Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet Recherche & Développement

2017-2018

Métaheuristiques bio-inspirées pour l'optimisation de l'affectation interne de la population de Nice aux centres d'accueil en cas de séisme

Entreprise
Polytech Tours

Tuteur entreprise
Nicolas MONMARCHE (Professeur)

Étudiant
Peng BI (DI5)

Tuteur académique
Nicolas MONMARCHE

2 avril 2018

Liste des intervenants

Entreprise

Polytech Tours 64 avenue Jean Portalis 37200 Tours, France polytech.univ-tours.fr

Nom	Email	Qualité
Peng BI	peng.bi@etu.univ-tours.fr	Étudiant DI5
Nicolas MONMARCHE	nicolas.monmarche@univ-tours.fr	Tuteur académique, Département Informatique
Nicolas MONMARCHE	nicolas.monmarche@univ-tours.fr	Tuteur entreprise, Professeur

Avertissement

Ce document a été rédigé par Peng BI susnommé l'auteur.

L'entreprise Polytech Tours est représentée par Nicolas MONMARCHE susnommé le tuteur entreprise.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Nicolas MONMARCHE susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnait assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnait qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document

Peng BI, *Métaheuristiques bio-inspirées pour l'optimisation de l'affectation interne de la population de Nice aux centres d'accueil en cas de séisme*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
    author={BI, Peng},
    title={Métaheuristiques bio-inspirées pour l'optimisation de l'affectation interne de la
       population de Nice aux centres d'accueil en cas de séisme},
    type={Projet Recherche \& Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2017-2018}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V2.2.5 (2017/05/02)

Table des matières

Table des figures

6 Conclusion

A Planification

B Description des interfaces externes du logiciel

C Spécifications fonctionnelles

E Cahier du développeur

H Cahier de test

Liste des tableaux

Liste des Algorithmes

1 Introduction

Dans ce chapitre, on décrit le contexte du projet et le structure du rapport ainsi que les acteurs, les objectifs, l'hypothèses, et les bases méthodologiques sur la gestion de projet.

1 Contexte de la réalisation et acteurs

Quaud le séisme a lieu, le Préfecture doit rapidement élaborer un plan de secours pour offrir les soutiens médicaux, et le Mairie doit bien organiser l'évacuation des sans-abris. Etant donné cette raison, ce projet est particulièrement conçu pour Nice en cas de séisme. Dans ce projet, il faut proteger le plus de sans-abris possible mais ne pas forcement proteger tout le monde.

Ce rapport est pour présenter en détail notre projet de recherche et développement. On collabore à faire ce projet avec le professeur et les étudiants en département d'aménagement et environnement. Les rôles de ce projet sont :

- Client : M. Kamal SERRHINI (le professeur en département d'aménagement et environnement) et certaine institution de Nice ;
- MOA : M. Nicolas MONMARCHE (le professeur en département d'informatique) ;
- MOE : Peng BI

Le projet de recherche et développement, comme le projet de fin d'étude, est le dernier cycle dans la vie d'étudiant, et c'est très important et obligatoire pour que les étudiants obtiennent leurs diplômes d'ingénieur. Il peut intégrer toutes les connaissances et les techniques que les étudiants apprennent de leurs formations, et mettre tous leurs acquis à la pratique.

Dans ce rapport, le chapitre 1 décrit le contexte de la réalisation, y compris l'objectif du logiciel, l'hypothèse et la méthodologie utilisée. Le chapitre 2 présente les descriptions générales du logiciel sur l'environnement, les utilisateurs, le structure, l'interface, et on expose en détaille la fonctionnalité et la non-fonctionnalité. Le chapitre 3 présente les techniques existantes relatives à ce projet. Le chapitre 4 explique les analyses de problème et montre la modélisation mathématique et la modélisation logicielle. Le chapitre 5 explique en détail l'algorithme proposé et des amélioration. On fait les bilans pour ce projet dans le chapitre 6. Dans l'annexe, il y a la planification, les fonctionnalités et les non-fonctionnalités détaillées, le cahier du développeur, le document d'installation des librairies et le document d'utilisation ainsi que le cahier de test.

2 Objectifs

Ce projet vise à proposer un algorithme métaheuristique pour développer un logiciel qui aide le gouvernement à optimiser l'affectation des sans-abris et à faire une meilleure décision en cas de séisme. Il y a deux objectifs espérés :

- Soit on veut minimiser la distance totale parcourue par les sans-abris pour rejoindre leur centre d'accueil en maximisant le nombre de personnes abritées ;
- Soit on veut minimiser la distance totale parcourue par les sans-abris pour rejoindre leur centre d'accueil en maximisant le nombre de bâtiments évacués.

C'est l'utilisateur qui décide lequel il prendra.

3 Hypothèse

Le langage de programmation que l'on choisit est Python. Cependant, si on veut utiliser « NVIDIA Jetson TX1 », on a aussi besoin de la platement CUDA. CUDA est une plateforme de calcul parallèle produit par NVIDIA, et les codes sur cette plateforme sont généralement écrits en langage C++. Si le langage Python n'est pas compatible sur cette plateforme, on utilisera le langage C++ pour la programmation.

