3 research paper fundamentals

Tuesday, October 21, 2025 5:56 PM

QPSO

Preliminaries

Quantum particle swarm optimization (QPSO)

QPSO uses quantum mechanics to optimize. The combination of classical and quantum principles allows researchers to find new optimization methods[16]. A population-based technique with changing particles solves difficulties. Quantum-inspired ideas like wave functions and Monte Carlo approaches could speed convergence and improve solution quality in QPSO, making it beneficial for optimization issues[18].

$$x_i(z+1) = \begin{Bmatrix} p + \beta * |MPV_i - x_i(z)| * V_n * u \, if \, k \geq 0.5 \\ p - \beta * |MPV_i - x_i(z)| * V_n * u \, if \, k < 0.5 \end{Bmatrix} \quad (1)$$

- $(z+1)$: Updated position of particle i at iteration $z+1$.
- p: Local attractor.
- β: Contraction–expansion coefficient.
- MPV: Personal best solution of particle i.
- V_n: Normalized velocity.
- u: Random value between 0 and 1.
- k: Random value between 0 and 1.

In the QPSO, Eq. 1 shows how to change the position of a particle.

$$p = \frac{(L_1 * f_1 * Perbest(i) + L_2 * (1 - f_2) * Globest)}{(L_1 + L_2)} \quad (3)$$

- p: Local attractor.
- $L1, L2$: Acceleration coefficients.
- $f1, f2$: Random numbers between 0 and 1.
- (i): Personal best solution of particle i.
- $Globest$: Global best solution.

$$MBV = \left(\frac{1}{N}\right) * \sum_{d=1}^{N} PVP_i(z)$$

- MBV: Global mean best.
- N: Number of particles.
- (z): Global best solution of particle i at iteration z.
- d: Dimensionality of the search space.

It uses Monte Carlo sampling and a random element to let particles explore the solution space and choose their best and the best for everyone. This equation allows population selection and particle movement toward better solutions.

$$\beta = \frac{1 - (1 - 0.5) * iPVP}{\max(iPVP)} \tag{2}$$

- β: Contraction–expansion coefficient.
- *iPVP*: Current iteration.

THIS FOLLOWS ALL THEORY IN PREVIOS NOTES

QPSO risks local stagnation due to imbalance between exploration and exploitation, while QGSA suffers from premature convergence. To overcome these issues, we propose QIGPSO,

QIGPSO

Combining exploration and exploitation in quantum search space The hybrid QIGPSO method combines QPSO's social exploration and QGSA's local search20. We combine these two algorithms to strike a balance between exploration and exploitation in a quantum search space, a key optimization trait.

$$x_i(z+1) = \left\{ \begin{array}{l} p + \alpha * |MBest_i - x_i(z)| * w_i(z)\, if s \geq 0.5 \\ p - \alpha * |MBest_i - x_i(z)| * w\_i(z) if s < 0.5 \end{array} \right\} \tag{11}$$

α represents the contraction–expansion coefficient, a parameter found in both QPSO and QGSA. In the QIGPSO algorithm, α is systematically reduced from 1 to 0.5 in a monotonic manner. This decrease in α is gradual and continuous.

$$\alpha = (1 - 0.5) * \left(\frac{maxiter - z}{maxiter + 0.5} \right) \tag{12}$$

where, z refers to the present iteration and maxiter refers to the maximum iterations.

The convergence can be accomplished if each particle converges to its local attractor p. Hence Eq. 2 is modified as

$$p = |gp| * Perbest_i + |GP| * Glbest \tag{13}$$

$|gp|$ and $|GP|$ represent absolute random numbers drawn from a Gaussian Probability Distribution (GPD)[21]. In this distribution, the mean is set to 0, and the variance is 1.

$$MBest = \frac{1}{N}\sum_{d=1}^{N} Glbest(z)$$

$$\omega_i(z) = \left(\frac{1}{rand_i}\right) * \omega_{i(z)} + accel_i(z)$$

The acceleration of the ith particle is computed as

$$accel_i(z) = \sum_{g \in kbest} Gravit(z) * \frac{M_i(z)}{dist_i(z)} * (Perbest_i(z) - x_i(z))$$

where $Dist_i(z)$ is the Hamming distance between the particle i and the best fitness particle Glbest. It is computed as

$$dist_j(z) = |Glbest(z) - x_j(z)| \tag{17}$$

The gravitational constant Gravit(z) is the decreasing coefficient over time z that is initially assigned to $Gravit_0$, the initial gravitational constant i.e. 1, and will be decreased towards zero over time z. It is computed as

