Approche Hybride BPSO-GSA pour le Problème du Sac à Dos Multiple

Selma Bettaieb

2 décembre 2024

1 Introduction à l'Approche Hybride

1.1 Motivation

L'hybridation de BPSO et GSA vise à combiner :
- La capacité d'exploration globale de GSA
- La convergence efficace de BPSO
- Les mécanismes de recherche complémentaires des deux algorithmes

2 Fondements Théoriques

2.1 BPSO (Binary Particle Swarm Optimization)

Les équations fondamentales de BPSO :

$$v_{id}^{t+1} = w \cdot v_{id}^{t} + c_1 r_1 (p_{id}^{t} - x_{id}^{t}) + c_2 r_2 (g_d^{t} - x_{id}^{t})$$

$$S(v_{id}^{t+1}) = \frac{1}{1 + e^{-v_{id}^{t+1}}}$$

$$x_{id}^{t+1} = \begin{cases} 1 & \text{si } rand() < S(v_{id}^{t+1}) \\ 0 & \text{sinon} \end{cases}$$

où :
- v_{id}^{t} : vitesse de la particule i dans la dimension d à l'itération t
- w : coefficient d'inertie
- c_1, c_2 : coefficients d'accélération
- r_1, r_2 : nombres aléatoires uniformes dans [0,1]
- p_{id}^{t} : meilleure position personnelle
- g_d^{t} : meilleure position globale

2.2 GSA (Gravitational Search Algorithm)

Les équations principales de GSA :

$$
\begin{aligned}
F_{ij}^d(t) &= G(t)\frac{M_i(t) \times M_j(t)}{R_{ij}(t) + \epsilon}(x_j^d(t) - x_i^d(t)) \\
a_i^d(t) &= \frac{F_i^d(t)}{M_{ii}(t)} \\
v_i^d(t+1) &= rand_i \times v_i^d(t) + a_i^d(t)
\end{aligned}
$$

où :
- $G(t)$: constante gravitationnelle
- $M_i(t)$: masse de l'agent i
- $R_{ij}(t)$: distance euclidienne entre les agents i et j
- ϵ : petite constante

3 Implémentation de l'Hybridation

3.1 Structure Principale

Algorithm 1 Algorithme Hybride BPSO-GSA

Initialiser population P avec solutions binaires aléatoires
while critère d'arrêt non atteint **do**
    Évaluer fitness de chaque solution
    Mettre à jour meilleure solution globale g_{best}
    Calculer masses et forces (GSA)
    Calculer accélérations (GSA)
    for chaque particule i **do**
        Calculer vitesse BPSO (v_{BPSO})
        Calculer vitesse GSA (v_{GSA})
        $v_{hybrid} \leftarrow \alpha \cdot v_{BPSO} + (1-\alpha) \cdot v_{GSA}$
        Appliquer fonction de transfert sigmoïde
        Mettre à jour position
        Appliquer réparation si nécessaire
    end for
    Mettre à jour α selon la performance
end while

3.2 Innovations Clés

3.2.1 Coefficient d'Hybridation Adaptatif

$$
\begin{aligned}
\alpha(t) &= \alpha_{min} + (\alpha_{max} - \alpha_{min}) \cdot e^{-\rho t} \\
\rho &= -\ln(\frac{fitness_{BPSO}}{fitness_{GSA}})
\end{aligned}
$$

3.2.2 Fonction de Transfert Modifiée

$$S(x) = \frac{1}{1 + e^{-\lambda x}}$$

où λ est ajusté dynamiquement selon la performance.

3.3 Mécanisme de Réparation

Algorithm 2 Réparation MKP

```
function RepairSolution(solution, weights, capacities)
    while solution non réalisable do
        ratio ← calculerRatioProfit(solution)
        item ← trouverPireRatio(ratio)
        solution[item] ← 0
    end while
    while possibilité d'amélioration do
        item ← trouverMeilleurCandidat(solution)
        if ajoutPossible(item) then
            solution[item] ← 1
        end if
    end while
    return solution
end function
```

4 Paramètres et Configuration

4.1 Paramètres Critiques

- Taille de la population : 30
- Nombre maximal d'itérations : 1000
- $\alpha_{min} = 0.2$, $\alpha_{max} = 0.8$
- $c_1 = c_2 = 2.0$ (BPSO)
- $G_0 = 100$ (GSA)

4.2 Adaptation pour MKP

- Fonction objectif adaptée :

$$f(X) = \sum_{j=1}^{n} p_j x_j - \beta \sum_{i=1}^{m} \max(0, \sum_{j=1}^{n} w_{ij} x_j - c_i)$$

  où β est le coefficient de pénalité
- Réparation spécifique au MKP
- Gestion des contraintes par pénalisation adaptative

5 Avantages de l'Approche

5.1 Performance

— Convergence plus rapide que BPSO ou GSA seuls
— Meilleure exploration de l'espace de recherche
— Équilibre optimal entre diversification et intensification

5.2 Résultats Expérimentaux

— Écarts-types réduits (stabilité accrue)
— Solutions optimales trouvées plus fréquemment
— Performance supérieure sur instances de petite et moyenne taille

5.3 Comparaison avec Autres Méthodes

— Plus stable que BGSA
— Plus précis que BPSO standard
— Meilleur compromis performance-stabilité

6 Code d'Implémentation

7 Code d'Implémentation

```python
class HybridBPSOGSA:
    def __init__(self, n_particles, max_iter, problem_size):
        self.n_particles = n_particles
        self.max_iter = max_iter
        self.problem_size = problem_size
        # Initialize velocities and positions
        self.velocities = np.zeros((n_particles, problem_size))
        self.positions = np.random.randint(2,
            size=(n_particles, problem_size))
```

Listing 1 – Initialisation de la classe

```python
def update_velocity(self, particle_idx):
    # BPSO component
    v_bpso = (self.w * self.velocities[particle_idx] +
              c1 * r1 * (self.p_best[particle_idx] -
                         self.positions[particle_idx]) +
              c2 * r2 * (self.g_best -
                         self.positions[particle_idx]))

    # GSA component
    v_gsa = self.calculate_gsa_velocity(particle_idx)

    # Hybrid velocity
    alpha = self.calculate_adaptive_alpha()
```

```python
    v_hybrid = alpha * v_bpso + (1 - alpha) * v_gsa

    return v_hybrid
```

Listing 2 – Mise à jour de la vitesse

```python
def update_position(self, velocity):
    s = 1 / (1 + np.exp(-self.lambda_param * velocity))
    return np.where(np.random.random(velocity.shape) < s, 1, 0)
```

Listing 3 – Mise à jour de la position

```python
def repair_solution(self, position):
    while not self.is_feasible(position):
        worst_item = self.find_worst_item(position)
        position[worst_item] = 0
    return position
```

Listing 4 – Réparation de solution

```python
def optimize(self):
    for iteration in range(self.max_iter):
        self.update_all_particles()
        self.update_global_best()
        self.update_parameters(iteration)
```

Listing 5 – Fonction d'optimisation principale