4 Bases méthodologiques

1. Les outils
   - Pour la gestion de projet, on utilise le diagramme de Gantt pour contrôler le déroulement du projet. Et on utilise la plateforme « Trello » pour gérer les tâches ;
   - Pour la modélisation logicielle, on utiliser les diagrammes d'UML ;
   - Pour la modélisation mathématique et la rédaction de rapport, on utilise « LaTex ».
2. La méthode de gestion de projet
   On utilise le modèle en « cascade »(Figure1) à gérer ce projet. Les phases de modèle en « cascade » sont effectuées par l'ordre de haut en bas. Et lors que chaque phrase finit, elle retournera à la phase précédente pour faire une rétroaction. Donc ce modèle est plus parfaitement correspondant que notre projet. Pour le semestre 9, ce que on doit faire est la

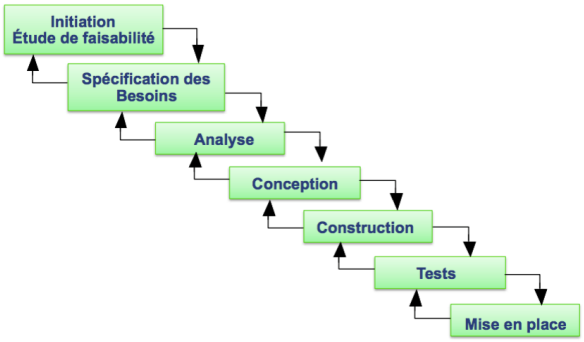


Figure 1 – *Le modèle en « cascade »*

recherche sur notre projet. Cela correspond aux phases de l'étude de faisabilité à l'analyse. Et pendant le semstre 10, on passera les phases de la conception à la mise en place.

La planification détaillée est décrite dans l'annexe « Planification ».

2 Description générale

Dans ce chapitre, on décrit l'environment du projet, les caractéristiques des utilisateur, les fonctionnalités du système, et la structure générale du système.

1 Environnement du projet

L'environnement de développement consiste aux points suivants :

- Le système d'exploitation : Windows ;
- Le langage de programmation : Python 3.6 ;
- L'IDE pour le développement : PyCharm ;
- La Matériel pour le développement : Inter® Core™ i7-6500U CPU @2.5GHZ, RAM 8GB.

2 Caractéristiques des utilisateurs

Il n'y a qu'un type de l'utilisateur pour ce logiciel. Ses caractéristiques sont décrites dans le tableau comme Tableau1.

Table 1 – *Le tableau de caractéristiques des utilisateurs*

	Connaissance de l'informatique	Expérience de l'application	Type de l'utilisateur	Droits d'accès
Réponse	Non	Non	Général et régulier	Tout droit
Commentaire	Il suffit importer les données dans le logiciel, et voir le résultat exécuté	Les opérations du logiciel est très simple	Le cible de l'utilisateur est uniquement le secteur sur l'séisme	Les utilisateurs peuvent faire toutes les opérations.

3 Fonctionnalités du système

Le diagramme de cas d'utilisation est comme Figure1.
Dans ce diagramme, il y a deux rôles : l'utilisateur et le système. L'utilisateur peut importer

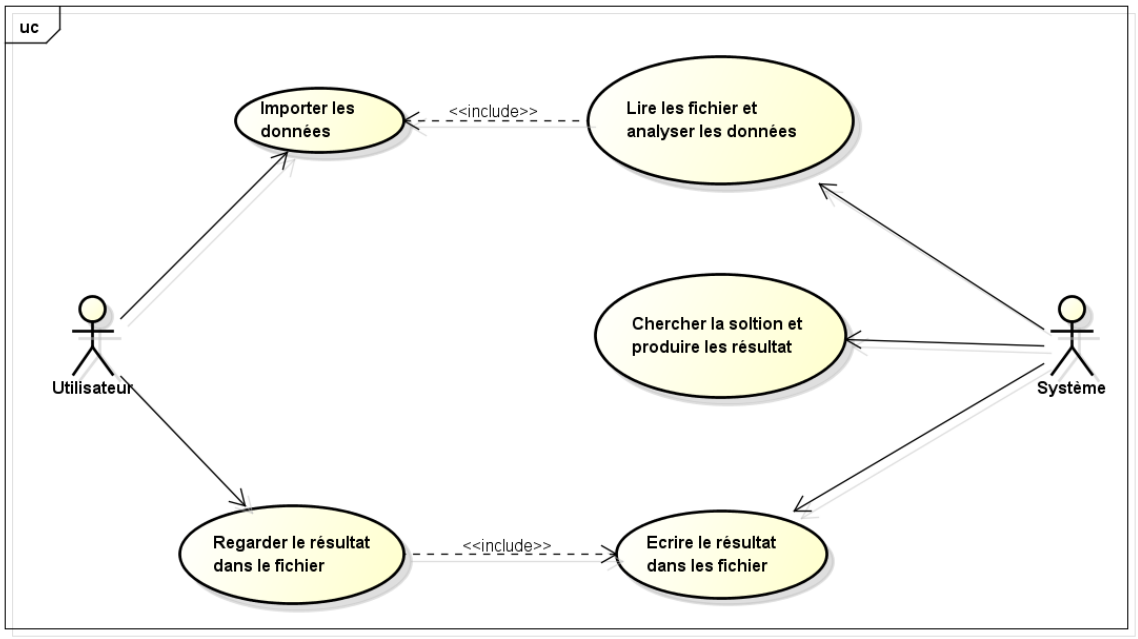


Figure 1 – *Le diagramme de cas d'utilisation*

les fichiers de données, et le système peut lire les fichiers et extraire les données alors il analyse les données pour chercher la solution. Enfin, le système fournit la meilleure solution trouvée à l'utilisateur.

L'interaction entre l'utilisateur et le système est présentée dans un diagramme de séquence comme Figure2.