$$Gravit(z) = Gravit_o * exp\left(-\alpha * \frac{i}{maxiter}\right) \tag{18}$$

Now as in QGSA, the gravitational mass $m_i(z)$ and the inertial mass $M_i(z)$ of the particles are calculated as

$$m_i(z) = \frac{fit_i z - worst(z)}{best(z) - worst(z)} \tag{19}$$

$$M_i(z) = \frac{m_i(z)}{\sum_{j=1}^{N} m_j(z)} \tag{20}$$

where, $fit_i(z)$ represents ith particle's fitness value at time z, best*f*(z) specifies best fitness value. It is the minimum fitness of particle j at time z and *worst*(z) is the worst fitness value specified as maximum fitness of particle j at time z. The best*f* (z) and worst*f* (z) is specified as (with respect to minimization problem)

$$bestf(z) = min_{j \in (1 \ldots N)} fit_j(z) \tag{21}$$

$$worstf(z) = max_{j \in (1 \ldots N)} fit_j(z) \tag{22}$$

Function QIGPSO (N, D, z, maxiter)

```
Initialize the population X using InitializePopulation(N)
Set Glbest to the best particle in X
Initialize other required parameters
global_best_fitness = Negative Infinity
for iteration in 1 to maxiter:
        for each particle in X:
                particle = UpdateParticle(particle, z)
                fitness = ComputeFitness(particle, z)
                bestf, worstf = ComputeBestAndWorst(z, X)
                Mbest = ComputeMbest(particle, z)
                ω = ComputeOmega(z)
                accel = ComputeAccel(z)
                dist = ComputeDistance(particle, Glbest, z)
                Gravit_0 = ComputeGravitationalForce(z)
                m_i, M_i = ComputeM(z)

                fitness = ComputeFitness(particle, z)
                mean_acc = MeanAccuracyUsingSVM(particle)
                fitness = ComputeFitness(particle, z)
                if fitness > particle.best_fitness:
                        particle.best_fitness = fitness
                        particle.Perbest = particle.position.copy()
                        if fitness > global_best_fitness:
                                global_best_fitness = fitness
                                Glbest = particle.position.copy()
Return Glbest
```

Whats the fitness function?

Well the fitness function for the case we are looking at (feature selection) would be the performance of a classification model on the features we had seleced

But this would make there have no penalty for chooosing a lot of features and increasing the dimensionality of our data so for that we impose a penalty for the no of features

5️⃣ Overall Structure

Let's group intuitively:

$$\text{fitness} = \underbrace{\alpha \log(\text{accuracy})}_{\text{reward for high accuracy}} - \underbrace{(1-\alpha)\left[\text{feature penalty} \times \text{accuracy} \times \text{golden ratio perturbation}\right]}_{\text{penalize too many features}}$$

🧠 Intuitive Explanation

Term	Meaning
$\alpha \log(\text{mean}_{acc})$	Rewards models with high classification accuracy
$(1-\alpha)\frac{n_{feat}}{\sqrt{N}}$	Penalizes models that use too many features
$(1+\sin(GR))$	Adds a small oscillation to encourage exploration
Overall goal	Find models that are accurate, simple (few features), and avoid premature convergence

---

Function InitializePopulation(N)

Create an empty list population

for i from 1 to N:

Generate a random particle X_i

Append X_i to population

Return the population

Function ComputeFitness(particle, z)

Compute fitness using Eq. (6.45)

Return fitness

Function ComputeBestAndWorst(z, population)

Compute bestf(z) and worstf(z) using Eq. (6.43) and Eq. (6.44)

Return bestf(z), worstf(z)

Function ComputeMbest(particle, z)

Compute $Mbest_i$ using Eq. (6.36)

Return $Mbest_i$

Function ComputeOmega(z)

Compute $\omega_i(z)$ using Eq. (6.37)

Return $\omega_i(z)$

Function ComputeAccel(z)

Compute $accel_i(z)$ using Eq. (6.38)

Return $accel_i(z)$

Function ComputeDistance(particle1, particle2, z)

Compute $dist_i(z)$ using Eq. (6.39)

Function ComputeDistance(particle1, particle2, z)

Compute $dist_i(z)$ using Eq. (6.39)

Return $dist_i(z)$

Function ComputeGravitationalForce(z)

Compute $Gravit_0$ using Eq. (6.40)

Return $Gravit_0$

Function ComputeM(z)

Compute mi(z) and Mi(z) using Eq. (6.41) and Eq. (6.42)

Return $m_i(z)$, $M_i(z)$

Function MeanAccuracyUsingSVM(particle)

Compute mean_acc using SVM for the given particle

Return mean_acc

Function UpdateParticle(particle, z)

Update the particle's position and velocity using Eq. (6.33) and Eq. (6.35)

Return updated particle

a