Artificial Feeding Birds (AFB): a new metaheuristic inspired by the behavior of pigeons

Jean-Baptiste Lamy

July 23, 2018

LIMICS (Laboratoire d'informatique médicale et d'ingénierie des connaissances en e-santé), Université Paris 13, Sorbonne Université, Inserm, 93017 Bobigny, France, jean-baptiste.lamy @ univ-paris13.fr

Abstract

Many optimization algorithms and metaheuristics have been inspired by nature. These algorithms often permit solving a wide range of optimization problems. Most of them were inspired by exceptional or extraordinary animal behaviors. On the contrary, in this chapter, we present Artificial Feeding Birds (AFB), a new metaheuristic inspired by the very trivial behavior of birds searching for food. AFB is very simple, yet efficient, and can be easily adapted to various optimization problems. We present application to unconstrained global nonlinear optimization, with several benchmark functions and the training of artificial neural networks (ANN), and to the resolution of ordering combinatorial optimization problems, with two examples: the traveling salesman problem and the optimization of rainbow boxes (a recent visualization technique for overlapping sets). We compare the results with those produced with Artificial Bee Colony (ABC), Firefly Algorithm (FA), Genetic Algorithm (GA) and Ant Colony Optimization (ACO), showing that AFB gives results equivalent or better than the other metaheuristics. Finally, we discuss the choice of inspiration sources from nature, before concluding.

1 Introduction

Many algorithms have been inspired by nature. Two well-known examples are Artificial Neural Networks (ANN) [1] and Genetic Algorithms (GA) [5]. More recently, researchers inspired themselves from the behavior of animals for inventing new optimization algorithms: social organization of insects [3] like ants [6] and honey bees [12], cohesion within a swarm in flight [18], communication by light between fireflies [23], parasitic behavior of cuckoo [22], ability of pigeons to orientate themselves spatially according to the sun position and the North pole direction [7]. These algorithms typically rely on *swarm intelligence*, *i.e.* they consider a population of agents that interact between themselves and with their environment [8]. These agents are very simple but they can achieve complex tasks together, and in particular they can solve optimization problems [2]. These algorithms are called *metaheuristics* when they provide a top-level strategy that can be used to guide a low-level heuristic search strategy [21, 16]. Consequently, a metaheuristic is not specific to a given type of problem; it can solve very different problems, depending on the chosen low-level search strategy.

Most of the meta-heuristics were actually inspired by *exceptional* or *extraordinary* animal behaviors. For instance, the ability of fireflies to emit light is exceptional: only a few animal species are able to emit light. Even the social organization of insects are quite rare: many species do not organize themselves in huge colonies (even not all bee species). However, from an evolutionary point of view, the most efficient behaviors lead to a higher chance of survival and thus, they are expected to be observed more frequently. Consequently, we might regard exceptional behaviors as poorly efficient ones (in terms of performances or capability of adaptation), and common behaviors as more efficient.

In this chapter, we propose Artificial Feeding Birds (AFB), a new metaheuristic that follows a different kind of inspiration: it has been inspired by a very *trivial* and *common* behavior that we observed on birds like pigeons, when they are searching for food on sidewalks or in a garden. Our hypothesis is that, if pigeons are so common, it is because their food search strategy is efficient, and thus it is an interesting inspiration source for algorithms. AFB presents several advantages: (1) the metaheuristic is very simple, (2) it provides good results, and (3) it is easy to adapt to new optimization problems, and in particular it makes no assumption on the solution space and does not require the computation of distances between solutions. Here, we will show the adaptability of AFB by applying it to unconstrained global nonlinear optimization, including the training of ANN, and to the resolution of ordering optimization problems, with two examples: the Traveling Salesman Problem (TSP) and the optimization of rainbow boxes [14]. Rainbow boxes are a recent information visualization technique for overlapping set that requires to solve a complex optimization problem.

The rest of the chapter is organized as follows. Section 2 presents a brief state of the art of nature-inspired optimization algorithms. Section 3 describes the behavior that we observed on pigeons and other birds searching for food. Section 4 presents the metaheuristic algorithm, and its adaptation to two problems: unconstrained global nonlinear optimization and ordering optimization. Section 5 presents various experiments performed for determining parameters values, and for testing AFB on several benchmark functions, on TSP and on rainbow boxes optimization, and comparing AFB with other metaheuristics. Finally, section 6 discusses the main results, the differences between AFB and other metaheuristics, and gives some perspectives.

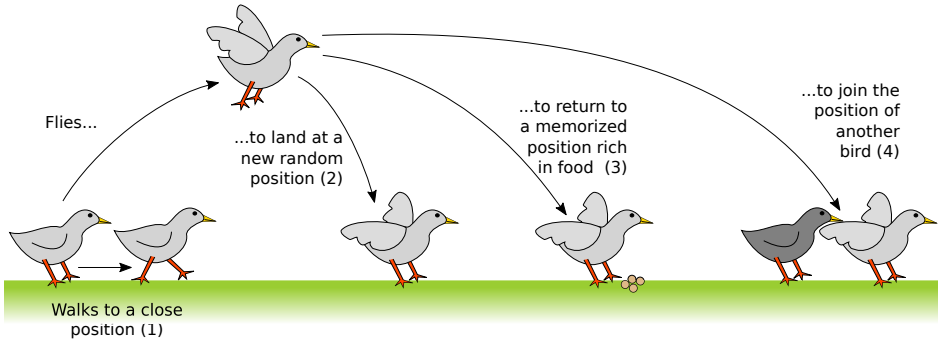

Figure 1: The four types of move observed on birds when they are searching for food.