Il y a deux acteurs dans ce diagramme, l'utilisateur est l'acteur principal, et le système

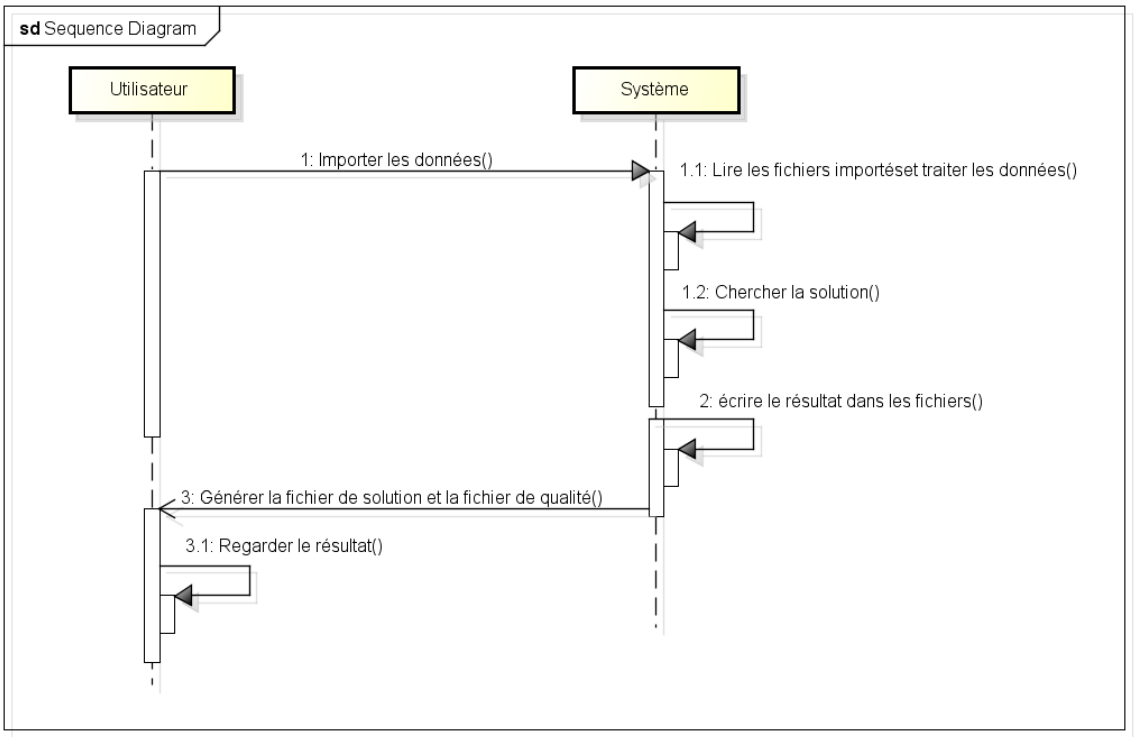


Figure 2 – *Le diagramme de séquence*

est l'acteur secondaire. Lors que l'utilisateur lance le logiciel, l'utilisateur envoie un nouveau

message synchrone qui contient les trois fichiers de données dont le système a besoins. Ensuite, le système lit les fichiers et extrait les données, puis analyse les données. Après que le système trouve la meilleure solution, il génère les fichiers qui stockent les résultats. L'utilisateur peuvent regarder la solution dans le fichier de solution.

Ce déclenchement d'événements peut être présente dans un diagramme d'activité comme Figure3.

Les descriptions détaillées des fonctionnalités sont dans l'annexe « Spécifications fonctionnelles ».

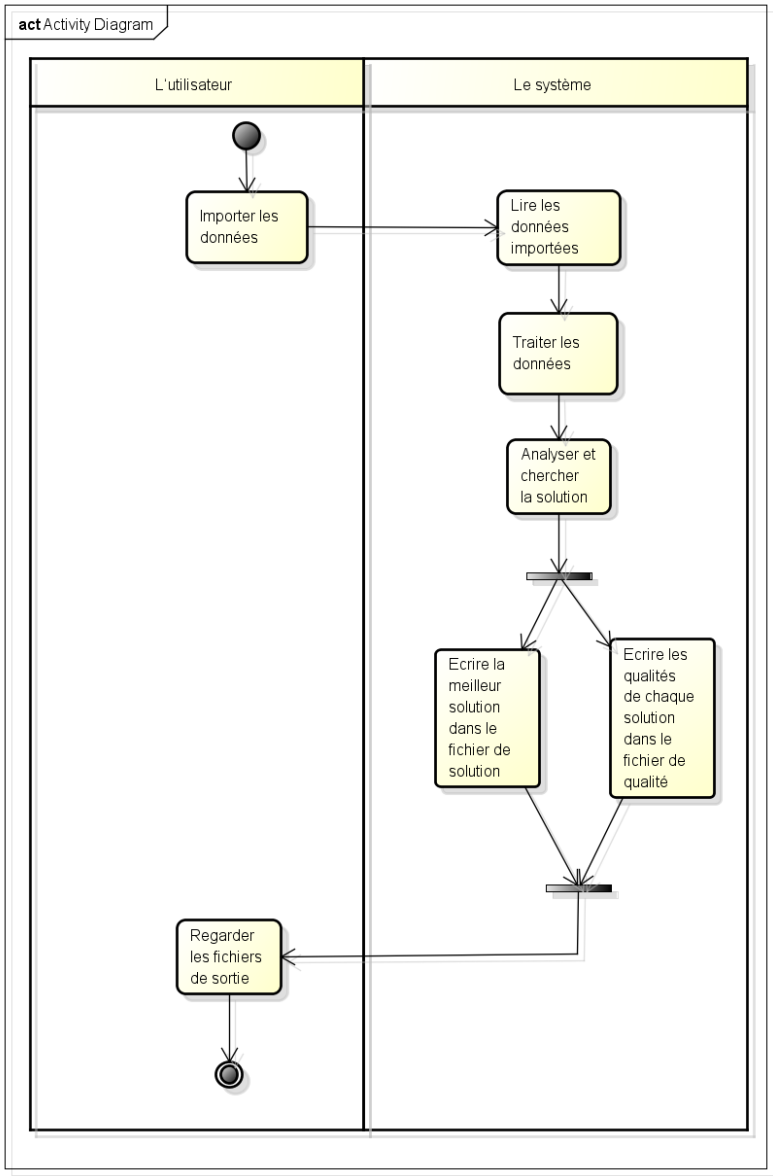


Figure 3 – *Le diagramme d'activité*

4 Structure générale du système

Pour décrire la structure interne, on conçoit un diagramme de composant comme Figure4.

