REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE

Faculté d'Informatique

# Mémoire de Licence

Filière
**Informatique**

Spécialité
**Informatique Académique**

## Thème

Algorithme Slime Mould pour le dispatching des ambulances d'urgence dans le contexte de la crise sanitaire du COVID-19

**Soutenu le : .../06/2022**

**Sujet proposé par :**
Dr. KHENNAK Ilyes

**Présenté par :**
Mr. BELHARDA Yagoub
Mr. SMAIL Mehrez

**Devant le jury composé de :**

| | | |
|---|---|---|
| Prof. AZZOUNE Hamid | ..... | Président |
| Mme. AMROUNI Radia | ..... | Membre |

Binôme n° : 115 / 2022

# Table des matières

# Liste des figures



# Liste des tableaux



# Liste des algorithmes

# Introduction générale

Récemment, les algorithmes d'intelligence en essaim ont reçu une attention considérable dans les communautés de l'optimisation et de l'intelligence artificielle en raison de leur simplicité, flexibilité, vitesse d'exécution et convergence rapide à résoudre des problèmes d'optimisation. Ces algorithmes s'inspirent principalement de l'intelligence collective et du comportement coopératif d'insectes et d'animaux à savoir les colonies de fourmis, les volées d'oiseaux, les bancs de poissons, les ruches d'abeilles, les troupeaux d'éléphants et les groupes de chauves-souris. Ce sont des techniques de recherche stochastique basées sur la population qui tentent de trouver des solutions quasi optimales. Dans ces algorithmes, nous sacrifions la garantie de trouver des solutions optimales au profit de l'obtention de bonnes solutions dans un laps de temps très court. Les algorithmes d'intelligence en essaim les plus connus sont : Algorithme de Colonies de Fourmis (ACO), Optimisation par Colonies d'Abeilles (BSO), Algorithme de Lucioles (FA) et Algorithme de Chauves-souris (BA). L'un des algorithmes d'intelligence en essaim le plus récent est l'Algorithme de Slime Mould (SMA), proposé par Li et al. en 2020. Cet algorithme est inspiré du comportement et des changements morphologiques de la moisissure visqueuse dans sa recherche de nourriture. L'intérêt manifesté par les chercheurs pour SMA augmente de jour en jour car il a une capacité de recherche globale, une vitesse de convergence rapide et une forte robustesse.

D'autre part, les Services d'Aides Médicales Urgentes (SAMU) fonctionnent dans des environnements très dynamiques et connaît des perturbations assez fréquentes. Le Dispatching des Ambulances d'Urgence (DAU) est parmi ces perturbations qui entraînent une dégradation progressive de la qualité de service. Ce dispatching consiste à choisir les bons véhicules d'urgence à déployer afin de transporter les malades aux hôpitaux les plus proches. Cependant, lorsque les capacités des ambulances sont considérées comme le cas de la pandémie COVID-19, il est important de minimiser le coût d'acheminement à savoir la distance, le temps de déplacement, et aussi le traitement des cas critiques.

Par conséquent, afin de minimiser le coût d'acheminement des véhicules ambulanciers et assurer un bon dispatching des ambulances d'urgence, nous proposons dans ce travail de

modéliser le DAU comme un problème de l'emplacement d'installations (FLP), puis le considérer comme un problème d'optimisation et de le résoudre avec l'algorithme SMA. Dans ce travail, nous évaluons de manière approfondie notre SMA proposé pour DAU. Nous effectuons d'abord des expérimentations préliminaires pour régler les paramètres de SMA. Ensuite, nous comparons les résultats de l'approche proposée avec ceux obtenus par l'algorithme APSO.

Le mémoire est organisé comme suit. Dans le chapitre 1, nous présentons quelques concepts, couvrant le service d'aide médicale urgente (SAMU), le problème de tournées de véhicules (VRP) et le problème de l'emplacement d'installations (FLP). Dans le chapitre 2, nous passons brièvement en revue les problèmes d'optimisation, les Métaheuristiques et l'Algorithme de Slime Mould (SMA). Dans le chapitre 3, nous présentons notre SMA pour le problème de dispatching des ambulances d'urgence. Les résultats expérimentaux sont présentés dans le chapitre 4. Pour finir, nous tirons des conclusions et présentons des perspectives.

# Chapitre 1

# Problème de dispatching de véhicules ambulanciers

Le Dispatching des Ambulances d'Urgence (DAU) est l'une des tâches principales et complexes du Service d'Aide Médicale Urgente (SAMU). Le DAU consiste à sélectionner les bons véhicules d'urgence à déployer afin de transporter les malades aux hôpitaux les plus proches. Le DAU est souvent modélisé comment un Problème de Tournées de Véhicules (VRP) ou un Problème de l'Emplacement d'Installations (FLP).

Dans ce chapitre, nous présentons un état de l'art sur les Services d'aide médicale urgente (section 1.1). Ensuite nous mettons l'accent sur le Problème de Tournées de Véhicules (section 1.2). Puis, nous exposons Problème de l'emplacement d'installations (section 1.3).

## 1.1. Service d'aide médicale urgente

### 1.1.1. Définition

Le Service d'Aide Médicale Urgente (SAMU) est un service hospitalier qui organise le traitement des urgences en dehors de l'hôpital. Ce service a pu montrer dans le passé ces capacités d'adaptation pour faire face à des événements exceptionnels avec un nombre important de victimes, jusqu'à l'apparition du virus COVID-19, la pandémie qui a été sans commune mesure avec tout ce qui avait pu être imaginé auparavant en termes d'ampleur pour une crise sanitaire. Cette situation inédite a nécessité de multiples adaptations successives afin de pouvoir faire face, d'une part, à la demande de soins de la population et, d'autre part, à la doctrine évolutive des autorités de santé [1].

### 1.1.2. Fonctionnement

Chaque SAMU comprend une unité de réception et de régulation médicale des appels et une autre unité, appelée service mobile d'urgence et de réanimation (SMUR) qui apporte les soins d'aide médicale urgente en dehors de l'hôpital. En outre, chaque SAMU gère, sur le plan de la régulation médicale, un ou plusieurs autres SMUR implantés dans les gouvernorats qu'il couvre. Les tâches principales d'un service SAMU sont : *(i)* assurer une écoute médicale

permanente, *(ii)* localiser et dispatcher les ambulances, *(iii)* donner des soins sur place et *(iv)*, transporter les patients aux hôpitaux [2].

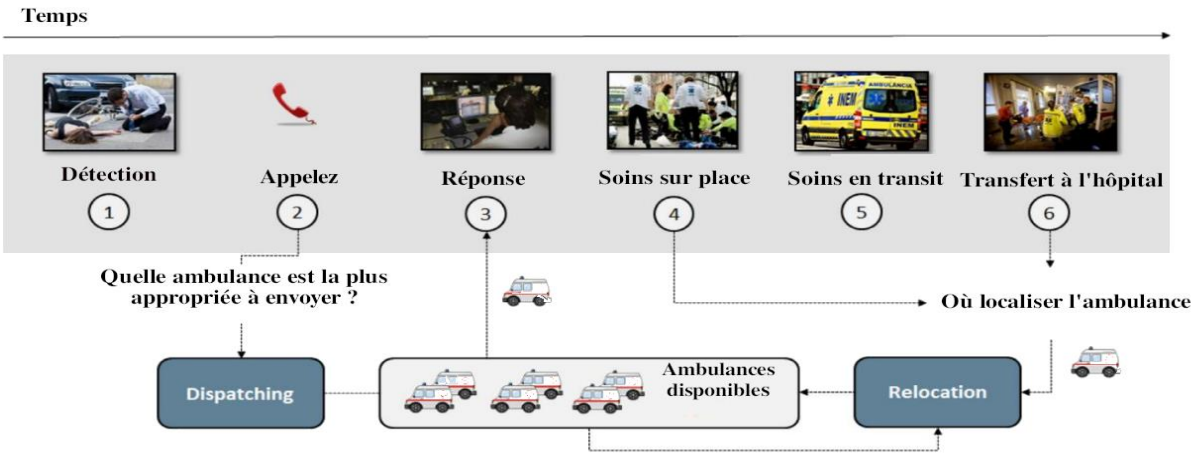



Figure 1.1 – Chaîne de développements SAMU.

### 1.1.3. Problème de dispatching de véhicules ambulanciers

Les responsables SAMU sont de plus en plus préoccupés par l'importance d'améliorer la performance des systèmes SAMU, cela a augmenté l'intérêt des chercheurs opérationnels pour étudier le problème de dispatching qu'y a un impact énorme sur l'optimisation du système SAMU. Le DAU consiste à sélectionner les bons véhicules d'urgence à déployer afin de transporter les malades aux hôpitaux les plus proches. Ainsi, les contributions liées au même problème se sont principalement concentrées d'assurer un plan équilibré entre les ambulances disponibles et maintenir une couverture adéquate afin de pouvoir répondre aux appels d'urgence dans les meilleurs délais [3]. On doit considérer dans ce problème l'affectation des patients aux ambulances et le séquencement le plus approprié des patients au sein de la tournée de chaque ambulance de façon à minimiser la distance totale parcourue. Par conséquent, afin de choisir les bons véhicules d'urgence à déployer, le dispatching des véhicules ambulanciers peut être modélisé comme un Problème de Tournées de Véhicules (VRP) ou un Problème de l'Emplacement d'Installations (FLP) [4].