2 Related works

Many optimization algorithms have been inspired by nature [24]. Many of them are based on the social behavior of insects [3] and their ability to communicate through chemical substances, moves or light. An example is the behavior of ants, which inspired Ant Colony Optimization (ACO) [6]. Ants explore their environment for searching food and they leave a track behind them using chemical substances named pheromones. These pheromones are then considered by other ants as signals indicating them the directions to follow or not to follow. Fireflies behavior and their use of light to attract sexual mates also inspired an algorithm [23]. The Firefly Algorithm (FA) has been used subsequently for training ANN [4].

The Artificial Bee Colony (ABC) algorithm [12] has been inspired by the communication between bees through their waggle dances when they are searching nectar. When a bee has found an interesting food source, she goes back to the hives and performs the waggle dance to "recruit" other bees for bringing them to the food source. ABC considers three types of bee: workers, onlookers and scouts. Each worker is associated with a food source, whose position corresponds to a solution of the optimization problem. Better solutions correspond to richer source food. On each cycle, each worker tries to improve her solution by trying a nearby solution, and keep this new solution if it is better than the current one. Then she communicates to onlookers the quality of her solution. Onlookers obtain this information from workers; on each cycle, each onlooker chooses a worker to help, the choice is random but with a higher probability to choose the workers with better solutions. Then, the onlooker tries to improve the solution, in a way similar to the worker. When a solution cannot be improved after a fixed number of trials, it is abandoned and the scout bee is in charge of finding a new random food source for the worker. The ABC metaheuristic has been adapted to unconstrained global nonlinear optimization [12], the optimization of constrained problems [9], the training of ANN [10] and clustering [11].

Particle Swarm Optimization (PSO) is a technique inspired by the behavior of animals that move in a swarm (flying insects or birds, fishes) [18]. In the swarm, the move performed by each individual at a given time depends on the position of the other individuals and of the quality of the solutions associated with their positions. FA can also be seen as an improvement of PSO.

More recently, several bird-inspired algorithms were published. XS Yang *et al.* inspired themselves from the parasitic behavior of cuckoo [22]. H Duan *et al.* proposed an algorithm inspired by pigeons and their ability to orientate themselves spatially according to the sun position and the North pole direction [7]. The algorithm has been used for air robot path planning.

3 Behaviors observed on birds

Our observations were carried initially on pigeons, and then in groups mixing various types of birds feeding at the same place. Pigeons are very common birds in European towns and they are easy to observe. They feed by pecking seeds or crumbs of food on the ground. When no food is in reach, they explore their environment, using the two mode of movement at their disposal: walking and flying.

We observed that a pigeon performs four types of move when searching for food (Figure 1) : (1) walking to a new position close to his current position (because they walk slowly), (2) flying and landing at an arbitrary semi-random position, (3) flying and returning to a memorized position rich in food (such as a picnic area), and (4) flying and landing close to another pigeon. Typically, a pigeon walks for searching food (one or several move 1). After a while, if no food is found, he flies and go to a random place (move 2), to a memorized position (move 3) or join another pigeon (move 4). Then, he begins to walk again (move 1), *etc.*

This simple behavior optimizes the food search. Move 1 (walk) allows a local search. This is meaningful because there is a high probability to find food close to a position where food has already been found (*e.g.* if crumbs of a sandwich are present somewhere, it is probable to find other crumbs of the same sandwich nearby). Move 2 (fly to random position) allows the random exploration of space. Move 3 (return to a memorized position) allows retrieving food, or continuing to look for it in the surroundings. Move 4 (join another bird) allows benefit from the food that the other bird might have found. This leads to big groups of pigeons when an important quantity of food is available in a given place.

These observations were carried on pigeons, however, many other birds present a similar behavior, including sparrows and gooses. When several species of birds are mixed together and feed at the same place, we observed that the size of the bird has an impact on move 4 (join another bird): a big bird can join a smaller one. On the contrary, a small bird is frightened by bigger ones and do not join them. We observed this behavior in population mixing gooses and pigeons.

4 Translation in algorithms

4.1 Metaheuristic

We designed a metaheuristic inspired by the bird feeding behavior. We consider a multi-agent system, each agent being an *artificial bird*. The position of each bird corresponds to a candidate solution for the optimization problem. Each bird also keeps in memory the best position he found, *i.e.* the one corresponding to the solution that minimizes the best the cost function. When the current position of a bird is better than the memorized position, the current position is memorized and the bird is considered to "have fed".

The metaheuristic performs several cycles. In each cycle, each bird performs one of the four moves described previously. For a given bird, the next move is determined as follows: if the bird has flown in the previous cycle, he walks. If the bird has eaten in the previous cycle, he walks. Otherwise, one of the four moves is randomly chosen, with different probabilities associated with each move. In addition, we considered two sizes of birds: small and big ones. Only big birds can perform move 4 and join another (small or big) bird. While this rule does not exactly match our observation, it efficiently avoids that all birds get stuck in a local minimum.

