previous best fitness value (Pbest) at time (t-1)
    • Set current value as the new Pbest
  END
  – Choose the particle with the best fitness value of all the particles as the gbest
  – For each particle
    • Update velocity
    • Update position
  – While maximum iterations not reached.
Algorithm-Example (1/8)

![Diagram showing a search space with arrows indicating movements. The x-axis is labeled as 'X', the y-axis as 'Y', and there is a color bar on the right indicating a fitness scale from min to max.]
Algorithm-Example (3/8)
Algorithm-Example (4/8)

Search space
Algorithm-Example (6/8)
Algorithm-Example (8/8)
PSO Characteristics

- **Pros [5]**
  - Simple implementation
  - Suitable for concurrent processing
  - Derivative free
  - Very few algorithm parameters
  - Very efficient global search algorithm

- **Cons [5]**
  - Premature convergence in mid optimum points
  - Slow convergence in refined search stage (weak local search ability)

---

PSO is a conventional algorithm

Applicable for continuous problems.

However, it cannot be applied to discrete problems directly.

Aiming at the discrete problems, Kennedy and Eberhart extended the PSO to BPSO in 1997 [5]

It is a binary variant of PSO.
How does it work? (1/2)

- In BPSO, population has a set of particles.
- Each individual particle represents a binary decision.
- This decision can be represented by either YES/TRUE=1 or NO/FALSE=0.
- All particles represent their positions through binary values which are 0 or 1.
- Velocity is restricted within the range \{0,1\}.
The velocity vector equation and position vector equation are defined as [5]:

- velocity vector equation:

\[
V_i^n (t + 1) = \frac{1}{1 + e^{-v_i^n(t)}}
\]

- position vector equation:

\[
x_i^n (t + 1) = \begin{cases} 
1 & \text{if } r < V_i^n \\
0 & \text{otherwise}
\end{cases}
\]

- \( r \) is the random number selected from a uniform distribution in [0, 1].
As an example, let’s say that we are dealing with a population of 5 bit binary particles and a population of 4 particles.

We are updating particle 2 (01011), bit 3(0)
Furthermore, we will assume that the current velocity of this bit to be a 1 is 0.25.

Furthermore, assume that the best value of this particle (to date) is 00100.

And the best value of the whole population (to date) is 01111.
BPSO-Example 1 (3/4)

\[ v_{23}(t-1) = 0.25 \leftrightarrow \text{Velocity of bit} \]
\[ x_{23}(t-1) = 0 \leftrightarrow \text{Position of bit} \]
\[ p_{23} = 1 \leftrightarrow P_{\text{best}} \]
\[ P_{g3} = 1 \leftrightarrow g_{\text{best}} \]
\[ c_1 = 2.5 \]
\[ c_2 = 1.5 \]
\[ v_{id}(t) = v_{id}(t-1) + c_1(p_{id} - x_{id}(t-1)) + c_2(p_{gd} - x_{id}(t-1)) \]
\[ v_{id}(t) = 0.25 + (0.25)(1-0) + (1.5)(1-0) = 4.43 \]
\[ f(v_{23}(t)) = \frac{1}{1 + e^{-4.43}} = 0.89 \]
Now, with the value for f, we generate a random number.

If the random number is less than f then bit $x$ becomes a 1 otherwise, it becomes a 0.
# Energy Optimization by BPSO (1/10) [6]

<table>
<thead>
<tr>
<th>No of Homes</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of Appliances</td>
<td>13</td>
</tr>
<tr>
<td>Timeslots</td>
<td>24 hours.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appliances</th>
<th>Power Rating (kW/h)</th>
<th>Appliances</th>
<th>Power Rating (kW/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lights</td>
<td>0.6</td>
<td>Coffee Maker (CM)</td>
<td>0.8</td>
</tr>
<tr>
<td>Fans</td>
<td>0.75</td>
<td>Washing Machine (CM)</td>
<td>0.78</td>
</tr>
<tr>
<td>Clothes Iron</td>
<td>1.5</td>
<td>Dish Washer (DW)</td>
<td>3.60</td>
</tr>
<tr>
<td>Microwave Oven</td>
<td>1.18</td>
<td>Cloth Dryer (CD)</td>
<td>4.40</td>
</tr>
<tr>
<td>Toaster</td>
<td>0.5</td>
<td>Air Conditioner (AC)</td>
<td>1.44</td>
</tr>
<tr>
<td>Refrigerator</td>
<td>0.73</td>
<td>Water Heater (WH)</td>
<td>4.45</td>
</tr>
<tr>
<td>Space Heater (SH)</td>
<td>1.50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Energy Optimization by BPSO (2/10)

Appliances

Fixed
- LOT cannot be modified
  - (Not Shiftable)
- Lighting
  - Fan, Iron, Microwave over, Toaster, Coffee maker

Shiftable
- Can be shifted without altering load profile.
  - (Uninterruptible)
- Washing machine, Dish washer, Clothes dryer

Elastic
- Can be shifted to a suitable time
  - (Interruptible)
- Air conditioner, water heater, space heater
## Parameters of shiftable appliances