## 1.2. Problème de tournées de véhicules

### 1.2.1. Définition

Le Problème de Tournées de Véhicules, ***Vehicle Routing Problem (VRP)***, est un nom générique donné à une classe entière de problèmes où un ensemble d'itinéraires doit être spécifié pour une flotte de véhicules dans un ou plusieurs dépôts pour un certain nombre de villes ou de clients géographiquement dispersés. L'objectif de VRP est de servir un groupe de clients ayant

des besoins connus sur les itinéraires de véhicules à moindre coût qui partent et se terminent dans le dépôt [5]. A travers la Figure 1.2, nous pouvons voir une image d'une entrée typique d'un problème VRP et l'une de ses sorties possibles :

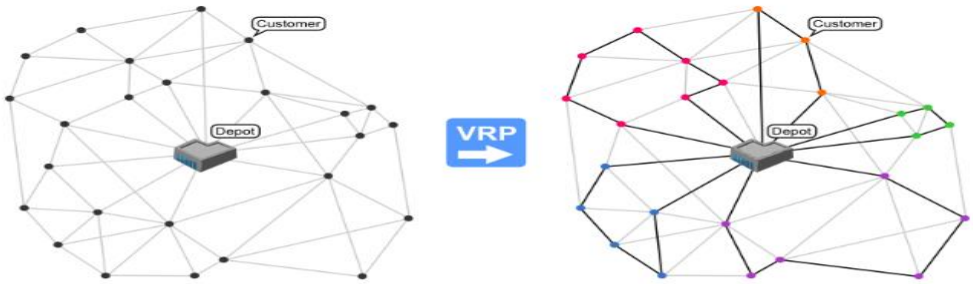



Figure 1.2 – Exemple d'un problème VRP.

## 1.2.2. Modélisation

Considérons un ensemble de clients représenté par $C = \{1, \dots, n\}$. Pour ses clients ces clients, nous devons concevoir des chemins pour une flotte de $K$ véhicules disponibles dans un seul dépôt. Chaque itinéraire doit commencer à partir d'un dépôt, visiter un sous-ensemble de clients, puis revenir au dépôt. Tous les clients doivent être visités une seule fois. Nous représentons le problème de VRP à l'aide d'un graphe $G(N, E)$, dans lequel $N = C \cup \{0, n+1\}$ est l'ensemble des nœuds associés aux clients dans $C$ et aux nœuds de dépôt $0$ et $n+1$. Nous utilisons deux nœuds pour représenter le même dépôt et nous imposons que toutes les routes doivent commencer de $0$ et revenir à $n+1$. L'ensemble $E$ contient les arcs $(i, j)$ pour chaque paire de nœuds $i, j \in N$. Le coût de d'un arc $(i, j) \in E$ est noté $c_{ij}$, on définit la variable de décision binaire $x_{ij}$ qui prend la valeur 1 si et seulement s'il existe une route qui va du client $i$ à $j$ directement, pour $i, j \in N$. L'objectif du problème est de déterminer un ensemble de routes à coût minimal qui satisfait à toutes les exigences et contraintes définies ci-dessous :

*fonction à minimizer*

$$\sum_{i=0}^{n+1} \sum_{j=0}^{n+1} c_{ij} x_{ij} \tag{1.1}$$

*Contraintes*

$$\sum_{\substack{j=1 \\ j \neq i}}^{n+1} x_{ij} = 1 \quad (1 \leq i \leq n) \tag{1.2}$$

$$\sum_{\substack{i=0 \\ i \neq h}}^{n} x_{ih} - \sum_{\substack{j=1 \\ j \neq h}}^{n+1} x_{hj} = 0 \quad (1 \leq h \leq n) \tag{1.3}$$

$$\sum_{j=1}^{n} x_{0j} \leq K \quad (1.4)$$

$$x_{ij} \in \{0,1\} \quad (0 \leq i,j \leq n+1) \quad (1.5)$$

La contrainte (1.2) assure que tous les clients sont visités une seul fois. La contrainte (1.3) garantit le bon flux de véhicules à travers les arcs, en assurant que si un véhicule arrive à un nœud $h \in N$, alors il doit partir de ce nœud. La contrainte (1.4) limite le nombre maximum d'itinéraires à $K$. La contrainte (1.5) assure que toutes les variables sont binaires. La fonction objectif est définie par (1.1) et vise à minimiser le coût total de déplacement A noter que différents types de contraintes sont proposées dans la littérature pour imposer les capacités des véhicules et/ou éviter les sous-tours.

### 1.2.3. Variantes de VRP

Il existe de nombreuses variantes et spécialisations du problème de direction :

**Problème de Tournées de Véhicules avec Capacité et Fenêtres Horaires (CVRPTW) :** Dans ce type de problème, il existe une limitation sur la capacité du véhicule et de la plage de temps de service.

**Problème de Tournées de Véhicules Multi-Dépôts (MDVRP) :** Il comporte plusieurs dépôts répartis sur un territoire géographique donné et dans lesquels les véhicules sont localisés. Chaque véhicule part de n'importe quel dépôt mais revient à son dépôt d'origine.

**Problème de Tournées de Véhicules avec Capacité (CVRP) :** Si les clients ont chacun une demande fixée et que les véhicules ont une capacité de stockage maximum, on parle alors du CVRP.

**Problème de Tournées de Véhicules avec Plusieurs Trajets (VRPMT) :** Ce problème consiste à définir un ensemble de routes qui minimise le coût total de transport.

## 1.3. Problème de l'emplacement d'installations

### 1.3.1. Définition

Le problème de l'emplacement d'installations, ***Facility Location Problem (FLP)***, consiste à trouver des emplacements pour de nouvelles installations de telle sorte que le coût d'affectation des demandes soit minimisé. Dans les applications pratiques que ce soit dans le secteur privé ou public, ces problèmes traitent de décisions stratégiques et à long terme impliquant des coûts d'investissement énormes. En général, lorsqu'une installation est positionnée, elle produit des effets, positifs ou négatifs, sur les utilisateurs, dont l'intensité peut être considérée en fonction de la distance mutuelle. Bien entendu si les effets sont positifs des positions effectives pour les installations sont attendues au plus près des points de demande. C'est le cas des sites d'utilité publique tels que les écoles, les hôpitaux, les magasins, etc. Au contraire, en cas d'effets négatifs les usagers souhaitent que les aménagements soient dans la

mesure du possible. Des exemples de ce genre concernent les centrales électriques ou nucléaires, les dépotoirs. Cependant, dans ces cas également, lorsque les installations sont trop éloignées des points de demande, des coûts supplémentaires doivent être payés en termes de coûts logistiques. Pour cette raison, il est nécessaire de trouver des solutions de compromis capables d'équilibrer les différents aspects [6].

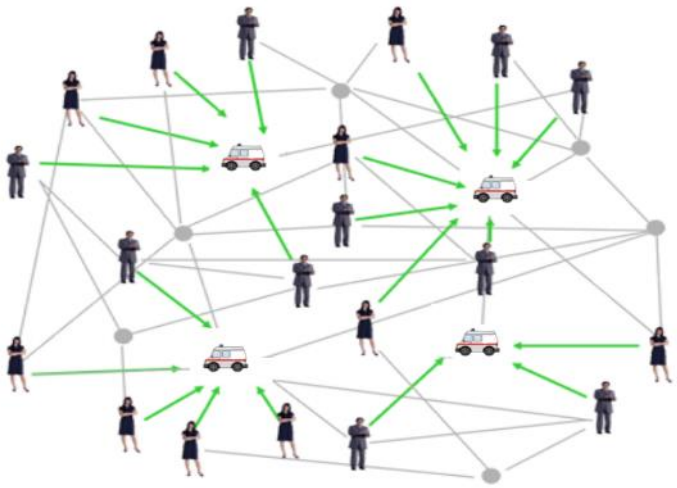


Figure 1.3 – Exemple d'un problème FLP.

## 1.3.2. Modélisation

Le problème de l'emplacement d'installations peut être médiatisé comme un problème $p$-médian. Le problème $p$-médian vise à minimiser la somme des distances entre un ensemble d'installations et un ensemble de demandes. Considérons un ensemble de demandes représentées par $I = \{1, \dots, n\}$ et un ensemble d'installations représentées par $P = \{1, \dots, m\}$. Nous représentons le problème de FLP à l'aide d'un graphe $G((N, M), E)$, dans lequel $N$ est l'ensemble des nœuds associés aux demandes dans $I$ et l'ensemble des nœuds associés aux ressources dans $P$. Chaque demande $i \in I$ doit être affectée à une seule installation $j \in P$. L'ensemble $E$ contient les $arcs(i,j)$ pour chaque paire de nœuds $i \in N$, $j \in M$. Le coût d'un arc $(i,j) \in E$ est noté $c_{ij}$ qui représente la distance entre la demande $i$ et l'installation $j$. Nous définissons la variable de décision binaire $x_{ij}$ qui prend la valeur 1 si et seulement si le point de demande $i$ est alloué à la ressource $j$, pour $i \in N, j \in M$. L'objectif du problème est de minimiser la somme des distances entre les demandes et les installations :

*fonction à minimizer*

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \tag{1.6}$$

*Contraintes*

$$\sum_{j=1}^{m} x_{ij} = 1 \quad (1 \le i \le n) \tag{1.7}$$

$$x_{ij} \in \{0,1\} \quad (1 \le i \le n, 1 \le j \le m) \tag{1.8}$$

La contrainte (1.7) garantit que chaque demande est assignée à une installation. La contrainte (1.8) assure que toutes les variables sont binaires. La fonction objectif est définie par (1.6) et vise à minimiser la somme des distances entre les demandes et les demandes.