Two moves (3 and 4) are generic and independent from the optimization problem. On the contrary, the two other moves (1 and 2, *i.e.* walk and random fly) are problem-dependent. Therefore, we can define an optimization problem as a triplet of three functions $(cost, fly, walk)$, as follows:

- $cost : A \to \mathbb{R}$, the cost function to minimize, where A is the admissible set of solutions for the cost function,
- $fly : \phi \to A$, a function that returns a random position,
- $walk : \mathbb{N} \to A$, a function that returns a random position close to the current position of the bird indicated by the given integer index.

The metaheuristic takes 5 parameters:

- n, the number of artificial birds,
- r, the ratio of small birds in the total bird population (the other being big birds),
- p_2, the probability that a bird chooses move 2,
- p_3, the probability that a bird chooses move 3,
- p_4, the probability that a bird chooses move 4.

The probability for move 1 is thus $p_1 = 1 - p_2 - p_3 - p_4$.

The metaheuristic defines 6 per-bird variables, $1 \leq i \leq n$:

- $x_i \in A$, the current position of bird i,
- $f_i \in \mathbb{R}$, the value of the cost function for x_i,
- $X_i \in A$, the best position found and memorized by bird i,
- $F_i \in \mathbb{R}$, the value of the cost function for X_i,
- $s_i \in \{0, 1\}$, the size of bird i (0 is a small bird, *e.g.* a pigeon, and 1 a big bird, *e.g.* a goose),
- $m_i \in \{1, 2, 3, 4\}$, the type of move performed by bird i at the previous cycle (1 walk, 2 fly to a random position, 3 fly to the memorized position, 4 fly to the position of another bird).

Algorithm 1 shows the metaheuristic. It initializes the variables, runs cycles and finally determines the best solution found. During initialization, the position x_i of each bird is randomly defined using the $fly()$ function, the current cost f_i is computed and m_i is set to 2 (because the random initialization is comparable to move 2).

Algorithm 1 The AFB metaheuristic in pseudo-code.

For $1 \leq i \leq n$:
  $x_i = X_i = fly()$
  $f_i = F_i = cost(x_i)$
  $m_i = 2$
  $s_i = 0$ if $i \leq r \times n$, 1 otherwise

Repeat:
  For $1 \leq i \leq n$:
    If $m_i \in \{2, 3, 4\}$ or $f_i = F_i$:
      $p = 1$
    Else, if $s_i = 0$:
      $p =$ random real number between $p4$ and 1
    Else:
      $p =$ random real number between 0 and 1

    If $p \geq p_2 + p_3 + p_4$:
      $m_i = 1$
      $x_i = walk(i)$
      $f_i = cost(x_i)$
    Else, if $p \geq p_3 + p_4$:
      $m_i = 2$
      $x_i = fly()$
      $f_i = cost(x_i)$
    Else, if $p \geq p_4$:
      $m_i = 3$
      $x_i = X_i$
      $f_i = F_i$
    Else:
      $m_i = 4$
      $j =$ random integer number between 1 and n, with $j \neq i$
      $x_i = x_j$
      $f_i = f_j$

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

    Check stopping condition

The best solution found is X_k, with $1 \leq k \leq n$ such as $F_k = min(\{F_i \mid 1 \leq i \leq n\})$

In each cycle, for each bird i, the algorithm chooses one of the four possible moves using the previously described rules, updates m_i with the chosen move, performs the move, and updates the best position if needed. Moves 1 and 2 call the $walk()$ and $fly()$ functions, respectively, and then the $cost()$ function. Moves 3 and 4 move the bird to the best memorized position or to the position of another random bird, respectively. These two moves do not test a new solution and thus do not require to call the $cost()$ function.

If the current cost f_i is lower or equal to the best memorized cost F_i, then the current position and cost are memorized. The condition "lower **or equal**" allows the modification of the memorized position in order to keep a solution that is not better than the previous one, but different; this potentially increases the diversity of the solutions memorized by the population of birds.

Finally, it is necessary to include a stopping condition in the algorithm. We suggest stopping the algorithm after a pre-defined number of solutions have been tested (*i.e.* to limit the number of calls to the $cost()$ function). As our

Algorithm 2 $fly()$ and $walk()$ functions for optimization problems in $\mathbb{R}^d$.

Function $fly()$:
    $x' \in \mathbb{R}^d$
    For $1 \leq k \leq d$:
        x'_k = random real number between x_{min} and x_{max}
    Return x'

Function $walk(i)$:
    $x' \in \mathbb{R}^d$, $x'_k = x_{ik}$ for $1 \leq k \leq d$
    j = random integer number between 1 and n, $j \neq i$
    k = random integer number between 1 and d
    $\Delta = |x_{ik} - x_{jk}|$
    if $\Delta = 0$: $\Delta = 0.001$
    r = random real number between -1 and 1
    $x'_k = x'_k + r \times \Delta$
    If $x'_k < x_{min}$: $x'_k = x_{min}$
    Else, if $x'_k > x_{max}$: $x'_k = x_{max}$
    Return x'

Algorithm 3 $fly()$ and $walk()$ functions for solving ordering problems.

