Les oiseaux picorant artificiels : une nouvelle méta-heuristique inspirée du comportement des pigeons

Jean-Baptiste Lamy

Les oiseaux picorant artificiels : une nouvelle méta-heuristique inspirée du comportement des pigeons

Jean-Baptiste Lamy[1]

[1] LIMICS, Université Paris 13, Sorbonne Paris Cité, 93017 Bobigny France, INSERM UMRS 1142, UPMC Université Paris 6, Sorbonne Universités, Paris, France
jean-baptiste.lamy@univ-paris13.fr

Résumé

De nombreux algorithmes et méta-heuristiques pour l'optimisation s'inspirent de la nature, par exemple du comportement social des abeilles pour l'algorithme ABC (Artificial Bee Colony). Ces algorithmes permettent souvent de résoudre des problèmes d'optimisation de nature très diverse. Dans cet article, nous proposons une nouvelle méta-heuristique, les oiseaux picorant artificiels (OPA), inspirée du comportement des pigeons lorsqu'ils sont à la recherche de nourriture. Cet algorithme est très simple, il donne de bons résultats et peut facilement s'adapter à des problèmes d'optimisation divers. Nous l'appliquons à l'optimisation globale non linéaire (y compris l'apprentissage de réseaux de neurones artificiels) et à la résolution du problème du voyageur de commerce, et nous montrons que OPA donne des résultats équivalents ou meilleurs que ABC sur plusieurs fonctions de test. Enfin, nous discuterons des différences entre les deux algorithmes et du choix des sources d'inspiration issues de la nature.

Abstract

Many optimization algorithms and metaheuristics have been inspired by nature, *e.g.* by the social behavior of honey bees for Artificial Bee Colony (ABC). These algorithms often permit solving a wide range of optimization problems. In this paper, we present Artificial Feeding Birds (AFB), a new metaheuristic inspired by the behavior of pigeons searching for food. AFB is very simple, yet efficient, and easy to adapt to various optimization problems. We apply it to unconstrained global nonlinear optimization (including the training of artificial neural networks) and to the resolution of traveling salesman problem. We also show that AFB gives results equivalent or better than ABC on several benchmark functions. Finally, we discuss the differences between the two algorithms and the choice of inspiration sources from nature.

1 Introduction

De nombreux algorithmes s'inspirent de la nature. Deux exemples bien connus sont les réseaux de neurones artificiels [1] et les algorithmes génétiques [5]. Plus récemment, les chercheurs se sont inspirés du comportement des animaux pour inventer de nouveaux algorithmes : l'organisation sociale des abeilles et des fourmis, la communication lumineuse chez les lucioles, la cohésion au sein d'un essaim d'oiseaux en vol ont chacun donné lieu à des algorithmes pour résoudre des problèmes d'optimisation tels que la recherche du minimum global d'une fonction numérique (optimisation globale non linéaire) ou la résolution du problème du voyageur de commerce. Ces algorithmes s'appuient généralement sur l'intelligence distribuée (*Swarm intelligence*), c'est-à-dire qu'ils font intervenir une population d'agents qui interagissent entre eux et avec leur environnement [8]. Ces agents sont très simples mais permettent d'accomplir des tâches complexes, et notamment de résoudre des problèmes d'optimisation [2]. Ces algorithmes sont appelés *méta-heuristiques* lorsqu'ils proposent une stratégie générale de « haut-niveau » qui peut être utilisée pour guider une recherche locale heuristique de plus « bas niveau » [16]. Par conséquent, une méta-heuristique n'est pas spécifique à un type donné de problème mais permet au contraire de résoudre des problèmes très différents, en fonction de la recherche de « bas niveau » utilisée.

Dans cet article, nous proposons les oiseaux picorant artificiels (OPA), une nouvelle méta-heuristique inspirée du comportement des pigeons lorsqu'ils sont à la recherche de nourriture. Ces oiseaux sont très communs et, en Europe, tout le monde a eu l'occasion de les observer en quête de nourriture sur les trottoirs. Notre hypothèse est la suivante : si le pigeon est aussi largement répandu, c'est que sa technique de recherche de nourriture est efficace et

donc qu'elle ferait une bonne source d'inspiration pour un algorithme. La méta-heuristique OPA résultante présente plusieurs avantages : elle est très simple, elle donne de bons résultats et elle peut facilement s'adapter à des problèmes d'optimisation divers. Nous montrerons l'adaptabilité d'OPA en l'appliquant à l'optimisation globale non linéaire, y compris l'entraînement de réseaux de neurones, et au problème du voyageur de commerce.

La suite de l'article est organisée de la manière suivante. La section 2 présente un état de l'art sur les algorithmes d'optimisation inspirés de la nature. La section 3 décrit le comportement que nous avons observé chez les pigeons en quête de nourriture. La section 4 décrit la méta-heuristique et son adaptation à l'optimisation globale non linéaire et à la résolution du problème du voyageur de commerce. La section 5 présente plusieurs expérimentations visant à fixer les valeurs des paramètres et à tester l'algorithme OPA. Enfin, la section 6 discute les principaux résultats, les différences entre OPA et ABC, le choix des sources d'inspiration issues de la nature, et propose des perspectives.

2 État de l'art

De nombreux algorithmes d'optimisation se sont inspirés de la nature [19]. La plupart s'inspirent des comportements sociaux des insectes [3] et de leurs capacités à communiquer entre eux par le biais de substances chimiques, de mouvements ou de signaux lumineux. Un premier exemple est le comportement des fourmis [6]. Les fourmis explorent leur environnement à la recherche de nourriture et laissent des traces sur leur passage sous forme de phéromones. Ces phéromones sont ensuite utilisées par d'autres fourmis comme des signaux leur indiquant les directions à suivre ou ne pas suivre. Le comportement des lucioles et leur utilisation de l'émission de lumière pour attirer leurs partenaires ont aussi donné lieu à un algorithme [18], qui a par la suite été appliqué à l'entraînement de réseaux de neurones artificiels [4]. L'optimisation par essaims particulaires (*particle swarm optimization*, PSO) s'inspire du comportement des animaux évoluant en formation (essaims d'insectes, vols d'oiseaux, bancs de poissons) [14]. Le déplacement de chaque individu de l'essaim dépend de la position des autres individus et de la qualité des solutions associées aux positions des éléments de l'essaim.