### 1.3.3. Variantes de FLP

Dans cette section, nous fournissons les modèles et les variantes les plus utilisés pour le problème FLP [7]:

**Problème *p*-médian :** ce problème vise à minimiser la somme pondérée des distances entre p installations à ouvrir et un ensemble de points de demande, dans le modèle il y a l'hypothèse importante qu'il faut localiser exactement p installations, c'est une situation considérable très proche de la pratique.

**Problème *p*-center :** ce problème cherche à minimiser la distance maximale entre toute demande et son installation la plus proche.

**Problème de couverture :** ce problème vise à s'assurer que chaque client est *couvert*, c'est-à-dire, chaque client est affecté à une installation si seulement si la distance entre le client et l'installation est inférieure à un certain seuil.

## 1.4. Conclusion

Ce chapitre, nous a permis de décrire le Service d'Aide Médicale Urgente (SAMU), de présenter ces tâches essentiels et de mettre l'accent sur le problème de dispatching des ambulances d'urgence qui peut être vu comme un Problème de Tournées de Véhicules (VRP) ou un Problème de l'Emplacement d'Installations (FLP).

Dans le chapitre qui suit, nous présentons un état de l'art sur les problèmes d'optimisation et les méthodes de résolution de ces problèmes à savoir les Métaheuristiques et les algorithmes d'Intelligence en Essaim.

# Chapitre 2

# Algorithme Slime Mould

Les algorithmes d'Intelligence en Essaim sont actuellement les techniques les plus couramment utilisées pour résoudre les problèmes d'optimisation difficiles. L'algorithme Slime Mould (SMA) fait partie des algorithmes d'intelligence en essaim les plus récents et les plus efficaces en raison de sa simplicité et de sa flexibilité.

Dans ce chapitre, nous présentons un état de l'art sur les problèmes d'optimisations (section 2.1), ensuite nous mettons l'accent sur les Métaheuristiques (section 2.2), en particulier, nous exposons l'Algorithme Slime Mould (section 2.3).

## 2.1. Problème d'optimisation

### 2.1.2. Définition

Un problème d'optimisation est obtenu à partir d'un problème de recherche en associant à chaque solution une valeur, Il consiste à trouver parmi un ensemble de solutions potentielles celle qui répond le mieux à certains critères décrits sous forme d'une fonction objectif (coût):

$$f\ (x_*) \leq\ f(x), \forall\ x\ \in\ S$$

L'objective est donc de trouver une solution optimale $x_* \in S$ pour lequel $f\ (x_*) \leq f$ pour tout $x \in\ S$.

### 2.1.3. Classification des problèmes d'optimisation

Pour résoudre un problème d'optimisation, il est important de bien identifier à quelle catégorie ce problème appartient. La classification des problèmes d'optimisation change d'un auteur à un autre, par exemple, on distingue [8]:

> **Problèmes d'optimisation discrets/continus :** dans certains cas, les variables de décision sont discrètes, le plus souvent sous la forme d'entiers ou de binaires. Ce problème d'optimisation est dit discret. Cependant, dans les problèmes d'optimisation continue, les variables peuvent prendre n'importe quelle valeur, ce sont des réels. Les problèmes d'optimisation continue sont généralement plus simples à résoudre.

**Problèmes d'optimisation mono-objectifs/multi-objectifs :** les problèmes mono-objectifs sont définis par une unique fonction objectif. Les problèmes multi-objectifs existent quand un compromis est à rechercher entre plusieurs objectifs contradictoires. Il est éventuellement possible de reformuler un problème multi-objectif a un ensemble de fonctions objectifs en une somme pondérée sous forme d'une combinaison des différents objectifs ou en transformant des objectifs sous forme de contraintes.

**Problèmes d'optimisation sans/avec contraintes :** il est important de bien distinguer les problèmes où des contraintes existent sur les variables de décision. Ces contraintes peuvent être simplement des bornes et aller jusqu'à un ensemble d'équations de type égalité et de type inégalité. Naturellement, les problèmes avec contraintes sont plus compliqués à résoudre.

### 2.1.4. Résolution des problèmes d'optimisation

La solution optimale à un problème d'optimisation ne peut que très rarement être déterminée en un temps polynomial. Il est donc souvent nécessaire de trouver des modes de résolution qui fournissent une solution de bonne qualité dans un laps de temps raisonnable [9].

**Méthodes exactes :** les méthodes exactes sont des méthodes qui garantissent la complétude de la résolution. Autrement dit, ces méthodes donnent à tous les coups la solution optimale. Le temps de calcul nécessaire de telles méthodes augmente en général exponentiellement avec la taille du problème à résoudre. Parmi les méthodes de résolution exactes les plus connus, nous pouvons citer : la méthode de *séparation et évaluation*, la *programmation dynamique,* la *programmation par contraintes* et la *programmation linéaire*.

**Méthodes approchées :** Contrairement aux méthodes exactes, les méthodes approchées ne procurent pas forcément une solution optimale, mais seulement une bonne solution en un temps de calcul aussi faible que possible. Les méthodes approchées les plus connus sont les *Métaheuristiques*.

## 2.2. Métaheuristiques

### 2.2.1. Définition

Une Métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficile pour lesquels on ne connaît pas de méthode classique plus efficace. Il existe un grand nombre de Métaheuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale. Ces méthodes utilisent cependant un haut niveau d'abstraction, leur permettant d'être adaptées à une large gamme de problèmes différents. Les Métaheuristiques sont souvent inspirées par des systèmes naturels, qu'ils soient pris en physique (cas du *Recuit Simulé*), en biologie (cas des *Algorithmes Génétiques*) ou encore en éthologie (cas des algorithmes de colonies de fourmis ou de l'optimisation par essaim de particule).

### 2.2.2. Intensification et diversification

Une Métaheuristique doit toujours garder un équilibre entre l'*Intensification* et la *Diversification* [10].

> **Intensification (exploitation) :** désigne le processus qui dirige la procédure pour récolter de l'information sur le problème à optimiser. La stratégie de diversification la plus simple consiste à redémarrer périodiquement le processus de recherche à partir d'une solution générée aléatoirement ou choisie judicieusement dans une région non encore visitée de l'ensemble des solutions admissibles.
>
> **Diversification (exploration) :** elle utilise l'information déjà récoltée pour explorer, en détail, les zones jugées prometteuses dans l'espace de recherche. Sa mise en œuvre réside, le plus souvent, dans l'élargissement temporaire du voisinage de la solution courante.

La diversification donne un comportement de recherche globale. Tandis que l'intensification donne à la Métaheuristique une caractéristique de recherche locale. Pour une Métaheuristique, la difficulté est de réaliser un équilibre entre ces deux principes afin de converger vers l'optimum global de l'espace de recherche, tout en évitant de rester bloqué sur un optimum local.

### 2.2.3. Classification des Métaheuristiques

Il existe plusieurs manières pour classifier les Métaheuristiques dans la littérature :

**Selon le nombre de solutions :**

*Métaheuristiques basées sur une solution unique* : les Métaheuristiques à solution unique, appelées aussi méthodes de trajectoire, commencent avec une seule solution initiale et procède à son amélioration progressivement, en construisant une trajectoire dans l'espace de recherche. Quelques exemples de ces Métaheuristiques sont : *Recuit Simulé (SA)* et *Recherche Tabou (TS)*.

*Métaheuristiques basées sur une population de solutions* : les Métaheuristiques à base de population améliorent une population de solutions tout au long du processus de recherche telles que les *Algorithmes Evolutionnaires*.

**Selon le type d'algorithme :**

*Métaheuristiques basées sur l'évolution* : ce sont des méthodes d'optimisation stochastique qui se basent sur des simulations brutes de l'évolution naturelle des populations. Précisément, ce sont des techniques de programmation qui s'inspirent du principe de l'évolution des espèces décrit par Darwin. Quelques exemples de ces Métaheuristiques sont : *Algorithme Génétique (GA)* et *Algorithme à Evolution Différentielle (DE)*.

*Métaheuristiques basées sur les essaims* : ce sont des méthodes qui s'inspirent de la nature et qui utilisent une population de solutions pour produire une solution

optimale au problème. Quelques exemples de ces Métaheuristiques sont : *Algorithme de Colonies de Fourmis (ACO)* et *Optimisation par Essaim de Particules (PSO).*

*Métaheuristiques basées sur le comportement humain* : ce sont des méthodes qui s'inspirent de l'immunologie théorique, des fonctions, des principes et des mécanismes humains observés.

# 2.3. Algorithme Slime Mould

## 2.3.1. Définition

L'algorithme de Slime Mould (SMA), proposé par Li et al. en 2020 [11], est une technique d'optimisation basée sur la population. L'inspiration principale de SMA est le comportement et les changements morphologiques de la moisissure visqueuse dans sa recherche des chemins les plus courts vers les sources de nourriture. Le comportement de recherche de nourriture de la moisissure visqueuse comprend principalement trois étapes : *approcher la nourriture*, *emballer la nourriture* et *saisir la nourriture*. En d'autres termes, lors de la recherche de nourriture, la moisissure visqueuse génère des ondes de propagation à l'aide d'oscillateurs biologiques pour augmenter le flux cytoplasmique à travers la veine et approcher des concentrations de nourriture plus élevées. Puis, elle entoure les aliments et sécrète des enzymes pour les digérer.

Figure 2.1 – Moisissure visqueuse.