Function $fly()$:
    x' = sequence of the elements in T, in a random order
    Return x'

Function $walk(i)$:
    $\Delta = 0$
    Repeat maximum 100 times:
        j = random integer number between 1 and $n, j \neq i$
        k = random integer number between 1 and $|T|$
        Δ' = position of x_{ik} in x_j − position of $x_{i(k-1)}$ in x_j
        If $1 < abs(\Delta') < |T| - 1$:
            $\Delta = \Delta'$
            Break
    If $\Delta = 0$: Δ = random integer number between 2 and $n - 1$
    $l = (k + \Delta)$ modulo $|T|$
    If $k > l$: Swap k and l
    x' = clone of sequence x_i
    Reverse the order of elements between x'_k and x'_l
    Return x'

metaheuristic does not test a new solution for each bird in each cycle, this stopping condition allows a fair comparison with other optimization algorithms that test more solutions per cycle.

At the end of the process, the best solution found is the best position memorized by the birds.

The $walk()$ and $fly()$ functions depend on the optimization problems. $fly()$ returns a random solution, and $walk()$ a new solution close to the one of the given bird. Simple $walk()$ functions just modify the bird position. More sophisticated $walk()$ functions (as the two presented below) first evaluate the local density of birds at the given bird's position. The local density is roughly estimated by Δ, a partial distance computed between the walking bird and another bird chosen randomly. Then $walk()$ modifies the solution of the bird by performing a move (or a change) that is proportional to Δ.

4.2 Adaptation to unconstrained global nonlinear optimization

In this section, we apply the AFB metaheuristic to the global optimization of a real function with d parameters, *i.e.* $A = \mathbb{R}^d$. The solutions are points in a space with d dimensions, whose coordinates are between x_{min} and x_{max}.

Algorithm 2 describes the $fly()$ and $walk()$ functions we propose for global nonlinear optimization. The $fly()$ function simply returns a random position. The $walk()$ function modifies a randomly chosen coordinate k of the current bird position x_i. The modification has a maximum amplitude which is Δ, the absolute value of the difference between coordinates x_{ik} and x_{jk}, where j is another randomly chosen bird index. This allows smaller amplitudes when birds are closer. This is a similar local search heuristic than the one proposed in the ABC algorithm [12]; however our metaheuristic differs.

4.3 Adaptation to ordering optimization

In this section, we apply the AFB metaheuristic to ordering problems, *i.e.* problems in which an optimal order of the elements of a set T must be found. Ordering problems are a subcategory of combinatorial optimization problem. Algorithm 3 describes the $fly()$ and $walk()$ functions we propose for solving ordering problems. Bird positions x_i are ordered sequences of the elements in T. The $fly()$ function generates a random order. The $walk()$ function corresponds to a variant of the 2-opt local search heuristic [17], in which the sequence is opened at two points, and reconnected after reversing one of the two parts (*e.g.* if the sequence ABCDEF is split between B-C and E-F, the resulting sequence is ABEDCF). We modified the heuristic in order to take into account the local similarity of the bird's position with the position of another random bird j (figure 2). The similarity is (roughly) estimated by Δ, the number of elements in x_j between the element located in position k (the first opened edge) and the previous element in x_i. If no satisfying value can be found for Δ after 100 tries, a default random value is used.

5 Experimentations

5.1 Implementation

The algorithms described in the previous section were implemented in the Python language and executed with the PyPy2 interpreter (a version of Python integrating a Just-In-Time (JIT) compiler). Other algorithms (ABC, FA, GA and ACO) have also been implemented in the same language, for comparison purpose.

The AFB implementation in Python is available online under the GNU LGPL Open Source license, including most of the examples of the chapter:

https://bitbucket.org/jibalamy/metaheuristic_optimizer

5.2 Benchmarks and tests

For nonlinear optimization, we selected five functions frequently used in benchmarks (Figure 3): a 5-dimension sphere function, the Rosenbrock function, the 10-dimension

