# 

# 

# 

# 

# 

# 

# 

* [**Introduction**](#_d4w61e53qd1w) **2**
* [**Qu’est-ce l’intelligence distribuée**](#_cyzp1eruecdp) **2**
  + [La résolution de problèmes distribués](#_t4a6r8iu7qhj) 4
  + [Les systèmes Multi-agent](#_ql0virjcarqg) 5
  + [L’intelligence en essaim](#_e5zr71gwkkzq) 5
* [Présentation de quelques algorithmes existants](#_m79au78jrw07) 7
  + [Ant Colony Optimization](#_xh6wde8kjl5b) 7
  + [Présentation d’un algorithme :](#_unvaxe7dchk6) 9
  + [Particle Swarm Optimization](#_b2f6opsub4h5) 10
  + [Simulated Bee Colony](#_st7br1ae4zev) 13
  + [Bat Algorithm](#_i1t5i0hz6k5i) 18
* [**La place des systèmes multi-agents dans les réseaux de neurones**](#_h101q3hqdk16) **21**
  + [Distributed Machine Learning & Distributed Deep Learning](#_poco0dc7xjpd) 22
  + [Federated Learning](#_3a9f7ynkkzaa) 23
* [**Cas réels d’utilisation des algorithmes présentés**](#_uu4h71dqv180) **25**
  + [Utilisation de Ant Colony Optimization](#_dpr9miq1pr1w) 25
  + [L’utilisation de Particle Swarm Optimization](#_nwtczn80wbly) 26
  + [L’utilisation de Simulated Bee Colony](#_rgtonq1sr1vm) 26
  + [L’utilisation de l’algorithme des chauves-souris](#_pi77orfi4f5q) 27
  + [Multi-Agent Learning - AlphaStar (Startcraft II)](#_nt7i8i4cnbow) 27
  + [Federated Learning - GBoard (Google)](#_rbarwv6e9uc8) 28
* [**Comparaison des algorithmes d’intelligence distribuée VS non distribuée**](#_yx1k7s2g31sm) **30**
  + [Comparaison de l’ACO vs Bruteforce A\*](#_5k9cvzwqwjp4) 30
  + [Comparaison des algorithmes d’optimisation](#_bu3oguz7z6dx) 35
  + [Fonction simple](#_sptbz5o46g5z) 35
  + [Rastrigin](#_dyo3bawb9fq8) 36
* [**Conclusion et place de l’intelligence distribuée**](#_c4qnk23w15qk) **38**
* [**Annexes**](#_w58hgobdpuwu) **40**

# **Introduction**

## Qu’est-ce l’intelligence distribuée

* L’intelligence distribuée (DI) est un ensemble de technologies et de méthodes allant de l’intelligence en essaim jusqu'à l'utilisation d'algorithmes multi-agents ayant pour but de trouver des solutions distribuées pour un problème spécifique.
* La DI est principalement utilisée pour des tâches d’apprentissage, de raisonnement ainsi que de planification.
* La DI est une sous-branche de l’intelligence artificielle qui se concentre sur la simulation d'environnements plutôt que sur la prédiction méta-heuristique.
* Dans ce type de système, l’apprentissage autonome d’agent (à grande échelle et chacun indépendant) permet de trouver des solutions en tirant partie de l'interaction et la communication entre ces agents. Un des gros avantages de la DI comparé à d'autres solutions comme les réseaux neuronaux reste la quantité de données nécessaire afin d’arriver à un résultat concluant; la DI nécessite beaucoup moins de données.  
    
  La DI peut être défini en trois grandes caractéristiques :
* **La distribution de tâches entre des agents,**
* **La répartition de l’impact d’un agent,**
* **La communication entre agents.**
* Ces caractéristiques citées en amont sont les bases nécessaires pour qu’un système soit considéré comme de la DI.
* Il existe d’autres éléments qui peuvent aider à mieux caractériser un système de DI :
* **La granularité des agents :**
  + Cette caractéristique permet de décrire comment un agent décompose une tâche.
* **La connaissance des agents :**
  + Est-ce que les agents possèdent tous le même savoir ou bien sont-ils spécialisés ?
* **Le type de comportement du système :** 
  + Est-ce que les agents doivent s’aider ou bien être en compétition ?
  + Est-ce que les rôles des agents sont statiques ou modulaires ?
  + Est-ce une équipe ou une hiérarchie ?
* **Les méthodes de communication entre agents.**
* Il faut donc prendre en compte dans la méthode de création d’un système de DI
* non seulement l’architecture de la simulation mais également toutes les caractéristiques des agents possibles vue précédemment.
* Toutes ces possibilités de configuration du système doivent être pensées afin de garantir la cohérence des agents et l’interaction entre les agents afin de s’assurer que les résultats peuvent être synthétisés et traduits en véritable solution.
* La DI peut être découpée entre deux grandes branches :
  + - **Les solutions distribuées** : plusieurs branches travaillent ensemble dans un but commun,
    - **Les solutions décentralisées** : Plusieurs agents travaillent de manière indépendante sans partager leurs résultats ; on sélectionne ensuite les meilleures solutions générées.
* Dans le premier cas, on peut parler de **résolution de problèmes distribués** et dans le deuxième cas de **système multi-agents**.

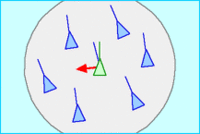
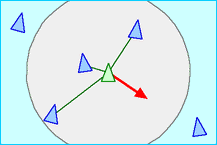
### La résolution de problèmes distribués

* Dans la résolution de problèmes distribués (DPS), plusieurs agents travaillent ensemble dans le but de résoudre un problème spécifique.
* La coopération entre les agents est la clé de ce type de système car aucun agent individuel ne possède ni les informations nécessaires, ni les connaissances ou bien les capacités de résoudre tout seul le problème. S’assurer que l’information et les capacités soient correctement réparties entre les agents afin qu’ils travaillent de manières complémentaires est primordial pour tout type de solution DPS.
* N'importe quelle solution obtenue grâce à ce type de système est le résultat de la **cohérence** (incentives qui pousse les agents à travailler ensemble) et de la **compétence** (comment les agents vont travailler ensemble) et dans la plupart des cas est appliquée à des problèmes de satisfaction de contraintes ou de problèmes d’optimisation de contraintes.
* Pour chaque type de problème, plusieurs algorithmes ont été créés mais ils ont pour la plupart tous la même approche qui consiste à réduire un problème en un ensemble de sous-tâches indépendantes. Les solutions indépendantes sont par la suite regroupées et permettent ainsi d’obtenir une solution au problème posé.
* On pourrait émettre l'hypothèse qu’avoir recours à une coopération entre agents augmente la complexité du problème de façon exponentielle et ainsi remet en cause l'utilisation des solutions DPS.
* Mais en fait, ce n’est pas le cas, il est souvent beaucoup moins coûteux de connecter plusieurs équipements d’un point de vue électronique que d’utiliser un processeur centralisé.
* De plus, la plupart des applications d’intelligence artificielle sont distribuées par conception.
* La capacité de découper un problème en un ensemble de problèmes plus simples offre ainsi de gros avantages de maintenance, débogage, etc.

### Les systèmes Multi-agent

* Les systèmes multi-agents sont des ensembles d’algorithmes, qui comme nous l’avons vu précédemment peuvent fonctionner grâce à la coopération des agents qui auront pour chacun un ensemble de règles et de contraintes prédéterminés et qui en conséquence permettent de créer un comportement qui a du sens au niveau macroscopique.
* Les interactions qui occurrent dans ce type de système ont lieu entre les agents et l'environnement.
* Dans la plupart des cas, un agent n’a pas connaissance comment il peut obtenir une récompense pour une tâche, ni comment maximiser cette récompense.
* Formulé autrement, un agent n’a pas connaissance de l’ensemble du problème, ni des capacités de résoudre ce problème tout seul. Un agent doit donc apprendre petit à petit, à son échelle, afin de résoudre un problème.