## 2.3.2. Modélisation

En simulation, chaque individu $i$ dans la moisissure visqueuse a une position actuelle $S_i$ et une valeur de fitness $f(S_i)$, et se déplace en direction de sources de nourriture de haute qualité. Formellement, soit $S$ un essaim ou une moisissure visqueuse avec $N$ individus où chaque

individu ou solution $i$ ($1 \leq i \leq N$) a une position $S_i = \{S_{i1}, S_{i2}, \dots, S_{iD}\}$ dans un espace de dimension $D$. Soit $f(S_i)$ la valeur objectif de l'individu $i$ et soit $S_* = \{S_{*1}, S_{*2}, \dots, S_{*D}\}$ le meilleur emplacement global avec la concentration d'odeur la plus élevée trouvée. A chaque itération $t$, chaque individu $i$ dans la population ajuste sa position $S_i^t$ selon deux paramètres. Le premier paramètre, noté $z$, assure l'équilibre entre les phases d'*exploration* et d'*exploitation*, tandis que le second paramètre, noté $p_i$, contrôle l'équilibre entre les phénomènes d'*approche* et d'*emballement*. La valeur de $p_i$ est calculée à l'aide de l'Equation **2.1.**:

$$p_i = \tanh\left|f(S_i^t) - f(S_*)\right| \tag{2.1}$$

Lorsque la valeur de $z$ est supérieure à un nombre aléatoire $r_1$, la phase d'exploration est activé et la position $S_i^t$ est mise à jour aléatoirement comme indiqué par l'Equation **2.2.**:

$$S_i^{t+1} = r_2(S_{max} - S_{min}) + S_{min} \tag{2.2}$$

Où :

$r_1$, $r_2 \in [0,1]$, sont des nombres réels aléatoires.

$S_{max}$, $S_{min}$, sont les bornes supérieure et inférieure de l'espace de recherche.

En revanche, lorsque $z$ est inférieur à $r_1$, la phase d'exploitation est activée et la position $S_i^t$ est mise à jour en fonction de la valeur de $p_i$. Si la valeur de $p_i$ est inférieure à un autre nombre aléatoire $r_3$, la nouvelle position de $i$ est déterminée en fonction du phénomène d'approche comme indiqué par l'Equation **2.3.**:

$$S_i^{t+1} = v_c x_i^t \tag{2.3}$$

Où :

$r_3 \in [0,1]$, est un nombre réel aléatoire.

$v_c$, est un paramètre d'oscillation qui varie de manière aléatoire dans la plage $[-b, b]$ et converge de manière monotone vers zéro. La valeur de $b$ ($0 \leq b \leq 1$) est calculée en fonction du numéro de l'itération actuel $t$ et le nombre maximal d'itérations $T$, comme défini par l'Equation **2.4.**:

$$b = 1 - \left(\frac{t}{T}\right) \tag{2.4}$$

Lorsque la valeur de $p_i$ est supérieure à $r_3$, la position $S_i^t$ est modifiée en fonction du phénomène d'emballement, comme déterminé par l'Equation **2.5.**:

$$S_i^{t+1} = S_* + v_b(w_i S_A^t - S_B^t) \tag{2.5}$$

Où :

$S_A^t$, $S_B^t$, sont les positions de deux individus choisis au hasard à partir de la population.

$v_b$, est un paramètre d'oscillation qui varie de manière aléatoire dans la plage $[-a, a]$ et diminue progressivement jusqu'à zéro. La valeur de $a$ est calculée en fonction du numéro de l'itération actuel $t$ et le nombre maximal d'itérations $T$, comme indiqué par l'Equation **2.6.**:

$$a = \operatorname{arctanh}\left(1 - \left(\frac{t}{T}\right)\right) \tag{2.6}$$

$w_i$, est le poids de l'individu $i$ dans la moisissure visqueuse, défini par l'Equation **2.7.**:

$$w_i = \begin{cases} 1 + r\log(g_i), & si\ i \in S_F \\ 1 - r\log(g_i), & sinon \end{cases}$$
$$g_i = \frac{f(S_b^t) - f(S_i^t)}{f(S_b^t) - f(S_w^t)} + 1 \qquad (2.7)$$

Où :

$S_F$, est l'ensemble des individus qui sont classés, en fonction de leurs valeurs objectifs, dans la première moitié de la population.

$f(S_b^t)$, est la meilleure valeur objectif dans l'itération actuelle.

$f(S_w^t)$, est la mauvaise valeur objectif dans l'itération actuelle.

$r \in [0,1]$, est un nombre réel aléatoire.

### 2.3.3. Algorithme

Les principales étapes de l'algorithme SMA sont présentées dans l'Algorithme **2.1**.

**Algorithme 2.1. :** Pseudo code de l'algorithme Slime Mould (SMA)

```
Générer aléatoirement une population initiale de N solutions
Répéter
    Evaluer la qualité de la population initiale
    Sélectionner la solution optimale S*
    Pour i de 1 à N (tous les N individuels) Faire
        Mettre à jour le poids w_i en utilisant l'Equation 2.7
    Fait;
    Pour i de 1 à N (tous les N individuels) Faire
        Si z > r_1 Alors
            Mettre à jour la position S_i^t en utilisant l'Equation 2.2
        Sinon
            Mettre à jour p_i en utilisant l'Equation 2.1
            Si p_i < r_3 Alors
                Mettre à jour b en utilisant l'Equation 2.4 et calculer v_c
                Mettre à jour la position S_i^t en utilisant l'Equation 2.3
            Sinon
                Mettre à jour a en utilisant l'Equation 2.6 et calculer v_b
                Mettre à jour la position S_i^t en utilisant l'Equation 2.5
            Fsi;
        Fsi;
    Fait;
Jusqu'à nombre maximum d'itérations atteint ou solution optimale trouvée
Classer les faucons et retourner la solution optimale S*
```

## 2.4. Conclusion

Dans le cadre de ce chapitre, nous avons présenté les principales notions et concepts des problèmes d'optimisation, des Métaheuristiques ainsi que la modélisation et l'algorithme d'optimisation Slime Mould.

Le chapitre suivant présente la démarche que nous avons élaborée pour résoudre le problème de dispatching de véhicules ambulanciers à l'aide de l'algorithme Slime Mould.

# Chapitre 3

# SMA pour le dispatching des ambulances d'urgence

Dans ce chapitre, nous présentons notre approche basée sur l'algorithme SMA pour résoudre le problème de dispatching des ambulances d'urgence et assurer une répartition utile et optimale des ambulances.

Nous commençons d'abord par discuter la complexité de l'espace de recherche de DAU (section 3.1). Ensuite, nous décrivons notre approche basée SMA pour résoudre le problème de DAU (section 3.2).

## 3.1. Complexité du problème DAU

### 3.1.1. Modélisation du problème DAU

Dans ce travail, nous considérons le problème de dispatching des ambulances d'urgence comme un problème FLP, tel que un appel d'urgence est considéré comme une demande, un hôpital est considéré comme une installation et chaque appel d'urgence sera satisfait par un hôpital, ou, en d'autres termes, chaque patient sera affecté à une ambulance (voir Figure 3.1).

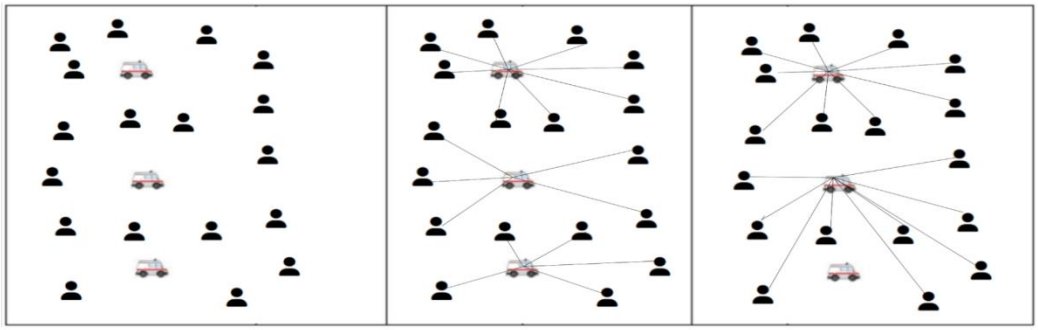


Figure 3.1 – Modélisation de DAU comme un problème FLP.

### 3.1.2. Complexité de l'espace de recherche de DAU

La complexité de l'espace de recherche de DAU est exponentielle. Cet espace de recherche inclut toutes les assignations potentielles patients-ambulances. Le nombre de ces affectations potentielles (solutions possibles) est donné par : $A^P$ avec $A$ et $P$ sont respectivement le nombre d'ambulances et le nombre de patients. Par exemple, pour un problème de DAU avec 2 patients et 3 ambulances, le nombre des affectations potentielles est donné par : $3^2 = 9$. Le Tableau 3.1 montre toutes les affectations possibles patients-ambulances pour 2 patients et 3 ambulances.

| | Patient n°1 | Patient n°2 |
|---|---|---|
| Affectation n°1 | Ambulance n°1 | Ambulance 1 |
| Affectation n°2 | Ambulance n°1 | Ambulance 2 |
| Affectation n°3 | Ambulance n°1 | Ambulance 3 |
| Affectation n°4 | Ambulance n°2 | Ambulance 1 |
| Affectation n°5 | Ambulance n°2 | Ambulance 2 |
| Affectation n°6 | Ambulance n°2 | Ambulance 3 |
| Affectation n°7 | Ambulance n°3 | Ambulance 1 |
| Affectation n°8 | Ambulance n°3 | Ambulance 2 |
| Affectation n°9 | Ambulance n°3 | Ambulance n°3 |

Tableau 3.2 – Affectations possibles pour 2 patients et 3 ambulances.

