Optimisation par essaim de particules
Application au problème des n-Reines

Antoine Dutot et Damien Olivier

Laboratoire Informatique du Havre
Université du Havre
25 rue Philippe Lebon
76600 Le Havre

Antoine.Dutot@univ-lehavre.fr
Damien.Olivier@univ-lehavre.fr

Résumé

L'objectif de ce document est de présenter une approche d'intelligence collective permettant de résoudre des problèmes d'optimisation et d'appliquer cela en particulier à un problème de permutation. L'approche retenue est l'optimisation par essaim de particules dont l'idée directrice est la simulation du comportement collectif des oiseaux à l'intérieur d'une nuée.

1 Optimisation par essaim de particules - Particle Swarm Optimization (PSO) -

Observez un champ entrain d'être labouré en automne, lorsque le soc de la charrue pénètre le sol pour la première fois le champ est vide de tout goéland et quelques minutes après une nuée accompagne le tracteur. Au début du labour un oiseau découvre la source de nourriture et très rapidement un autre arrive et ainsi de suite. Que s'est il passé ? L'information concernant un festin potentiel s'est largement diffusé au sein du groupe de goéland. Les goélands volaient à la recherche de nourriture de façon plus ou moins ordonnée et le rassemblement s'est effectué par un échange (volontaire ou non) social d'informations entre individus de la même espèce. L'un d'entre-eux à trouvé une solution et les autres se sont adaptés en copiant sa solution, ceci offre un caractère adaptatif à la méthode.

Au départ J. Kennedy et R. Eberhart (Kennedy and Eberhart, 1995) cherchaient à simuler la capacité des oiseaux à voler de façon synchrone et leur aptitude à changer brusquement de direction tout en restant en une formation optimale. Le modèle qu'ils ont proposé à ensuite été étendu en un algorithme simple et efficace d'optimisation.

Les particules sont les individus et elles se déplacent dans l'hyperespace de recherche. Le processus de recherche est basé sur deux règles :

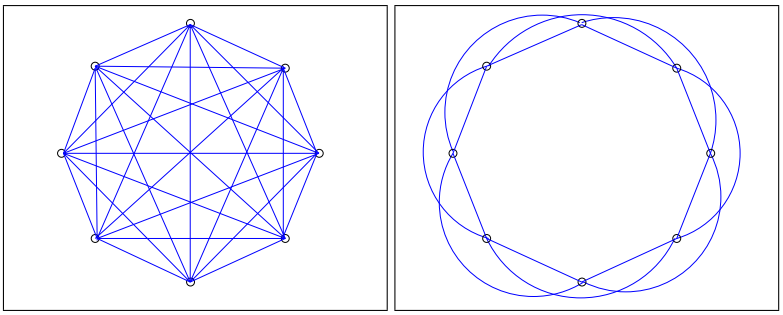
1. Chaque particule est dotée d'une mémoire qui lui permet de mémoriser le meilleur point par lequel elle est déjà passée et elle a tendance à retourner vers ce point.

2. Chaque particule est informée du meilleur point connu au sein de son voisinage et elle va tendre à aller vers ce point.

1.1 Le voisinage

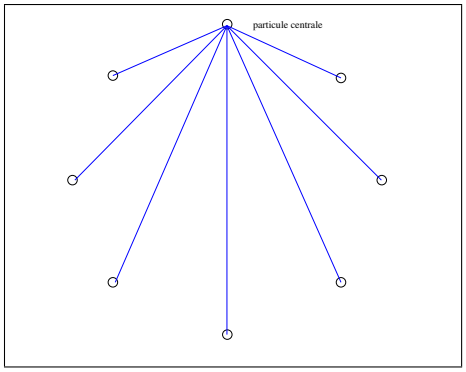
Le voisinage constitue la structure du réseau social. Les particules à l'intérieur d'un voisinage communiquent entre-elles. Différents voisinages ont été étudiés (Kennedy, 1999) et sont considérés en fonction des identificateurs des particules et non des informations topologiques comme les distances euclidiennes dans l'espace de recherche :

- Topologie en étoile (figure 1(a)) : le réseau social est complet, chaque particule est attirée vers la meilleure particule notée `gbest` et communique avec les autres.
- Topologie en anneau (figure 1(b)) : chaque particule communique avec n ($n = 3$ 1(b)) voisines immédiates. Chaque particule tend à se déplacer vers la meilleure dans son voisinage local notée `lbest`.
- Topologie en rayon (figure 1(c)) : une particule "centrale" est connectée à toute les autres. Seule cette particule centrale ajuste sa position vers la meilleure, si cela provoque une amélioration l'information est propagée aux autres.