Dans ce diagramme, il y a 3 composants principaux, deux interfaces de service et deux interfaces de demande. Parce qu'il n'y a pas la base de données dans ce projet, l'utilisateur doit toujours importer les fichiers de données.

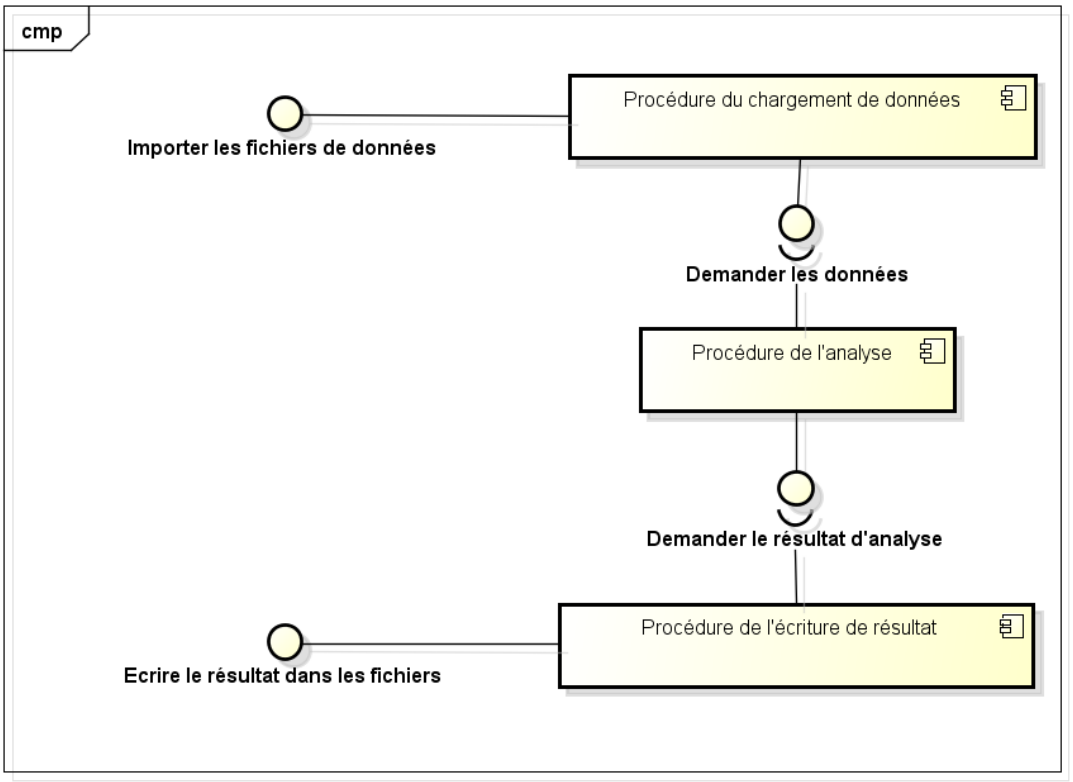


Figure 4 – *Le diagramme de composant*

Quand le système exécute le composant « Procédure de l'analyse », il demande les données importées au composant « Procédure du chargement de données » par l'interface de demande « Demander les données », donc le composant « Procédure du chargement de données » fournit aussi une interface de service « Importer les fichiers de données » à l'utilisateur pour lui faire importer les données à partir des fichiers.

Après l'analyse, le composant « « Procédure de l'écriture de résultat » demande le résultat analysé par l'interface de demande « Demander le résultat d'analyse » et écrit le résultat dans le fichier par l'interface de service « Ecrire le résultat dans les fichiers ».

3 Etat de l'art

Dans ce chapitre, on décrit des problèmes classiques qui sont semblables au problème de ce projet, et on explique des algorithmes existant pour résoudre ces problèmes.

1 Problèmes classiques

Dans cette section, il s'agit des problèmes très classiques et très connus qui donnent l'inspiration à ce projet.

1.1 Variantes du problème du sac à dos

Le problème du sac à dos est un problème d'optimisation combinatoire. Il appartient au problème NP-Complets. On l'étudie intensivement depuis vingtième siècle [WWW3] et il est appliqué à beaucoup de domaines de nos jours, par exemple, la commerce, la combinatoire, la théorie de la complexité, La cryptographie, etc. Il y a de plus en plus de variantes sur le problème du sac à dos sont proposés avec la recherche plus profonde sur ce problème.

1.1.1 Problème du sac à dos 0-1 (KP)

Le problème du sac à dos 0-1 est la variante la plus simple et la plus classique. Toutes les autres variantes sont basées sur ceci.

1. La définition
   - Il n'y a qu'un sac à dos avec sa capacité C ;
   - Il y a n objets, et le nombre de chaque objet est limité à 1. Chaque objet i a sa propre valeur p_i et son propre poids w_i ;
   - Le but est de choisir des objets à remplir le sac afin de maximiser la valeur totale des objets dans le sac sans dépasser sa capacité.
2. La formulation
   - Variable :

$$x_i = \begin{cases} 1 & \text{si l'objet i est sélectionné} \\ 0 & \text{sinon} \end{cases} \quad \forall i \in \{1, ..., n\}$$

- Contrainte :

$$\sum_{i\in\{1,\dots,n\}} w_i \times x_i \leq \mathrm{C}$$

- Fonction objective :

$$f(x) = \max \sum_{i\in\{1,\dots,n\}} p_i \times x_i$$

.