### 3.1.3. Résolution de l'espace exponentiel de DAU

Afin de traiter l'espace de recherche exponentiel de DAU et trouver la meilleure affectation dans un délai raisonnable, nous considérons le problème de dispatching des ambulances d'urgence comme un problème d'optimisation et nous résolvons à l'aide des algorithmes d'intelligence en essaim, plus précisément l'Algorithme Slime Mould (SMA). Le DAU peut être donc considéré comme un problème d'optimisation, tel que :

- Toutes les affectations possibles patients-ambulances représentent l'ensemble des solutions potentielles, notée $S$.
- Une affectation patient-ambulance représente une solution potentielle, notée $s$.
- La fonction définie dans la modélisation de FLP (Equation 1.6), qui vise à minimiser la somme des distances entre les demandes et les installations représente la fonction objectif, notée $f$. Cette fonction est étendue en donnant la priorité aux cas critiques. fonction objectif $f$ est déterminée dans la Section 3.2.2.

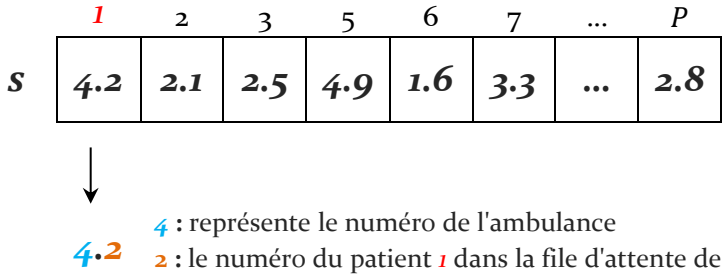
# 3.2. SMA pour le problème de DAU

Afin d'appliquer le SMA pour le problème de DAU, nous déterminons d'abord la représentation de la solution et la fonction objectif. Puis, nous présentons la mise à jour des affectations patients-ambulances à l'aide de l'algorithme SMA. Afin d'appliquer le SMA pour résoudre le problème de DAU,

Pour appliquer le SMA pour le problème de dispatching des ambulances d'urgence, nous devons décrire l'encodage de la solution, la fonction objectif et l'ajustement des affectations.

## 3.2.1. Encodage de la solution

Une solution $s$ du problème est une affectation possible patient-ambulance représentée par un vecteur de réels de $P$ éléments, comme suit :




Une solution est dite réalisable ou acceptable si seulement si $1 \leq s[i] < A + 0.9$ tel que : s[i] est générée aléatoirement, $A$ représente le nombre d'ambulances et $P$ représente le nombre de patients.

## 3.2.2. Fonction objectif

La fonction objectif $f$ inclut deux fonctions objectifs. La première fonction $f_1$ vise à minimiser la somme des distances entre les hôpitaux et les patients. Cette fonction est donnée par l'Equation 3.1.:

$$f_1(s) = min \sum_{i=1}^{P} \sum_{j=1}^{A} x_{ij} d_{ij} \qquad (3.1)$$

Où :

$x$, est une matrice binaire qui décrit une affectation patient-ambulance, tel que $x_{ij} = 1$ si seulement si le patient $i$ est affecté à l'ambulance $j$.

$d$, est une matrice de réels qui contient toutes les distances patients-ambulances, tel que $d_{ij}$ est la distance Euclidienne entre le patient $i$ et l'ambulance $j$.

La deuxième fonction $f_2$ vise à donner la priorité aux patients dans des situations critiques Cette fonction est donnée par l'Equation 3.2.:

$$f_2(s) = min \sum_{i=1}^{P} \sum_{j=1}^{A} y_{ij} h_{ij} \quad (3.2)$$

Où :

$y$, est une matrice d'entiers qui contient les rangs des patients dans les files d'attentes des ambulances, tel que $y_{ij}$ est le rang du patient $i$ dans la file d'attente de l'ambulance $j$.

$h$, est un vecteur binaire qui contient des informations sur les cas critiques, tel que $h_i = 1$ si le patient $i$ est prioritaire, 0 sinon. La fonction objectif f est donc donnée par l'Equation 3.3.:

$$f(s) = \min\big(f_1(s) + f_2(s)\big) \quad (3.3)$$

Le problème traité peut être considéré comme un problème d'optimisation multi-objectifs. Comme mentionné dans le deuxième chapitre (Section 2.1.3), les problèmes multi-objectifs peuvent être reformulés en transformant les objectifs en une somme pondérée. Par conséquent, nous transformons la fonction objectif ou l'ensemble de fonctions objectifs en une somme pondérée, comme suit :

$$f(s) = \min\big(w_1 f_1(s) + w_2 f_2(s)\big) \quad (3.4)$$

Où :

$w_1 + w_2 = 1$

Afin de donner plus de sens à la fonction objectif, les deux fonctions constituant la fonction $f$ sont normalisées entre 0 et 1 à l'aide de la méthode de Normalisation Min-Max, comme suit :

$$f_1(s)_N = \frac{f_1(s) - f_1(s)_{MIN}}{f_1(s)_{MAX} - f_1(s)_{MIN}} \quad (3.5)$$

$$f_2(s)_N = \frac{f_2(s) - f_2(s)_{MIN}}{f_2(s)_{MAX} - f_2(s)_{MIN}} \quad (3.6)$$

Où :

$f_1(s)$ et $f_2(s)$, indiquent respectivement les valeurs objectifs retournées par les fonctions $f_1$ et $f_2$ avant normalisation.

$f_1(s)_N$ et $f_2(s)_N$, indiquent respectivement les valeurs objectifs retournées par les fonctions $f_1$ et $f_2$ après normalisation.

$f_1(s)_{MIN}$ et $f_2(s)_{MIN}$, indiquent respectivement les valeurs objectifs minimales retournées par les fonctions $f_1$ et $f_2$. Pour notre problème, $f_1(s)_{MIN}$ et $f_2(s)_{MIN}$ sont initialisées à 0.

$f_1(s)_{MAX}$ et $f_2(s)_{MAX}$, indiquent les valeurs objectifs maximales renvoyées par $f_1$ et $f_2$. Elles représentent la situation où les patients sont transportés par une seule ambulance.

### 3.2.3. Ajustement des affectations patients-ambulances à l'aide de SMA

Les principales étapes de l'algorithme SMA pour le problème DAU sont présentées dans l'Algorithme 3.1.

**Algorithme 3.1. :** Pseudo code de l'algorithme SMA pour le problème de DAU

***Générer aléatoirement une population de N solutions***
Chaque affectation ou solution $i$ possède une position $s_i^0$
$s_i^0$ est un vecteur de réels de $P$ éléments

***Répéter***
***Evaluer la population***
Pour $i$ de 1 à $N$ Faire
Calculer $f(s_i)$ en utilisant l'Equation 3.1

***Trouver la solution optimale $s_*$***
$s_* \leftarrow s_1$
$f(s_*) \leftarrow f(s_1)$
Pour $i$ de 2 à $N$ Faire Faire
Si $(f(s_i) < f(s_*))$ Alors
$s_* \leftarrow s_1$
$f(s_*) \leftarrow f(s_1)$

***Ajustement des affectations patients-ambulances***
Pour $i$ de 1 à $N$ Faire
Mettre à jour le poids $w_i$ en utilisant l'Equation 2.7
Fait;
Pour $i$ de 1 à $N$ Faire
Si $z > r_1$ Alors
Mettre à jour la position $s_i^t$ en utilisant l'Equation 2.2
Sinon
Mettre à jour $p_i$ en utilisant l'Equation 2.1
Si $p_i < r_3$ Alors
Mettre à jour $b$ en utilisant l'Equation 2.4 et calculer $v_c$
Mettre à jour la position $s_i^t$ en utilisant l'Equation 2.3
Sinon
Mettre à jour $a$ en utilisant l'Equation 2.6 et calculer $v_b$
Mettre à jour la position $s_i^t$ en utilisant l'Equation 2.5
Fsi;
Fsi;
Fait;
***Jusqu'à*** *nombre maximum d'itérations atteint ou solution optimale trouvée*
*Classer les affectations et renvoyer la solution optimale* $s_*$

## 3.3. Conclusion

Dans le cadre de ce chapitre, nous avons discuté la complexité de l'espace de recherche du problème de DAU. Afin de traiter son espace de recherche exponentiel, nous avons considéré ce problème comme un problème d'optimisation et nous l'avons résolu à l'aide de SMA

Dans le chapitre suivant nous présentons les expérimentations réalisées pour valider notre approche proposée.

# Chapitre 4

# Tests et résultats expérimentaux

Nous présentons dans ce chapitre les expérimentations que nous avons effectuées. Ces expérimentations ont pour objectif de valider notre algorithme SMA pour le problème de dispatching des ambulances d'urgence.

D'abord nous introduisons l'environnement de test (section 4.1). Ensuite, nous présentons les différents tests effectués pour régler les paramètres de SMA (section 4.2). Enfin, nous décrivons les résultats expérimentaux (section 4.3).

## 4.1. Présentation de l'environnement de test

### 4.1.1. Collection de test

Une collection de test est une carte constituée d'un ensemble de patients et d'un ensemble d'hôpitaux répartis sur plusieurs zones. Chaque patient est défini par les coordonnées $(x, y)$. Les patients sont divisés en deux catégories : cas normaux et cas critiques. Les hôpitaux sont aussi définis par les coordonnées $(x, y)$. La collection de test est générée aléatoirement et pour simuler un cas naturel, les positions des patients sont générées en utilisant la loi normale (voir la Figure 4.1).