(a) Etoile

(b) Anneau



(c) Rayon

FIG. 1 – Trois topologies différentes

1.2 Algorithmes

Chaque particule représente une solution potentielle dans l'espace de recherche. La nouvelle position d'une particule est déterminée en fonction de sa propre valeur et celle de ses voisines. Soit $\overrightarrow{x_i}(t)$ la position de la particule P_i au temps t, sa position est modifiée en ajoutant une vitesse $\overrightarrow{v_i}(t)$ à sa positon courante :

$$\overrightarrow{x_i}(t) = \overrightarrow{x_i}(t-1) + \overrightarrow{v_i}(t) \tag{1}$$

C'est le vecteur vitesse qui dirige le processus de recherche et reflète la "sociabilité" des particules. Si l'on considère N particules et que chaque particule compare sa nouvelle position à sa meilleure position obtenue, cela donne l'algorithme 1, $\mathcal{F}$ étant la fonction de fitness.

[Les variables et paramètres de l'algorithme]
N nombre de particules
$\overrightarrow{x_i}$ position de la particule P_i
$\overrightarrow{v_i}$ vitesse de la particule P_i
$pbest_i$ meilleure fitness obtenue pour la particule P_i
$\overrightarrow{x}_{pbest_i}$ position de la particule P_i pour la meilleure fitness
ρ valeur aléatoire positive
[———————————————————————————————-]

[Initialisations]
Initialiser aléatoirement la population
...
[Traitement]
Répéter
  Pour i **de** 1 **à** N **faire**
    Si ($\mathcal{F}(\overrightarrow{x_i}) > pbest_i$) **Alors**
      $pbest_i \leftarrow \mathcal{F}(\overrightarrow{x_i})$
      $\overrightarrow{x}_{pbest_i} \leftarrow \overrightarrow{x_i}$
    Fin Si
    $\overrightarrow{v_i} \leftarrow \overrightarrow{v_i} + \rho(\overrightarrow{x}_{pbest_i} - \overrightarrow{x_i})$
    $\overrightarrow{x_i} \leftarrow \overrightarrow{x_i} + \overrightarrow{v_i}$
  Fin Pour
jusqu'à ce que (le processus converge)

Algorithme 1 – Algorithme de base

Ce premier algorithme ne prend pas en compte le voisinage, puisqu'on utilise uniquement l'amélioration obtenue sur la particule elle-même. En considérant un voisinage en étoile l'algorithme 1 devient 2.

[Les variables et paramètres de l'algorithme]
N nombre de particules
$\overrightarrow{x_i}$ position de la particule P_i
$\overrightarrow{v_i}$ vitesse de la particule P_i
$pbest_i$ meilleure fitness obtenue pour la particule P_i
$\overrightarrow{x}_{pbest_i}$ position de la particule P_i pour la meilleure fitness
$\overrightarrow{x}_{gbest}$ position de la particule ayant la meilleure fitness de toutes
ρ_1, ρ_2 valeurs aléatoires positives
[———————————————————————————————-]

[Initialisations]
Initialiser aléatoirement la population
...

[Traitement]
Répéter
  Pour i **de** 1 **à** N **faire**
    Si ($\mathcal{F}(\overrightarrow{x_i}) > pbest_i$) **Alors**
      $pbest_i \leftarrow \mathcal{F}(\overrightarrow{x_i})$
      $\overrightarrow{x}_{pbest_i} \leftarrow \overrightarrow{x_i}$
    Fin Si
    Si ($\mathcal{F}(\overrightarrow{x_i}(t)) > gbest$) **Alors**
      $gbest \leftarrow \mathcal{F}(\overrightarrow{x_i})$
      $\overrightarrow{x}_{gbest} \leftarrow \overrightarrow{x_i}$
    Fin Si
  Fin Pour
  Pour i **de** 1 **à** N **faire**
    $\overrightarrow{v_i} \leftarrow \overrightarrow{v_i} + \rho_1(\overrightarrow{x}_{pbest_i} - \overrightarrow{x_i}) + \rho_2(\overrightarrow{x}_{gbest} - \overrightarrow{x_i})$
    $\overrightarrow{x_i} \leftarrow \overrightarrow{x_i} + \overrightarrow{v_i}$
  Fin Pour
jusqu'à ce que (le processus converge)