1.1.2 Problème du sac à dos multiple (MKP)

Dans le problème du sac à dos 0-1, il n'y a qu'un sac à dos. Mais dans le problème du sac à dos multiple, il y a plusieurs sacs à dos.[5]

1. La définition
   - Il y a ***m*** sacs à dos, et chaque sac à dos ***j*** a sa propre capacité c_j ;
   - Il y a ***n*** objets, et le nombre de chaque objet est limité à 1. Chaque objet ***i*** a sa propre valeur p_i et son propre poids w_i ;
   - Le but est de choisir des objets à remplir les sacs afin de maximiser la valeur totale des objets dans les sacs sans dépasser la capacité de chaque sac.
2. La formulation
   - Variable :

$$x_{i,j} = \begin{cases} 1 & \text{si l'objet i est affecté au sac j} \\ 0 & \text{sinon} \end{cases} \quad \forall i \in \{1,\dots,n\}, \forall j \in \{1,\dots,m\}$$

   - Contraintes :

$$\sum_{j\in\{1,\dots,m\}} x_{i,j} \leq 1 \quad \forall i \in \{1,\dots,n\}$$

$$\sum_{i\in\{1,\dots,n\}} w_i \times x_{i,j} \leq c_j \quad \forall j \in \{1,\dots,m\}$$

   - Fonction objective :

$$f(x) = \max \sum_{i\in\{1,\dots,n\}} \sum_{j\in\{1,\dots,m\}} p_i \times x_{i,j}$$

1.1.3 Problème du sac à dos multidimensionnel (MDKP)

Dans le problème du sac à dos 0-1, le sac à dos n'a qu'une dimension – la capacité. Pourtant, dans le problème du sac à dos multidimensionnel, on ajoute des autres dimensions, telles que la taille, le nombre, etc. Ces dimensions représentent les ressources du sac à dos. [1]

1. La définition
   - Le sac à dos a ***m*** ressource (capacité, taille, nombre, . . .), et la quantité totale disponible de ressource ***j*** est b_j ;
   - Il y a ***n*** objets (s'il y a une dimension qui indique le nombre de sac à dos, il n'y aura plus la limite à 1 pour le nombre de chaque objet, et c'est possible de séparément mettre les articles de même objet dans des sacs différents), et chaque objet ***i*** a sa propre valeur p_i et la consomption $r_{i,j}$ de ressource j ;
   - Le but est de choisir des objets à remplir le(s) sac(s) afin de maximiser la valeur totale des objets dans le(s) sac(s) sans dépasser sa(ses) capacité(s).
2. La formulation

- Variable :

$$x_i = \begin{cases} 1 & \text{si l'objet i est sélectionné} \\ 0 & \text{sinon} \end{cases} \quad \forall i \in \{1,...,n\}$$

- Contrainte :

$$\sum_{i \in \{1,...,n\}} r_{i,j} \times x_i \leq b_j \quad \forall j \in \{1,...,m\}$$

- Fonction objective :

$$f(x) = \max \sum_{i \in \{1,...,n\}} p_i \times x_i$$

.

1.1.4 Problème du sac à dos multi-objectif (MOKP)

Dans le problème du sac à dos 0-1, il y a qu'un sac à dos et qu'une fonction objective, et chaque objet n'a qu'un article. Mais dans le problème du sac à dos multi-objectif, il y a plusieurs sacs à dos et plusieurs fonctions objectives, et chaque objet possède plusieurs articles.[7]

1. La définition
   - Il y a $\boldsymbol{m}$ sacs à dos, et chaque sac à dos $\boldsymbol{j}$ a sa propre capacité c_j ;
   - Il y a $\boldsymbol{n}$ objets, les articles peuvent être mis aux différents sacs. $p_{i,j}$ représente la valeur de l'objet $\boldsymbol{i}$ dans le sac à dos $\boldsymbol{j}$, et $w_{i,j}$ représente le poids de l'objet $\boldsymbol{i}$ dans le sac à dos $\boldsymbol{j}$;
   - Le but est de choisir des objets à remplir les sacs afin de maximiser un vecteur de fonction profit sans dépasser sa capacité.
2. La formulation
   - Variable :

$$x_i = \begin{cases} 1 & \text{si l'objet i est sélectionné} \\ 0 & \text{sinon} \end{cases} \quad \forall i \in \{1,...,n\}$$

   - Contrainte :

$$\sum_{i \in \{1,...,n\}} w_{i,j} \times x_i \leq c_j \quad \forall j \in \{1,...,m\}$$

   - Fonctions objectives :

$$f(x) = \max[f_1(x), f_2(x), ..., f_j(x)]$$

$$f_j(x) = \sum_{i \in \{1,...,n\}} p_{i,j} \times x_i \quad \forall j \in \{1,...,m\}$$

1.1.5 Problème du sac à dos multichoix (MCKP)

Dans le problème du sac à dos multichoix, on divise l'ensemble des objets en plusieurs sous-ensembles. Tous les sous-ensembles contiennent des objets. [6]

1. La définition
   - Il n'y a qu'un sac à dos avec sa capacité $\boldsymbol{C}$;
   - L'ensemble des objets est divisé en $\boldsymbol{m}$ sous-ensembles des objets, et Chaque sous-ensemble $\boldsymbol{j}$ possède n_j objets ;
   - Chaque objet $\boldsymbol{i}$ n'a qu'un article, et il a sa propre valeur $p_{i,j}$ et son propre poids $w_{i,j}$;
   - Le but est de choisir seulement un objet à partir de chaque sous-ensemble à remplir le sac afin de maximiser la valeur totale des objets dans le sac sans dépasser sa capacité.
2. La formulation

- Variable :

$$x_{i,j} = \begin{cases} 1 & \text{si l'objet i est sélectionné de sous-ensemble j} \\ 0 & \text{sinon} \end{cases} \quad \forall i \in \{1,...,n_j\}, \forall j \in \{1,...,m\}$$

- Contraintes :

$$\sum_{i \in \{1,...,n_j\}} \sum_{j \in \{1,...,m\}} w_{i,j} \times x_{i,j} \leq C$$

$$\sum_{i \in \{1,...,n_j\}} x_{i,j} = 1 \quad \forall j \in \{1,...,m\}$$

- Fonction objective :

$$f(x) = \max \sum_{i \in \{1,...,n_j\}} \sum_{j \in \{1,...,m\}} p_{i,j} \times x_{i,j}$$