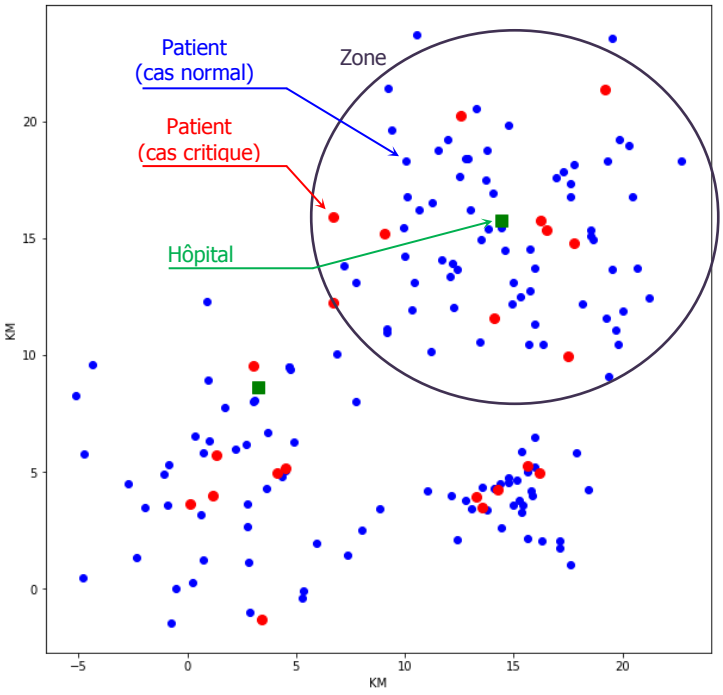



Figure 4.1 – Collection de test représentant les cas normaux et les cas critiques dans une région.

Afin d'évaluer notre SMA pour DAU, nous utilisons plusieurs collections de test. Tableau 4.1. présente les différentes collections de test générées.

| *Collection* | *n°1* | *n°2* | *n°3* | *n°4* |
|---|---|---|---|---|
| *Nombre de zones* | 2 | 3 | 2 | 3 |
| *Nombre de patients* | 50 | 50 | 100 | 100 |
| *Nombre de patients : cas critiques* | 10 | 15 | 10 | 15 |
| *Nombre d'hôpitaux* | 2 | 2 | 2 | 2 |
| *Nombre d'ambulances* | (2, 1) | (2, 1) | (2, 3) | (2, 3) |

Tableau 4.1 – Les détails des collections de test générées.

### 4.1.2. Outils et mesures d'évaluation

**Software :** afin de réaliser notre SMA pour le problème de DAU, nous utilisons le langage Python à travers les logiciels suivants :

*PyCharm* est un environnement de développement intégré (IDE) utilisé pour programmer en Python. Il a pour rôle de faciliter l'écriture des codes ainsi que l'analyse des codes.

*NumPy* est une bibliothèque utilisée en Python, elle permet de manipuler des vecteurs à plusieurs dimensions et elle propose des fonctions mathématiques complètes.

*PyQt* est composée d'une bibliothèque Python Qt et d'une application Qt designer permettant de créer des interfaces graphique.

*Pandas* est un outil d'analyse et de manipulation de données open source rapide, puissant, flexible et facile à utiliser, construit sur le langage de programmation Python.

*Jupyter* est un outil de développement interactif basé sur le Web pour le code et les données, son interface flexible permet aux utilisateurs de configurer et d'organiser des flux de travail en science des données. Une conception modulaire invite les extensions à étendre et enrichir les fonctionnalités.

**Hardware :** la machine suivante est utilisée pour effectuer les expérimentations :

| Processeur | RAM | Système d'exploitation |
|---|---|---|
| Intel® Core™ i5-10 génération CPU @ 1.70GHz × 8 | 16 Go | Ubuntu |

Tableau 4.2 – Le hardware utilisé pour évaluer l'approche proposée.

**Mesure d'évaluation :** Le temps d'exécution et le coût retourné par la fonction objectif $f$ sont utilisés pour évaluer l'approche proposée.

**Algorithme de comparaison :** Les résultats de SMA proposé sont comparés avec ceux de l'algorithme d'Optimisation Accélérée des Essaims de Particules (APSO), qui est une version améliorée de PSO.

## 4.2. Réglages des paramètres de SMA

Avant de comparer les résultats de l'algorithme proposé avec ceux obtenus par l'algorithme SMA, nous fixons d'abord les paramètres de SMA, à savoir la taille de la population, notée $N$, le nombre maximum d'itérations, notée $T$, et la constante utilisée pour assurer l'équilibre entre les phases d'exploration et d'exploitation, notée $Z$. Dans ces tests initiaux, nous exécutons notre algorithme 50 fois et nous varions dans chaque exécution les paramètres $N, T$ *et* $Z$ aléatoirement. Afin de garantir des résultats de test fiables et précis, lors du réglage des paramètres, l'algorithme SMA est exécuté 50 fois pour chaque variation de paramètres, puis une moyenne des résultats est calculée. La Figure 4.2 présente les résultats de coût $f$ en fonction de temps d'exécution.

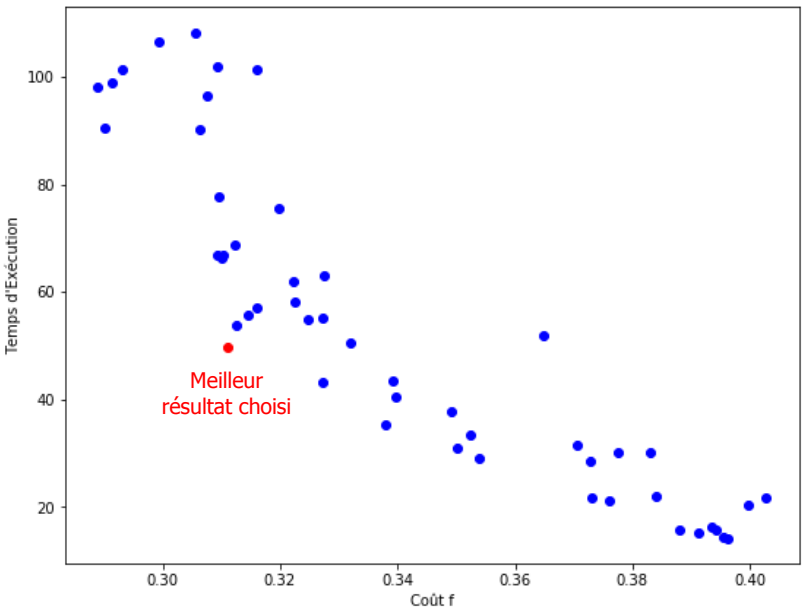



Figure 4.2 – Les résultats de coût $f$ en fonction de temps d'exécution.

Sur la base de ces tests préliminaires, les valeurs des paramètres de SMA, utilisées dans tous les tests suivants, sont présentées dans le Tableau 4.3.:

| | |
|---|---|
| ***N*** *(taille de la population)* | 19 |
| ***T*** *(nombre d'itérations)* | 37 |
| ***Z*** *(constante)* | 0.05 |

Tableau 4.3 – Les paramètres de SMA après le réglage.

## 4.3. Résultats expérimentaux

Dans cette première phase d'expérimentations, les résultats de SMA sont comparés avec ceux de l'algorithme APSO. Dans ces tests, le poids $w_1$ est fixé à 0.4, puis à 0.8. Les Figures 4.3 et 4.4 présentent respectivement les temps d'exécution de chaque algorithme et les coûts retournés par la fonction f en fonction du nombre d'itérations $T$.

Cas $w_1 = 0.4$ Cas $w_1 = 0.8$

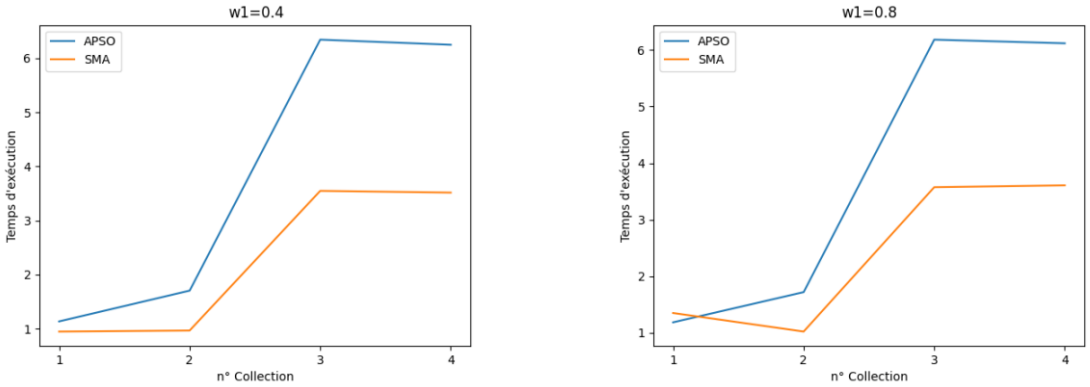



Figure 4.3 –Comparaison des performances de SMA et d'APSO en termes de temps d'exécution.