Algorithme 2 – Algorithme avec un voisinage en étoile

Plus une particule est éloignée de la meilleure solution globale et de sa meilleure solution, plus sera importante la variation de sa vitesse afin de faire bouger la particule vers les meilleures solutions. Les variables aléatoires ρ_1 et ρ_2 peuvent être définie de la façon suivante :

$$\begin{cases} \rho_1 = r_1 c_1 \\ \rho_2 = r_2 c_2 \end{cases} \tag{2}$$

r_1 et r_2 suivent une loi uniforme sur $[0..1]$ et c_1 et c_2 sont constantes et représentent une accélération positive, avec $c_1 + c_2 \leq 4$. L'algorithme s'exécute tant qu'un critère de convergence n'a pas été atteint. Cela peut être :

- Un nombre fixe d'itérations ;
- En fonction de la fitness ;
- Lorsque la variation de vitesse est proche de 0.

Cet algorithme peut être facilement modifié pour utiliser un voisinage en anneau par exemple. Les boucles devront se faire sur le voisinage et on déterminera $\overrightarrow{x}_{lbest}$ qui remplacera $\overrightarrow{x}_{gbest}$.

1.3 Paramètres de l'algorithme

Six paramètres rentrent en ligne de compte :

1. La dimension du problème ;
2. Le nombre de particules ;
3. Les valeurs des coefficients ρ ;
4. La taille du voisinage ;
5. La vitesse maximale ;
6. L'inertie.

Nous allons nous intéresser plus particulièrement aux deux derniers.

1.3.1 Vitesse maximale

Pour éviter que les particules se déplacent trop rapidement d'une région à une autre dans l'espace de recherche, on fixe une vitesse maximale V_{max}. Ainsi si $v_{ij}(t)$ est la vitesse de la particule P_i au temps t dans la dimension j,

$$v_{ij}(t) = V_{max} \text{ si } v_{ij}(t) > V_{max} \text{ et } v_{ij}(t) = -V_{max} \text{ si } v_{ij}(t) < -V_{max}$$

V_{max} est généralement dépendant de l'échelle du problème. V_{max} n'est pas obligatoire (Clerc and Kennedy, 2002) si on utilise un coefficient de constriction (resserrement) κ

$$\begin{aligned} &\overrightarrow{v_i}(t) = \kappa(\overrightarrow{v_i}(t-1) + \rho_1(\overrightarrow{x}_{pbest_i} - \overrightarrow{x_i}(t)) + \rho_2(\overrightarrow{x}_{gbest} - \overrightarrow{x_i}(t))) \\ &\text{avec } \kappa = 1 - \tfrac{1}{\rho} + \tfrac{\sqrt{|\rho^2 - 4\rho|}}{2} \text{ et } \rho = \rho_1 + \rho_2 > 4 \end{aligned} \tag{3}$$

1.3.2 Facteur d'inertie

Pour contrôler l'influence de la vitesse obtenue au pas précédent on peut introduire un facteur d'inertie Φ qui décroît en fonction du temps.

$$\overrightarrow{v_i}(t) = \Phi\overrightarrow{v_i}(t-1) + \rho_1(\overrightarrow{x}_{pbest_i} - \overrightarrow{x_i}(t)) + \rho_2(\overrightarrow{x}_{gbest} - \overrightarrow{x_i}(t)) \tag{4}$$

Un grand facteur d'inertie provoque une grande exploration de l'espace de recherche alors qu'un petit facteur d'inertie concentre la recherche sur un petit espace. La convergence n'est pas assurée quelques soient les valeurs de Φ et également de c_1 et c_2 (van den Bergh, 2002), les études expérimentales montrent

$$\frac{1}{2}(c_1 + c_2) - 1 < \Phi \leq 1 \tag{5}$$

1.3.3 Voisinage basé sur une distance euclidienne

Kennedy et Eberhart utilisent un voisinage basé sur les indices des particules, Suganthan (Suganthan, 1999) utilise un voisinage spatial entre les particules. Une particule P_b est voisine d'une particule P_a si :

$$\frac{||\overrightarrow{x}_a - \overrightarrow{x}_b||}{d_{max}} < \xi \tag{6}$$

avec d_{max} la plus grande distance entre deux particules et

$$\xi = \frac{3t + 0.6t_{max}}{t_{max}} \tag{7}$$

avec t l'itération courante et t_max le nombre maximal d'itérations. On peut remarquer que la taille du voisinage croît avec le temps.