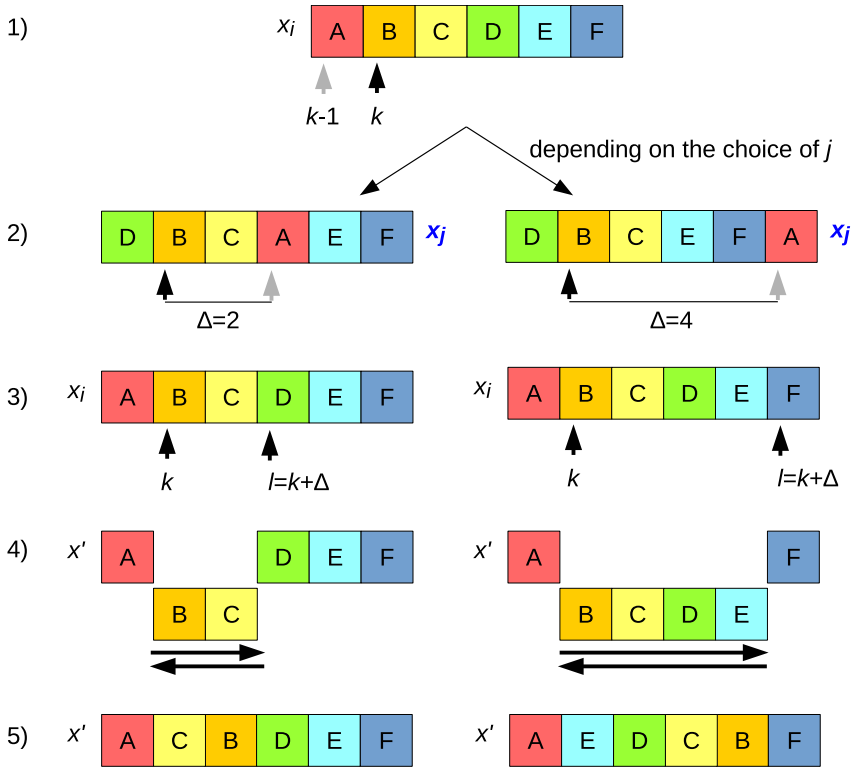

Figure 2: Examples of the modified 2-opt local search heuristic on sequences of 6 elements, ABCDEF. Notice how the order of another random bird j affects the 2-opt for bird i (the schema shows the results for two different birds j). Variable names (i, j, k, l, Δ,...) correspond to those in algorithm 3.

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

$$Rosenbrock(x, y) = (1 - x^2) + 100 \times (y - x^2)^2 \qquad -2.048 < x < 2.048 \text{ and } -2.048 < y < 2.048$$

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

$$Eggholder(x, y) = -(y + 47) \sin\left(\sqrt{\left|y + \tfrac{x}{2} + 47\right|}\right) - x \sin\left(\sqrt{|x - (y + 47)|}\right) + 959.640662720851 \qquad -512 < x, y < 512$$

$$Himmelblau(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \qquad -6 < x, y < 6$$

Figure 3: The five benchmark functions for experimentation.

Rastrigin function, the Eggholder function (we added a constant to this function, so as its global minimum is about 0, as for the other functions) and the Himmelblau function.

In addition, we tested the training of ANN, using the "Xor6" problem, which is also commonly used for benchmarking [4]. Its consist of an ANN with 2 input neurons I_1 and I_2, 2 hidden neurons H_1 and H_2, and 1 output neuron O. Neurons have no bias, there are thus 6 coefficients to optimize: I_1-H_1, I_1-H_2, I_2-H_1, I_2-H_2, H_1-O and H_2-O. The four learning samples (I_1, I_2, O) are $(0, 0, 0)$, $(0, 1, 1)$, $(1, 0, 1)$, $(1, 1, 0)$; they correspond to a logical exclusive or. The cost function takes 6 parameters corresponding to the 6 coefficients of the ANN, and returns the Mean Squared Error (MSE). We tested the Xor6 problem with two activation functions: sinus and sigmoid, leading to 2 functions to optimize: $Xor6_{sin}$ and $Xor6_{sig}$, with $x_{min} = -100$ and $x_{max} = 100$.

For ordering optimization, we tested two problems. The first one is the Traveling Salesman Problem (TSP). It is a well-known optimization problem in which a traveling salesman must visit a set T of towns and then return to his starting town. The objective is to find the optimal order for visiting the towns, in order to minimize the total distance of the trip. We used the FRI26 dataset, which includes 26 towns.

The second problem is the optimization of rainbow boxes. Rainbow boxes [14, 13] are an information visualization technique that we recently proposed for *overlapping sets*. We applied this technique for the visualization of drug properties [15]. Several elements and sets are to be visualized, each element can belong to several sets and each set can contain several elements. In rainbow boxes, each element is represented by a column, and each set by a rectangular box that covers the columns corresponding to the elements belonging to the set (see example in figure 4). When these elements are not presented in adjacent columns, holes are present in the box. The optimization problem consists in finding the optimum column order to minimize the

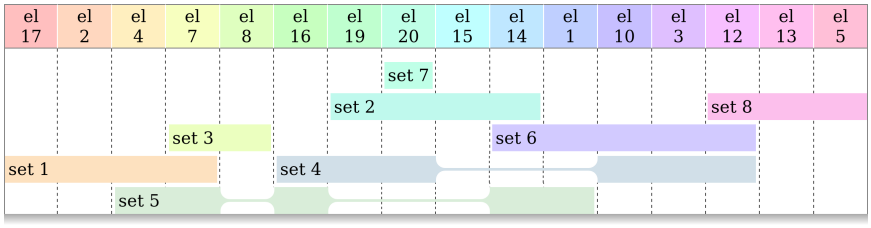

Figure 4: Example of rainbow boxes, corresponding to one of the small random dataset ("el" stands for "element"). Here, after optimization, there are 3 holes, one in set 4 and two in set 5.

	p_1 (walk)	p_2 (random fly)	p_3 (memory)	p_4 (join other)
$Sphere()$	0	0	0.64	0.36
$Rosenbrock()$	0.54	0.05	0.36	0.05
$Rastrigin()$	0	0	0.58	0.42
$Eggholder()$	0.37	0.01	0.31	0.31
$Himmelblau()$	0	0.20	0.66	0.14
$Xor6_{sin}()$	0.06	0.09	0.52	0.33
$Xor6_{sig}()$	0.20	0	0.51	0.29
Retained values	**0.25**	**0.01**	**0.67**	**0.07**

Table 1: The best values obtained for parameters p_2, p_3 and p_4 for each test function, and the retained values.

number of holes. Therefore, the $cost()$ function computes and returns the number of holes produced by a given column order. We tested two previously presented rainbow boxes datasets [13], amino acid and histones, and we generated two sets of 100 random datasets, the first with small datasets (20 elements, 8 sets and 30 membership relations) and the second with larger ones (30 elements, 15 sets and 60 membership relations).