Le comportement des abeilles a également fait l'objet d'un algorithme qui repose sur le comportement des abeilles à la recherche de nectar et la communication entre abeilles par le biais de « danses » effectuées en vol [12]. Ces danses permettent à une abeille qui a trouvé un endroit riche en nectar de « recruter » d'autres abeilles afin de les conduire à cet endroit. L'algorithme ABC (*Artificial Bee Colony*) proposé par D Karaboga [12] considère 3 types d'abeilles : les ouvrières, les spectatrices et les scoutes. Chaque ouvrière est associée à une source de nectar, qui correspond à une solution du problème d'optimisation. Les meilleures solutions correspondent à des sources de nectar plus riches. À chaque itération, l'ouvrière essaie d'améliorer sa solution en essayant une solution voisine, puis elle communique aux spectatrices la qualité et la position de sa solution. Les abeilles spectatrices obtiennent les informations des ouvrières ; à chaque itération, chaque spectatrice choisit d'aider une ouvrière, le choix étant aléatoire mais avec une probabilité plus grande de choisir les ouvrières qui ont des solutions plus intéressantes. La spectatrice essaie alors d'améliorer la solution de l'ouvrière, de la même manière que celle-ci. Lorsqu'une solution n'a pas pu être améliorée après un nombre défini de tentatives, celle-ci est abandonnée et l'abeille scoute est chargée de la remplacer par une solution aléatoire. L'algorithme ABC a été adapté à l'optimisation globale non linéaire [12], à l'optimisation de problèmes contraints [9], à l'entraînement de réseaux de neurones artificiels [10] et au *clustering* [11].

Plus récemment, plusieurs algorithmes se sont inspirés du comportement des oiseaux. XS Yang *et al.* se sont inspirés du comportement parasitique du coucou pour l'optimisation de fonctions numériques [17]. H Duan *et al.* ont proposé un algorithme inspiré du vol des pigeons et de leurs capacités à s'orienter dans l'espace en fonction de la position du soleil et à déterminer l'orientation du pôle Nord [7]. Cet algorithme a permis d'optimiser le parcours d'un robot volant.

3 Description du comportement des pigeons

Le pigeon est un oiseau très commun dans les villes européennes et facile à observer. Il se nourrit en picorant des graines ou des miettes de nourriture présentes sur le sol. Lorsqu'il n'a pas de nourriture à sa portée immédiate, le pigeon explore son environnement à la recherche de nourriture, en utilisant les deux modes de déplacement à sa disposition : la marche et le vol.

Nous avons observé que le pigeon accomplit quatre types de déplacement différents afin de chercher sa nourriture (figure 1) : (1) marcher vers une position proche de sa position actuelle, (2) s'envoler et se poser à une nouvelle position choisie de manière arbitraire ou aléatoire par le pigeon, (3) s'envoler et se poser à côté d'un autre pigeon, et (4) s'envoler et retourner à une position mémorisée riche en nourriture (nous avons constaté que les pigeons tendent à visiter préférentiellement certains endroits, comme les aires de pique-nique, et nous ferons l'hypothèse qu'ils gardent en mémoire ce genre d'emplacement). Typiquement, le pigeon marche à la recherche de nourriture (déplacement 1 effectué de manière répétitive). Au bout d'un certain temps, s'il ne trouve rien, il s'envole et se pose soit à un endroit aléatoire (déplacement 2), soit à côté d'un autre pigeon (déplacement 3), soit vers une position mémorisée (déplacement 4). Puis il recommence à marcher (retour au

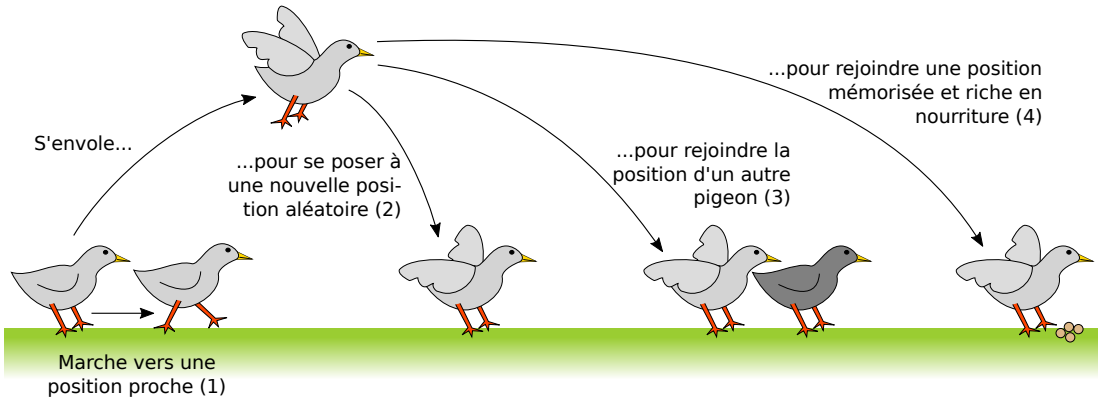

FIGURE 1 – Les quatre types de déplacement observés chez un pigeon lorsqu'il recherche de la nourriture.

déplacement 1), et ainsi de suite.