1.2 Problème du bin packing

Le problème du bin packing est un problème d'optimisation combinatoire très classique, qui appartient au problème NP-Difficile. Ce problème est appliqué à beaucoup de secteurs industriels et informatique, tels que la coupe du tissu dans l'industrie du vêtement, le chargement de conteneurs dans l'industrie du transport, le découpage du matériel dans l'industrie de la transformation, le rangement de fichiers sur un support informatique, etc.[WWW2]

1. La définition
   - Il y a un nombre infini boîtes, et toutes les boîtes ont les capacités identiques $\boldsymbol{C}$;
   - Il y a $\boldsymbol{n}$ objets, et chaque objet $\boldsymbol{i}$ a son propre poids w_i ;
   - Le but est de minimiser le nombre de boîtes utilisées pour ranger tous les objets aux boîtes.
2. La formulation
   - Variables :

$$x_{i,j} = \begin{cases} 1 & \text{si l'objet i est rangé dans la boîte j} \\ 0 & \text{sinon} \end{cases} \quad \forall i, \forall j \in \{1,...,n\}$$

$$y_j = \begin{cases} 1 & \text{si la boîte j est utilisée} \\ 0 & \text{sinon} \end{cases} \quad \forall j \in \{1,...,n\}$$

   - Contraintes :

$$\sum_{i \in \{1,...,n\}} w_i \times x_{i,j} \leq C \times y_j \quad \forall j \in \{1,...,n\}$$

$$\sum_{j \in \{1,...,n\}} x_{i,j} = 1$$

   - Fonction objective :

$$f(x) = \min \sum_{j \in \{1,...,n\}} y_j$$

1.3 Problème du p-médian

Le problème du p-médian est premièrement proposé par Hakimi en 1964. Garey et Johnson ont privé que ce problème est un problème NP-Difficile en 1979. Il est appliqué principalement à choisir les sites des centres de service. [4]

1. La définition
   - Il y a ***m*** sites candidats de centre de service ;
   - Il y a ***n*** clients, et chaque client ***i*** a une quantité de demande h_i ;
   - On connaît la distance $d_{i,j}$ entre chaque client ***i*** et chaque centre de service ***j*** ;
   - Le but est de choisir ***P*** centre de service entre m sites candidats pour minimiser la somme de distances avec les poids (les demandes) entre les clients et leurs propres centres de service les plus proche.
2. La formulation
   - Variables :

$$x_{i,j} = \begin{cases} 1 & \text{si le client i est servi par le centre de service j} \\ 0 & \text{sinon} \end{cases} \quad \forall i \in \{1,...,n\}, \forall j \in \{1,...,m\}$$

$$y_j = \begin{cases} 1 & \text{si le site canditat j est sélectionné} \\ 0 & \text{sinon} \end{cases} \quad \forall j \in \{1,...,m\}$$

   - Contraintes :

$$\sum_{j \in \{1,...,m\}} x_{i,j} = 1 \quad \forall i \in \{1,...,n\}$$

$$x_{i,j} - y_j \leq 0 \quad \forall i \in \{1,...,n\} \, et \, j \in \{1,...,m\}$$

$$\sum_{j \in \{1,...,m\}} y_j = \mathrm{P}$$

   - Fonction objective :

$$f(x) = \min \sum_{i \in \{1,...,n\}} \sum_{j \in \{1,...,m\}} h_i \times d_{i,j} \times x_{i,j}$$

2 Algorithmes

Dans cette partie, il s'agit des algorithmes heuristiques et métaheuristiques qui supportent de résoudre ces problèmes. Parce que ce projet est plus similaire au problème de sac à dos multiple (cela est expliqué dans le chapitre 4 « Comparaison et identification du problème », ici on ne décrit que les algorithmes existants qui sont pour résoudre le problème de sac à dos multiple. On a étudié deux algorithmes, l'un est heuristique, et l'autre est métaheuristique.

L'algorithme heuristique est un algorithme qui peut obtient des solutions faisables pour un problème spécifie en temps raisonnable et dans l'espace raisonnable. Mais on n'est pas sûr que ces solutions soient les meilleures solutions.

L'algorithme métaheuristique est l'amélioration de l'algorithme heuristique. Généralement, il ne dépend pas de conditions spécifiées de certain problème, donc il peut être plus largement appliqué. Et il est souvent inspiré par des phénomènes dans la nature.

2.1 Algorithme MTHM – heuristique

Cet algorithme est proposé par Martello et Toth. Elle est spécialement pour résoudre le problème du sac à dos multiple.[2]

Dans cet algorithme, tout d'abord, les objets sont triés par l'ordre décroissante selon leurs quotients de profil divisé par poids, et les sacs sont tries par l'ordre croissante selon leurs capacités.

$$Trier \ les \ objets : \frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_i}{w_j}$$

$$Trier\ les\ sacs : c_1 \leq c_2 \leq ... \leq c_j$$

Par exemple, on a 2 sacs et 9 objets :

- les capacité de sacs sont : {75, 125} ;
  On trie les sacs par l'ordre croissante : {sac1(75), sac2(125)} ;
- les paires « profil/poids » d'objets sont : {35/50, 30/40, 45/70, 20/65, 50/25, 13/20, 40/35, 30/60, 35/30}.
  On trie les objets par l'ordre décroissante : {objet5(50/25), objet7(40/35), objet9(35/30), objet2(30/40), objet1(35/47), objet3(50/70), objet6(13/20), objet8(30/60), objet4(20/65)}.
- la solution de départ est S=[0,0,0,0,0,0,0,0,0].