1.4 Exemple développé

Considérons par exemple la fonction $f(x, y, z, w) = 5x^4 + 4y^3 - (z/4 * y)^2 + 5$ que l'on cherche à annuler. L'espace de recherche est donc de dimension 4 et la fonction de fitness `calculFitness()` dépend de la fonction $f(x, y, z, w)$ elle-même.

```
/*-------------------------------------------------------------------------------
Variables de contrôle
Les valeurs sont données à titre d'exemple
*/
nombreDeParticules       ←   40
nombreDeVoisins          ←   4
maxItérations            ←   10000

/* Vitesse maximale */
deltaMinVitesse          ←  -4.0
deltaMaxVitesse          ←   4.0

/* Fitness */
fitnessInitiale          ←  -100000
fitnessVoulue            ←   0

/* Dimension de l'espace de recherche */
dimensionEspace          ←   4

/* Coefficient individuel et collectif */
c1                       ←   2
c2                       ←   2
/*-------------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------------
Initialisations
*/
POUR j  ←  1 ↗  nombreDeParticules FAIRE
  particule  ←  essaim[j]
  /* Position et vitesse */
  POUR d  ←  1  ↗  dimensionEspace FAIRE
    particule.nouvellePosition[d]  ←  random(...)
    particule.vitesse[d]  ←  random(deltaMinVitesse, deltaMaxVitesse)
  FPOUR
  particule.meilleureFitness  ←  fitnessInitiale
  /* Voisinage */
  POUR n  ← 1  ↗  nombreDeVoisins FAIRE
    particule.voisin[n]  ←  obtenirVoisin(essaim, j, n)
  FPOUR
FPOUR
/*-------------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------------
PSO
*/
TANTQUE i <= maxItérations FAIRE
  POUR j  ←  1  ↗  nombreDeParticules FAIRE
    particule  ←  essaim[j]
    POUR d  ← 1  ↗  dimensionEspace FAIRE
      particule.positionCourante[d]  ←  particule.nouvellePosition[d]
    FPOUR
    fitness  ←  calculFitness(p)
    SI fitness > particule.meilleureFitness
      ALORS
        particule.meilleureFitness  ←  fitness
POUR d  ←  1  ↗  dimensionEspace FAIRE
  particule.meilleurePosition[d]  ←  particule.positionCourante[d]
FPOUR
    FSI
    SI fitness = fitnessVoulue
      ALORS Retourner particule.meilleurePosition
    FSI
  FPOUR
  POUR j  ← 1  ↗  nombreDeParticules FAIRE
    particule  ←  essaim[j]
    meilleureVoisine  ←  obtenirMeilleureVoisine(essaim, j, nombreDeVoisins)
    POUR d  ← 1  ↗  dimensionEspace FAIRE
      rho_1  ←  c1 * random(0,1)
      rho_2  ←  c2 * random(0,1)


      particule.vitesse[d]  ←
          vitesseBornée(particule.vitesse[d] +
          rho_1 * (particule.meilleurePosition[d] - particule.positionCourante[d]) +
          rho_2 * (meilleureVoisine.meilleurePosition[d] - particule.positionCourante[d]))
      particule.nouvellePosition[d]  ←  particule.positionCourante[d] + particule.vitesse[d]
    FPOUR
  FPOUR
  i  ←  i+1
FTQ
```

2 Le problème des n-Reines

Il s'agit d'utiliser l'algorithme d'optimisation par essaim de particules au problème des n-reines (Hu et al.,). On utilise un vecteur pour représenter l'échiquier (figure 2).