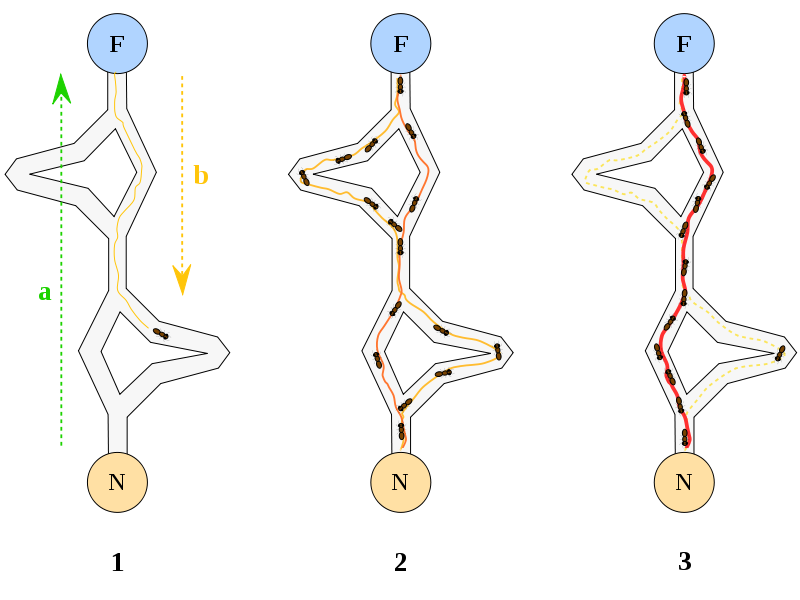
### L’intelligence en essaim

* La première apparition du terme “intelligence en essaim” créé par Gerardo Beni date de 1989 (Proceedings of the Seventh Annual Meeting of the Robotics Society of Japan) : ***“L’intelligence en essaim est une propriété de systèmes de robots non-intelligents qui montrent collectivement un comportement intelligent”***
* L’idée première était de reproduire des comportements que l’on peut observer dans la nature. Les exemples dans la nature sont assez variés mais pour en donner une idée plus concrète, la formation de bancs de poissons ou de vols d’oiseaux constitue une forme d’intelligence par essaim ; l’ensemble réussit ainsi à maintenir une forme cohérente et groupée malgré le nombre élevé d’individus.
* C’est dans le but de récréer le groupement d’un vol d’oiseaux (également applicable à un banc de poissons) que Craig Reynolds inventa les **Boids** (contraction de “bird-oid object”) en 1986. Le but est de recréer le comportement d’un déplacement d’un groupe d’oiseaux et pour cela il va donner à chaque Boid trois règles simples :
* **Cohésion** : s’il est trop éloigné, le Boid doit se rapprocher des autres Boids les plus proches,
* **Séparation** : si le Boid est trop proche d’un autre Boid il doit s’écarter pour éviter une collision,
* **Alignement** : le Boid doit s’aligner dans le sens global du groupe de Boids.
* 
  + *Cohésion* *Séparation* *Alignement*
* A partir de l’exemple des Boids, il est ainsi possible de définir les éléments qui constituent une intelligence distribuée :
* Un groupe d’individus sans leader
* Les individus sont capables de réagir les uns envers les autres
* Chaque individu a une tâche ou un but à accomplir qui permet au groupe d’avancer
* Chaque individu peut agir pour accomplir ladite tâche
* Chaque individu possède une liste de règles simples pour atteindre son objectif
* Chaque individu est dépendant des autres pour accomplir sa tâche
* Cette façon de concevoir des algorithmes en imitant la nature a donné naissance à trois écoles de pensée sur le sujet :
* **Algorithmique** : Développement et applications mathématiques des algorithmes.
* **Organisationnelle** : Recherche des principes qui peuvent être appliqués aux processus et structures avec des personnes pour surpasser les modèles conventionnels.
* **Automatisation** : Automatisation d’appareils (robots, PC, logiciels, …) pour améliorer l’efficacité de certaines tâches.
* Il est assez récurrent d’utiliser des intelligences distribuées sous nos yeux sans qu’on s’en rende compte comme, par exemple, au cinéma on peut citer Jurassic Park qui utilise l’algorithme des Boids pour créer des troupeaux de Gallimimus ou encore le Seigneur des Anneaux qui utilise une forme, plus poussée, d’intelligence distribuée pour créer des batailles à grande échelle.
* Il est également possible d'observer l’utilisation d’intelligence distribuée dans les spectacles de drones notamment en Chine. En effet, il n’est pas possible de contrôler manuellement plus de 1000 drones pour faire un spectacle ni faire suivre un paterne précis. Une seule erreur créerait une réaction en chaîne. Il est donc nécessaire de recourir à l’intelligence distribuée pour ce genre de spectacle.
* Mais c’est surtout dans l’informatique que l’intelligence distribuée semble avoir la meilleure perspective de développement que ce soit dans les domaines des réseaux, de l’IOT ou plus globalement le nombre d’appareils et logiciels communicants qui ne fait que d’augmenter.

## Présentation de quelques algorithmes existants

* Nous allons maintenant voir quatre algorithmes d’intelligences distribuées :
* L’Ant Colony Optimization,
* Le Particle Swarm Optimization,
* Simulated Bee Colony,
* Bat Algorithm.
* Ces quatre algorithmes s’inspirent de comportements du vivant. L’Ant Colony et la Bee colony s’inspirent évidemment d’insectes (les fourmis et les abeilles) et le Particle Swarm s’inspire d’algorithme comme celui des Boids en se basant sur le mouvement d’oiseaux ou de bancs de poissons pour créer une représentation de comportement sociaux. Enfin, le Bat Algorithm, s’inspire de l'écholocation des chauves-souris.

### **Ant Colony Optimization**

* L’algorithme des fourmis est un algorithme qui a pour but d’imiter les différents comportements que peut avoir une colonie de fourmis ou un ensemble d'individus formant un superorganisme. Cet algorithme a pour but de trouver les meilleurs chemin au sein d’un graphe (en général le chemin le plus court).
* Principe :
* Cet algorithme a vu le jour à la suite de plusieurs expériences menées par des équipes de scientifiques. Leurs observations ont permis de conclure qu’une colonie de fourmis tend toujours à trouver le chemin le plus court vers un objectif (dans le cas des fourmis de la nourriture).
* Fonctionnement de l’algorithme :
* Dans un premier temps, nous allons initialiser N fourmis. Il est également nécessaire de définir un nombre de cycles à effectuer. Ces fourmis qui vont être initialisées vont chacune proposer une solution.
* Chaque fourmi aura un quota de phéromones que l’on va initialiser. Ces phéromones seront utilisées par chaque fourmi afin de pouvoir les appliquer sur l'entièreté d’un chemin parcouru par une fourmi.
* À la fin de chaque cycle, chacune des solutions va être améliorée par des recherches locales. Les meilleures d’entre elles seront alors récompensées en leurs ajoutant des phéromones.
* Finalement, pour chaque fourmi, seul le chemin le plus court pour atteindre le point X sera conservé, et donc seules les phéromones appliquées sur ce chemin seront conservées. Nous pourrons donc alors conserver uniquement les chemins les plus intéressants.
  + - Recherche du plus court chemin
  + [](https://commons.wikimedia.org/wiki/File:Aco_branches.svg?uselang=fr)
* L’algorithme appliqué à un problème plus complexe :
* Nous avons vu dans l’algorithme précédent que ce dernier était très efficace pour déterminer le chemin le plus court vers un point.
* Ce même algorithme avec quelques modifications peut également résoudre le problème du “voyageur” (trouver le chemin le plus court entre plusieurs points).
* Afin d'accomplir ce but, les individus de la colonie suivent le comportement et les règles suivantes : Une fourmi ne peut passer qu'une seule fois par point. Plus le point est loin, moins il a de chance d'être visité (visibilité). Plus le dépôt de phéromone sur une piste est élevé, plus il a de chance d'être choisi comme chemin par la fourmi. Si le trajet est court, les fourmis déposent à nouveau des phéromones, incitant ainsi encore plus les autres fourmis à passer par ce chemin. Enfin, les phéromones déposées par les fourmis disparaissent petit à petit à chaque itération laissant ainsi le chemin le plus court reliant tous les points uniquement en évidence.

### 

#### Présentation d’un algorithme :

|  |
| --- |
| * **SI** nous parcourons cette étape du chemin pour la première fois :   + **next\_location = random(available\_next\_locations)** * **SINON**, pour chaque prochaine étape possible :   + **attractiveness = pow(pheromone, alpha) \* pow(1/distance, beta)**   + **total\_attractiveness += attractiveness** * **cumule = 0** * pour chaque prochaine étape :   + **weight = attractiveness / total\_attractiveness**   + **SI random() <= weight + cumule :**      - **next\_location = current\_location**   + **cumule += weight** |