Après que les objets et les sacs sont tous bien triés, l'algorithme se démarre. Son exécution suit les étapes suivantes :

1. Générer la solution initiale :
   On remplit le premier sac dont capacité est c1 dans la liste triée en utilisant l'algorithme Greedy, puis on remplit le deuxième de même manière jusqu'à ce que l'on ait rempli le mème sac. Ainsi, la solution initiale est générée.
   - On remplit sac1 = {objet5(50/25), objet7(40/35)} :
     - Le profil total du sac1 : 50+40=90 ;
     - Le poids total du sac1 : 25+35=60 (la capacité restante : 15).
   - On remplit sac2 = {objet9(35/30), objet2(30/40), objet1(35/47)} :
     - Le profil total du sac2 : 35+30+35=100 ;
     - Le poids total du sac2 : 30+40+47=117(la capacité restante : 8).
   - On obtient la solution initiale : S = [2,2,0,0,1,0,1,0,2] :
     - Le profil total : 90+100=190.
2. Réarranger les objets :
   Dans cette étape, on fait le réarrangement des objets sur la solution initiale. On échange des objets entre des sacs différents, et on regarde si c'est possible d'insérer des objets restant dans certains sacs après l'échange local.
   - On fait l'échange entre l'objet5 dans le sac1 et objet l'objet2 dans le sanc2 :
     - sac1 = {objet2(30/40), objet7(40/35)} :
       - ⋆ Le profil total du sac1 : 30+40=70 ;
       - ⋆ Le poids total du sac1 : 40+35=75 (la capacité restante : 0).
     - Sac2 = {objet9(35/30), objet5(50/25), objet1(35/47)} :
       - ⋆ Le profil total du sac2 : 35+50+35=120 ;
       - ⋆ Le poids total du sac2 : 30+25+47=102(la capacité restante : 23).
   - On ajoute l'objet6 dont le poids est de 20 dans le sac2 :
     - Sac2 = {objet9(35/30), objet5(50/25), objet1(35/47), objet6(13/20)} :
       - ⋆ Le profil total du sac2 : 35+50+35+13=133 ;
       - ⋆ Le poids total du sac2 : 30+25+47+20=122(la capacité restante : 3).
   - On obtient la solution initiale : S = [2,1,0,0,2,2,1,0,2] :
     - Le profil total : 70+133=203.
3. Remplacer les objets :
   On peut considérer de replacer les objets sélectionnés par un ou plusieurs objets restant pour élever le profil total.
   - On ne fait rien sur le sac1 :
     - sac1 = {objet2(30/40), objet7(40/35)} ;
       - ⋆ Le profil total du sac1 : 30+40=70 ;
       - ⋆ Le poids total du sac1 : 40+35=75 (la capacité restante : 0).
   - On remplace l'objet1 et l'objet6 par l'objet3 dans le sac2 :
     - sac2 = {objet9(35/30), objet5(50/25), objet3(50/70)} :
       - ⋆ Le profil total du sac2 : 35+50+50=135 ;

★ Le poids total du sac1 : 30+25=70 (la capacité restante : 0).

- On obtient la solution initiale : S = [0,1,2,0,2,0,1,0,2] :
  - Le profil total : 70+135=205.

Ainsi, on obtient la meilleure solution ***S***.

2.2 Algorithme de colonies de fourmis – métaheuristique

L'algorithme de colonies de fourmis est un algorithme métaheuristique. Il y a beaucoup de variante de colonies de fourmis, telles que ACO(Ant Colony Optimization), ACS(Ant Colony System), MMAS(Max-Min Ant System), BWAS(Best-Worst Ant System), etc. La variante la plus polulaire et la plus commun est ACO, et on l'explique en détail dans cette section.[3]

2.2.1 Inspiration

L'algorithme de colonies de fourmis simule le comportement de colonies de fourmis quand elles cherchent les nourritures. La clé qu'elles peuvent trouver un chemin le plus court entre leur nid et la ressource de nourriture consiste à la phéromone, avec laquelle les fourmis se communiquent pendant leurs mouvements.

Quand une fourmi prend un chemin, elle déposera la phéromone sur ce chemin, et la phéromone peut aussi s'évaporer avec la marche du temps. Les fourmis ont la capacité de perception sur la phéromone. Elles prendront le chemin dont la quantité de phéromones est la plus grande. Et si plus de fourmis prennent ce chemin, la quantité de phéromone sur ce chemin est plus grande. A mesure que le temps passe et les phéromones s'évaporent, toutes les fourmis se concentrent sur un chemin, et ce chemin est évidemment le plus court.

2.2.2 Présentation

Dans l'algorithme de colonies de fourmis, la phéromone est l'information numérique. Les fourmis utilisent ces phéromones à construire les solutions pour le problème du sac à dos multiple, et ces phéromones représentent leurs expériences de cherche pendant leurs mouvements.

Pendant l'exécution de l'algorithme, chaque fourmi construit une solution faisable. Donc cet algorithme est dans une boucle. La fois d'itération représente le numéro de fourmi. Pour déterminer le nombre de fourmi (i.e. le nombre d'itérations), on a trois façons à faire :

- On définit un nombre fixe d'itérations ;
- On définit une limite de temps d'exécution ;
- On définit une borne supérieure ou inférieure de la solution. Quand le résultat est suffisamment proche de la borne, l'algorithme s'arrête.

Au début, on initialise la quantité de phéromones, et chaque fourmi départ d'un état aléatoire. Ensuite, la fourmi se déplace de l'état i à l'état j selon la probabilité suivante la formulaire(1) :

$$p_{i,j} = \begin{cases} \frac{\tau_{i,j} \times \eta_{i,j}}{\sum_{l \in EtatsRestantFaisables} \tau_{i,l} \times \eta_{i,l}} & \text{si } j \in EtatsRestantFaisables \\ 0 & \text{sinon} \end{cases} \quad \forall i \in \{1,...,n\}, \forall j \in \{1,...,m\} \qquad (1)$$

Avec :

- $\tau_{i,j}$ est la quantité de phéromones déposée par la fourmi de l'état ***i*** à l'état ***j***. Donc la valeur initiale ne peut pas être 0, sinon la probabilité est égale à 0 et l'algorithme s'arrêtera au début ;

- $\eta_{i,j}$ est la désirabilité de déplacer de l'état ***i*** à l'état ***j***. Par exemple, sa valeur peut être égale à $\frac{1}{d_{i,j}}$ ($d_{i,j}$ est la distance entre l'état ***i*** et l'état ***j***). Ça veut dire que la distance est moins, la désirabilité de déplacer de l'état ***i*** à l'état ***j*** est plus ;
- *EtatsRestantFaisables* est l'ensemble des états auxquelles ne sont pas passés par la fourmi et la fourmi peut directement aller de l'état ***i***.