Ce comportement simple permet d'optimiser la recherche de nourriture. Le déplacement 1 (marche) permet une recherche locale. Ce comportement est censé car une position avec de la nourriture à de fortes chances d'être placée à proximité d'une autre position avec de la nourriture : par exemple si une miette de sandwich est présente à une position x, il est fort probable que d'autres miettes du même sandwich soient présentes dans des positions voisines ou proches x', x'', *etc*. Le déplacement 2 (nouvelle position aléatoire) permet l'exploration aléatoire. Le déplacement 3 (rejoindre un autre pigeon) permet de profiter de la nourriture qu'aurait éventuellement trouvée le pigeon rejoint. Cette action peut conduire à former de grands groupes de pigeons lorsqu'une quantité importante de nourriture est disponible à un endroit donné. Le déplacement 4 (retour à une position mémorisé) permet de retrouver de la nourriture, ou de continuer à en chercher aux alentours.

Nos observations ont porté sur les pigeons, cependant de nombreux autres oiseaux (moineaux par exemple) semblent adopter un comportement similaire.

4 Traduction en algorithmes

4.1 Méta-heuristique

Le comportement du pigeon optimise la recherche de nourriture. En nous inspirant de ce comportement, nous pouvons aboutir à une méta-heuristique pour minimiser une fonction de coût. Nous considérerons un système multi-agents où chaque agent est un « oiseau artificiel ». Chaque oiseau possède une position qui est une solution candidate pour la fonction de coût ; il dispose également d'une mémoire qui lui permet de retenir la meilleure position qu'il a trouvée. La position initiale de l'oiseau est aléatoire. Lorsque la position actuelle de l'oiseau minimise autant ou davantage la fonction de coût que la meilleure position qu'il a en mémoire, la position actuelle devient la meilleure position et nous considérons que l'oiseau « a trouvé de la nourriture » et « a mangé ».

La méta-heuristique est un processus cyclique. À chaque itération, l'oiseau artificiel effectue l'un des quatre déplacements décrits à la section précédente : (1) il « marche », c'est-à-dire qu'il modifie légèrement sa position actuelle, (2) il « s'envole » vers une position aléatoire, c'est-à-dire qu'il réinitialise sa position actuelle avec des valeurs aléatoires, (3) il « s'envole » et rejoint un autre oiseau, c'est-à-dire qu'il choisit un oiseau j au hasard et réinitialise sa position actuelle avec la position actuelle de j, ou bien (4) il « s'envole » pour retourner à la meilleure position qu'il a mémorisée. Le choix du déplacement se fait de manière suivante : si l'oiseau vient de se poser au sol, il marche. Si l'oiseau vient de « manger », il marche. Sinon, le choix du déplacement est aléatoire, avec des probabilités spécifiques à chaque type de déplacement.

Pour définir l'algorithme général, nous considérons un problème d'optimisation défini par une fonction de coût dont les solutions appartiennent à l'ensemble admissible A. La recherche de nourriture par les pigeons telle que décrite en section 3 peut être considérée comme la recherche d'une solution optimale dans un espace à deux dimensions puisque les graines sont posées au sol ; dans ce cas, $A = \mathbb{R}^2$. Deux des quatre déplacements (3 et 4) sont génériques et indépendants du problème d'optimisation. Au contraire les deux autres (1 et 2) dépendent du problème. Nous définirons donc le problème d'optimisation à résoudre par un triplet de trois fonctions $(coût, vol, marche)$ définies comme suit :

- $coût : A \to \mathbb{R}$, la fonction de coût à minimiser
- $vol : \emptyset \to A$, une fonction sans argument générant une position (ou solution) aléatoire
- $marche : \mathbb{N} \to A$, une fonction générant une nouvelle position aléatoire proche de la position de l'oiseau dont l'index est passé en paramètre

La méta-heuristique des oiseaux picorant artificiels (OPA)

que nous proposons prend 4 paramètres :

— n, le nombre d'oiseaux artificiels
— p_2, la probabilité qu'un oiseau choisisse le déplacement 2
— p_3, la probabilité qu'un oiseau choisisse le déplacement 3
— p_4, la probabilité qu'un oiseau choisisse le déplacement 4

La probabilité de choisir le déplacement 1 (marche) est donc $p_1 = 1 - p_2 - p_3 - p_4$.

L'algorithme définit les 5 variables suivantes pour chaque oiseau ($1 \leq i \leq n$) :

— $x_i \in A$ est la position actuelle de l'oiseau i
— $f_i \in \mathbb{R}$ est la valeur de la fonction de coût pour la position actuelle de l'oiseau i
— $X_i \in A$ est la meilleure position trouvée et gardée en mémoire par l'oiseau i
— $F_i \in \mathbb{R}$ est la valeur de la fonction de coût pour la meilleure position de l'oiseau i
— $d_i \in \{1, 2, 3, 4\}$ est le type de déplacement effectué par l'oiseau à l'itération précédente (1 marche, 2 vol aléatoire, 3 vol vers un autre oiseau, 4 vol vers la position mémorisée)

L'algorithme 1 permet d'initialiser les variables, d'effectuer les itérations et de déterminer la meilleure solution trouvée en fin du processus. Lors de l'initialisation, la position x_i de chaque oiseau est définie de manière aléatoire avec la fonction *vol*(), le coût actuel f_i est calculé et la variable d_i est initialisée à 2 (car la position initiale correspond à celle à l'issue du déplacement 2).

À chaque itération, pour chaque oiseau i, l'algorithme choisit l'un des quatre déplacements possibles, effectue le déplacement choisi, met à jour la variable d_i avec le type de déplacement effectué, et met à jour la meilleure position mémorisée si nécessaire. Le choix du déplacement effectué se fait de la manière suivante : si le déplacement précédent était un vol (déplacement 2, 3 ou 4, c'est-à-dire $d_i \in \{2, 3, 4\}$) ou si le coût actuel f_i est égal au meilleur coût mémorisé F_i, l'oiseau fera le déplacement 1 (marche), sinon, l'oiseau choisira aléatoirement le déplacement 1, 2, 3 ou 4 avec les probabilités p_1, p_2, p_3 et p_4, respectivement.