<table>
<thead>
<tr>
<th>Appliances</th>
<th>Start time</th>
<th>End time</th>
<th>Waiting time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Washing machine</td>
<td>8</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>Dish washer</td>
<td>7</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>Clothes dryer</td>
<td>6</td>
<td>18</td>
<td>5</td>
</tr>
</tbody>
</table>

## Parameters of elastic appliances

<table>
<thead>
<tr>
<th>Appliances</th>
<th>Start time</th>
<th>End time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air conditioner</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>Water heater</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>Space heater</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>BPSO Parameters</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>Swarm size</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Vmax (maximum velocity)</td>
<td>4 m/s</td>
<td></td>
</tr>
<tr>
<td>Vmin (minimum velocity)</td>
<td>-4 m/s</td>
<td></td>
</tr>
<tr>
<td>Max iterations</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
### Energy Optimization by BPSO (5/10)

**Key terms corresponding to Smart grid optimization**

<table>
<thead>
<tr>
<th>PSO Parameters</th>
<th>Role in HEM</th>
<th>General use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle</td>
<td>A timeslot (One possible solution)</td>
<td>Possible solution in search area</td>
</tr>
<tr>
<td>swarm</td>
<td>Set of possible solutions</td>
<td>a set of particles</td>
</tr>
<tr>
<td>Dimension</td>
<td>Number of appliances</td>
<td>Search area</td>
</tr>
<tr>
<td>Fitness function</td>
<td>Designed objective function with constraints</td>
<td>Objective function</td>
</tr>
<tr>
<td>Position</td>
<td>Initialize vector for appliances states randomly.</td>
<td>Initialization point in search area</td>
</tr>
<tr>
<td>Velocity</td>
<td>Probability of the bit to be 1. (To turn on the appliance.)</td>
<td>Randomly generated</td>
</tr>
<tr>
<td>Particle best (Pbest)</td>
<td>Local best values for state array that satisfy objective function</td>
<td>Evaluated fitness function answer</td>
</tr>
<tr>
<td>Global best (Gbest)</td>
<td>Globally best solution that satisfy all constraints (A timeslot)</td>
<td>Evaluated fitness function answer</td>
</tr>
</tbody>
</table>
• **Step1:**
  – Initialize Particles with random number (randomly generate population)
  – Set initial position of particles as Pbest.

**Code to randomly generate population**

```matlab
for j=1:swarm
    for i=1:n
        if rand(1)>0.5
            X=1;
        else
            X=0;
        end
        x1(j,i)=X;
    end
end
```

<table>
<thead>
<tr>
<th>WM</th>
<th>DW</th>
<th>CD</th>
<th>AC</th>
<th>WH</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
• **Step 2:**
  
  — For Each Particle
  • Calculate the fitness

**Fitness Function:**

1. function [FF]=obj(electricity_cost, power rating, x1, swarm, d)
2. for i=1:swarm
3. FF(i,1)=electricity_cost*x1(i,:);’
4. err=c_electricity_cost*x1(i,:)'-d;
5. FF(i,1)=FF(i,1)+1000*abs(err);
6. end

<table>
<thead>
<tr>
<th>WM</th>
<th>DW</th>
<th>CD</th>
<th>AC</th>
<th>WH</th>
<th>SP</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2.4894</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.4916</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.4956</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.4888</td>
</tr>
</tbody>
</table>
Energy Optimization by BPSO (8/10)

- **Step 3:**
  - Choose the **Timeslot (solution)** with the best fitness value among all possible solutions as the *gbest*.

<table>
<thead>
<tr>
<th>WM</th>
<th>DW</th>
<th>CD</th>
<th>AC</th>
<th>WH</th>
<th>SP</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2.4894</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.4916</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.4956</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.4888</td>
</tr>
</tbody>
</table>

*Gbest solution*
Energy Optimization by BPSO (9/10)

• **Step 4:**
  – For each Particle
    • Update its velocity.
      
      \[
      V_i^n(t + 1) = \frac{1}{1 + e^{-v_i^n(t)}}
      \]
    • Update its position.
      
      \[
      x_i^n(t + 1) = \begin{cases} 
      1 & \text{if } r < V_i^n \\
      0 & \text{otherwise}
      \end{cases}
      \]

• **Step 5:**
  – While maximum iterations not reached.

  Note: Repeat this process 24 times to get the most optimal solution for each timeslot.
MATLAB CODE:

1. for i = 1:swarm
2.     for j = 1:n
3.         v(i,j) = v(i,j)+c1*rand(1)*(pbest(i,j)-x_BP(i,j))+c2*rand(1)*(gbestt_BP(1,j)-x_BP(i,j));
4.         if ( (v(i,j) <= vmax) && (v(i,j)>=vmin) )
5.             v(i,j) = v(i,j);
6.         elseif ( v(i,j) < vmin )
7.             v(i,j) = vmin;
8.         elseif ( v(i,j) > vmax )
9.             v(i,j) = vmax;
10.     end
11.     sig(i,j) = 1/(1+exp(-v(i,j)));
12.     if rand(1) < sig(i,j)
13.         x_BP(i,j) = 1;
14.     else
15.         x_BP(i,j) = 0;
16.     end
17. end
18. end

This shows the velocity of the particle should stay in a limit
Apply sigmoidal function on the velocity
Any questions?