5.3 Parameter values

First, n was set to 20 semi-arbitrarily: this value gives good results and is used in several other population-based algorithms, such as ABC. Similarly, r was set to 0.75 (corresponding to 15 small birds and 5 big ones).

Finally, for studying and determining the values of the parameters p_2, p_3 and p_4, we created two instances of the AFB metaheuristic, one in charge of optimizing the parameters of the other. Table 1 shows the best parameter values found for each benchmark functions. Since each parameter p_{1-4} correspond to one of the four moves, we can see that some moves are not pertinent for optimizing some functions (However, notice that $p_1 = 0$ does not mean that the birds never walk, because they *always* walk after landing or feeding; it just mean that the bird never walk after an unsuccessful walk move). However, additional tests showed that the values found for p_4 are overestimated in table 1: when optimizing parameter values, we computed the mean of 20 runs for each set of parameter values, but there is still a certain variability in the results. A high p_4 value means that the AFB algorithm will focus more on the best solutions found, and this possibly benefits more to better runs, which are those selected during the optimization process.

The last line of table 1 shows the parameter values that we retained. These values will be systematically used in the rest of the chapter.

5.4 Results on unconstrained global non-linear optimization

We tested AFB on the optimization of the benchmark functions presented in section 5.2, and we compared the results with those obtained with ABC and FA. The stopping condition was fixed to 40,000 tested solutions. For ABC, parameters were the following: $n = 20$ (number of bees), $limit = 100$ (a food source is considered exhausted if it cannot be improved after 100 cycles). These values correspond to the conditions used by Karaboga for the ABC algorithm: 2000 cycles for 20 bees [12]. For FA, we were unable to find a set of parameter values that performed well on all tests. We used the following values: $n = 20$ (number of fireflies), $\alpha = 0.022$, $\alpha_{fade} = 1.0$, $\beta = 0.442$, $\gamma = 3.413$ for $Rosenbrock()$, $\alpha = 0.268$, $\alpha_{fade} = 1.0$, $\beta = 0.128$, $\gamma = 9.807$ for $Eggholder()$, and $\alpha = 0.37$, $\alpha_{fade} = 0.98$, $\beta = 0.91$, $\gamma = 0$ for others. These values were obtained by running another optimization algorithm on the parameter values.

Table 2 gives the results. For ABC, they are similar to the ones published [12, 4]. AFB gives the best results for all functions but $Xor6_{sig}()$ and $Himmelblau()$, but for these two functions, AFB is close to the best results. Compared to ABC, AFB performs much better for two functions, $Rosenbrock()$ and $Eggholder()$. We explain this difference as follows. In ABC, a single coordinate of the solution is modified, and then a greedy selection is performed between the new solution and the previous one. This implies that the solution can move only in a single axis, and the move needs to improve the results to be conserved. But the $Rosenbrock()$ function has a narrow "valley" that requires diagonal moves. In AFB, walk moves also involve a single coordinate, however, several walks can be performed before returning to the best memorized position. This permits diagonal moves.

Finally, computation times were similar between AFB and ABC, but higher for FA.

		AFB	ABC	FA
Sphere()	Result	**5.07e-81**	6.23e-17	2.36e-32
	Std. deviation	2.88e-80	3.05e-17	1.62e-32
	Time (ms)	15	15	310
Rosenbrock()	Result	**2.64e-05**	8.73e-03	2.84e-03
	Std. deviation	8.05e-05	1.40e-02	3.56e-02
	Time (ms)	13	10	176
Rastrigin()	Result	**0** (*)	7.94e-15	95.04
	Std. deviation	0	1.02e-13	113.30
	Time (ms)	26	28	383
Eggholder()	Result	**0** (*)	0.48	11.3
	Std. deviation	0	2.86	18.16
	Time (ms)	17	17	185
Himmelblau()	Result	6.00e-31	5.80e-17	**2.46e-31**
	Std. deviation	3.37e-31	3.17e-17	4.04e-31
	Time (ms)	11	11	156
$Xor6_{sin}$()	Result	**1.24e-06**	9.31e-06	1.34e-04
	Std. deviation	2.78e-06	1.13e-05	8.39e-04
	Time (ms)	37	39	337
$Xor6_{sig}$()	Result	4.40e-02	**4.05e-02**	1.26e-01
	Std. deviation	3.77e-02	3.10e-02	1.75e-02
	Time (ms)	41	39	49

Table 2: Comparison of the results obtained when minimizing various functions with AFB and ABC. Results are the means over 250 runs, and the lower values are the best. (*) 0 results are understood at double precision (which is 1e-323).

5.5 Results on ordering problems

We compared the results obtained with AFB with those obtained with GA and ACO. The GA we implemented is the random-key algorithm proposed by Snyder *et al.* [19] for generalized TSP. We tested the algorithm both with and without local optimizations (2-opt and swap). We used two ACO algorithms: ACO-pants[1], a Python ACO implementation for TSP with some TSP-specific optimizations, for TSP and the *MAX-MIN Ant System* (MMAS) [20] for rainbow boxes optimization (without any specific optimization).

