Project ID - 51

Group 2 - Agrima, Piyush.

PART 1 - Basic Overview

What is meant by community in a social network?

➔ Imagine you're on Instagram. You follow your school friends, your family, some influencers, and maybe people from your coding club.

Now think of this as a **network**:

- Each **person** is a **node (or point)**
- If two people follow each other or interact (like, comment, message), there's a **connection (or edge)** between them

A **community** in this network is a group where:

- People are more connected to each other **within the group**
- And **less connected to people outside** that group

So:

- Your school friends could form one community
- Your coding club another
- Family could be a third

These communities **aren't assigned**–they **form naturally** based on how people interact.

Why is community detection important?

➔ By finding these communities, we can:

- Understand **who interacts with whom**
- Spot **influential users** in a group (e.g., someone who can start a viral trend)
- Improve **recommendations** (like suggesting friends or content)
- Strengthen **cybersecurity** by spotting unusual groups or fake accounts
- Optimize **marketing strategies** (targeting specific communities)

So, **detecting communities helps us make sense of big, messy networks**.

Quick Glimpse of PSO (Particle Swarm Optimization)

➔ To detect communities, we use **algorithms** - smart mathematical methods that can find patterns in complex networks.

One such algorithm is **Particle Swarm Optimization (PSO)**.

Here's a super simple way to think of it:

Imagine a group of birds flying together to find food. Each bird doesn't know where the food is, but they follow the bird that's closest to the food - slowly, the whole flock gets closer to the goal.

That's how PSO works - each "bird" (or **particle**) is a potential solution. They move around in the search space, learn from each other, and eventually find the best solution (in our case, the best community structure).

Why Do We Need to Improve It?

➔ While PSO is smart and useful, it's not perfect. Sometimes:

- It **gets stuck too early** (before finding the best solution)
- It **misses better answers** because it doesn't explore enough
- It **takes time** to fine-tune the communities

That's where our improvements come in…

PART 2 - Why Basic PSO Isn't Enough (and How We Fix It)

The Problem with Basic PSO

➔ While PSO is a great algorithm for solving many complex problems (like community detection), it has some **real limitations**, especially when dealing with huge and complicated social networks.

Here's what usually goes wrong:

1. Premature Convergence

PSO can sometimes "settle too early" on a solution that looks good at first - but isn't actually the best.

Imagine the birds in the flock all decide to follow the wrong bird who thought he saw food - but actually didn't. They stop exploring, thinking they've already found the best spot.

2. Lack of Diversity

If all the particles start too similarly, or if they quickly start copying each other, the algorithm doesn't explore enough new possibilities.

This means the solution space (all possible ways of dividing communities) isn't fully explored - and we might miss better communities.

3. Balancing Exploration vs. Exploitation

PSO needs to balance two things:

- **Exploration** (searching for completely new possibilities)

- **Exploitation** (improving the current best solutions)

If it does too much of one and not enough of the other, the performance drops.

How Do “WE” Fix These Issues?

➔ That’s where our **Enhanced PSO (EPSO)** named as ***OBPSO-AIW*** comes in - we added three smart techniques to make it better:

1. Opposition-Based Learning (OBL)

Before the particles even start searching, we use a trick called **Opposition-Based Learning**.

For every initial solution, we also look at its **opposite** version. Then we choose whichever is better to start with.

This **boosts diversity** right at the beginning and gives us a wider search area. It’s like starting the birds in different corners of the sky - some will get closer to the food faster!

In this step, for every solution we generate, we also create its opposite version using a simple formula:

$$X' = A + B - X$$

Where:

- X is the current value (solution)
- A is the lower limit
- B is the upper limit
- X' is the opposite value

We then compare both X and X', and choose the better one to start with.

This helps the algorithm start with more variety in solutions and improves the chances of finding a good result early on.

2. Adaptive Inertia Weight (AIW)

In PSO, **inertia weight** controls how much a particle "trusts" its previous direction.

We made this **adaptive** - it starts high (to explore more) and gradually lowers (to focus on refining the best answers).

This helps balance **exploration and exploitation** smartly over time, making sure we don’t get stuck early.

$$\omega = \omega_{\max} - \left(\frac{\omega_{\max} - \omega_{\min}}{\mathrm{N}_{\max}}\right) \times \text{iteration}$$

Where :

- ω_{max} = initial (high) inertia weight
- ω_{min} = final (low) inertia weight
- N_{max} = total number of iterations
- iteration = current step number

3. Local Search Mechanism

Once we've found a pretty good solution, we do an extra step - we **locally fine-tune** it to squeeze out even better results.

This increases **modularity**, which means cleaner, higher-quality communities.

End Result?

➔ With **OBL + AIW + Local Search**, our Enhanced PSO:

- Finds better communities
- Avoids getting stuck in bad solutions
- Works faster and more accurately
- Handles both **small and large networks**

We've tested it on real social network datasets - and it's outperforming the regular PSO and other methods out there.

PART 3 - Understanding EPSO Step by Step (The Algorithm Flow)

Step 1: Initialize the Particles

- Start with a random set of possible solutions (particles).
- Each particle has a position (community structure) and a velocity (how it changes).
- We also calculate the opposite version of each solution using Opposition-Based Learning (OBL) to get more variety from the start.

Step 2: Evaluate Each Particle

- Use the modularity score to check how good each community structure is.
- The better the modularity, the stronger the community detection.

Step 3: Update Velocity and Position

- For each particle, update its velocity using the Adaptive Inertia Weight (AIW):
  - ➢ It starts high to explore widely and reduces gradually to fine-tune results.
- The new position (community assignment) is calculated using this velocity.

Step 4: Apply Local Search

- After updating, we apply a local search to each solution.
- This means we slightly change some node assignments to see if it improves modularity.
- If it does, we keep the new version.

Step 5: Update Best Values

- Keep track of the best solution each particle has found (pBest).
- Also update the global best solution found by the whole swarm (gBest).

Step 6: Repeat Until Max Iterations

- Continue steps 2 to 5 for a fixed number of iterations or until results stop improving.

Final Output

➔ After all iterations, the particle with the highest modularity score is chosen.
This gives us the best community structure for the network.