### 

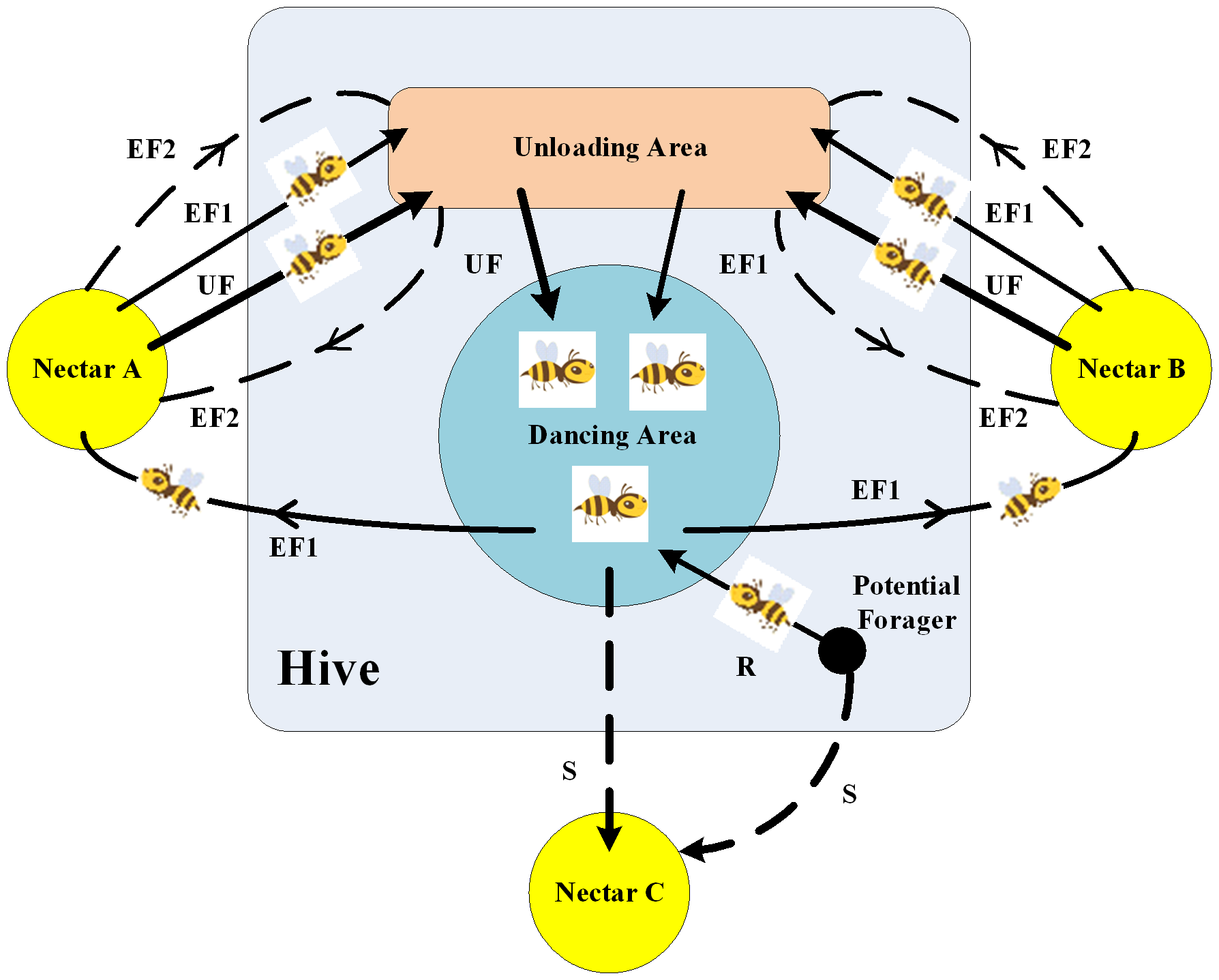
### **Particle Swarm Optimization**

* Le Particle Swarm Optimization (PSO) ou Optimisation par essaims particulaires est un algorithme d’optimisation locale. Il s’inspire de l’algorithme de déplacement d’oiseaux de Craig Reynolds défini en introduction du chapitre. Il va simuler le déplacement d’un essaim pour explorer la fonction souhaitée.
* Un algorithme d’optimisation locale a pour but de trouver la valeur minimisant ou maximisant le résultat d’une fonction ***y = f(x).*** Avec ***x*** et ***y*** des vecteurs appartenant à . Le schéma ci-dessous nous montre le fonctionnement de l’algorithme dans un contexte pour lequel ***x*** est un vecteur de dimension 2 (abscisse et ordonnée) et ***y*** est une valeur comprise entre -5 et 20. Ici les particules cherchent à minimiser la fonction.
* [](https://commons.wikimedia.org/wiki/File:ParticleSwarmArrowsAnimation.gif)
* Sur le schéma, nous pouvons voir que, dans un premier temps les particules vont explorer l’espace par rapport à leur position de départ. Dans un deuxième temps, elles vont ensuite majoritairement explorer autour de leur meilleure position et de la meilleure position globale jusqu’à stagner autour de la meilleure position trouvée par l’essaim.
* **Fonctionnement de l’algorithme :**
* Nous allons créer un nombre ***i*** de particules dans un espace à ***n*** dimensions. Ces particules vont être initialisées à des positions choisies soit de manière aléatoire, soit de manière équidistante par exemple. Ces positions que l’on va noter ***X*** vont être définies par ***i*** vecteurs de taille ***n*** tel que pour la position d’une particule :
* ***Xi = (Xi1 , Xi2 , …, Xin )***
* Un plus grand nombre de particules utilisées va dans la majorité des cas, nous fournir de meilleurs résultats au prix d’un temps de calcul plus élevé.
* On va enregistrer ensuite pour chaque particule, la position qui a maximisé ou qui a minimisé la fonction que l’on cherche à optimiser. On notera la meilleure position obtenue par une particule ***Pi*** tel que :
* ***Pi*** = ***(Pi1 , Pi2 , …, Pin )***
* De la même manière, on notera ***Pg*** la meilleure position trouvée parmi l’ensemble des particules :
* ***Pg*** = ***(Pg1 , Pg2 , …, Pgn )***
* Enfin, chaque particule possède un vecteur vitesse qui à un moment ***t***, va lui permettre de changer sa position afin d’explorer la fonction et de se regrouper par la suite vers une même valeur. On notera le vecteur vitesse ***Vi*** d’une particule à un moment ***t*** de la manière suivante :
* ***Vit = (Vit1 , Vit2 , …, Vitn )***
* On calcule le vecteur vitesse d’une particule à un moment ***t+1*** de la manière suivante :
* ***Vit+1 = Vit + r1 x (Pi - Xi) + r2 x (Pg - Xi)***
* Avec ***r1*** et ***r2*** , deux nombres décimaux tirés aléatoirement dans .
* Pour une particule ***i***, le sens et l’intensité du vecteur dépendent du vecteur vitesse au moment ***t***, de sa meilleure position obtenue et de la meilleure position obtenue par l’ensemble de l’essaim. ***r1*** et ***r2*** permettent d’impacter de manière aléatoire l’influence de la meilleure position de la particule et celle de l’essaim (respectivement ***Pi*** et ***Pg***).
* Il existe plusieurs variations de cet algorithme, certains utilisent une valeur d’inertie ou bien des coefficients d’accélération ***a1*** et ***a2*** tel que :
* ***Vit+1 = ( x Vit) + r1 x (Pi - Xi) + r2 x (Pg - Xi)***
* ou
* ***Vit+1 = Vit + a1 x r1 x (Pi - Xi) + a2 x r2 x (Pg - Xi)***
* Ces variations nous permettent d’influer ainsi sur l'accélération du vecteur mais surtout, d’appliquer des poids qui vont influencer sur l’importance des valeurs. Ces valeurs peuvent-être le vecteur vitesse précédent ou la meilleure position obtenue par la particule et l’essaim.
* Si on prend  ***= 1.50***, ***a1 = 0*** et ***a2 = 1*** alors la particule ne sera plus influencée par sa meilleure position obtenue. L’influence de la meilleure position de l’essaim sera la même et enfin l’accélération obtenue par le précédent vecteur vitesse sera augmentée.
* L’influence de ces paramètres est donc importante et il sera nécessaire de mettre des paramètres plus pertinents que ceux utilisés pour cet exemple.

|  |
| --- |
| * On cherche a maximiser une fonction f(x) * **Initialiser** X la position des particules; * **Initialiser** P = X, les meilleurs positions des particules; * **Initialiser** Pg = max(P) la meilleure des positions; * **Initialiser** aléatoirement V la vitesse des particules; * **Tant que** le critère d’arrêt n’est pas atteint :   + **Pour** i allant de 1 à length(X) :     - r1 = rand(0, 1);     - r2 = rand(0, 1);     - V[i] = V[i] + r1 x (P[i] - X[i]) + r2 x (Pg - X[i])     - X[i] = Application du vecteur V[i] à X[i]     - **SI** f(X[i]) > f(P[i]) :       * P[i] = X[i];       * **SI** f(X[i]) > f(Pg) :         + Pg = X[i]; * **Retourner** Pg |

### **Simulated Bee Colony**

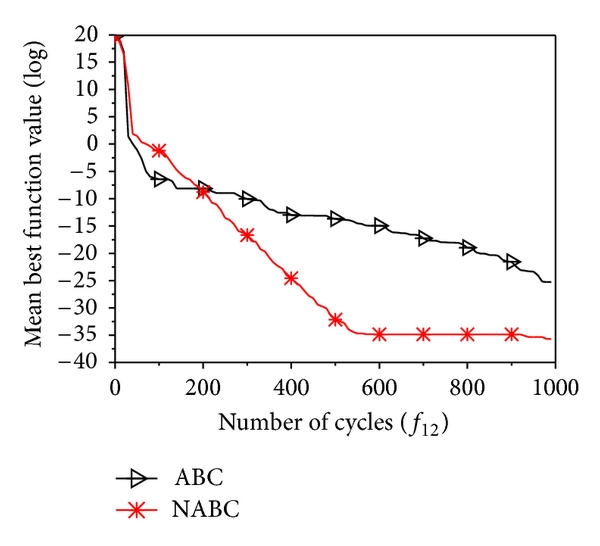
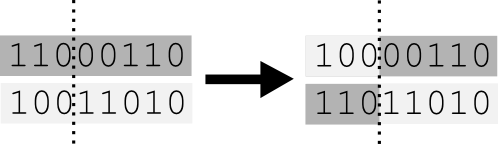
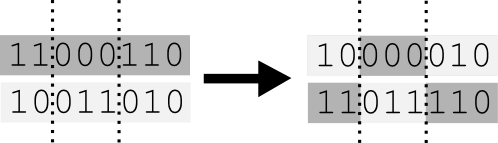
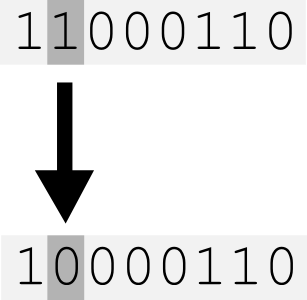
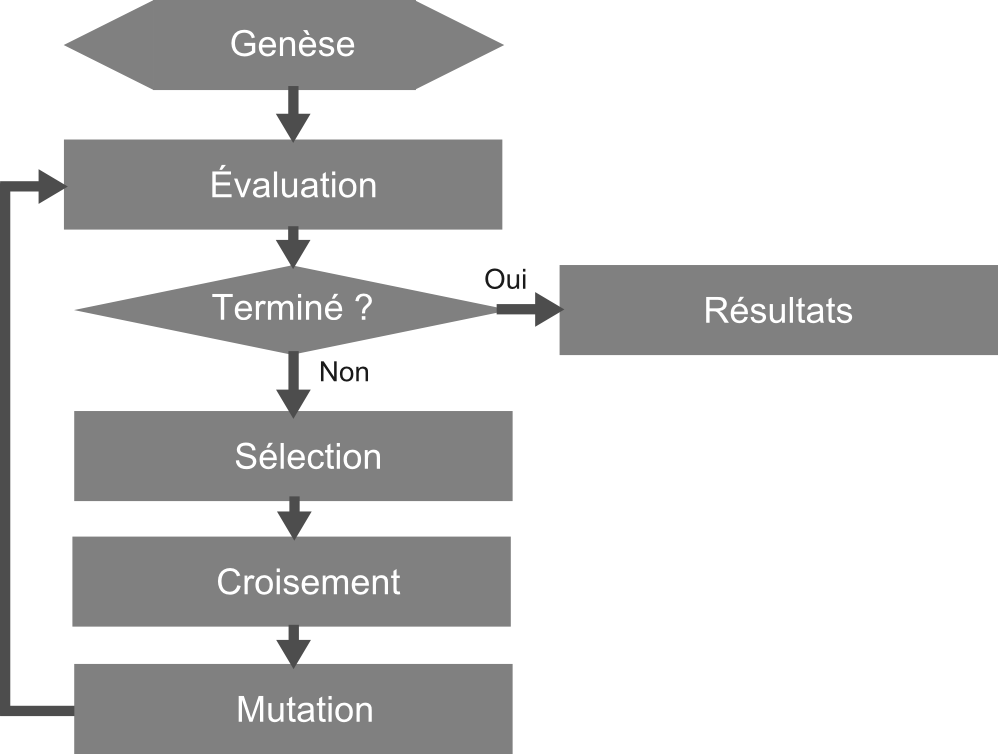
* L’algorithme des abeilles est une méthode d’optimisation inspirée du comportement des abeilles qui vont récolter du nectar. Sachant que nous avons différents types d’abeilles (élites, travailleuses, suiveuses), nous arrivons assez rapidement à un système multi-agents. Celui-ci est utilisé dans des problèmes d’optimisation de fonction continue et combinatoire.
* **Fonctionnement de l’algorithme :**
* Dans un premier temps, nous allons initialiser notre nombre d’abeilles élites (ou scoot), nos abeilles travailleuses, nos autres abeilles, ainsi que le "Patch Size", qui représente la taille du voisinage (qui va diminuer au fur et à mesure de notre itération).
* Les abeilles éclaireuses vont explorer aléatoirement notre domaine de définition et vont permettre de localiser les meilleurs sites. Ces meilleurs sites vont être exploités par nos abeilles élites qui vont appliquer une recherche locale sur nos bons sites. Pendant ce temps, les abeilles éclaireuses vont continuer leurs recherches. De plus chaque abeille classique va générer une solution proche qui va suivre l’équation suivante :
* Xk représente une solution aléatoire, j est un index aléatoire de la dimension entre [1,D] et ***ij ,*** est un nombre aléatoire entre [-1;1]. Vi,j représentera alors une nouvelle “greedy” solution , celle-ci sera uniquement concernée dans le cas où elle est meilleure que son parent, si c’est le cas X sera alors remplacé par V.
* Pendant toute la durée de l’algorithme les meilleurs sites seront continuellement exploités tandis qu’une partie des abeilles éclaireuses va continuer de rechercher de meilleurs sites. Nous allons donc avoir une sélection des meilleurs sites car le nombre de “Meilleurs sites” doit toujours être inférieur aux nombres de sites "Classiques". De plus, une fois que les abeilles seront de retour à la ruche elles feront une “danse”.
* Cependant, l'algorithme fait également une sélection : si on dépasse un certain nombre d’itérations sur un des sites, ce site sera considéré comme épuisé. Il sera alors abandonné et les abeilles seront donc attribuées à un nouveau site.



***Schema représentant le fonctionnement de l’algolrithme ABC***

* **Pseudo code de l’algorithme ABC**

|  |
| --- |
| * **Initialiser** ProblemeSize, BeeNum et EliteBee. * **Initialiser le critère d’arrêt** * **Tant que** le critère d’arrêt n’est pas atteint : * Evaluation de la population * Récupérer la meilleure abeille dans la population * On diminue notre patchSize * **Pour tous les sites**    + **Si** le nombre d’abeilles est inférieur à i   + Les abeilles recrutés devienne des abeilles Elite   + **Sinon**   + Les abeilles recrutés devienne d’autres abeilles   + On assimile nos abeilles à des quartier   + On récupére les meilleures solutions de la zone * On réduit notre PatchSize * Pour les abeilles restante :   + On crée de nouvelle abeilles pour la prochaine génération * Population <- NextGeneration * **Retourner** BestBee |

* Comme nous avons pu l’évoquer précédemment, l’algorithme des abeilles peut être utilisé dans plusieurs problématiques :
* Les problèmes de planifications,
* Les problèmes de VRP (Vehicle Routing Problem),
* La segmentation d’image,
* La classification et l'entraînement de réseaux de neurones.
* De plus, pour améliorer les performances de l’algorithme, nous pouvons retirer les abeilles suiveuses.
* Cependant, il existe un algorithme appelé NABC (New Artificial Bee Colony) qui est une combinaison entre l’algorithme ABC (vu durant ce chapitre) et l’algorithme “differential evolution”. L’évolution différentiel (DE) est également un algorithme d’optimisation.
* **Son fonctionnement est le suivant** **:**
* Dans un premier temps, nous allons choisir un ensemble de solutions. Sur cet ensemble de solutions, nous allons appliquer des recombinaisons ce qui va nous permettre d’avoir de nouvelles solutions. Souvent basés sur la différence pondérée entre différents membres de cette population, ces membres sont sélectionnés aléatoirement. Cet état de fait va créer des changements dans la population par rapport à l’ancienne population.
* **Concernant le NABC** **:**
* Nous allons stocker les meilleures solutions dans un de nos essaims, soit celui avec qui nous allons travailler actuellement. Ce qui permet au final d’obtenir une meilleure convergence.
* 
* Mais nous ne verrons pas celui-ci pendant ce cours.
* Il existe également des métaheuristiques d’optimisation qui quant à elles ne sont pas distribuées. Nous pouvons prendre l’exemple des algorithmes génétiques.
* Ces algorithmes se reposent en général sur la théorie d’évolution des espèces (découverte par Darwin en 1859).
* Cette théorie repose sur trois principes qui sont les suivants :
* **La** **variation** c’est à dire que chaque individu sera unique (une solution unique) et que ces différences lui permettront ou non d'être conservé pendant le processus de sélection,
* **L’ adaptation** des meilleurs individus (meilleure survie des plus adaptés) qui seront donc conservés,
* **L’hérédité**: certaines caractéristiques doivent être héréditaires pour être conservées par leur descendance. Cependant, les algorithmes génétiques vont uniquement conserver trois opérateurs qui sont les suivants :
* **La sélection** : Choix des individus (solutions) les plus adaptés
* Il existe bien évidemment différentes méthodes de sélection. Comme évoqué juste avant, il s’agit de choisir les solutions les plus proches d’une convergence globale et non locale. Nous avons dans un premier temps la méthode de la sélection uniforme. Celle-ci a pour but de choisir aléatoirement les individus sans aucune intervention.
* Ensuite, nous avons la sélection par tournoi. Dans cette méthode, nous allons créer des paires d’individus. Les paires avec les meilleurs scores d’adaptation sont alors sélectionnées.
* Il existe également la méthode de sélection de probabilité proportionnelle à l’adaptation. Le fonctionnement de cette méthode de sélection est le suivant : chaque solution a une probabilité d’être choisie qui sera plus importante en fonction de son adaptation au problème.
* Enfin, il y a la sélection par rang. Cette méthode va uniquement sélectionner les meilleures solutions (avec le score le plus important).
* **Le croisemen**t : Nous allons reproduire des individus sélectionnés dans notre population afin de mélanger leurs caractéristiques.
* Concernant le croisement, il y a uniquement deux méthodes :
* Le **simple enjambement** va consister à fusionner les spécificités d’un individu et d’un autre en fonction d’un pivot. Par exemple, avec le simple pivot, on aura ceci :
* 
* Le **doublement enjambement** quant à lui va être presque semblable au simple enjambement sauf que nous allons avoir deux pivots :
* 
* On aura alors, les enfants issus de notre reproduction qui vont alors subir une ou plusieurs mutations.
* **La mutation** : Modification aléatoire d’une ou plusieurs caractéristiques d’un de nos individus
* La mutation va être la modification d’une particularité d’un individu selon une variable X ; cet opérateur va suivre la théorie expliquée plus tôt, et va nous permettre d'éviter la convergence locale.
* 
* Les fonctionnements que vont suivre les algorithmes génétiques sont les suivants :
* 
* Le schéma précédent décrit le fonctionnement d’un algorithme génétique.
* Dans un premier temps, nous allons créer une population d’individus aléatoires.
* Ensuite, nous allons étudier et analyser nos solutions pour conserver uniquement celles qui sont valables. Nous pourrons ainsi déterminer la solution grâce au score d’adaptation via des fonctions déterminant le coût ou les erreurs.

### **Bat Algorithm**

* Le BAT algorithme (BA) est une métaheuristique d'optimisation développée par Xin She Yang en 2010. Cet algorithme va se baser sur l’utilisation des echolocations. Ces echolocations sont énormément utilisées par les plus petites chauves-souris afin d'identifier leurs proies, repérer les obstacles, définir des distances, mais également pour se diriger car elles ont une faible vision.
* Cet algorithme est énormément utilisé car il présente de nombreux avantages. En effet, dans un premier temps, il a une convergence assez rapide dès le début de l’algorithme que ce soit au niveau de l’exploration ou celui de l’exploitation.
* L'algorithme se divise en plusieurs étapes. Dans un premier temps nous allons initialiser notre population de chauve-souris. Il est à noter que cette initialisation va devoir respecter certaines règles qui sont les suivantes : notre population sera générée aléatoirement et chacune des chauves-souris sera alors représentée par un vecteur réel avec une dimension D. Pour définir notre population nous utiliserons l’équation suivante :
* *𝐗𝐢 = 𝐗𝐦𝐢𝐧𝐣 + 𝐫𝐚𝐧𝐝 𝟎, 𝟏 (𝐗𝐦𝐚𝐱 − 𝐗𝐦𝐢𝐧)*
* Où Xmin et Xmax définissent les bornes de notre dimension.
* Par la suite, à chaque étape (i), nous allons modifier les positions (x) des différentes chauves-souris ainsi que leurs vitesses (V). De plus, parmi l’ensemble des solutions, il existe une solution optimale (S). Cette solution sera obtenue via des comparaisons avec les solutions de toutes nos autres chauves-souris. Pour définir les règles évoquées juste avant nous avons les équations suivantes :
* *fj = f𝐦𝐢𝐧 + f𝐦𝐚𝐱 − f𝐦𝐢𝐧 𝛃*
* Ici 𝛃 représente un vecteur généré aléatoirement.
* 𝐕j i = 𝐕j i−𝟏 + Sj i − 𝐗 ∗ 𝐟j
* Une fois que nous avons obtenu notre meilleure solution, nous allons réattribuer de nouvelles solutions à tous nos individus N. qui seront définis par l’équation qui suit.
* E sera une variable aléatoire comprise entre -1 et 1, tandis que Ai est la moyenne des volumes de nos individus au moment I.
* Enfin, nous allons mettre à jour le volume et le taux de pulsations ( ‘R’ ) à chaque passage si de meilleures solutions sont toujours existantes et que donc, nos chauves-souris peuvent encore se déplacer vers la proie (représentée par notre solution) . En effet, au cours de l’algorithme, notre intensité va diminuer tandis que la valeur de R quant à elle va augmenter. Si notre volume atteint son minimum alors, c’est que nous avons une de nos solutions. Les équations suivantes présentent ce que nous venons d’expliquer.
* Comme vous avez pu le comprendre, dans cet algorithme nos hyperparamètres vont être très importants car il est très sensible à ceux-ci.
* Pour résumer, l’ensemble des explications que nous venons de faire sur l'algorithme des chauves-souris, le pseudo code de cet algorithme est présenté ci-après.
* **Pseudo code du Bat Algorithm**

|  |
| --- |
| * **Définition de notre fonction objective** * **initialisation** de notre population de chauve-souris xi et vi * **Définition** des fréquences de pulsation de fi au moment xi * **Initialisation** des fréquences de pulsation ri et de volume AI * **Tant que** i < nbIter:   + Génération de nouvelle solution en modifiant les fréquences, la vitesse et les positions.   + Si (rand > ri)     - On sélectionne aléatoirement une des meilleures solutions     - On génère une solution local autour de cette solution via des * déplacements aléatoires.   + Si ( rand < Aiter et f(xi) < f(xbest) )   + On récupère la nouvelle solution   + On augmente Ri et diminue Ai   + On définie les meilleurs individus et les stock dans nos meilleures solutions * **Retourner** BestSoluce |

* Maintenant que nous connaissons le fonctionnement de certains algorithmes d'intelligence distribuée et d’algorithmes semblables non distribués, nous allons pouvoir faire une comparaison entre ces deux types d’algorithmes que nous verrons par la suite.

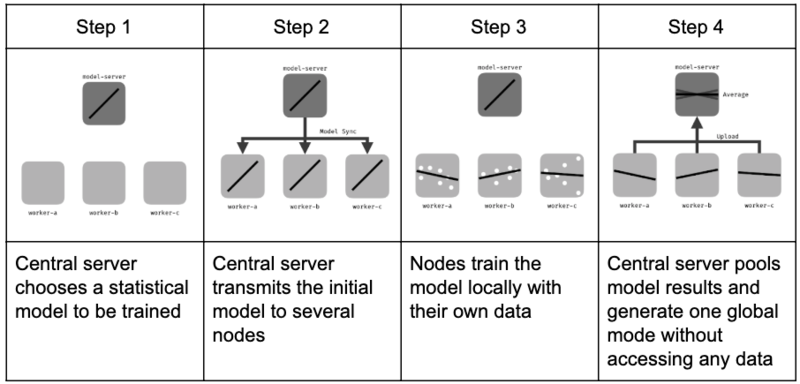
# La place des systèmes multi-agents dans les réseaux de neurones

* L’utilisation de systèmes multi-agents est aussi possible dans le cas de tâches de Machine/Deep Learning. Il s’agit d’une branche de l’intelligence distribuée et du Machine Learning nommée Multi-agent Learning. Cependant, contrairement à ce que nous avons présenté précédemment, il ne s’agit pas d’algorithmes qui seront utilisables par un modèle mais d’architectures logicielles permettant d’utiliser plusieurs modèles.
* Dans un scénario de Machine Learning classique, nous avons un unique modèle (donc un unique agent) qui va utiliser des données en entrée afin d’ajuster ses poids pour fournir une prédiction la plus précise possible. Ce modèle est donc l’unique responsable du résultat obtenu car il est le seul à donner une prédiction et à s’entraîner sur des données.
* 
* Dans un système de multi-agents Learning on utilisera plusieurs modèles, généralement x agents workers et au moins un agent master. Les workers sont des modèles qui utilisent un échantillon des données différent des autres workers. Chaque modèle va effectuer de manière autonome, le même travail qu’un modèle de machine learning traditionnel. L’agent master va récupérer des logs (résultats, poids ou données traitées par exemple) des différents workers et va avoir pour rôle de les utiliser au mieux pour fournir un résultat globalement meilleur qu’un agent worker unique.
* 
* Le schéma ci-dessus décrit le modèle présenté plus haut mais il existe de nombreuses configurations possibles dans une architecture multi-agents.
* Comme pour les algorithmes vus précédemment, nous utiliserons donc plusieurs individus/agents qui vont apprendre et explorer de manière autonome, puis, qui vont communiquer ensemble ou via un intermédiaire afin d’obtenir des résultats qu’ils n’auraient pas pu atteindre de manière autonome. Il est fortement possible qu’un worker puisse obtenir de meilleurs résultats que le master pour une entrée mais dans l’ensemble, le master va obtenir de meilleurs résultats que n’importe quel worker.
* Pour des tâches complexes, comme nous pourrons le voir par la suite avec AlphaStar, les systèmes multi-agents permettent d’avoir de très bons résultats qui ne pourraient pas être obtenus par un système avec un seul agent en autant de temps. Bien sûr, étant donné que plusieurs agents travaillent simultanément (bien que pas nécessairement) ce gain de temps se fait au détriment des ressources nécessaires pour effectuer les calculs et prédictions.
* Il existe plusieurs types de Multi-agent Learning :

### Distributed Machine Learning & Distributed Deep Learning

* Afin d’avoir un très bon modèle et qu’il soit performant, nous allons avoir besoin d’énormément de temps ce qui est dû aux nombreuses données traitées. C’est à ce moment-là que va apparaître le machine learning distribué et le deep learning distribué. Ces deux méthodes vont nous permettre d'entraîner nos modèles sur plusieurs machines, disques ou serveurs et tout cela en parallèle. En effet, en utilisant ces méthodes, nous allons pouvoir améliorer nos modèles que ce soit sur les performances ou bien sur la précision.
* Cette méthode se divise en trois sous-méthodes, qui sont les suivantes :
* La méthode du **parallélisme** des données :
  + Cette technique va plus ou moins avoir le même fonctionnement que HDFS avec un fichier lambda. En effet, nos données peuvent être divisées en différentes partitions indépendantes les unes des autres. Nous aurons ainsi autant de partitions que de ressources disponibles pour nos calculs et chacun des nœuds (des ressources de calculs) va faire le traitement sur sa propre partie. Ensuite, chacune de nos ressources actualise leurs états afin que nos algorithmes puissent converger vers une version optimale. Nous allons donc gagner en temps avec cette technique mais le coût en ressources sera alors plus important.
* La **technique** pour le modèle
  + La technique du parallélisme de modèle va plus ou moins avoir le même fonctionnement que celle des données. En effet, au lieu de redistribuer nos données sous forme de partition dans différentes ressources, nous allons redistribuer ces divisions sur notre modèle comme évoqué auparavant. Cependant, dans ce cas, les partitions vont être exécutées au même moment, mais sur le même ensemble de données. L’avantage que nous allons obtenir, si on compare à la méthode précédente, c’est que nous n'avons pas besoin de synchroniser toutes nos ressources.
* Pour finir, nous avons la méthode **hybride** qui est la combinaison des deux méthodes précédentes, ou la moitié de nos ressources sera utilisée pour une des méthodes et l’autre moitié sera utilisée pour la seconde méthode. Nous avons donc pu voir que le machine learning distribué et le deep learning distribué sont des méthodes de machine learning multi-agents.

### Federated Learning

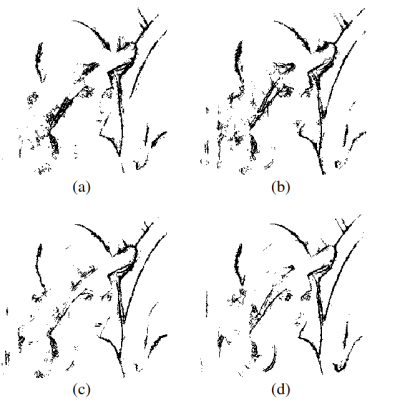
* [](https://commons.wikimedia.org/wiki/File:Federated_learning_process_central_case.png)
* Comme on peut le voir sur le schéma ci-dessus, le Federated Learning est un système multi-agents composé de x modèles workers et d’un modèle master. Chaque workers va récolter et travailler sur ses propres données (celles qu’il a récoltées). Une fois que le modèle worker a fini son entraînement, il va envoyer ses résultats (poids des neurones et résultats) au modèle master. Ce modèle va s’adapter en fonction des résultats obtenus par ses workers. La différence entre un système multi-agents et le Federated Learning provient de la confidentialité des données. En effet, les données restent sur les workers et le master n’y a pas du tout accès.
* Ensemble Learning
* L’ensemble learning est une méthode qui va avoir pour but de combiner différents algorithmes afin d’améliorer notre modèle. Grâce à celui-ci, nous pouvons avoir des niveaux de précision assez importants avec des résultats bien meilleurs que si nous avions utilisé les algorithmes séparément. Cependant, en fonction des résultats que nous attendons ou de nos besoins (dans le cas où notre but serait d’avoir des résultats homogènes), nous utiliserons alors un seul et même type d'algorithme. Cependant, si notre but est d’avoir des résultats hétérogènes, il sera alors judicieux d’utiliser différents types d’algorithmes. Comme vous avez pu le comprendre, l’ensemble learning se base sur une méthode d'assemblage de modèles, où nous allons utiliser une méthode qui consiste à mettre en commun plusieurs algorithmes pour obtenir de meilleurs résultats (prédictions).
* Cependant, il existe deux types de méthodes ensemblistes (utilisées par l’ensemble learning). En effet, elles peuvent être séquentielles ou parallèles. Dans les premier cas, les algorithmes seront entraînés à la chaîne, c'est-à-dire, que les algorithmes vont apprendre des erreurs passées et la fusion (ou combinaison) ne sera faite qu’à la fin de l'exécution. Pour que ces erreurs soient prises en compte, nous allons devoir jouer sur les poids de ces différentes erreurs. En effet, elles sont élevées pour le modèle qui suit et ainsi de suite.
* Dans la méthode parallèle, les modèles seront entraînés en même temps et c’est seulement à la fin de cette exécution que l’observation des différences sera faite et exploitée.
* Ce type de méthode pourrait alors être utilisée pour combiner des algorithmes d’optimisation par exemple le PSO et l’ABC.

# 

# Cas réels d’utilisation des algorithmes présentés

### **Utilisation de Ant Colony Optimization**

* Au début de ce cours, nous avons pu voir l’algorithme Ant Colony Optimization qui imitait le comportement de fourmis au sein d’une colonie pour accéder à un plus court chemin. Cet algorithme est utilisé sur plusieurs cas réels, le plus évident étant sur des problèmes de routing de véhicules tels que le Capacitated vehicle routing problem ou le Multi-depot vehicle routing problem. En effet, l’utilisation de l’ACO permet d’obtenir de bons résultats tout en restant peu coûteux mais peut également être modifiée pour rendre les résultats encore meilleurs. C’est dans le cadre de la résolution du multi-depot vehicle routing problem qu’un improved ant colony optimization a été créé et qui permet, tout en restant sur de mêmes bases, de donner des résultats plus qualitatifs et de réduire le coût en performance de ce dernier.
* Cet algorithme est également utilisé dans le processing d’images notamment dans l’edge detection et l’edge linking. Dans le cas de l’edge detection, on fait chercher les fourmis dans un espace de recherche constitué de nœuds et de bords. La probabilité d’une transition de pixels est reflétée par le fait qu’une fourmi bouge d’un nœud à un autre. Les valeurs de phéromones sont stockées dans une matrice et mises à jour à chaque recherche. Le but final est que la matrice de phéromones constitue les bords détectés dans l’image. Avec cette utilisation de l’ACO on obtient de bons résultats d’edge detection



*  
* Échantillons des résultats des recherches d’Anna Veronica Baterina et Carlos Oppus
* Comme dernier exemple, on peut parler de l’utilisation de l’ACO dans ce qui touche le réseau. Dans ce cas, il est nécessaire d’acheminer les paquets par différents chemins possibles avec une destination fixe. Cela semble propice à l’ACO qui permettrait de définir quel chemin choisir et avoir du contrôle sur d’éventuelles congestions. C’est certainement ce que ce sont dit de nombreux ingénieurs avant de l’appliquer ! Comme pour le cas du multi-depot vehicle routing problem, des ingénieurs de l’université de Pékin sont partis sur un improved ACO qui a pour principe que chaque fourmi évalue la qualité de la route pour ses phéromones et permet donc d’acheminer plus rapidement différents paquets tout en réduisant le coût en communication.
* Ces exemples ne donnent qu’une petite idée des utilisations de l’ACO. Il peut être utilisé dans des situations différentes et donne des résultats intéressants dans la majorité des cas.

### L’utilisation de Particle Swarm Optimization

* Le deuxièmealgorithme dont nous avons parlé est le PSO. Ce dernier possède également des utilisations dans des cas réels. Par exemple, dans le domaine d’analyse de signaux, un problème complexe est la détermination de fréquences et les amplitudes de fonctions dites “quasi-périodiques”. En adaptant PSO à ce problème on obtient de meilleurs résultats qu’avec des méthodes plus classiques comme l’analyse de Fourier. Le principe est que chaque particule du système représente un signal à trois fréquences et six paramètres avec des valeurs aléatoires et sur lesquelles on va appliquer une fonction de fit.
* Un autre exemple est l’optimisation des poids d’un réseau de neurones. Dans un réseau de neurones nous avons une entrée de données, des couches cachées constituées de poids qui détermine la donnée de sortie. Le but est d’optimiser ces poids pour obtenir le meilleur résultat possible en sortie. Des tests ont été faits en utilisant un réseau de neurones sur un pendule où l’on doit déterminer le mouvement dudit pendule en partant d’un angle supérieur à ½ **π**. En optimisant les poids avec PSO on obtient une précision de 91,5%.

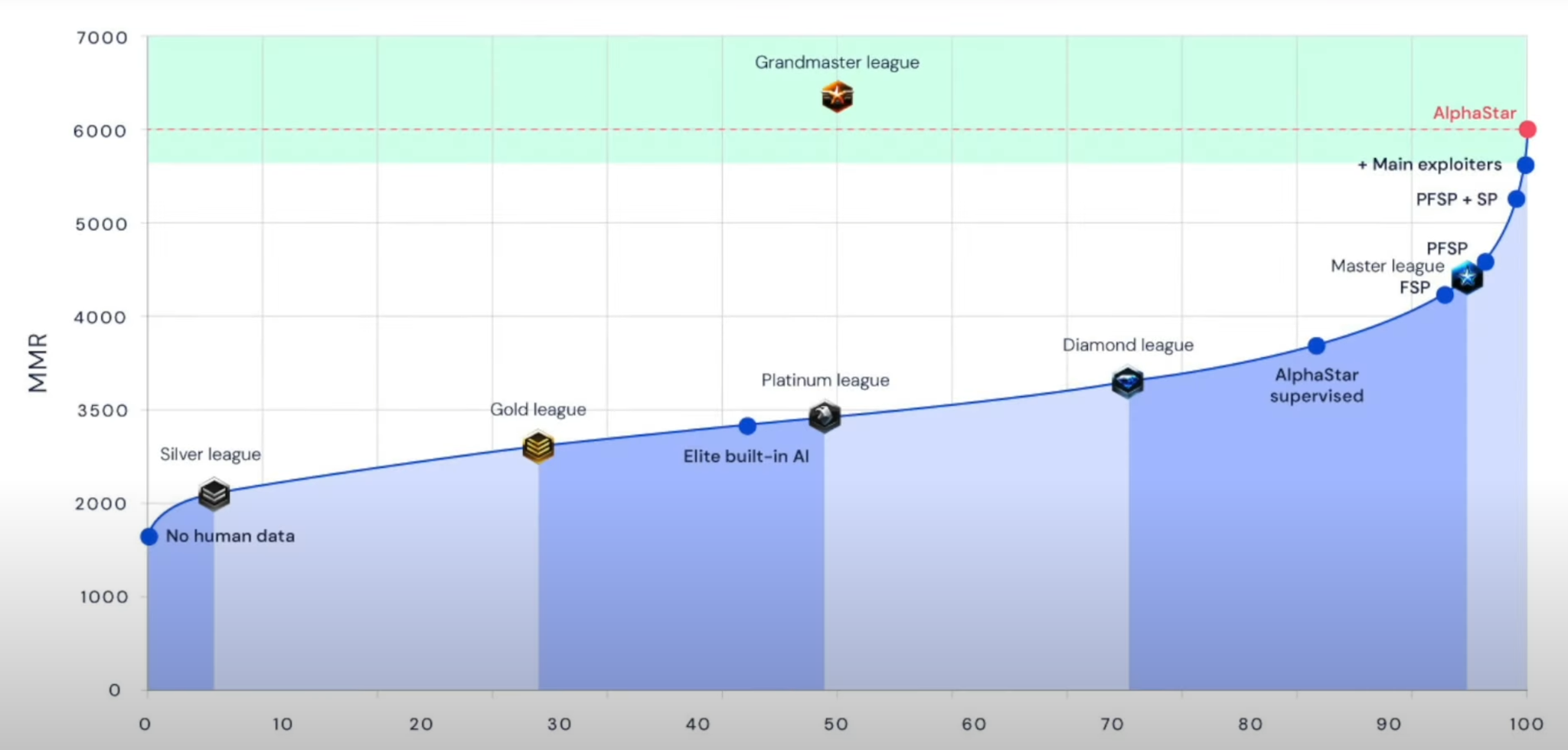
### L’utilisation de Simulated Bee Colony

* Comme évoqué précédemment, l’Artificial Bee Colony est un algorithme utilisé pour résoudre les problèmes d’optimisation combinatoire. Cependant, cet algorithme est également utilisé pour différents cas qui sont les suivants :
* Dans un premier temps, nous pouvons évoquer l'utilisation de l’algorithme des abeilles pour la planification des déplacements d’un robot (notamment utilisé pour le robot gynames). L'intérêt d’utiliser cet algorithme dans ce cas-là, est qu'avec les différentes ruches et sources de nourritures, notre robot pourra alors commencer à planifier différents chemins et finir par choisir le chemin optimum.
* Ensuite, nous pouvons évoquer, l'utilisation d'algorithmes des abeilles pour les problèmes d’ordonnancement de jobs. En effet, le principe est de trouver un ensemble de tâches avec nos machines qui sont représentées par les abeilles, ce qui permettra de minimiser notre temps de production ou d'exécution de notre job. Dans ce cas-là, chacune de nos “configurations” serait alors un chemin de notre ruche jusqu’à l’une des sources de nourritures. Concernant notre fonction de fitness, elle sera représentée par le temps d'exécution (dans l’algorithme représenté par la source de nourriture avec la distance). Nos meilleures fonctions de fitness représenteront ainsi nos chemins les plus rentables, c’est-à-dire les exécutions réalisables et les plus rapides.
* L’algorithme est également utilisé pour la résolution des distances et d'angles par rapport au soleil (Coordonnées polaires). Cet algorithme est également utilisé dans de nombreux autres domaines comme la conception mécanique, le groupement de données et la linéarisation des composants mécaniques.

### L’utilisation de l’algorithme des chauves-souris

* Le dernier algorithme que nous avons étudié est l'algorithme des chauves-souris. Celui-ci est utilisé dans de nombreux de cas réels. Dans un premier temps, il peut être utilisé pour la résolution du problème du voyageur. Bien évidemment, nous pouvons également l’utiliser pour tout ce qui est chargement et la gestion de fréquence.
* Il peut également être utilisé pour tout ce qui est classification, le data mining ainsi que les capteurs wifi étant donné que ces cas se rapprochent fortement du comportement des chauves-souris (utilisation d’ondes etc) ce qui assez simple à réaliser avec l’algorithme des chauves-souris. Il est également utilisé pour tout ce qui est reconnaissance d’image et pour les seuillages d’images sur différents niveaux.

### Multi-Agent Learning - AlphaStar (Startcraft II)

* AlphaStar est l’IA de deepmind qui a réussi à battre des joueurs professionnels d’eSport du jeu Starcraft II. Cette dernière utilise, entre autres, du multi-agent learning et réussi à jouer à haut niveau en un contre un dans les mêmes contraintes qu’un joueur, c’est à dire, avec et contre toutes les races disponibles dans le jeu (Zerg, Protoss, Terran) qui ont chacune des spécificités, un gameplay différent et donc un niveau de complexité différent. Toutes les parties ont été faites sur le client officiel de StarCraft II, sur les serveurs de battle.net (client pour accéder au jeu) et ont leurs replays disponibles directement sur le site de DeepMind. AlphaStar atteint donc un rang GrandMaster (au-dessus de 99.8% des joueurs) sur les trois races jouables. Grâce à la collaboration de Blizzard (créateurs du jeu), l’équipe de DeepMind a pu consulter le code source du jeu pour pouvoir commencer leur travail sur AlphaStar et commencer ainsi à implémenter la base de tout jeu de versus : réussir à battre son adversaire en anticipant et exploitant ses failles tout en analysant ses propres failles pour s’améliorer.
* L’exploration de la carte était aussi un point important à prendre en compte pour DeepMind. En effet, sur StarCraft II, comme sur de nombreux RTS (Real-Time Strategy), un brouillard de guerre est mis en place, c’est à dire qu’on ne dispose que d’informations limitées sur la carte et qu’on obtient la vision sur ce brouillard qu’en construisant des bâtiments ou en déplaçant des unités. Il faut donc explorer la carte à l’aide d’unités pour obtenir des informations cruciales telles que l’emplacement de l’ennemi, de son armée s’il en a une, etc… L’équipe de DeepMind a déterminé à 1026 possibles actions pour chaque agent à chaque étape. Pour passer ce point bloquant, ils ont employé un mélange d’imitation learning, de réseaux de neurones avancés et des techniques de language modeling. Cela leur a permis d’obtenir une policy qui permet à l’IA de jouer mieux que 84% des joueurs.
* C’est à partir de cette expérience et des données des joueurs et de parties qu’AlphaStar n’a fait que progresser en commençant en dessous du rang Silver (meilleur que 9.5% des joueurs) pour atteindre celui de GrandMaster
  + - 
  + Montée en MMR (MatchMaking Ranking) d’AlphaStar sur SC2 extrait de [la vidéo des résultats de DeepMind](https://youtu.be/xP7LwZxq0ss)

### Federated Learning - GBoard (Google)

* Le Federated Learning est une architecture conçue par Google et dont l’article a été publié en 2017 ([lien de l’article](https://ai.googleblog.com/2017/04/federated-learning-collaborative.html)).
* Google a décidé d’utiliser cette technologie sur GBoard, un outil intégré aux messages sur tous les smartphones. GBoard est un clavier intelligent permettant d’associer une recherche à un lien pertinent. Il peut s’agir d’un site web, d’une vidéo ou bien encore d’une localisation géographique.
* Le fonctionnement du Federated Learning associé à GBoard est le suivant :
* Le serveur (ou master) va envoyer aux smartphones utilisant GBoard, le modèle à utiliser pour obtenir le meilleur résultat pour la recherche,
* Le modèle va se mettre à jour avec les données déjà présentes sur le smartphone (obtenues via les précédentes utilisations de GBoard).
* Lorsqu’un utilisateur va utiliser GBoard sur son smartphone pour envoyer un lien, la recherche va être associée au lien envoyé pour avoir un couple donnée - prédiction qui sera stocké sur le smartphone,
* Le smartphone va alors utiliser les nouvelles données pour mettre à jour son modèle avec une version minimaliste de TensorFlow,
* Les différences entre le modèle de base et le modèle mis à jour avec ses données vont alors être envoyées au serveur,
* Le serveur va alors mettre à jour son modèle en fonction des différences envoyées par les différents smartphones.
* Les données sont donc privées car elles restent sur le smartphone de l’utilisateur. Il est à noter cependant que chaque smartphone apprend des autres grâce au serveur qui fait l’intermédiaire entre l’ensemble des smartphones.

# Comparaison des algorithmes d’intelligence distribuée VS non distribuée

* Nous avons donc vu de nombreux algorithmes distribués mais pour chacun de ceux que nous avons décrit, il existe d'autres algorithmes non distribués qui permettent d’arriver au même objectif. Pour comparer leur efficacité, nous allons maintenant confronter ces algorithmes sur plusieurs points.
* Tout d’abord, il est important de mettre en place différents types de problèmes et sur des données plus ou moins importantes.
* Ensuite les algorithmes seront jugés sur :
* Leur vitesse d’exécution,
* La place en mémoire,
* Les meilleurs résultats obtenus,
* La moyenne des résultats obtenue,
* Vous trouverez ci-après les algorithmes qui seront utilisés pour faire notre comparaison :

|  |  |  |
| --- | --- | --- |
|  | * Distribué | * Non distribué |
| * Optimisation locale | * Particles Swarm Optimization, Artificial Bee Colony | * Recherche Local Naïve, Recul Simulé |
| * Recherche du meilleur chemin | * Ant Colony Optimization | * Dijkstra, Bruteforce A\* |

### Comparaison de l’ACO vs Bruteforce A\*

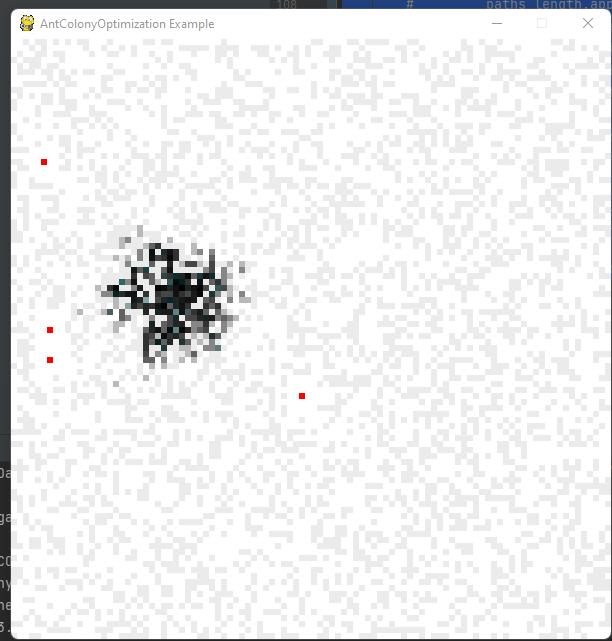
* Afin de comparer ces deux algorithmes, nous avons créé un environnement similaire à un problème du voyageur (problème d’optimisation de chemin le plus court entre multiples points).
* C’est donc dans cet environnement que nous avons décidé de comparer le temps de convergence pour trouver le chemin le plus court entre deux algorithmes :
* Ant Colony Optimisation que nous vous avons présenté en amont,
* une version modifiée de l’algorithme A\*.
* En effet, l’algorithme A\* ne permettant que de trouver le chemin le plus court entre deux points uniquement, nous avons dû construire autour de cet algorithme une fonction de "Brute Force" permettant pour un ensemble de points de trouver la combinaison la plus courte pour compléter ce chemin.
* **Pseudocode de l’algorithme Bruteforce A\***

|  |
| --- |
| * **Initialiser** astar\_paths = [] **Initialiser** all\_possible\_paths = itertools.permutation(**[A, B]**, r=2)   **[[A,B], [B,A]]**   * *# Liste de l’ensemble des paths possible* * **Pour tous** idx, possible\_path **in** all\_possible\_paths: * **for** i **in range(len(**possible\_path**)-1):** * **Initialiser** start\_point = possible\_path[**i**] * **Initialiser** end\_point = possible\_path[**i+1**] * **astar\_paths[idx][i] = astar(matrix, start\_point, end\_point)** * *# Triage des paths possible par longueur* * **astar\_paths = astar\_paths.sort(key=len)** * *#Affichage du path le plus court* * print(f’ shortest path found : {**astar\_paths[0]**}’) |

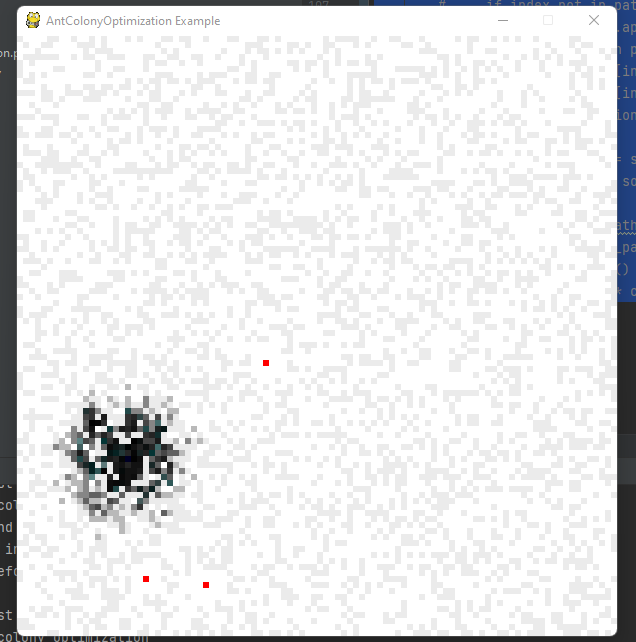
* Afin toujours de comparer ces deux algorithmes, nous avons créé différents “test case” dans lequel nous faisons varier certaines variables telles que :
* Le nombre de points à relier,
* La distance minimale entre ces points,
* La distance maximale entre ces points.
* **Voir ci-dessous la présentation de nos résultats :**

# 

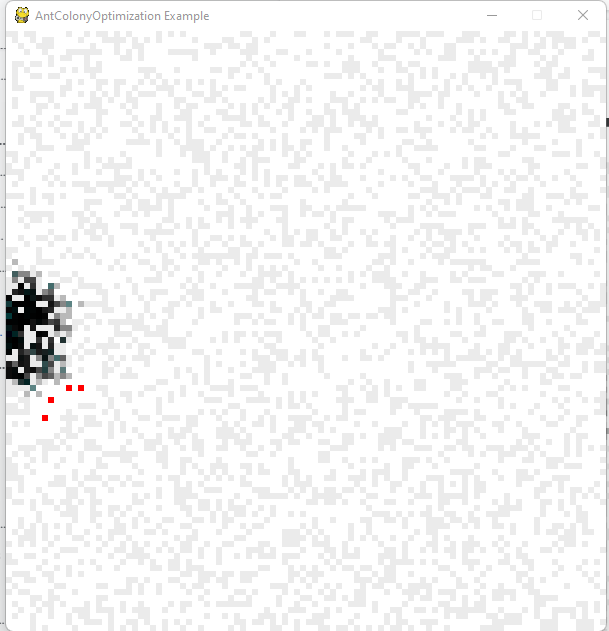
|  |
| --- |
| * Comparison Test ACO vs Bruteforce A\* : 2 Points TSP [min\_distance: 20, max\_distance:25] Starting ant colony optimization Ants have found the correct path in 37 iterations. ACO completed in 2.0311238765716553 Starting Bruteforce A\* The shortest path found **by** bruteforce A\* : {'id': 0, 'cost': 22, 'iterations': 22} in 45 total iterations. Bruteforce A\* completed in 0.0029981136322021484 |

* Comme attendu, l’utilisation de l'algorithme A\* est bien plus rapide pour trouver le chemin le plus court entre deux points comparé à l’algorithme ACO.
* L’utilisation de l’ACO dans un problème du voyageur a deux points est de l’overkill.
* 

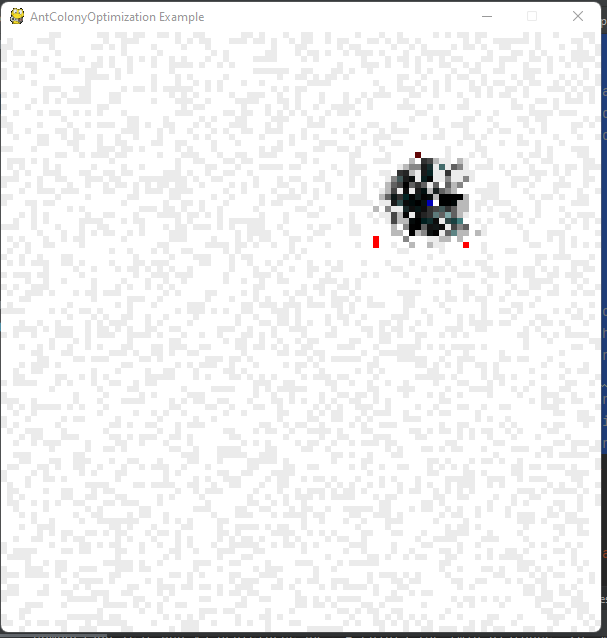
|  |
| --- |
| * Comparison Test ACO vs Bruteforce A\* : 5 Points TSP [min\_distance: 20, max\_distance:25] Starting ant colony optimization Ants have found the correct path in 142 iterations. ACO completed in 6.106964349746704 Starting Bruteforce A\* The shortest path found **by** bruteforce A\* : {'id': 38, 'cost': 44, 'iterations': 44} in 14976 total iterations. Bruteforce A\* completed in 2.418032169342041 |

* Pour ce test, nous pouvons constater que l'écart du temps d'exécution entre les deux algorithmes commence à se réduire.
* La complexification du problème (augmentation du nombre de points à relier) laisse envisager que l’ACO est bien plus efficace sur des problèmes plus complexes.
* 

|  |
| --- |
| * Comparison Test ACO vs Bruteforce A\* : 4 Points TSP [min\_distance: 20, max\_distance:25] Starting ant colony optimization Ants have found the correct path in 52 iterations. ACO completed in 2.6850528717041016 Starting Bruteforce A\* The shortest path found **by** bruteforce A\* : {'id': 0, 'cost': 44, 'iterations': 47} in 1524 total iterations. Bruteforce A\* completed in 0.14104366302490234 |

* La théorie développée plus haut se confirme ici. La complexification du problème donne un clair avantage à l'algorithme Bruteforce A\* en temps d'exécution.
* 

|  |
| --- |
| * Comparison Test ACO vs Bruteforce A\* : 5 Points TSP [min\_distance: 10, max\_distance:15] Starting ant colony optimization Ants have found the correct path in 39 iterations. ACO completed in 1.444000244140625 Starting Bruteforce A\* The shortest path found **by** bruteforce A\* : {'id': 9, 'cost': 41, 'iterations': 41} in 9360 total iterations. Bruteforce A\* completed in 1.201052188873291 |

* Le résultat de ce test est intéressant car c’est le premier cas environnement ou l’ACO est légèrement plus rapide en temps d'exécution que le Bruteforce A\* alors que nous avions obtenu un résultat différent dans le test **numéro trois** (cinq points TSP, écart entre les points réduits)
* 