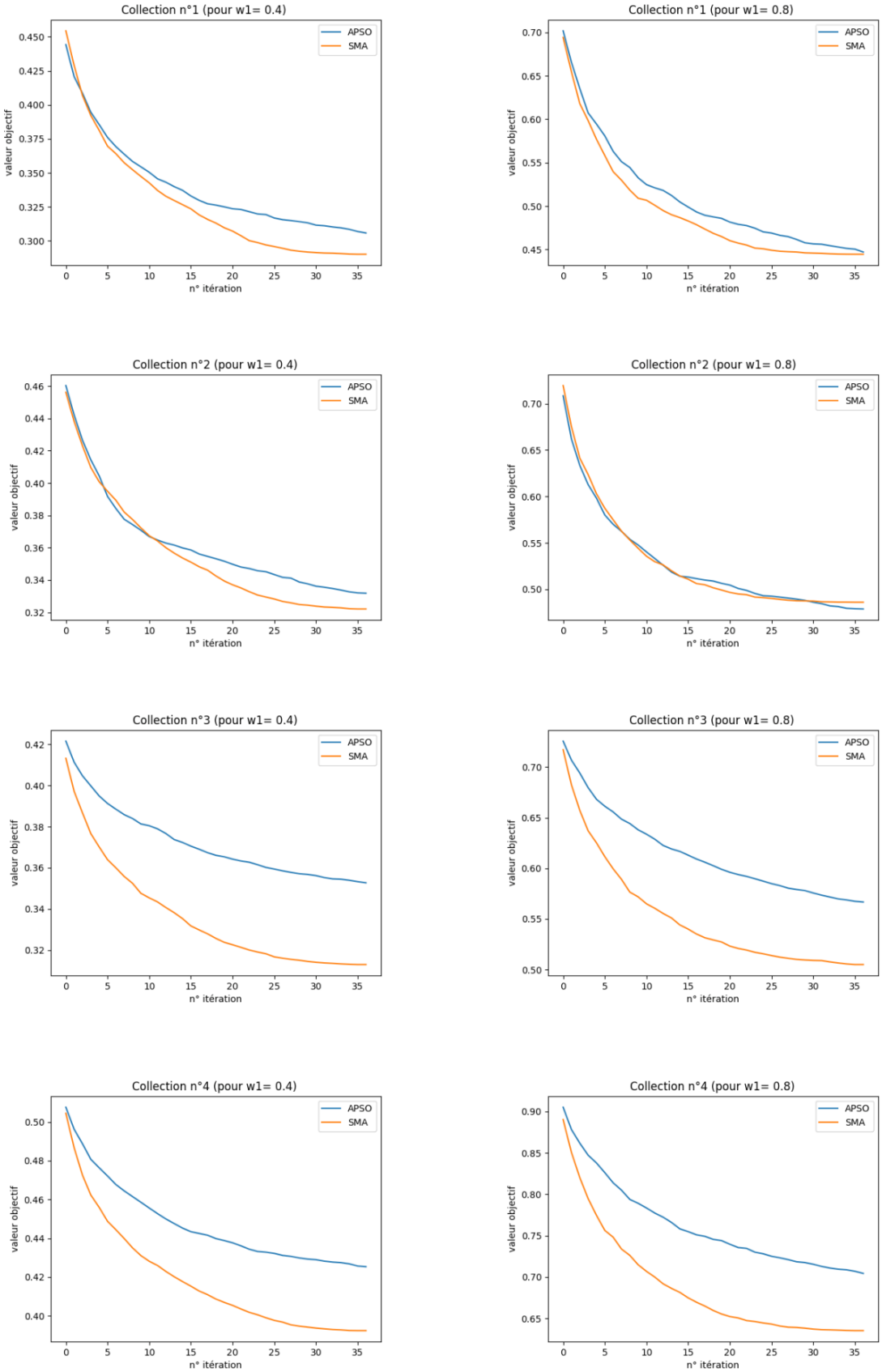



Figure 4.4 – Les coûts $f$ de SMA et d'APSO en fonction du nombre d'itérations.

A partir de la Figure 4.3, nous pouvons constater que l'approche proposée produit le temps de calcul le plus court dans tous les cas et réalise des améliorations significatives par rapport à l'algorithme APSO. De plus, la Figure 4.4 montre que l'approche proposée produit des valeurs objectifs meilleurs par rapport à l'algorithme APSO.

Dans la prochaine série d'expérimentations, le poids $w_1$ est varié de 0.0 à 1.0. La Figure 4.5 présente les meilleurs coûts retournés par SMA et APSO.

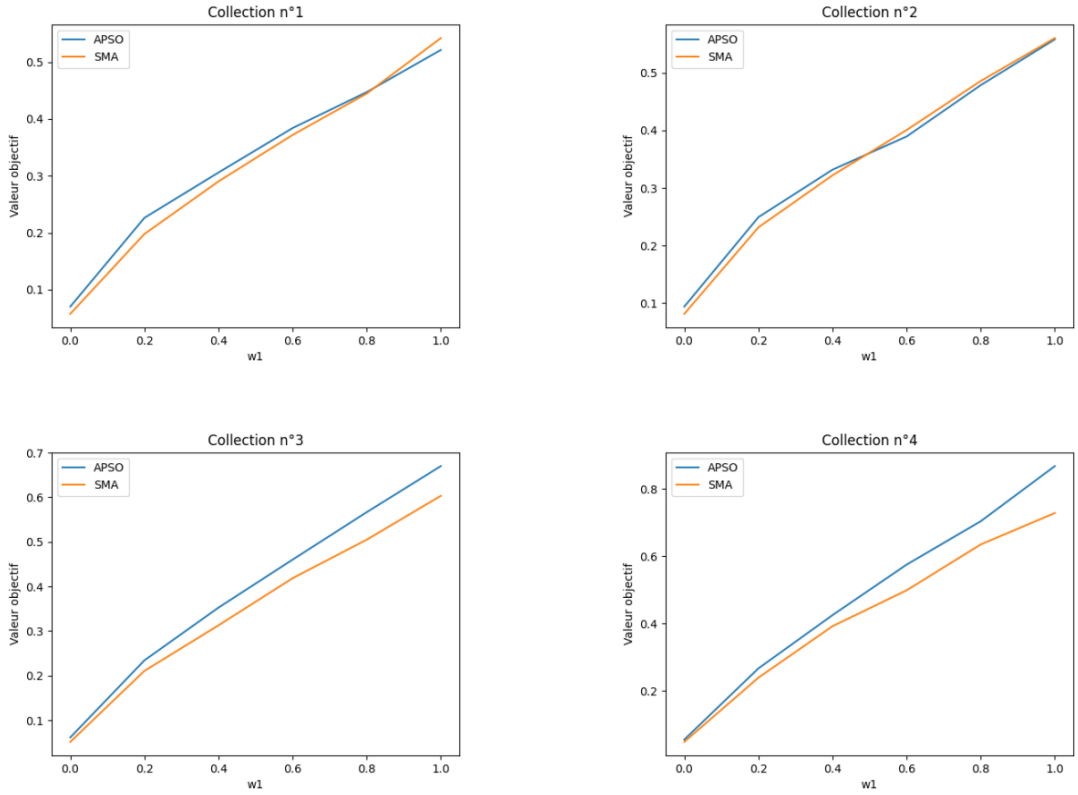



Figure 4.5 – Les coûts $f$ de SMA et d'APSO en fonction du poids $w_1$.

A travers les résultats de la Figure 4.5, nous pouvons observer que l'approche proposée atteint les meilleurs coûts et donne une grande amélioration par rapport à l'algorithme APSO en termes de coût $f$.

Dans cette dernière phase de tests, nous visualisons les meilleurs coûts retournés par SMA et APSO. La Figure 4.6 présente les coûts retournés par la fonction $f$ à chaque exécution.