For TSP, the stopping condition was fixed to 40,000 tested solutions and we performed 250 runs for each algorithm. The best possible tour has a distance of 937. For rainbow boxes optimization, the stopping condition was fixed to 40,000 for large random datasets, and to 10,000 for the others. We performed 250 runs for each of the amino acid and histone dataset, and one run for each of the random datasets. The best known results are 4 for amino acids and 6 for histones. Table 3 gives the results. AFB performed better than other algorithms. For TSP, AFB yielded a mean distance of 941.0. This represents a 0.43% error margin compared to the best possible solution. ACO performed well on rainbow boxes small datasets, but poorly on large ones.

Results obtained with GA for TSP are not on par with those published by Snyder *et al.* [19]. Two reasons can explain that. First, the number of tested solutions was much lower in our experiment and the GA needs to run longer, especially if local optimizations are used because they perform a lot of calls to the cost function. Second, we did not implement the TSP-specific optimizations proposed by Snyder *et al.* (such as considering the two tours ABCD and BCDA as identical), since we were targeting ordering problems in general.

[1]; https://github.com/rhgrant10/Pants

6 General discussion

In this chapter, we presented Artificial Feeding Birds (AFB), a new metaheuristic, inspired by the behavior of pigeons searching for food. We showed that AFB was able to solve various problems: unconstrained global nonlinear optimization, including training of neural networks, traveling salesman problem and rainbow boxes optimization. We also showed that AFB competes well with ABC, FA, GA and ACO algorithms.

We focused on rapid tests and rather small datasets, because our short-term goal is to optimize visualizations, such as rainbow boxes, and they often need to be produced dynamically, on the fly. In particular for TSP, the results presented here need to be confirmed on bigger datasets and with longer computation times. For rainbow boxes optimization, we previously proposed a specific heuristic algorithm [14, 13], however, it was limited to 25 elements or less, due to computation time. Here, we showed that AFB could be used up to 30 elements, and possibly even more.

Lones [16] identified several metaheuristic aspects that are shared by many nature-inspired optimization algorithms. The following ones are found in AFB:

(a) *Neighborhood search* consists of testing new solutions that are close (or similar) to the previously tested solution. In AFB, the *walk* move performs a local search, by producing a position that is close to the current position.

(b) *variable neighborhood search* is similar to neighborhood search, but considers moves with variable steps, depending *e.g.* on the position of other agents. In AFB, the two *walk*() functions we proposed perform variable neighborhood search: the step of the walk move is proportional to Δ, which depends on the position of another bird.

			AFB	GA	GA+local op	ACO
TSP	*(26 towns)*	Result	**941.0**	1082.0	983.3	968.21
		Std. deviation	7.1	66.2	36.1	13.6
		Time (ms)	133	574	103	4203
Rainbow boxes	*Amino-acid dataset*	Result	**4.04**	7.16	6.46	4.82
		Std. deviation	0.20	1.24	1.42	0.69
		Time (ms)	205	219	263	648
	Histone dataset	Result	**6**	6.39	6.53	6.27
		Std. deviation	0	0.73	0.82	0.5
		Time (ms)	550	512	557	623
	Small random datasets	Result	**3.85**	5.39	4.92	3.87
		Std. deviation	1.08	1.16	1.25	1.02
		Time (ms)	184	194	208	658
	Large random datasets	Result	**12.59**	16.63	15.59	21.21
		Std. deviation	1.90	1.94	1.58	1.53
		Time (ms)	1052	3496	4528	61523

Table 3: Comparison of the results obtained on ordering problems. Lower values are the best.

(c) *hill climbing* consists of trying to improve a given solution incrementally. If the new solution is better than the previous one, it is kept. Otherwise, it is discarded and the previous solution is kept. In AFB, when a *walk* move leads to a better solution than the one memorized by a bird, a second walk is systematically performed in the next cycle.

(d) *accepting negative moves* consists of keeping a new solution that is worse than the previous one; it is somehow the opposite of hill climbing. In AFB, when a *walk* move does not lead to a better solution, a second walk is still possible, but not systematic.

(e) *population-based search* consists of considering multiple agents. AFB considers a population of artificial birds, each bird having its own position and memory.

In particular, when using specific parameter values, AFB performs like well-known algorithm. If $p_1 = 1$ and $p_2 = p_3 = p_4 = 0$, AFB performs a random walk. If $p_3 = 1$ and $p_1 = p_2 = p_4 = 0$, AFB performs like a hill-climbing algorithm.

AFB presents some similarities with ABC. In both algorithms, agents perform the four following tasks: random exploration of the solution space, local search, reversion to the best solution found so far, and concentration of several agents on the most promising solutions. In AFB, these tasks correspond to the four moves of the birds: move 2 allows random exploration, move 1 local search, move 3 reversion to the best solution, and move 4 allows a bird to "join his force" with another bird and eventually to adopt his best solution. In ABC, these tasks are associated with the three types of bee. Scouts are in charge of the random exploration, workers of local search with reversion to the best position found in case of failure, and onlookers allow concentrating more agents on the best solutions. However, there is an important difference between ABC and AFB: in ABC, local search and reversion to the best position are grouped in the worker behavior, while in AFB, we separated them in two distinct moves (1 and 3). This separation allows accepting negative moves (*i.e.* "diagonal moves" in nonlinear optimization), and we have seen in section 5.4 that it improved the results for some benchmark functions. While our *walk*() function for global nonlinear optimization was inspired by ABC, our metaheuristic differs, hence the difference observed in the results.