Si le déplacement 1 est choisi, la position de l'oiseau x_i est modifiée avec la fonction *marche*() et le coût f_i est recalculé en appelant la fonction *coût*(). Si le déplacement 2 est choisi, la position de l'oiseau x_i est modifiée avec la fonction *vol*() et le coût f_i est recalculé en appelant la fonction *coût*(). Si le déplacement 3 est choisi, la position de l'oiseau x_i et son coût actuel f_i sont remplacés par les valeurs x_j et f_j d'un autre oiseau choisi au hasard. Si le déplacement 4 est choisi, la position de l'oiseau x_i et son coût actuel f_i sont remplacés par les valeurs X_i et F_i mémorisées précédemment. Les deux derniers mouvements ne testent pas une nouvelle solution, donc aucun appel n'est fait à la fonction de coût.

Algorithme 1 Algorithme des oiseaux picorant artificiels (OPA) en pseudo-code.

Pour $1 \leq i \leq n$:
  $x_i = X_i = vol()$
  $f_i = F_i = coût(x_i)$
  $d_i = 2$

Répéter :
  Pour $1 \leq i \leq n$:
    Si $d_i \in \{2, 3, 4\}$ ou $f_i = F_i$:
      $p =$ nombre réel aléatoire entre 0 et 1

    Si $p > p_2 + p_3 + p_4$:
      $d_i = 1$
      $x_i = marche(i)$
      $f_i = coût(x_i)$
    Sinon, si $p < p_2$:
      $d_i = 2$
      $x_i = vol()$
      $f_i = coût(x_i)$
    Sinon, si $p < p_2 + p_3$:
      $d_i = 3$
      $j =$ nombre entier aléatoire entre 1 et n
      $x_i = x_j$
      $f_i = f_j$
    Sinon :
      $d_i = 4$
      $x_i = X_i$
      $f_i = F_i$

    Si $f_i \leq F_i$:
      $X_i = x_i$
      $F_i = f_i$

  Vérifier la condition d'arrêt

La meilleure solution trouvée est X_k, avec $1 \leq k \leq n$ tel que $F_k = min(F_i)$ pour $1 \leq i \leq n$

Si le nouveau coût actuel f_i est inférieur ou égal au meilleur coût mémorisé F_i, alors la position et le coût actuels sont mémorisés (la condition « inférieur **ou égal** » permet de modifier la valeur mémorisée afin de garder une solution qui n'est pas meilleure que la précédente, mais qui est différente ; cela permet d'augmenter la « diversité » des solutions et de tester d'autres solutions).

Enfin, il est nécessaire d'ajouter une condition d'arrêt à l'algorithme. Nous suggérons de limiter l'algorithme à un nombre prédéfini de solutions testées (c'est-à-dire d'appels à la fonction *coût*()). Comme notre méta-heuristique effectue un nombre variable d'appels à la fonction *coût*() par cycle, cette condition d'arrêt facilite la comparaison entre différents algorithmes ou entre différents résultats obtenus avec un même algorithme pour des paramètres différents.

Algorithme 2 Fonctions *vol*() et *marche*() pour résoudre des problèmes d'optimisation dans $\mathbb{R}^d$.

Fonction *vol*() :
  $x' \in \mathbb{R}^d$
  Pour $1 \leq k \leq d$:
    x'_k = nombre réel aléatoire entre x_{min} et x_{max}
  Retourne x'

Fonction *marche*(i) :
  $x' \in \mathbb{R}^d$, $x'_k = x_{ik}$ for $1 \leq k \leq d$
  j = nombre entier aléatoire entre 1 et n, $j \neq i$
  k = nombre entier aléatoire entre 1 et d
  r = nombre réel aléatoire entre -1 et 1
  $x'_k = x'_k + r \times \left| x_{ik} - x_{jk} \right|$
  If $x'_k < x_{min}$: $x'_k = x_{min}$
  If $x'_k > x_{max}$: $x'_k = x_{max}$
  Retourne x'

En fin de processus, la meilleure solution correspond à la « meilleure meilleure position » parmi l'ensemble des oiseaux (c'est-à-dire X_k pour k conduisant à la plus petite valeur F_k).

4.2 Adaptation à l'optimisation globale non linéaire non contrainte

Dans cette section, nous appliquerons la méta-heuristique OPA lorsque $A = \mathbb{R}^d$, c'est-à-dire pour des problèmes d'optimisation de fonctions numériques avec d dimensions, dont les solutions sont des séquences de d nombres réels compris entre x_{min} et x_{max}.

L'algorithme 2 décrit les fonctions *vol*() et *marche*() que nous proposons pour ce type d'optimisation. La fonction *vol*() génère simplement une solution aléatoire. La fonction *marche*() modifie l'une des coordonnées k (choisie au hasard) de la position x_i qui lui est passée en argument. La modification se fait avec une amplitude maximale qui est égale à la valeur absolue de la différence des coordonnées x_{ik} et x_{jk}, où x_i est la position actuelle de l'oiseau qui marche et x_j est la position d'un oiseau artificiel choisi aléatoirement. Cette fonction de marche est directement inspirée du déplacement des abeilles dans l'algorithme ABC [12].

4.3 Adaptation au problème du voyageur de commerce

Le problème du voyageur de commerce est un problème d'optimisation bien connu dans lequel un voyageur de commerce doit visiter un ensemble de villes V puis revenir à son point de départ. L'objectif est de trouver l'ordre optimal dans lequel visiter les villes de sorte à minimiser la distance totale parcourue. Dans cette section, nous appliquerons l'algorithme méta-heuristique OPA au problème du voyageur de commerce.

Algorithme 3 Fonctions *vol*() et *marche*() pour résoudre des problèmes d'optimisation combinatoire.