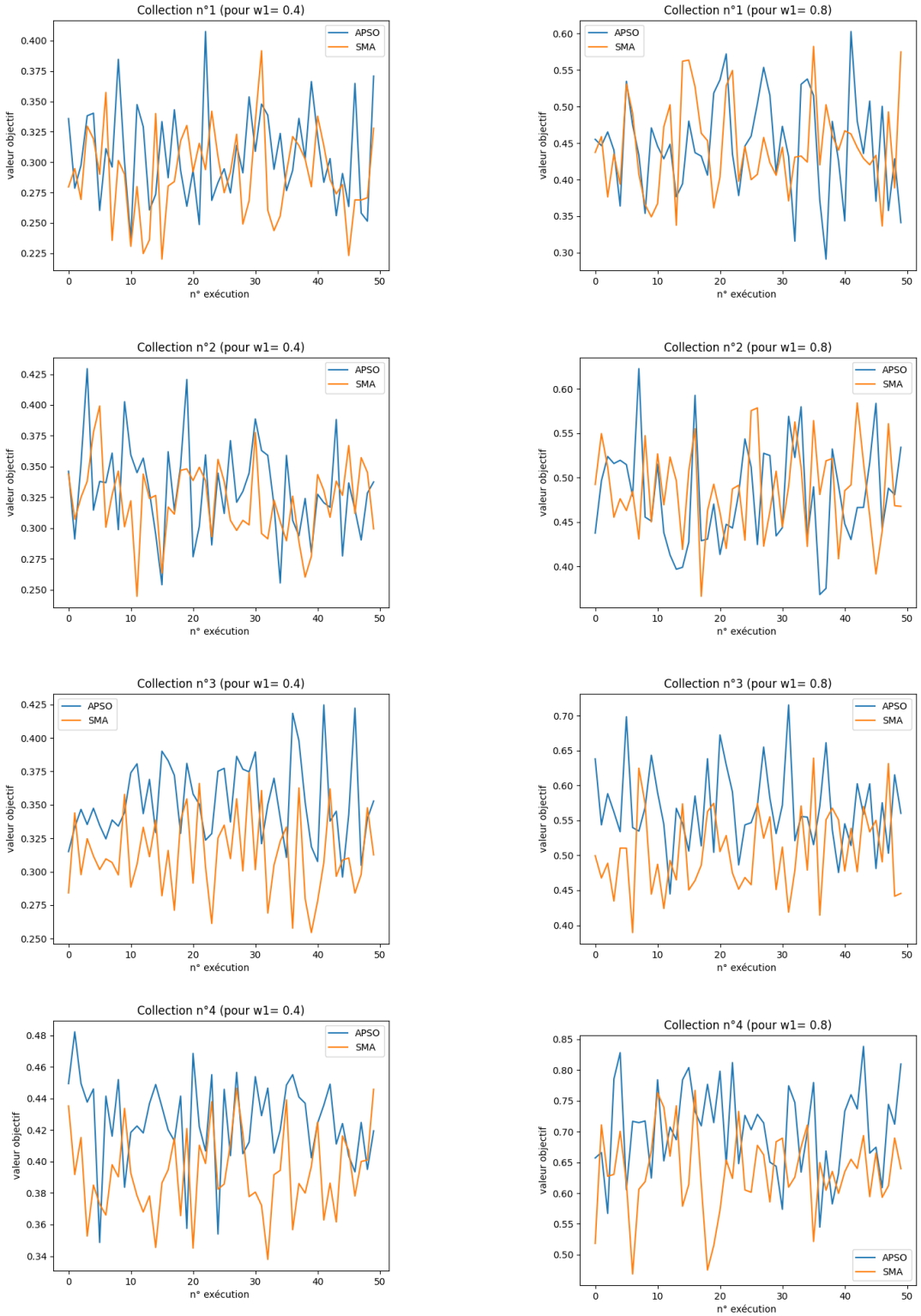



Figure 4.6 – Les meilleurs coûts retournés par SMA et APSO à chaque exécution.

Nous pouvons voir à partir de la Figure 4.6 que l'approche proposée atteint à chaque exécution, et dans la majorité des cas, les meilleurs résultats et donne une grande amélioration par rapport à l'algorithme APSO.

## 4.4. Conclusion

Dans ce chapitre, nous avons comparé les résultats obtenus par notre approche avec ceux obtenus par l'algorithme APSO. Nous constatons bien que l'approche proposée a permis d'améliorer considérablement le temps de calcul. De plus, le SMA proposé pour le problème de dispatching des ambulances d'urgence a obtenu les meilleures performances en termes de coût $f$.

# Conclusion générale

Dans ce travail, nous avons montré que le dispatching des ambulances d'urgence (DAU) est une tâche complexe et ne pourrait pas être résolue efficacement par des méthodes exactes. De plus, nous avons montré que l'espace de recherche de DAU est exponentiel et inclut toutes les affectations patients-ambulances possibles. Par conséquent, pour faire face à ce problème, il est nécessaire de considérer le problème de DAU comme un problème d'optimisation et de le résoudre avec une technique d'intelligence en essaim. Pour cela, nous avons employé l'Algorithme Slime Mould (SMA) pour résoudre ce problème.

Le SMA est l'un des algorithmes d'intelligence en essaim le plus récent et le plus efficace en raison de sa simplicité, de sa grande flexibilité et de sa vitesse de convergence rapide. Nous avons donc introduit dans ce travail une modélisation multi-objectifs de DAU avec SMA en présentant notamment l'encodage de la solution, l'espace des solutions potentielles, la fonction objectif et l'ajustement des solutions.

Pour obtenir de meilleurs résultats, nous avons effectué des tests préliminaires pour fixer les paramètres de SMA. Puis, nous avons évalué et comparé les résultats de notre algorithme SMA avec ceux de l'algorithme APSO. Les résultats expérimentaux montrent que l'approche proposée est prometteuse et réussit à améliorer les performances en termes de coût $f$ et en termes de temps de calcul.

A l'avenir, nous essaierons de comparer le SMA proposé avec de nouveaux types d'algorithmes d'intelligence en essaim tels que l'Algorithme de chauves-souris (BA) et l'Algorithme de Lucioles (FA). En outre, sur la base des résultats prometteurs présentés dans ce travail, plus de tests et d'expérimentations seront nécessaires pour vérifier si ces résultats ont une validité plus générale. Une autre direction de recherche possible consiste à tester l'approche proposée sur des ensembles de données réels et volumineux qui nécessiteraient l'utilisation de machines offrant de grands espaces de stockage ainsi qu'une vitesse de calcul rapide comme les GPUs.

# Références bibliographiques


[1] Lavoie, G. (2019). Analyse des politiques d'affectation d'un service pré-hospitalier d'urgence par simulation. *Mémoire de Maîtrise*, Polytechnique Montréal, Montréal, Canada.

[2] Bandara, D., Mayorga, M.E., & McLay, L.A. (2014). Priority dispatching strategies for EMS systems. *Journal of the Operational Research Society*, vol. 65, n°4, p. 572–587.

[3] Gendreau, M., Laporte, G., & Semet, F. (2001). A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. Parallel computing, vol. 27, n°12, p. 1641–1653.

[4] Carvalho, A.S., Captivo, M.E., & Marques, I. (2020). Integrating the ambulance dispatching and relocation problems to maximize system's preparedness. *European Journal of Operational Research*, vol. 282, n°3, p. 1064-1080.

[5] Mahmat, Z., Sua, L. S., & Balo, F. (2021). Mathematical Modeling in Disaster Logistics: Multi-Depot Vehicle Routing Problem. *Computational Research Progress in Applied Science and Engineering*, vol. 7, n°2, p. 1-9.

[6] Tassone, J., & Choudhury, S. (2020). A comprehensive survey on the ambulance routing and location problems.

[7] Barbati, M. (2013). Models and algorithms for facility location problems with equity considerations. *Thèse de Doctorat*, Université de Naples Federico II, Naples, Italie.

[8] Dirion, J.L. (2015). Optimisation. *Support de cours*, Ecole Nationale Supérieure des Mines Albi, Toulouse, France.

[9] Bouri, S. (2007). Optimisation de la production et de la structure d'énergie électrique par les colonies de fourmis. *Thèse de Doctorat*, Université Jilali Liabès, Sidi Bel Abbes, Algérie.

[10] Ahmia, I. (2019). Une nouvelle Métaheuristique pour les problèmes d'optimisation combinatoire : La Monarchie Métaheuristique. *Thèse de Doctorat*, USTHB, Alger, Algérie.

[11] Li, S., Chen, H., Wang, M., Heidari, A.A., Mirjalili, S. (2020). Slime Mould Algorithm: a new method for stochastic optimization. *Future Generation Computer Systems*, p. 300-323.

# Résumé

L'Algorithme Slime Mould (SMA) est l'un des algorithmes d'intelligence en essaim les plus récents et les plus efficaces pour la résolution des problèmes d'optimisation complexes. En revanche le Service d'Aide Médicale Urgente (SAMU) connaît des perturbations assez fréquentes. Le problème de Dispatching des Ambulances d'Urgences (DAU) est parmi ces perturbations qui entraînent une dégradation progressive de la qualité du service. Par conséquent, nous proposons dans ce projet de considérer le problème de DAU comme un problème d'optimisation et de le résoudre avec l'algorithme Slime Mould. Nous effectuons des expérimentations préliminaires pour régler les paramètres de SMA. Ensuite, nous comparons les résultats de l'approche proposée avec ceux obtenus par APSO. L'analyse expérimentale montre que le SMA proposé pour DAU est très compétitif et donne une amélioration substantielle par rapport à l'algorithme APSO en termes de performances et de complexité de calcul.



# Résumé

The Slime Mold Algorithm (SMA) is one of the newest and most efficient swarm intelligence algorithms for solving complex optimization problems. On the other hand, the Emergency Medical Serivces (EMS) experiences frequently disruptions. The problem of Emergency Ambulance Dispatching (DAU) is one of those disruptions that lead to a gradual deterioration in the quality of service. Therefore, we propose in this project to consider the DAU problem as an optimization problem and solve it with the Slime Mold algorithm. We perform preliminary experiments to tune the SMA parameters. Then, we compare the results of the proposed approach with those obtained by APSO. The experimental analysis shows that the proposed SMA for DAU is very competitive and gives a substantial improvement over the APSO algorithm in terms of performance and computational complexity.



# ملخص

تعد خوارزمية Slime Mould Algorithm (SMA) واحدة من أحدث خوارزميات ذكاء الأسراب وأكثرها كفاءة لحل مشكلات التحسين المعقدة. من ناحية أخرى، تعاني خدمات الطوارئ الطبية (EMS) من اضطرابات متكررة . إن مشكلة إرسال سيارات الإسعاف في حالات الطوارئ (DAU) هي إحدى تلك الاضطرابات التي تؤدي إلى تدهور تدريجي في جودة الخدمة .لذلك، نقترح في هذا المشروع اعتبار مشكلة DAU كمشكلة تحسين معقدة وحلها باستخدام خوارزميةSMA. نجري تجارب أولية لضبط إعدادات SMA ثم نقارن نتائج النهج المقترح مع تلك التي حصل عليها APSO. يوضح التحليل التجريبي أن SMA المقترح لـ DAU تنافسي للغاية ويعطي تحسنًا كبيرًا على خوارزمية APSO من حيث الأداء والتعقيد الحسابي.