The inspiration source of AFB is particular at two levels. First, in section 2, we noticed that most inspiration sources were exceptional or extraordinary animal behaviors, such as light emission. Here, on the contrary, we successfully inspired ourselves from a very trivial behavior: birds searching for food. From an evolutionary point of view, the most efficient behaviors lead to a higher chance of survival and thus, they are expected to be encountered more frequently. Consequently, it should be more interesting to inspire ourselves from very common behaviors, widely spread over many species, rather than exceptional behaviors. However, this hypothesis needs additional verification.

Second, most inspiration sources include communication between animals, using chemical signs (ants), dances (bees) or light (fireflies). On the contrary, we did not observe communication when pigeons are feeding; their behavior seemed to us rather "individualistic". For example, when a pigeon finds some food, he does not seem to call other pigeons. When a pigeon joins another one, it is not following a call, but rather following a simple observation ("another bird is there, he seems to feed, let's get closer!"). In consequence, in AFB, observation replaces communication. In Algorithm 1, each agent accesses only his own information and variables, as well as the position of other agents, which can be obtained through simple observation. On the contrary, an agent never access to the best position found by another agent. Surprisingly, when we modified the metaheuristic by adding communication of the best position, *i.e.* changing move 4 so as it moves the bird on the best position found by another bird ($x_i = X_j$ and $f_i = F_j$) rather than his current position, the results did not improve significantly (and they were even poorer for *Rosenbrock*()).

A first strength of AFB is its extreme simplicity. Nature-inspired algorithms are often surprisingly simple, with regards to their performance [24]. This is especially true for AFB: the metaheuristic (Algorithm 1) is very simple and, in particular, does not need complex computations, contrary to many other algorithms (*e.g.* ABC has a complex formula for computing the probability of onlooker bees to choose a given food source).

A second strength of AFB is its generic nature. It can be run on any optimization problem that can be defined by a $(cost, fly, walk)$ triplet of functions, and we have shown

that it performs well on very different kinds of problems. In the presentation of our algorithms, we clearly separated the AFB metaheuristic from its adaptation to the two problems (global nonlinear optimization and ordering problems). This separation greatly facilitates the adaptation to new problems, since one only has to define the two functions $fly()$ and $walk()$. Usually, in other metaheuristics, the separation between the problem-specific and the problem-independent part of the algorithm is not so clear. Furthermore, AFB makes no assumption about the optimization problem and the solution space, and in particular, it does not require to compute distance between solutions. Distance computation is not trivial in some problems, such as TSP, in terms of computation method and computation time. Finally, AFB is rather insensitive to parameter values, since we used the same default values across all our experiments. This allows to use the metaheuristic without having to tune the algorithm for a specific problem.

Perspectives of this work include the adaptation of the AFB metaheuristic to other optimization problems, such as clustering, and its use in real life applications. AFB could also be improved, for example by adding additional moves, possibly inspired by other metaheuristics. Finally, the extreme simplicity of AFB could also make it interesting for educational purpose.

References

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

[2] C Blum and X Li. *Natural computing series, Swarm intelligence: introduction and applications*, chapter Swarm intelligence in optimization, pages 43–85. 2008.

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

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

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

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

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

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

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

[11] D Karaboga and 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] J B Lamy, H Berthelot, C Capron, and M Favre. Rainbow boxes: a new technique for overlapping set visualization and two applications in the biomedical domain. *Journal of Visual Language and Computing*, 43:71–82, 2017.

[14] J B Lamy, H Berthelot, and M Favre. Rainbow boxes: a technique for visualizing overlapping sets and an application to the comparison of drugs properties. In *International Conference Information Visualisation (iV)*, pages 253–260, Lisboa, Portugal, 2016.

[15] J B Lamy, H Berthelot, M Favre, A Ugon, C Duclos, and A Venot. Using visual analytics for presenting comparative information on new drugs. *J Biomed Inform*, 71:58–69, 2017.

[16] Lones MA. Metaheuristics in nature-inspired algorithms. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1419–1422, Vancouver, BC, Canada, 2014.

[17] Marinakis Y. *Encyclopedia of optimization*, chapter Heuristic and metaheuristic algorithms for the traveling salesman problem, pages 1498–1506. Springer-Verlag, 2009.

[18] R Poli, J Kennedy, and T Blackwell. Particle swarm optimization - An overview, 2007.

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

[20] T Stützle and H H Hoos. MAX-MIN Ant System. *Future Generation Computer Systems*, 16(9):889–914, 2000.

[21] Voss F. *Encyclopedia of optimization*, chapter Metaheuristics, pages 2061–2075. Springer-Verlag, 2009.

[22] X S Yang and S Deb. Cuckoo search via Levy flights. In *World Congress on Nature & Biologically Inspired Computing*, pages 210–214, 2009.

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

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