Fonction *vol*() :
  x' = séquence composée des éléments de V dans un ordre aléatoire
  Retourne x'

Fonction *marche*(i) :
  Répéter :
    j = nombre entier aléatoire entre 1 et n, $j \neq i$
    k = nombre entier aléatoire entre 1 et $|V|$
    Δ = position de x_{ik} dans x_j
      − position de $x_{i(k-1)}$ dans x_j
    Si $abs(\Delta) > 1$: Quitter la boucle
  $l = (k + \Delta)$ modulo $|V|$
  Si $k > l$: Échanger les valeurs de k et l
  x' = clone de la séquence x_i
  Inverser l'ordre des éléments entre x'_k et x'_l
  Retourne x'

L'algorithme 3 décrit les fonctions *vol*() et *marche*() que nous proposons pour ce problème. La fonction *vol*() génère simplement une séquence avec un ordre aléatoire. La fonction *marche*() correspond à une variante de l'algorithme de recherche heuristique locale « 2-opt » [13], dans lequel une sous-séquence est choisie de manière aléatoire dans la séquence de ville, et l'ordre des éléments dans cette sous-séquence est inversé. Nous avons modifié l'algorithme « 2-opt » pour prendre en compte la similarité locale avec la position d'un autre oiseau aléatoire. Cette similarité est estimée (de manière grossière) par Δ, le nombre de villes séparant la ville localisée en position k (correspondant à la première extrémité de la sous-séquence) et la ville précédente dans x_i. Δ correspondra ensuite à la longueur de la sous-séquence.

Notons que ces deux fonctions ne sont *a priori* pas spécifiques du problème du voyageur de commerce, mais qu'elles pourraient être utilisées pour d'autres problèmes d'optimisation combinatoire.

5 Expérimentation

5.1 Implémentation

Les algorithmes décrits à la section précédente ont été implémentés en langage Python et exécutés avec l'interpréteur PyPy2 (une version de Python incluant un compilateur *juste-à-temps*, JIT). L'algorithme ABC a également été implémenté pour servir de référence.

5.2 Fonctions de tests

Nous avons utilisé quatre fonctions couramment utilisées pour tester les algorithmes d'optimisation : une fonc-

$$Sphère(x_1, ..., x_n) = \sum_{i=1}^{n} x_i^2, \qquad n = 5 \text{ et } -100 < x_i < 100$$

$$Rosenbrock(x, y) = (1 - x^2) + 100 \times (y - x^2)^2, \qquad -2,048 < x < 2,048 \text{ et } -2,048 < y < 2,048$$

$$Rastrigin(x_1, ..., x_n) = \sum_{i=1}^{n} x_i^2 - 10 \times \cos(2\pi x_i) + 10, \qquad n = 10 \text{ et } -600 < x_i < 600$$

$$Eggholder(x, y) = -(y + 47) \sin\left(\sqrt{\left|y + \tfrac{x}{2} + 47\right|}\right) - x \sin\left(\sqrt{|x - (y + 47)|}\right) + 959,640662720851, \qquad -512 < x, y < 512$$

FIGURE 2 – Les quatre fonctions de test utilisées pour l'expérimentation.

tion de sphère en 5 dimensions, la fonction de Rosenbrock, la fonction de Rastrigin avec 10 dimensions et la fonction d'Eggholder (à laquelle nous avons ajouté une constante, de sorte que son minimum global soit de 0 comme pour les trois autres fonctions, Figure 2).

Pour tester l'entraînement de réseaux de neurones artificiels, nous utiliserons le problème « Xor6 ». Il s'agit d'un réseau comprenant 2 neurones d'entrée E_1 et E_2, 2 neurones intermédiaires I_1 et I_2 et 1 neurone de sortie S. Les neurones n'ont pas de biais, il y a donc 6 coefficients à optimiser en tout : E_1-I_1, E_1-I_2, E_2-I_1, E_2-I_2, I_1-S et I_2-S. Les 4 exemples d'apprentissage (E_1, E_2, S) sont $(0, 0, 0)$, $(0, 1, 1)$, $(1, 0, 1)$, $(1, 1, 0)$ et correspondent à un « ou exclusif » logique. La fonction de coût prend 6 paramètres correspondant aux 6 coefficients du réseau, et retourne l'erreur quadratique moyenne. Nous testerons le réseau Xor6 avec deux fonctions d'activation : la fonction sinus et la fonction sigmoïde, ce qui aboutit à deux fonctions à optimiser, $Xor6_{sin}$ et $Xor6_{sig}$ avec $x_{min} = -100$ et $x_{max} = 100$.

Sphère()	p_3				
p_2	0,0	0,001	0,002	0,003	0,005
0,0	4,13e-73	1,80e-73	1,14e-73	2,13e-73	**3,9e-74**
0,015	1,36e-69	1,40e-69	1,87e-69	1,26e-70	8,30e-71
0,03	1,60e-66	1,03e-66	4,58e-67	2,18e-67	1,69e-67
0,05	6,38e-63	6,16e-62	5,93e-62	1,65e-62	2,59e-64
0,1	1,68e-54	3,90e-55	5,75e-55	4,88e-55	7,71e-56

Rastrigin()	p_3				
p_2	0,0	0,001	0,002	0,003	0,005
0,0	8,26e-17	1,51e-11	3,31e-11	3,41e-06	3,98e-03
0,015	**0**	7,11e-18	1,70e-14	1,07e-09	9,95e-04
0,03	1,24e-17	8,81e-16	8,11e-12	1,51e-07	2,85e-05
0,05	8,45e-15	8,26e-17	7,37e-17	7,64e-16	2,17e-08
0,1	4,38e-14	4,48e-12	1,79e-14	5,30e-12	1,16e-08

TABLE 1 – Résultats obtenus pour l'optimisation de deux fonctions de test pour différentes valeurs de p_2 et p_3, avec 20 oiseaux et une condition d'arrêt de 40 000 solutions testées. Les résultats sont des moyennes obtenues sur 250 exécutions, et les plus petites valeurs sont les meilleures.

5.3 Études des paramètres

Pour étudier et fixer les valeurs des paramètres n, p_2, p_3 et p_4, d'une part nous avons observé les déplacements des agents au fil du temps à l'aide d'une interface graphique présentée en figure 3, et d'autre part nous avons généré des tableaux montrant l'évolution des résultats en fonction de deux paramètres. Le tableau 1 montre l'impact des paramètres p_2 et p_3 sur l'optimisation des fonctions *Sphère*() et *Rastrigin*(). Pour la fonction *Sphère*(), nous constatons que le déplacement 2 (vol aléatoire) impacte négativement les résultats, en revanche le déplacement 3 (vol vers un autre oiseau) améliore les résultats. Minimiser la fonction *Sphère*() peut se faire en « descendant la pente », par conséquent repartir vers une position aléatoire à de fortes chances de provoquer un retour en arrière. Au contraire, pour la fonction *Rastrigin*(), le déplacement 2 semble améliorer les résultats, contrairement au déplacement 3. Dans ce cas, le déplacement 3, s'il est trop fréquent, risque d'amener tous les oiseaux au même endroit, et ceux-ci risquent alors de rester bloqués dans un minimum local.

Nous proposons les valeurs suivantes, déterminées de manière empirique, qui semblent convenir à différents types de problèmes : $n = 20$, $p_2 = 0,03$, $p_3 = 0,003$, $p_4 = 0,6$ (et donc $p_1 = 0,367$). Ces valeurs seront systématiquement utilisées dans la suite du papier.

5.4 Optimisation globale non linéaire et comparaison avec l'algorithme ABC

Nous avons testé la méta-heuristique OPA sur les fonctions de test décrites en section 5.2, et nous avons comparé les résultats avec ceux obtenus en utilisant l'algorithme ABC. Les paramètres de l'algorithme ABC ont été fixés comme suit : $n = 20$ (nombre d'abeilles), *limite* $= 100$ (une source de nourriture est considérée comme épuisée si elle n'a pas pu être améliorée pendant *limite* itérations). Pour l'algorithme OPA, nous avons utilisé les valeurs données à la section précédente. Pour les deux algorithmes, la condition d'arrêt a été fixée à 40 000 solutions testées (ce qui, pour l'algorithme ABC, correspond approximativement à 2000 cycles pour 20 abeilles, c'est-à-dire aux conditions expérimentales utilisées par D. Karaboga).

Le tableau 2 donne les résultats. Les résultats obte-

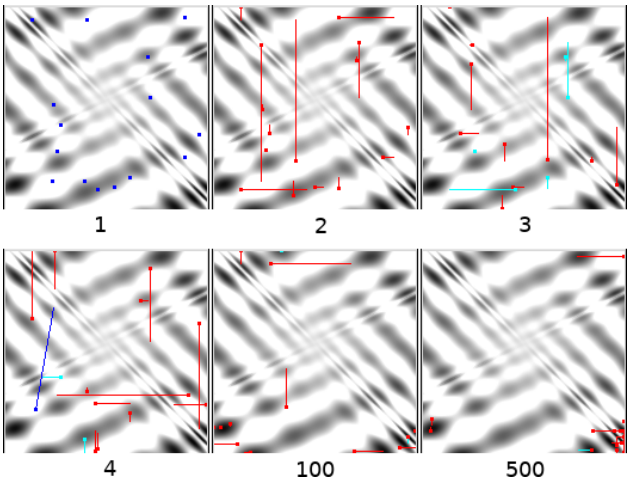

FIGURE 3 – Copie d'écran d'une interface graphique montrant les déplacements des oiseaux artificiels au fil du temps. Les copies d'écran correspondent aux itérations 1, 2, 3, 4, 100 et 500. L'image en noir et blanc montre la fonction *Eggholder*(), les valeurs les plus basses correspondant aux pixels les plus sombres. Chaque point coloré indique la position d'un oiseau et chaque ligne son déplacement. La couleur indique le type de déplacement effectué (rouge : marche, bleu : vol aléatoire, cyan : vol vers la meilleure position, vert : vol vers un autre oiseau). À l'itération 100, nous constatons que les oiseaux se sont regroupés en deux principaux *clusters*, en bas à gauche et en bas à droite. À l'itération 500, la quasi-totalité des oiseaux sont localisés en bas à droite, dans la région où se situe le minimum global de la fonction.

nus pour l'algorithme ABC sont similaires à ceux publiés par les auteurs [12] ou retrouvés dans la littérature [4]. Nous constatons que l'algorithme OPA donne de meilleurs résultats pour toutes les fonctions sauf pour la fonction $Xor6_{sig}()$, où l'écart reste faible. En revanche, l'algorithme OPA donne des résultats nettement meilleurs sur les fonctions *Rosenbrock*() et *Eggholder*(). Cela peut s'expliquer car, dans l'algorithme ABC, une seule coordonnée de la solution est modifiée à la fois puis, si la nouvelle solution n'est pas meilleure que la précédente, elle n'est pas conservée. Or la fonction *Rosenbrock*() décrit une « vallée » étroite en diagonale, il est donc nécessaire d'effectuer un déplacement simultané dans les deux dimensions. Dans l'algorithme OPA, nous modifions également une seule coordonnée de la solution à la fois, cependant la méta-heuristique permet d'effectuer plusieurs déplacements avant de retourner à la meilleure position (lorsque le déplacement 1 est choisi plusieurs fois de suite). Un problème similaire est rencontré dans l'optimisation de la fonction *Eggholder*(). Par ailleurs, le temps de calcul est comparable entre les deux algorithmes, avec un temps légèrement plus court pour OPA.