1	4	7	9	2	5	8	6	3

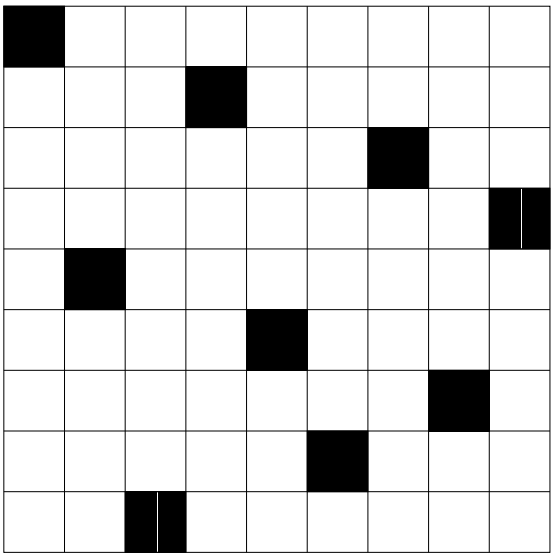


FIG. 2 – Échiquier et sa représentation

Par cette représentation les conflits entre les reines au niveau des lignes et des colonnes sont supprimés, donc pour trouver une solution il faut éliminer les conflits au niveau des diagonales.
La fonction de fitness $\mathcal{F}$ est alors le nombre de conflits au niveau des diagonales, on remarquera que cette approche est très voisine de celle utilisée avec le recuit simulé. Le problème consiste donc à rechercher une permutation dont la fonction de fitness est nulle.

Pour appliquer PSO à un problème de permutation, il faut adapter l'algorithme. En effet les particules codent les solutions dans l'espace de résolution et plusieurs positions peuvent avoir les mêmes valeurs ce qui n'est pas possible dans notre problème de permutation, puisqu'une particule doit être représentée par son vecteur dont chacune des valeurs qui représente une position est différente. La vitesse conditionne la possibilité de changement dans la séquence, donc plus la vitesse est grande plus il est probable qu'une permutation soit effectuée. Chaque composante de la vitesse est donc normalisée (cf. 1).

On détermine ensuite si il doit y avoir permutation ou non en fonction de cette probabilité. La composante k dans l'exemple doit donc être modifiée (probabilité = 1) et la composante $k + 4$ à une forte probabilité. Si un échange doit avoir lieu,

		k	k + 1	k + 2	k + 3	k + 4	k + 5	
$\overrightarrow{v}_i(t)$	...	50	10	5	35	40	25	...
$\lvert\overrightarrow{v}_i(t)\rvert$	...	1	0,2	0,1	0,7	0,8	0,5	...

TAB. 1 – Normalisation de la vitesse

il est effectué de la façon suivante (cf. 2) :

$$x_i^{(k)} \leftrightarrow x_i^{(m)} \text{ avec } x_{lbest}^m = x_i^{(k)}$$

		k	k + 1	k + 2	k + 3	k + 4	k + 5	
$\overrightarrow{x}_{lbest}$	...	7	11	15	30	10	13	...
$\overrightarrow{x}_i(t-1)$	...	30	8	12	1	17	7	...
$\overrightarrow{x}_i(t)$	...	1	8	12	30	17	7	...

TAB. 2 – Permutation de composante

Afin de ne pas rester sur un minimum local ($\overrightarrow{x}_{lbest} = \overrightarrow{x}_i$) qui peut être atteint, on échange aléatoirement deux composantes, dans ce cas.

2.1 Mise en œuvre

Dans un premier temps vous testerez cette approche en utilisant un voisinage en anneau avec $n = 2$, une population de dix particules, une vitesse maximale bornée par le nombre de reines, un facteur d'inertie de $0,5 + \frac{Rand()}{2}$ et $c_1 = c_2 = 1,49445$. Vous pourrez dans un second temps modifier ces paramètres et montrer leurs influences l'obtention de la solution en calculant le nombre de génèrations nécessaires pour obtenir une solution.

Références

Clerc, M. and Kennedy, J. (2002). The Particle Swarm : Explosion, Stability, and Convergence in a Multi-Dimensional Complex Space. In *Proceedings of the IEEE Transactions on Evolutionary Computation*, volume VI, pages 58–73.

Hu, X., Eberhart, R., and Shi, Y. Swarm Intelligence for Permutation Optimization : A Case Study of n-Queens Problem.

Kennedy, J. (1999). Small Worlds and Mega-Minds : Effects of Neighborhood Topology on Particle Swarm Performance. In *IEEE Congress on Evolutionary Computation*, volume III, pages 1932–1938.

Kennedy, J. and Eberhart, R. (1995). Particle Swarm Optimization. In *Proceedings of IEEE International Conference on Neural Network*, volume IV, pages 1942–1948.

Suganthan, P. (1999). Particle Swarm Optimizer with Neighborhood Operator. In *IEEE Congress on Evolutionary Computation*, volume III, pages 1958–1961.

van den Bergh, F. (2002). *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria.