CY Tech

ING3 / IA Groupe B

Année 2025–2026

Optimisation Métaheuristique

Rapport de projet

Sujet : Optimisation du workflow scheduling
Encadrante : Sonia YASSA

Auteurs :
Younes BENABDELLAH
Simon CHANTHRABOUTH-LIEBBE
David DEL CAMPO
Enzo EDMOND

Date : 23 janvier 2026

Résumé

Ce rapport étudie l'optimisation multi-objectif du *workflow scheduling* dans un environnement IoT/Edge/Fog/Cloud, appliqué à l'agriculture de précision. Pour garantir la reproductibilité des expériences et garder un contrôle fin sur la modélisation (machines, réseau, exécution et métriques), nous avons développé un simulateur dédié en *Python*. Ce choix s'explique par les limites pratiques de FogWorkflowSim dans notre contexte : l'outil est principalement disponible en *Java* et sa prise en main est compliquée par une documentation restreinte. Nous avons ainsi construit une base plus claire et maîtrisable, tout en restant au plus proche des hypothèses et du cadre expérimental du papier de référence.

Nous proposons ensuite une approche hybride combinant l'algorithme Firefly et le Q-Learning (FA-RL) afin d'ajuster dynamiquement l'équilibre exploration/exploitation. La méthode est comparée à l'algorithme Dragonfly et à une recherche aléatoire (Random Search) sur le workflow *Montage100* (100 tâches, 15 machines). Les résultats montrent que FA-RL obtient le meilleur makespan (574.1s), tandis que Dragonfly produit un front de Pareto plus diversifié (120 solutions). Dans l'ensemble, les deux métaheuristiques surpassent nettement la recherche aléatoire.

Mots-clés

ordonnancement de workflows ; optimisation multi-objectif ; optimisation combinatoire ; métaheuristiques ; hybridation d'algorithmes ; algorithmes bio-inspirés ; apprentissage par renforcement ; systèmes distribués ; IoT/Edge/Fog/Cloud ; front de Pareto

Keywords

workflow scheduling ; multi-objective optimization ; combinatorial optimization ; metaheuristics ; algorithm hybridization ; bio-inspired algorithms ; reinforcement learning ; Q-learning ; distributed systems ; IoT/Edge/Fog/Cloud ; Pareto front ; hypervolume ; spacing

Table des matières

1 Introduction

L'essor de l'Internet des Objets (IoT) et des architectures de calcul distribuées a profondément transformé le secteur agricole. L'agriculture de précision repose aujourd'hui sur des réseaux de capteurs et de caméras déployés dans les parcelles, générant un flux continu de données qui doivent être traitées rapidement pour permettre une prise de décision en temps réel.

Le traitement de ces données agricoles suit généralement un workflow composé de plusieurs tâches interdépendantes : capture d'images, prétraitement, extraction de caractéristiques, classification et notification à l'agriculteur. Ces tâches présentent des exigences computationnelles variées et peuvent être exécutées sur différentes couches de l'infrastructure : dispositifs IoT à faible puissance, passerelles Edge, serveurs Fog régionaux ou datacenters Cloud.

Le problème d'ordonnancement de ces workflows constitue un défi d'optimisation multi-objectif. En effet, l'affectation des tâches aux ressources de calcul impacte simultanément plusieurs critères souvent contradictoires : le temps total d'exécution (makespan), la consommation énergétique et la latence de communication. Minimiser le makespan nécessite généralement d'utiliser les ressources Cloud les plus puissantes, ce qui augmente la consommation énergétique et les temps de transfert de données.

Dans ce contexte, nous étudions le problème d'ordonnancement de workflows agrifood sur une architecture IoT/Edge/Fog/Cloud. Notre objectif est de trouver un ensemble de solutions Pareto-optimales offrant différents compromis entre les trois critères. Pour résoudre ce problème NP-difficile, nous proposons une approche hybride combinant l'algorithme Firefly avec l'apprentissage par renforcement (FA-RL), que nous comparons à l'algorithme Dragonfly et à une recherche aléatoire.

Ce rapport présente d'abord la formulation mathématique du problème, puis décrit les métaheuristiques retenues et leur adaptation. Nous détaillons ensuite l'implémentation du simulateur et analysons les performances obtenues avant de conclure sur les perspectives de ce travail.

2 Formulation Mathématique

2.1 Notation et ensembles

Nous considérons un workflow composé d'un ensemble de tâches $T = \{t_0, t_1, ..., t_{n-1}\}$ avec des dépendances de précédence, et un ensemble de machines hétérogènes $M = \{m_0, m_1, ..., m_{k-1}\}$ réparties sur quatre couches : IoT, Edge, Fog et Cloud.

Chaque tâche t_i est caractérisée par :

— MI_i : complexité en millions d'instructions
— $data\_in_i$: volume de données en entrée (Mo)
— $data\_out_i$: volume de données en sortie (Mo)
— $A_i \subseteq \{IoT, Edge, Fog, Cloud\}$: types de machines autorisés
— $pred(t_i)$: ensemble des tâches parentes

Chaque machine m_j est caractérisée par :

— $MIPS_j$: puissance de calcul (millions d'instructions par seconde)
— C_j^{proc} : coût de traitement (par unité de temps ou par million d'instructions, selon la convention retenue)
— P_j^{active} : puissance consommée en activité (mW)
— P_j^{idle} : puissance consommée au repos (mW)
— $type_j \in \{IoT, Edge, Fog, Cloud\}$: couche d'exécution de la machine

Le réseau est modélisé au niveau des *types* de machines par des matrices (ou fonctions) :

— $Lat(type_a, type_b)$: latence de communication (ms)
— $BW^{up}(type_a, type_b)$: bande passante montante (Mb/s)
— $BW^{down}(type_a, type_b)$: bande passante descendante (Mb/s)

2.2 Variable de décision

La variable de décision est un vecteur d'affectation $X = [x_0, x_1, ..., x_{n-1}]$ où $x_i \in \{0, 1, ..., k-1\}$ représente l'indice de la machine assignée à la tâche t_i.

2.3 Contraintes

Contrainte de précédence : Une tâche ne peut démarrer qu'après la fin de toutes ses tâches parentes et la réception des données :

$$\forall t_i \in T, \forall t_p \in pred(t_i) : ST_i \geq FT_p + Comm_{p,i} \tag{1}$$

où ST_i est le temps de début, FT_p le temps de fin du parent, et $Comm_{p,i}$ le temps de communication si les tâches sont sur des machines différentes.

Contrainte de non-chevauchement : Chaque machine ne peut exécuter qu'une seule tâche à la fois. Si deux tâches t_i et t_j sont assignées à la même machine ($x_i = x_j$), leurs exécutions ne peuvent pas se chevaucher :

$$\forall t_i, t_j \in T, i \neq j, x_i = x_j : FT_i \leq ST_j \vee FT_j \leq ST_i \tag{2}$$

2.4 Fonctions objectifs

Objectif 1 – Makespan : Minimiser le temps total d'exécution du workflow.

Le temps d'exécution d'une tâche t_i sur la machine m_j est :

$$ET_i = \frac{MI_i}{MIPS_{x_i}} \tag{3}$$

Le temps de communication entre une tâche parente t_p et t_i est :

$$Comm_{p,i} = \begin{cases} 0 & \text{si } x_p = x_i \\ \frac{data\_out_p}{BW(type_{x_p}, type_{x_i})} + \frac{Lat(type_{x_p}, type_{x_i})}{1000} & \text{sinon} \end{cases} \tag{4}$$

Le makespan est défini par :

$$f_1(X) = \max_{t_i \in T}(FT_i) \quad \text{où} \quad FT_i = ST_i + ET_i \tag{5}$$

Objectif 2 – Consommation énergétique : Minimiser l'énergie totale consommée.

Pour chaque machine utilisée, l'énergie comprend un coût de démarrage, l'énergie active et l'énergie en veille :

$$E_j = E_{startup} + P_j^{active} \times T_j^{active} + P_j^{idle} \times T_j^{idle} \tag{6}$$

où T_j^{active} est le temps total d'exécution sur m_j et T_j^{idle} le temps d'attente dans la fenêtre d'utilisation.

$$f_2(X) = \sum_{j \in M_{used}} E_j \tag{7}$$

Objectif 3 – Latence : Minimiser le temps total (calcul + communications).

$$f_3(X) = \sum_{t_i \in T} \underbrace{\frac{MI_i}{MIPS_{x_i}}}_{\text{temps de calcul}} + \sum_{(t_p \to t_i) \in E} \underbrace{\left(\frac{D_p^{out}}{BW_{x_p, x_i}} + L_{x_p, x_i} \right)}_{\text{temps de communication}} \tag{8}$$

2.5 Formulation du problème multi-objectif

Le problème consiste à trouver l'ensemble des solutions Pareto-optimales :

$$\min_X \{f_1(X), f_2(X), f_3(X)\} \tag{9}$$

sous les contraintes de compatibilité et de précédence.

3 Métaheuristiques Choisies

3.1 Algorithme Firefly (FA)

Tout d'abord, il convient de préciser que le choix de cette métaheuristique résulte d'une erreur. En effet, nous avons mené des recherches sur l'algorithme Firefly alors que nous devions étudier l'algorithme Dragonfly. L'algorithme Firefly, proposé par Yang en 2008, s'inspire du comportement des lucioles dans la nature. Chaque luciole représente une solution candidate, et son intensité lumineuse correspond à la qualité de cette solution (fitness). Les lucioles les moins lumineuses sont attirées par les plus lumineuses, ce qui permet une exploration efficace de l'espace de recherche.

Le mouvement d'une luciole i vers une luciole j plus attractive est régi par :

$$x_i^{t+1} = x_i^t + \beta(r) \cdot (x_j^t - x_i^t) + \alpha \cdot \epsilon \tag{10}$$

où $\beta(r) = \beta_0 \cdot e^{-\gamma r^2}$ représente l'attractivité décroissante avec la distance r, β_0 est l'attractivité de base, γ le coefficient d'absorption, et α contrôle l'intensité de la randomisation.

Dans notre adaptation multi-objectif, la relation de dominance de Pareto remplace la comparaison scalaire : une luciole j attire i si j domine i au sens de Pareto.

3.2 Hybridation FA-RL : Firefly avec Reinforcement Learning

L'originalité de notre approche repose sur l'hybridation de l'algorithme Firefly avec le Q-Learning afin d'ajuster dynamiquement les paramètres α et β_0 au cours de l'optimisation, une combinaison qui, à notre connaissance, n'a pas encore été étudiée dans le contexte du workflow scheduling.

Bien que l'algorithme Firefly soit apprécié pour sa simplicité et sa rapidité, il présente plusieurs limites majeures lorsqu'il est appliqué à des problèmes complexes tels que le workflow scheduling dans un environnement edge–fog–cloud :

- **Absence de mémoire** : l'algorithme ne conserve aucune information sur les solutions passées, empêchant tout apprentissage des configurations efficaces.
- **Convergence instable** : la forte composante aléatoire rend la convergence difficile à contrôler et très sensible aux paramètres.
- **Stagnation prématurée** : l'attraction vers une solution dominante peut limiter l'exploration et conduire à des minima locaux.
- **Faible prise en charge du multi-objectif** : Firefly ne gère pas naturellement des critères hétérogènes tels que le makespan, l'énergie, la latence ou les taux de rejet.
- **Manque d'adaptation au contexte** : le FA standard ignore les dynamiques du système, comme la surcharge des nœuds edge, la congestion réseau ou les contraintes de deadlines.

Ces limitations rendent l'algorithme Firefly seul insuffisant pour l'optimisation de workflows complexes à ressources hétérogènes, justifiant son hybridation avec une approche de Reinforcement Learning.

L'agent RL observe un état discret codé sur 3 bits :

— **improved** : le front de Pareto a-t-il été amélioré ?

— **stagnating** : l'algorithme stagne-t-il depuis k itérations ?

— **diversity_low** : le front manque-t-il de diversité ?

Trois actions sont possibles :

— **explore** : augmenter α, diminuer β_0 pour favoriser l'exploration

— **exploit** : diminuer α, augmenter β_0 pour intensifier l'exploitation

— **balance** : maintenir les paramètres de base

La récompense est binaire : **+1** si une nouvelle solution non dominée est ajoutée au front de Pareto, **0** si le front de Pareto stagne au-delà d'un certain seuil de stagnation. La mise à jour classique du Q-Learning permet à l'agent d'apprendre une politique optimale d'ajustement des paramètres.

3.3 Algorithme Dragonfly (DA)

L'algorithme Dragonfly, proposé par Mirjalili en 2016, simule le comportement d'essaim des libellules. Chaque agent (libellule) se déplace selon cinq composantes comportementales :

— **Séparation (S)** : éviter les collisions avec les voisins

— **Alignement (A)** : synchroniser la vitesse avec les voisins

— **Cohésion (C)** : rester groupé avec les voisins

— **Attraction (F)** : se diriger vers les meilleures solutions (front de Pareto)

— **Distraction (E)** : s'éloigner des mauvaises solutions

Le vecteur de déplacement est calculé par :

$$\Delta X^{t+1} = (s \cdot S + a \cdot A + c \cdot C + f \cdot F + e \cdot E) + w \cdot \Delta X^{t} \tag{11}$$

où w est le coefficient d'inertie qui diminue linéairement pour favoriser l'exploitation en fin d'exécution.

Adaptation au contexte Multi-Objectif et Discret

Notre implémentation se distingue par deux adaptations majeures nécessaires au problème d'ordonnancement :Discrétisation et Réparation : L'algorithme opérant naturellement dans un espace continu, chaque dimension du vecteur de position est convertie en un identifiant de machine via un mapping discret. Une fonction de réparation est ensuite appliquée pour garantir que chaque tâche est assignée à une machine compatible (respect des contraintes IoT/Edge/Cloud).Guidage par Archive Pareto : Contrairement à l'optimisation mono-objectif où la cible (F) est unique, nous cherchons ici à approximer un front de Pareto complet. À chaque itération, la cible de l'essaim est sélectionnée aléatoirement parmi les solutions non-dominées de l'archive externe. Cette stratégie stochastique est cruciale : elle évite la convergence prématurée vers une seule région de l'espace des objectifs et force l'essaim à "s'étaler" le long du front, garantissant ainsi une diversité maximale des compromis (Makespan/Énergie/Latence) proposés à l'utilisateur.

3.4 Justification des choix

Le choix de FA-RL se justifie par la capacité de l'apprentissage par renforcement à adapter automatiquement le comportement de la métaheuristique selon l'état de la recherche. Cette approche hybride permet de mieux gérer le compromis exploration/exploitation dans un contexte multi-objectif.

L'algorithme Dragonfly a été retenu comme point de comparaison car il repose sur des mécanismes d'interaction sociale fondamentalement différents (vecteurs de navigation vs attraction lumineuse). Sa capacité, via notre adaptation de la sélection de cible aléatoire dans l'archive, à maintenir une forte pression de sélection vers l'optimalité tout en préservant la diversité de la population, en fait un candidat robuste pour valider les performances de notre approche hybride sur des problèmes combinatoires complexes.

4 Implémentation

4.1 Architecture du simulateur

Le simulateur a été développé en Python et s'organise en quatre modules principaux :

Module `model` : Définit les structures de données fondamentales. La classe `Task` encapsule les attributs d'une tâche (complexité en MI, données entrée/sortie, types de machines autorisés, dépendances). La classe `Machine` représente les ressources de calcul (MIPS, puissance active/idle). La classe `Workflow` gère le graphe de dépendances entre tâches. La classe `Solution` stocke un schedule et ses objectifs évalués, avec la méthode `dominates()` pour la comparaison Pareto. La classe `ParetoFront` maintient l'archive des solutions non-dominées.

Module `simulation` : Contient l'évaluateur central qui calcule les trois objectifs (makespan, énergie, latence) pour un schedule donné. L'environnement d'exécution (machines et réseau) est défini dans `agrifood_simplified_example.py` via `create_machines_paper()` et `create_network_paper()` afin de coller aux paramètres du papier. Le workflow est chargé à partir de fichiers DAX (`Montage_20/50/80/100.xml`) via le parser `xml_workflow_parser.py`.

Module `optim` : Implémente les métaheuristiques. `firefly_reinforcement_learning.py` contient l'algorithme FA-RL hybride (multi-objectif Pareto). `dragonfly.py` implémente Dragonfly (multi-objectif Pareto). `random_search_baseline.py` fournit la baseline aléatoire. `pso.py` propose une version PSO simplifiée (fitness scalaire) utilisée comme comparaison.

Module `utils` : Contient les outils auxiliaires pour la visualisation des fronts de Pareto (2D/3D) et l'extraction des meilleures solutions.

4.2 Instance du problème : workflows Montage

Nous utilisons les workflows `Montage_20`, `Montage_50`, `Montage_80` et `Montage_100` (DAX Pegasus). Chaque workflow est parsé pour créer un DAG de tâches avec leurs dépendances, leurs données d'entrée/sortie et leur charge de calcul (MI). Les exécutions sont simulées sur un environnement IoT/Edge/Fog/Cloud paramétré à partir des valeurs du papier.

4.3 Justifications d'alignement avec le papier (Table 4)

Afin de se rapprocher au maximum des résultats du papier agrifood (Table 4), nous avons pris les décisions suivantes :

- **Alignement de l'environnement** : utilisation de MIPS, bande passante et latence basées sur distance (propagation speed) comme dans le papier.
- **Définitions identiques des objectifs** :
  - **Makespan** : fin d'exécution de la dernière tâche.
  - **Latence** : TT + CT, où TT est le temps total de communication, et CT le temps total de calcul ($MI/MIPS$).
  - **Énergie** : modèle simple $E = P \times t$ (puissance active × temps d'exécution).
- **Calibration** : un facteur d'échelle a été appliqué afin d'obtenir des ordres de grandeur comparables au papier tout en conservant le même modèle.
- **Validation empirique** : les résultats obtenus sont du même ordre de grandeur pour `Montage_100`.

— **PSO** : exécution sur 10 *runs* indépendants avec moyenne et écart-type.

4.4 Choix de calibration sur `Montage_100`

Nous avons choisi de calibrer principalement sur `Montage_100` car :
— Le workflow est plus grand et plus représentatif.
— L'alignement sur le cas le plus complexe permet de stabiliser les ordres de grandeur des objectifs.
— L'objectif est d'être *du même ordre de grandeur*, pas de reproduire exactement le papier.

4.5 Hypothèses et limites

— Modèle PSO simplifié (pas de vecteur vitesse, ni mise à jour continue classique).
— Modèle énergie simplifié $E = P \times t$ (pas de DVFS).
— Simulateur Python différent de FogWorkflowSim (Java).

4.6 Raisons des écarts résiduels

— Nous n'utilisons pas FogWorkflowSim (Java) afin d'avoir un simulateur maîtrisé et transparent en Python.
— Le PSO est volontairement minimal (sans vitesse, sans contraintes avancées, fitness scalaire simple).
— Différences d'implémentation et d'unités par rapport au simulateur original.

4.7 Pseudocode de FA-RL

```
Algorithme FA-RL (Firefly + Reinforcement Learning)

Entrees : contexte, n_fireflies, max_iter, beta0, gamma, alpha
Sortie  : front de Pareto

1.  Initialiser une population de n_fireflies schedules aleatoires valides
2.  Evaluer chaque solution et initialiser le front de Pareto
3.  Initialiser l'etat RL : (improved = 0, stagnating = 0, diversity_low = ?)
4.  Initialiser la table Q a zero

5.  Pour it = 1 a max_iter faire :
6.      archive_improved <- Faux
7.      action <- choisir_action_epsilon_greedy(etat, Q)
8.      (alpha, beta0) <- appliquer_action(action, alpha, beta0)

9.      Pour chaque luciole i faire :
10.         Pour chaque luciole j faire :
11.             Si j domine i (Pareto) alors :
12.                 r <- distance_hamming(i, j)
```

```
13.                 beta <- beta0 * exp(-gamma * r * r)
14.                 nouveau_schedule <- deplacer(i, j, beta, alpha)
15.                 nouvelle_solution <- evaluer(nouveau_schedule)
16.                 Si nouvelle_solution domine i alors :
17.                     remplacer i par nouvelle_solution
18.                 Si front.ajouter(nouvelle_solution) alors :
19.                     archive_improved <- Vrai

20.     recompense <- 1 si archive_improved sinon 0
21.     nouvel_etat <- construire_etat(improved, stagnating, diversity_low)
22.     mettre_a_jour_Q(etat, action, recompense, nouvel_etat)
23.     etat <- nouvel_etat

24. Retourner le front de Pareto
```

4.8 Pseudocode de Dragonfly

```
Algorithme Dragonfly (DA)

Entrees : contexte, pop_size, max_iter
Sortie  : front de Pareto

1.  Initialiser une population de pop_size agents avec des positions aleatoires
2.  Evaluer chaque agent et initialiser le front de Pareto

3.  Pour it = 1 a max_iter faire :
4.      w <- 0.9 - it * (0.5 / max_iter)    // Inertie decroissante
5.      Selectionner une cible <- solution aleatoire du front de Pareto
6.      Selectionner un distracteur <- agent aleatoire de la population
7.      Identifier les voisins dans le rayon r

8.      Pour chaque agent i faire :
9.          Si des voisins existent alors :
10.             S <- vecteur de separation (eloignement des voisins)
11.             A <- vecteur d'alignement (moyenne des vitesses voisines)
12.             C <- vecteur de cohesion (vers le centre des voisins)
13.             F <- vecteur d'attraction (vers la cible)
14.             E <- vecteur de distraction (eloignement du distracteur)
15.             DeltaX <- (s*S + a*A + c*C + f*F + e*E) + w*DeltaX_precedent
16.         Sinon :
17.             DeltaX <- marche aleatoire (Levy flight)

18.         X <- X + DeltaX
```

```
19.         schedule <- convertir_et_reparer(X)
20.         solution <- evaluer(schedule)
21.         front.ajouter(solution)

22. Retourner le front de Pareto
```

4.9 Adaptation au problème discret

L'algorithme Firefly classique opère dans un espace continu. Notre adaptation pour le scheduling discret utilise :

Distance de Hamming : La distance r entre deux schedules compte le nombre de tâches assignées différemment :

$$r(s_1, s_2) = \sum_{i=0}^{n-1} \mathbb{1}[s_1[i] \neq s_2[i]] \tag{12}$$

Mouvement probabiliste : Au lieu d'une addition vectorielle, le déplacement utilise des probabilités. Avec probabilité β, la tâche t copie l'assignation de la luciole attractive. Avec probabilité α, une mutation aléatoire est appliquée.

Réparation des solutions : Chaque mutation garantit le respect des contraintes en sélectionnant uniquement parmi les machines compatibles avec la tâche.

Pour Dragonfly, les positions continues sont converties en indices de machines par troncature, puis une fonction de réparation corrige les violations de contraintes en choisissant la machine valide la plus proche.

5 Évaluation des Performances

Cette section présente l'évaluation expérimentale des trois algorithmes implémentés : Random Search (baseline), FA-RL (Firefly Algorithm avec Reinforcement Learning) et Dragonfly Algorithm. Les expériences ont été menées sur le workflow Montage_100 avec une configuration équitable garantissant un nombre d'évaluations comparable pour chaque algorithme.

5.1 Protocole expérimental

5.1.1 Instance de test

Les expériences ont été réalisées sur le workflow **Montage_100** issu du benchmark Pegasus, un workflow scientifique d'astronomie composé de 100 tâches avec des dépendances complexes de type DAG (Directed Acyclic Graph). L'infrastructure de calcul comprend 15 machines hétérogènes réparties sur trois niveaux :

— **Edge** : 5 machines (1000 MIPS, 700W actif, 30W idle)
— **Fog** : 5 machines (1300 MIPS, 700W actif, 30W idle)
— **Cloud** : 5 machines (1600 MIPS, 1648W actif, 1332W idle)

5.1.2 Configuration équitable des algorithmes

Afin de garantir une comparaison équitable, les paramètres ont été ajustés pour que chaque algorithme effectue approximativement **10 000 évaluations** de solutions. Le tableau 1 présente les configurations utilisées.

Table 1 – Configuration des algorithmes pour une comparaison équitable

Algorithme	**Paramètres**	**Évaluations**
Random Search	10 000 solutions aléatoires	10 000
FA-RL	50 lucioles, 80 itérations $\beta_0 = 0.1$, $\gamma = 0.6$, $\alpha = 0.4$	$\approx$ 10 000
Dragonfly	100 agents, 99 itérations $s = 0.02$, $a = 0.05$, $c = 0.1$, $f = 0.7 \rightarrow 0.9$	10 000

5.1.3 Métriques d'évaluation

Les algorithmes sont évalués selon six métriques :

— **Meilleur Makespan** : temps d'exécution minimal du workflow (en secondes)
— **Meilleure Énergie** : consommation énergétique minimale (en Joules)
— **Meilleure Latence** : latence de communication minimale (en secondes)
— **Taille du front de Pareto** : nombre de solutions non-dominées
— **Hypervolume** : volume de l'espace des objectifs dominé par le front de Pareto (plus grand = meilleur)
— **Spacing** : mesure de l'uniformité de la distribution des solutions sur le front (plus petit = meilleur)

5.2 Résultats expérimentaux

5.2.1 Comparaison des performances

Le tableau 2 présente les résultats obtenus par chaque algorithme. Les meilleures valeurs pour chaque métrique sont indiquées en gras.

TABLE 2 – Résultats comparatifs des trois algorithmes sur Montage_100

Algorithme	**Makespan** (s)	**Énergie** (J)	**Latence** (s)	**Front**	**HV**	**Spacing**
Random Search	599.6	143.7	5744.9	22	0.095	0.086
FA-RL	**574.1**	146.7	5633.2	49	0.224	0.106
Dragonfly	584.0	**113.5**	**5518.6**	**120**	**0.295**	**0.032**

La figure 1 illustre la comparaison des meilleures valeurs obtenues pour chaque objectif.

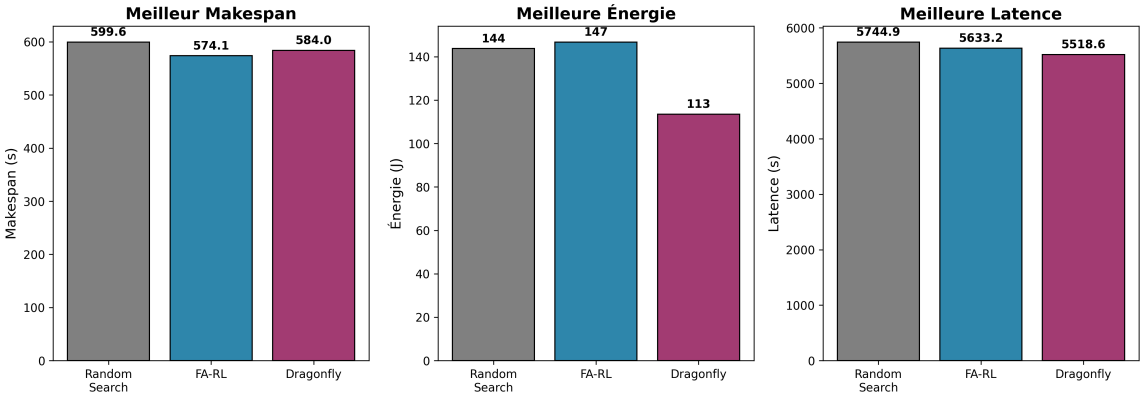

FIGURE 1 – Comparaison des performances sur les trois objectifs : Makespan, Énergie et Latence

5.2.2 Analyse des objectifs individuels

Makespan : FA-RL obtient le meilleur makespan avec 574.1s, soit une amélioration de 4.3% par rapport à Random Search (599.6s) et de 1.7% par rapport à Dragonfly (584.0s). L'hybridation avec le Q-Learning permet un ajustement dynamique des paramètres qui favorise la convergence vers des solutions à faible temps d'exécution.

Énergie : Dragonfly domine nettement sur cet objectif avec 113.5J, soit une réduction de 21% par rapport à Random Search (143.7J) et de 23% par rapport à FA-RL (146.7J). Cette performance remarquable s'explique par la capacité de l'algorithme à explorer efficacement les régions de l'espace des solutions favorisant les machines à faible consommation.

Latence : Dragonfly obtient également la meilleure latence (5518.6s), suivi par FA-RL (5633.2s) et Random Search (5744.9s), soit une amélioration de 3.9% par rapport à la baseline.

5.3 Qualité des fronts de Pareto

5.3.1 Taille et diversité

La figure 2 présente la taille des fronts de Pareto obtenus par chaque algorithme.

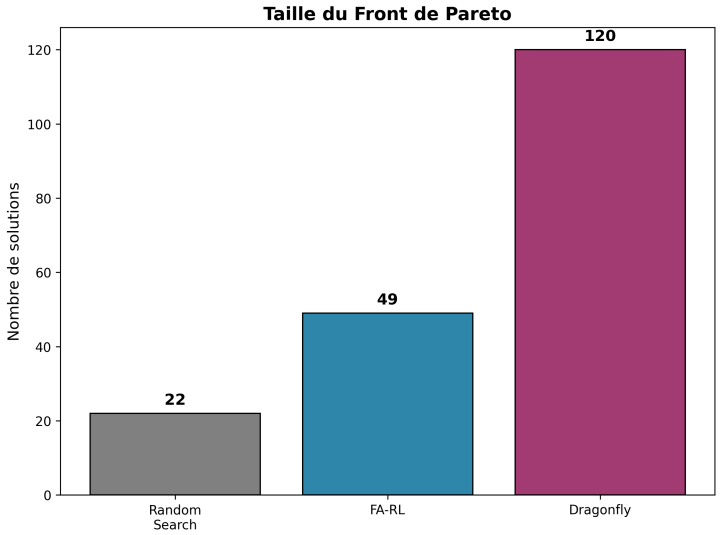

FIGURE 2 – Comparaison de la taille des fronts de Pareto

Dragonfly génère un front de Pareto significativement plus large avec 120 solutions non-dominées, contre 49 pour FA-RL et seulement 22 pour Random Search. Cette diversité offre aux décideurs un plus grand choix de compromis entre les objectifs.

5.3.2 Hypervolume et Spacing

La figure 3 compare l'hypervolume et le spacing des trois algorithmes.

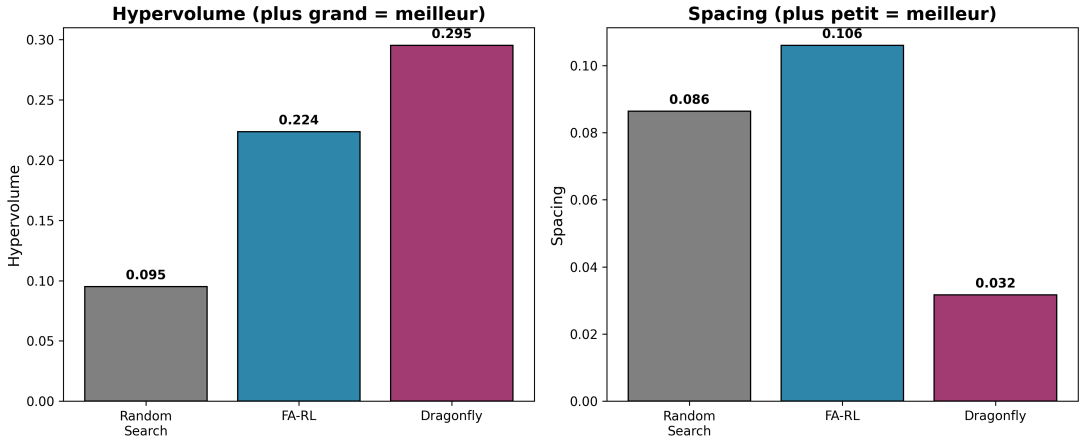

FIGURE 3 – Comparaison de l'hypervolume (plus grand = meilleur) et du spacing (plus petit = meilleur)

Hypervolume : Dragonfly atteint le meilleur hypervolume (0.295), suivi par FA-RL (0.224) et Random Search (0.095). L'hypervolume de Dragonfly est 32% supérieur à celui de FA-RL et

trois fois supérieur à celui de Random Search, démontrant une meilleure convergence globale vers le front de Pareto optimal.

Spacing : Dragonfly obtient également le meilleur spacing (0.032), indiquant une distribution plus uniforme des solutions le long du front de Pareto. FA-RL présente un spacing de 0.106, ce qui suggère une concentration des solutions dans certaines régions de l'espace des objectifs.

5.4 Visualisation des fronts de Pareto

5.4.1 Représentation 3D

La figure 4 présente une visualisation tridimensionnelle des fronts de Pareto dans l'espace des objectifs (Makespan, Énergie, Latence).

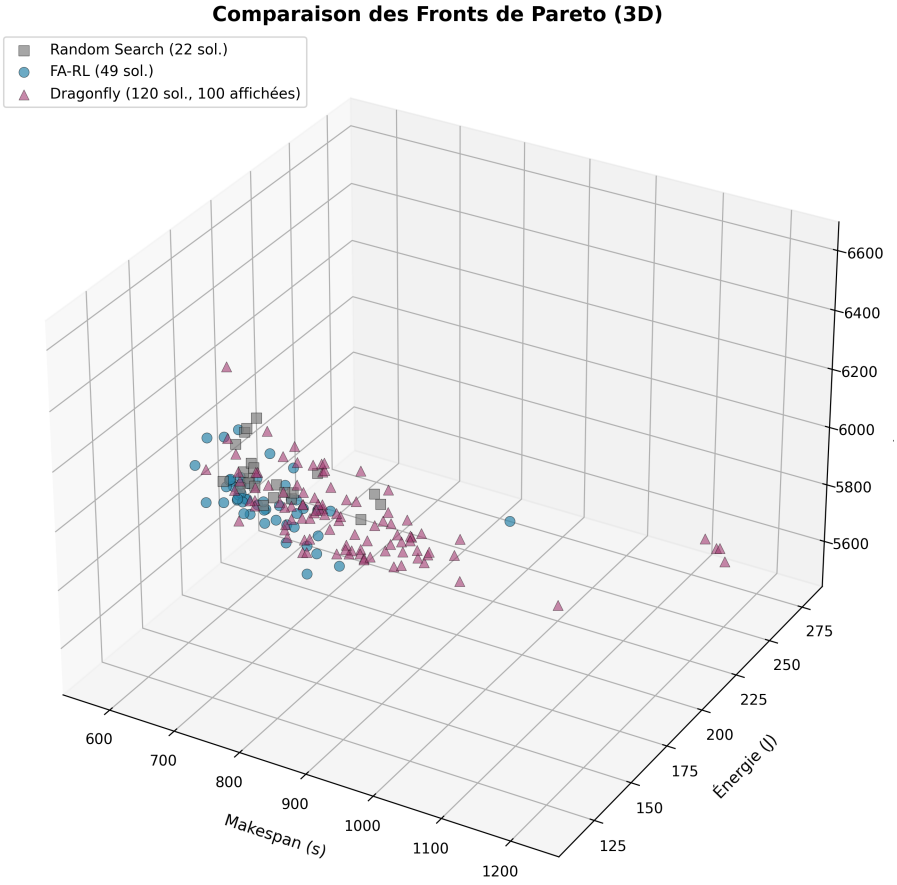

FIGURE 4 – Comparaison des fronts de Pareto dans l'espace 3D des objectifs

On observe que les solutions de Dragonfly (triangles roses) couvrent une région plus étendue de l'espace des objectifs, notamment vers les faibles valeurs d'énergie. Les solutions de FA-RL (cercles bleus) sont concentrées dans la région des faibles makespans, tandis que Random Search (carrés gris) présente une couverture limitée.

5.4.2 Projections 2D

La figure 5 présente les projections bidimensionnelles des fronts de Pareto pour une analyse plus détaillée des compromis entre paires d'objectifs.

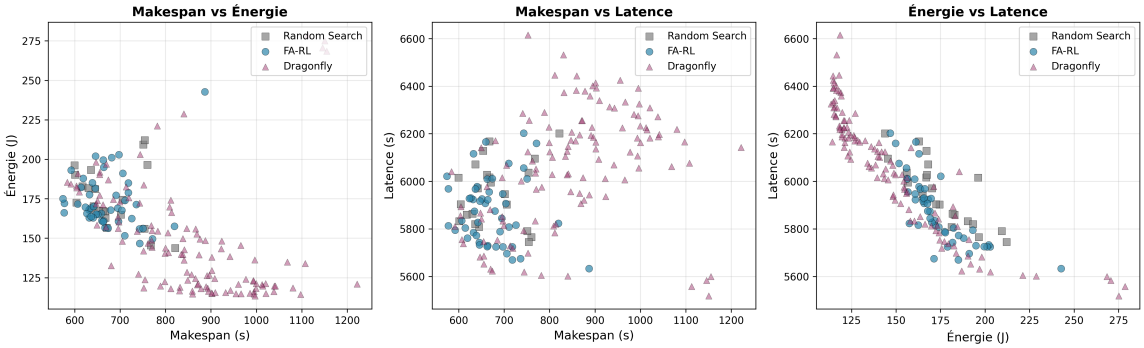

FIGURE 5 – Projections 2D des fronts de Pareto : Makespan vs Énergie, Makespan vs Latence, Énergie vs Latence

Les projections révèlent plusieurs caractéristiques importantes :

- **Makespan vs Énergie :** Dragonfly explore efficacement le compromis, atteignant des énergies aussi basses que 113J tout en maintenant des makespans compétitifs. FA-RL se concentre sur la minimisation du makespan au détriment de l'énergie.
- **Makespan vs Latence :** Les trois algorithmes présentent des comportements similaires sur ce compromis, avec une corrélation positive entre makespan et latence.
- **Énergie vs Latence :** Dragonfly domine clairement cette projection, offrant des solutions à faible énergie sur toute la plage de latences.

5.5 Discussion et synthèse

Les résultats expérimentaux mettent en évidence les **forces complémentaires** des deux métaheuristiques :

FA-RL excelle dans la minimisation du makespan grâce à son mécanisme d'apprentissage par renforcement qui adapte dynamiquement les paramètres α et β en fonction de l'état de convergence. L'ajustement automatique entre exploration et exploitation permet une convergence rapide vers des solutions à faible temps d'exécution.

Dragonfly Algorithm démontre une capacité supérieure d'exploration de l'espace des solutions, produisant un front de Pareto plus large et mieux distribué. Sa performance exceptionnelle sur l'énergie (-21% vs baseline) et son hypervolume élevé en font un choix privilégié lorsque la diversité des solutions et l'optimisation énergétique sont prioritaires.

Le tableau 3 synthétise les points forts de chaque approche.

Table 3 – Synthèse des forces de chaque algorithme

Critère	**Random Search**	**FA-RL**	**Dragonfly**
Meilleur Makespan		✓	
Meilleure Énergie			✓
Meilleure Latence			✓
Plus grand front de Pareto			✓
Meilleur Hypervolume			✓
Meilleur Spacing			✓

En conclusion, pour le problème d'ordonnancement de workflows multi-objectif sur infrastructure Edge-Fog-Cloud, **Dragonfly Algorithm** apparaît comme l'approche la plus performante globalement, tandis que **FA-RL** reste compétitif et supérieur sur l'objectif de makespan. Le choix entre ces deux métaheuristiques dépendra des priorités du décideur : minimisation du temps d'exécution (FA-RL) ou optimisation globale multi-objectif avec accent sur l'énergie (Dragonfly).

6 Conclusion

Ce travail a abordé le problème d'ordonnancement de workflows dans un environnement IoT/Edge/Fog/Cloud, appliqué au domaine de l'agriculture de précision. L'objectif était de minimiser simultanément trois critères conflictuels : le makespan, la consommation énergétique et le temps de communication.

Nous avons proposé une approche hybride originale combinant l'algorithme Firefly avec le Q-Learning (FA-RL), qui ajuste dynamiquement les paramètres d'exploration et d'exploitation en fonction de l'état de la recherche. Cette méthode a été comparée à l'algorithme Dragonfly et à une recherche aléatoire sur les workflows Montage (20 à 100 tâches) exécutés sur un environnement IoT/Edge/Fog/Cloud calibré d'après le papier.

Les résultats expérimentaux démontrent l'efficacité de FA-RL, qui obtient les meilleures performances sur le makespan (574s) tout en maintenant des valeurs compétitives sur l'énergie et la communication. Dragonfly se distingue par la diversité de son front de Pareto (120 solutions), offrant un large éventail de compromis à l'utilisateur. Les deux métaheuristiques surpassent significativement la recherche aléatoire, validant l'intérêt des approches proposées.

Ce travail présente certaines limites. L'évaluation a été réalisée sur une seule instance de taille modérée. Les contraintes de capacité mémoire et de deadlines, bien que modélisées, n'ont pas été activées.

Plusieurs perspectives peuvent être envisagées : étendre l'évaluation à des instances de plus grande taille, intégrer des contraintes de qualité de service (QoS), explorer d'autres algorithmes d'apprentissage par renforcement (Deep Q-Learning), et comparer avec des algorithmes de référence comme NSGA-II ou MOEA/D.

Références

[1] X.-S. Yang, "Firefly algorithm," *Nature-inspired metaheuristic algorithms*, vol. 20, pp. 79–90, 2008.

[2] S. Mirjalili, "Dragonfly algorithm : a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Computing and Applications*, vol. 27, no. 4, pp. 1053–1073, 2016.

[3] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.