		OPA	ABC
S phère()	Résultat	**3,82e-67**	6,23e-17
	Écart-type	2,21e-66	3,05e-17
	Temps (ms)	14	15
Rosenbrock()	Résultat	**5,44e-06**	8,73e-03
	Écart-type	3,49e-05	1,40e-02
	Temps (ms)	10	10
Rastrigin()	Résultat	**1,42e-17**	7,94e-15
	Écart-type	1,58e-16	1,02e-13
	Temps (ms)	26	28
Eggholder()	Résultat	**0**	0.48
	Écart-type	0	2.86
	Temps (ms)	15	17
$Xor6_{sin}()$	Résultat	**4,54e-06**	9,31e-06
	Écart-type	7,77e-06	1,13e-05
	Temps (ms)	37	39
$Xor6_{sig}()$	Résultat	6,63e-02	**4,05e-02**
	Écart-type	3,46e-02	3,10e-02
	Temps (ms)	37	39

TABLE 2 – Comparaison des résultats obtenus avec les oiseaux picorant artificiels (OPA) et les colonies d'abeilles artificielles (ABC) pour l'optimisation de différentes fonctions numériques. Les résultats sont des moyennes sur 250 exécutions, les valeurs les plus faibles sont les meilleures.

5.5 Test sur le problème du voyageur de commerce

Pour tester le problème du voyageur de commerce, nous avons utilisé le jeu de donnée FRI26 comprenant 26 villes. Nous avons fixé la condition d'arrêt à 40 000 solutions testées et nous avons effectué la moyenne de 250 exécutions. Nous obtenons un résultat de 940,7 (écart-type 6,9, temps moyen 144ms), ce qui représente une marge d'erreur de 0,4% par rapport à la meilleure solution possible correspondant à une distance de 937.

À titre de comparaison, nous avons également testé un algorithme génétique utilisant la technique des clefs aléatoires (*random-key*) proposée par Snyder *et al.* [15], sans ajouter d'amélioration locale (l'algorithme OPA n'en utilisant pas). Nous avons obtenu un résultat moyen de 1 059,0 (écart type 52,8, temps moyen 564ms).

6 Discussion

Dans cet article, nous avons présenté une nouvelle méta-heuristique, les oiseaux picorant artificiels (OPA), inspirée du comportement des pigeons à la recherche de nourriture. Nous avons montré que cette méta-heuristique était capable de résoudre des problèmes variés en l'adaptant à l'optimisation globale non linéaire (y compris l'entraînement de réseaux de neurones) et à la résolution du problème du voyageur de commerce. Nous avons également montré que OPA

conduit à des résultats proches voire meilleurs que l'algorithme ABC inspiré du comportement social des abeilles.

Les algorithmes inspirés de la nature sont réputés pour leur simplicité souvent surprenante au regard de leur performance [19]. Cela est particulièrement vrai pour la méta-heuristique OPA que nous proposons : l'algorithme 1 est très simple et, en particulier, il ne fait pas appel à des calculs complexes contrairement à l'algorithme ABC (pour le calcul des probabilités associées au choix de chaque source de nourriture par les spectatrices) ou à celui inspiré des lucioles. Par ailleurs OPA ne fait aucune supposition sur le problème d'optimisation en lui-même ou sur l'espace des solutions, en particulier il ne nécessite pas de calcul de distances. Les calculs de distance peuvent être complexes à mettre en oeuvre dans certaines situations, par exemple dans le problème du voyageur de commerce, calculer la distance entre deux solutions n'est pas trivial.

L'algorithme OPA présente certaines similarités avec l'algorithme ABC. Les agents des deux algorithmes réalisent les tâches suivantes : l'exploration aléatoire de l'espace des solutions, la recherche locale, le retour aux meilleures solutions trouvées et la concentration de plusieurs agents sur les solutions les plus prometteuses. Dans OPA, ces différentes tâches sont associées aux quatre déplacements des oiseaux : le vol aléatoire (déplacement 2) permet l'exploration aléatoire, la marche (déplacement 1) permet la recherche locale, le vol vers la position mémorisée (déplacement 4) permet de retourner à la meilleure position rencontrée, et le vol vers un autre oiseau (déplacement 3) permet à un oiseau de « joindre ses forces » à un autre oiseau et éventuellement d'adopter sa meilleure solution. Dans ABC, les différentes tâches sont associées aux différents types d'abeilles : ouvrières, spectatrices et scoutes. Les scoutes sont chargées de l'exploration aléatoire, les ouvrières de la recherche locale avec un retour à la meilleure position mémorisée en cas d'échec, et les spectatrices permettent de concentrer davantage de moyens sur les solutions les plus prometteuses. Cependant, une différence importante entre les deux algorithmes est que, dans ABC, l'amélioration itérative et le retour à la meilleure position sont associés en un seul comportement chez les ouvrières alors que dans OPA nous les avons dissociés en deux déplacements (1 et 4). Nous avons vu dans la section 5.4 que cette séparation permet d'améliorer nettement les résultats dans certaines situations.

La source d'inspiration de notre algorithme est particulière à deux titres. Premièrement, dans la section 2, nous avons constaté que la plupart des algorithmes d'optimisation issus de la nature se sont inspirés de comportements animaux jugés « exceptionnels » ou « extraordinaires » : l'organisation sociale des abeilles ou des fourmis, la capacité des lucioles à émettre de la lumière, celle des pigeons à déterminer la direction du pôle Nord, ou des coucous à pondre leurs oeufs dans les nids des autres oiseaux, *etc*. Dans ce travail, nous nous sommes au contraire inspirés avec succès d'un comportement extrêmement « trivial », celui des oiseaux à la recherche de nourriture. D'un point de vue évolutionnaire, les comportements les plus performants conduisent à une plus grande survie des individus et sont donc les plus fréquemment rencontrés. Par conséquent, il pourrait être intéressant de s'inspirer des comportements largement répandus parmi de nombreuses espèces, plutôt qu'à des comportements exceptionnels. Cette hypothèse reste cependant à vérifier sur d'autres comportements.