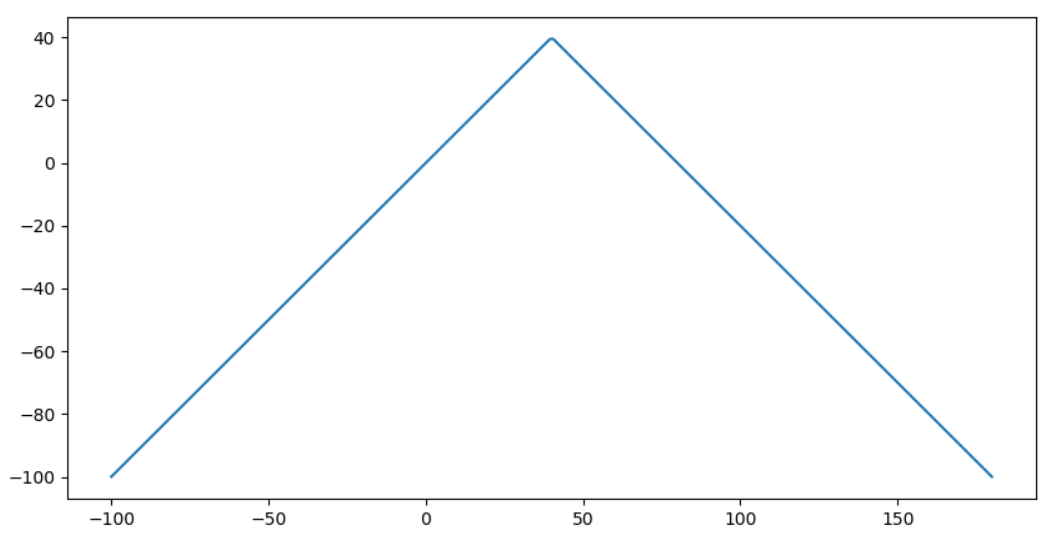
|  |
| --- |
| * Comparison Test ACO vs Bruteforce A\* : 7 Points TSP [min\_distance: 5, max\_distance:10] Starting ant colony optimization Ants have found the correct path in 12 iterations. ACO completed in 0.6350538730621338 Starting Bruteforce A\* The shortest path found **by** bruteforce A\* : {'id': 3295, 'cost': 31, 'iterations': 31} in 231840 total iterations. Bruteforce A\* completed in 9.210999488830566 |

* Pour conclure, ce dernier environnement de test, nous pouvons constater un large écart entre les deux algorithmes à l'avantage de l’ACO.
* En effet, les résultats de ces tests présentent parfaitement ce à quoi nous nous attendions, c'est-à-dire une augmentation de l'écart du temps d'exécution entre les deux algorithmes suite à la complexification du problème.

### Comparaison des algorithmes d’optimisation

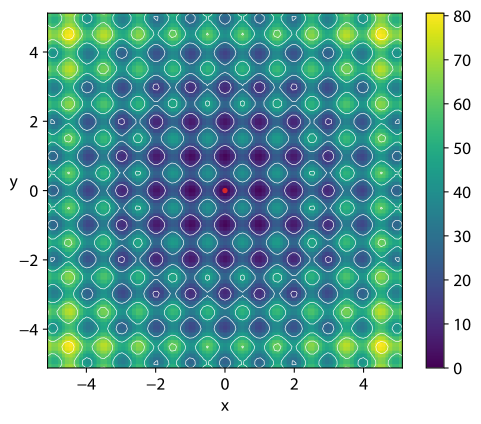
* Nous allons maintenant tester les performances des algorithmes d’optimisation.
* Pour réaliser ces tests, nous avons utilisé des fonctions plus ou moins complexes de dimension 2. Les algorithmes utilisés sont Ant Colony Optimization, PSO, Naive Search et Annealing Search. Ces tests prennent en compte la durée d'exécution de la recherche et la précision du résultat obtenu. L’intervalle de recherche des algorithmes est situé entre -600 et 600. L’agrandir augmente la difficulté de trouver une valeur correcte. Au niveau des hyperparamètres, on utilise 20 particules pour PSO et 60 abeilles pour ABC.

#### Fonction simple

* 
* 20 recherches par algorithmes avec une marge d’erreur de 0.2

|  |  |  |  |
| --- | --- | --- | --- |
|  | * Nombre d’itérations moyen | * Temps d'exécution moyen (en ms) | * Résultat moyen |
| * PSO | * 4 | * 0.0017623 | * 40.03795255 |
| * ABC | * 9 | * 0.065226 | * 39.99233717 |
| * Naive Search | * 1522 | * 0.027115 | * 39.99435131 |
| * Annealing Search | * 2458 | * 0.070178 | * 40.01188516 |

#### Rastrigin

* Voici la formule associée à la fonction Rastrigin (n est le nombre de dimensions donc deux dans notre cas):
* avec
* [](https://en.wikipedia.org/wiki/File:Rastrigin_contour_plot.svg)
* Vingt recherches par algorithmes avec une marge d’erreur de 0.4

|  |  |  |  |
| --- | --- | --- | --- |
|  | * Nombre d’itérations moyen | * Temps d'exécution moyen (en ms) | * Résultat moyen |
| * PSO | * 5 | * 0.0021996 | * 0.02210561 |
| * ABC | * 8 | * 0.061841 | * 0.05190725 |
| * Naive Search | * 3252 | * 0.055494 | * 0.02650762 |
| * Annealing Search | * 2326 | * 0.052713 | * -0.00289304 |

* Comme nous pouvons le voir avec les différents tests effectués, l’algorithme PSO est le meilleur en termes de résultats et de temps d'exécution. L’algorithme ABC affiche de bons résultats mais la recherche est très coûteuse en temps. Ces résultats sont comparables aux deux algorithmes non distribués.
* Cependant il faut noter que les algorithmes distribués sont plus coûteux en mémoire que les algorithmes non distribués. Cela est dû au fait que toutes les positions vont devoir être gardées en mémoire.

# Conclusion et place de l’intelligence distribuée

* + En conclusion, nous avons pu aborder ce qu’était l’intelligence distribuée et son intérêt. Que ce soit ACO, PSO, ABC ou BA que nous avons étudiés à travers ce cours ou d’autres dont nous n’avons pas eu le temps de citer ou détailler, tous ces algorithmes permettent de résoudre des problèmes variés et n’ont cependant qu’une utilisation infime de nos jours. Nous avons pu voir pendant ce cours l’efficacité de tels algorithmes et ce grâce à des instructions relativement simples à mettre en place.
* Le fait que les problèmes pour lesquels l’intelligence distribuée est utilisée soit de catégories variées (traitement d’images, de coordonnées, de chemin, …) nous laisse penser que l’intelligence distribuée est encore sous exploitée et peut encore être exploitée dans bien d’autres domaines. Plus le temps avance et plus les systèmes ont à gérer des masses de données qui ne font qu’augmenter. Rien que dans les domaines du réseau et du Big Data, l’intelligence distribuée pourrait mener à des combinaisons qui donneraient des résultats intéressants.
* Dans le cadre de système de réseaux, l’utilisation accrue de l’intelligence distribuée pourrait permettre le transit d’encore plus de données dans un temps encore plus court surtout si on combine cela avec des technologies telles que la fibre optique qui est bien implémentée actuellement.
* Pour ce qui est du Big Data, il est assez facile d’imaginer l’intelligence distribuée comme concurrent à certaines technologies du Big Data tel qu’Hadoop. Hadoop fonctionne avec un (ou plusieurs nœuds) master qui donne des tâches à des nœuds workers mais il est tout à fait possible d’imaginer l’intelligence distribuée avoir le même objectif, qui est la répartition d’une grosse tâche sur de nombreux nœuds, mais cette fois individuels sans nœud maître. L’avantage serait d’obtenir la même efficacité tout en réduisant les points où un échec entraînerait un défaut de fonctionnement majeur.
* Nous pouvons également nous projeter sur des projets tels que le Metaverse qui pourrait exploiter de tels algorithmes. Le Metaverse pourrait avoir besoin de gérer des foules, que ce soit d’animaux ou autres, et pourrait rendre cela faisable avec un coût minime et un nombre limité de ressources. Nous voyons déjà ce genre d’utilisation dans certains jeux-vidéo avec d’autres algorithmes d’IA en concordance. Mais un projet d’une telle ampleur nécessitera peut-être des optimisations extrêmes de systèmes que nous avons pu voir afin d’obtenir une expérience stable et fluide pour l’utilisateur.
* Au tout début du cours nous avons cité succinctement l’utilisation d’intelligence distribuée pour la gestion de drones. Ce dernier exemple n’est qu’à usage divertissant mais dans la même idée on pourrait l’appliquer à d’autres machines et d’autres tâches. Nous pouvons naïvement penser à ABC qui serait appliqué à des abeilles mécaniques dans le cas d’une population d’abeilles trop faible voire disparue. Le problème majeur de ce genre d’utilisation serait les failles de sécurité que cela pourrait engendrer pour toute personne qui peut mettre à jour les différentes instructions dans l’algorithme. Dans les cas des abeilles mécaniques, la série britannique Black Mirror a mis en scène dans un de ses épisodes une dérive possible de la mise en place d’un tel système et qui est également dépeint dans une nouvelle d’Isaac Asimov. Il faudrait donc prévoir en amont des systèmes de sécurité adaptés à des systèmes d’intelligence distribuée. Et bien évidemment si le problème est soulevé pour des choses aussi simples que des abeilles, il faudra également, malheureusement, prendre en compte toute possible utilisation militaire de ce type d’algorithmique.
* Effectivement, dans un cadre ludique, on peut utiliser des drones pour faire des spectacles de lumières qui sont beaux à voir mais dans un cadre militaire, que ce soit à petite ou grande échelle, cela pourrait tourner au massacre assez rapidement vu la simplicité à mettre en place et le tout sûrement en coordinations avec d’autres méthode d’IA ou technologies.
* Pour finir sur une note plus positive, l’intelligence distribuée semble avoir un bel avenir devant elle qui permettra sûrement un développement, notamment en informatique, plus qu’intéressant. Il est envisageable que cela puisse s’appliquer à des nanobots pour soigner une personne… seul l’avenir nous le dira. Même si beaucoup reste à venir on peut déjà apprécier son utilisation que ce soit dans le domaine des spectacles de divertissement en plein air ou dans celui de la gestion de foules dans les paysages vidéoludique et cinématographique qui continuent de s’améliorer et de nous impressionner.

# Annexes

* [Swarm intelligence](https://en.wikipedia.org/wiki/Swarm_intelligence#Models_of_swarm_behavior)
* [A Comprehensive Review of Swarm Optimization Algorithms](https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0122827)
* [Algorithmes en essaim](https://complex-systems-ai.com/algorithmes-dessaims/)
* **Artificial Bee Colony :**
* [Wikipedia](https://en.wikipedia.org/wiki/Artificial_bee_colony_algorithm)
* [Artificial Bee Colony (ABC) Visualized - Artificial Intelligence](https://www.youtube.com/watch?v=3qQr1eZwz5E&ab_channel=JCOpUntukIndonesia)
* **Multi-Agent Learning :**
* [AlphaStar Starcraft 2](https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learning)
* [Article AlphaStar](https://www.nature.com/articles/s41586-019-1724-z.epdf?author_access_token=lZH3nqPYtWJXfDA10W0CNNRgN0jAjWel9jnR3ZoTv0PSZcPzJFGNAZhOlk4deBCKzKm70KfinloafEF1bCCXL6IIHHgKaDkaTkBcTEv7aT-wqDoG1VeO9-wO3GEoAMF9bAOt7mJ0RWQnRVMbyfgH9A%3D%3D)
* **Federated Learning :**
* [AIGoogleBlog](https://ai.googleblog.com/2017/04/federated-learning-collaborative.html)