Deuxièmement, la plupart des sources d'inspiration font intervenir la communication entre les animaux, que ce soit par le biais de signaux chimiques (fourmis), de mouvements (abeilles) ou de lumière (lucioles). Au contraire, en ce qui concerne les pigeons en quête de nourriture, nous n'avons pas observé de communication entre oiseaux : par exemple, lorsqu'un pigeon trouve à manger, il ne semble pas appeler d'autres pigeons à le rejoindre. Lorsqu'un pigeon en rejoint un autre, ce n'est pas suite à un appel, mais plutôt suite à une observation. De même, les deux fonctions marche que nous avons proposées ne sont pas entièrement aléatoires puisqu'elles prennent en compte la position actuelle d'un autre oiseau. Cela permet de réduire l'amplitude de la marche lorsqu'un autre oiseau est proche, ce qui peut simuler l'encombrement spatial : un oiseau marche plus vite lorsqu'il est seul que lorsqu'il est pris dans une « foule ». Cependant, l'obtention de cette position ne nécessite pas une vraie communication entre oiseaux. Par conséquent, dans OPA, l'observation remplace la communication. Dans l'algorithme que nous avons proposé (algorithme 1), chaque agent n'accède qu'à ses propres informations, ainsi qu'à la position des autres agents (pour le déplacement 1 et 3), laquelle peut être déduite par simple observation. Au contraire, un agent n'accède jamais à la meilleure position mémorisée par un autre agent. De manière assez surprenante, si nous modifions l'algorithme pour intégrer la communication, en faisant en sorte que le déplacement 3 amène l'oiseau sur la meilleure position mémorisée par un autre oiseau plutôt que sur sa position actuelle ($x_i = X_j$ et $f_i = F_j$), cela n'améliore pas les résultats, voire les détériore.

Dans la présentation de l'algorithme OPA, nous avons séparé la méta-heuristique de son adaptation aux deux problèmes (optimisation globale non linéaire et voyageur de commerce). Cette séparation facilite l'adaptation à de nouveaux problèmes, puisqu'il suffit de définir les deux fonctions *vol*() et *marche*(). Dans la plupart des algorithmes méta-heuristiques existants, la séparation n'est pas aussi claire et l'adaptation plus complexe.

Les perspectives de ce travail incluent (1) une évaluation plus poussée de la méta-heuristique et sa comparaison à d'autres méta-heuristiques ou des approches différentes (solveur numérique non linéaire par exemple), (2)

l'application de la méta-heuristique à d'autres problèmes d'optimisation, tels que le *clustering* et l'optimisation sous contraintes, ainsi que (3) l'amélioration de l'algorithme. Pour cela, une piste serait l'ajout de la biodiversité, en considérant plusieurs populations d'oiseaux aux caractéristiques différentes. La simplicité extrême de l'algorithme OPA pourrait aussi le rendre intéressant dans l'éducation.

Références

[1] Abraham A: *Handbook of Measuring System Design*, chapitre Artificial neural networks. John Wiley & Sons, 2005.

[2] Blum, C et X Li: *Natural computing series, Swarm intelligence : introduction and applications*, tome 43-85, chapitre Swarm intelligence in optimization. 2008.

[3] Bonabeau, E, M Dorigo et G Theraulaz: *Inspiration for optimization from social insect behaviour*. Nature, 406 :39–42, 2000.

[4] Brajevic, I et M Tuba: *Training feed-forward neural networks using firefly algorithm*. Dans *Recent advances in knowledge engineering and systems science*, 2013.

[5] Darrell W: *A genetic algorithm tutorial*. Statistics and Computing, 4 :65–85, 1994.

[6] Dorigo, M, M Birattari et T Stutzle: *Ant Colony Optimization - Artificial ants as a computational intelligence technique*. IEEE Comput. Intell. Mag, 1 :28–39, 2006.

[7] Duan, H et P Qiao: *Pigeon-inspired optimization : a new swarm intelligence optimizer for air robot path planning*. International journal of intelligent computing and cybernetics, 7(1) :24–37, 2014.

[8] Garnier, S, J Gautrais et G Theraulaz: *The biological principles of swarm intelligence*. Swarm intelligence, 1(1) :3–31, 2007.

[9] Karaboga, D et B Basturk: *Artificial Bee Colony (ABC) optimization algorithm for solving constrained optimization problems*. Lecture Notes in Computer Science, 4529 :789–798, 2007.

[10] Karaboga, D et C Ozturk: *Neural networks training by artificial bee colony algorithm on pattern classification*, 2009.

[11] Karaboga, D et C Ozturk: *A novel clustering approach : Artificial Bee Colony (ABC) algorithm*. Applied Soft Computing, 11(1) :652–657, 2011.

[12] Karaboga D: *An idea based on honey bee swarm for numerical optimization*. Technical report, 2005.

[13] Marinakis Y: *Encyclopedia of optimization*, tome 1498-1506, chapitre Heuristic and metaheuristic algorithms for the traveling salesman problem. Springer-Verlag, 2009.

[14] Poli, R, J Kennedy et T Blackwell: *Particle swarm optimization - An overview*, 2007.

[15] Snyder, L V et M S Daskin: *A random-key genetic algorithm for the generalized traveling salesman problem*. European Journal of Operational Research, 174(1) :38–53, 2015.

[16] Voss F: *Encyclopedia of optimization*, tome 2061-2075, chapitre Metaheuristics. Springer-Verlag, 2009.

[17] Yang, X S et S Deb: *Cuckoo search via Levy flights*. Dans *World Congress on Nature & Biologically Inspired Computing*, tome 210-214, 2009.

[18] Yang XS: *Firefly algorithms for multimodal optimization*. Stochastic Algorithms : Foundations and Applications - Lecture Notes in Computer Sciences, 5792 :169–178, 2009.

[19] Yang XS: *Nature-inspired metaheuristic algorithms (second edition)*. Luniver Press, 2010.