Quand une fourmi prendre un chemin, elle déposera des phéromones sur ce chemin. La quantité de phéromones que la fourmi dépose sur le chemin de l'état ***i*** à l'état ***j*** est noté $\Delta\tau_{i,j}$, sa valeur suit la formulaire(2) :

$$\Delta\tau_{i,j}\begin{cases}\rho\times f(S) & \text{Si c'est un problème de maximisation}\\ \frac{\rho}{f(S)} & \text{Si c'est un problème de minimisation}\end{cases}\quad \forall i\in\{1,...,n\},\forall j\in\{1,...,m\} \qquad (2)$$

Avec :

- ρ est le taux d'évaporation, $0<\rho<1$;
- *S* est la solution faisable générée par une fourmi et *f(S)* est la valeur de la fonction objective selon cette solution.

Après que la fourmi fait son déplacement, la quantité de phéromone sera aussi être mise à jour, la nouvelle quantité de l'état i à l'état j suit la formulaire(3) :

$$\tau_{i,j}=(1-\rho)\times\tau_{i,j}+\Delta\tau_{i,j} \qquad (3)$$

Enfin, chaque fourmi construit sa propre solution faisable. On met respectivement ces solutions à la fonction objective, on peut obtenir la meilleure solution selon les valeurs de la fonction objective.

En fait, il y a deux possibilités de déposer les phéromones :

- soit déposer les phéromones sur le nœud ;
- soit déposer les phéromones sur le chemin.

Dans le problème du sac à dos, les deux façons sont possibles :

- si la phéromone est dépose sur le nœud : le nœud est l'objet. L'objet a plus de phéromones, il a plus de probabilités d'être sélectionné.
- si la phéromone est dépose sur le chemin : le chemin est :
  - soit entre les objets : le chemin a plus de phéromones, il y a plus de probabilités de sélectionner l'objet ***j*** après que l'objet ***i*** est sélectionné ;
  - soit entre l'objet et le sac, le chemin a plus de phéromones, il y a plus de probabilités d'affecter l'objet ***j*** au sac ***i***.

4 Analyse et conception

Dans ce chapitre, on identifie ce problème d'affectation de sans-abirs, fait la modélisation mathématique.Et on explique en detail l'algorithme de colonies de fourmis que l'on propose et adapte.

1 Comparaison et identification du problème

Ce problème ressemble aux 3 problèmes classiques :

- Le problème du bin packing ;
- Le problème du sac à dos multiple ;
- Le problème du p-médian.

On essaie de trouver un modèle de problème classique qui est le plus proche de notre problème. Pour identifier notre problème, on fait 3 comparaisons entre notre problèmes et les 3 problèmes classiques.

1.1 Comparaison avec le problème du bin packing

La comparaison entre notre problème et le problème du bin packing est présenté dans le tableau comme Tableau1.

- La condition de ce modèle : le nombre de centres d'accueil est inconnu. C'est-à-dire que tous les sans-abris doivent sûrement être affectés à un centre d'accueil.On veut déterminer que l'on a besoin de combien de centres d'accueil ;
- Mais dans notre problème, le nombre de centres d'accueil est fixe ;

1.2 Comparaison avec le problème du sac à dos multiple

La comparaison entre notre problème et le problème du sac à dos multiple est présenté dans le tableau comme Tableau2.

Table 1 – *Le tableau de comparaison avec le problème du bin packing*

	Problème du bin packing	Notre problème
Description	***n*** objets numérotés par l'indice ***i*** variant de 1 à ***n***	***n*** bâtiments numérotés par l'indice ***i*** variant de 1 à ***n***
	m boîtes numérotés par l'indice ***j*** variant de 1 à ***m***	***m*** centres d'accueil numérotés par l'indice ***j*** variant de 1 à ***m***
	l'objet numéro ***i*** a la taille v_i	le bâtiment numéro ***i*** a le nombre de sans-abris P_i
	la taille de chaque boîte V_j	la capacité de chaque centre d'accueil W_j
Objectif	Minimiser le nombre de boîtes utilisées	Minimiser le nombre de centres d'accueil utilisés

Table 2 – *Le tableau de comparaison avec le problème du sacs à dos multiple*

	Problème du sac à dos multiple	Notre problème
Description	***n*** objets numérotés par l'indice ***i*** variant de 1 à ***n***	***n*** bâtiments numérotés par l'indice ***i*** variant de 1 à ***n***
	m sacs à dos numérotés par l'indice ***j*** variant de 1 à ***m***	***m*** centres d'accueil numérotés par l'indice ***j*** variant de 1 à ***m***
	l'objet numéro ***i*** a les poids w_i et le coût p_i	le bâtiment numéro ***i*** a le nombre de sans-abris P_i et la distance avec chaque centre d'accueil $d_{i,j}$
	la capacité de chaque sac à dos W_j	la capacité de chaque centre d'accueil W_j
Objectifs	Maximiser le coût total des objets pour remplir les sacs à dos	Minimiser la distance totale parcourue par les sans-abris pour rejoindre leurs centres d'accueil
	Minimiser les poids totaux des objets pour remplir les sacs à dos	Maximiser le nombre de sans-abris hébergés par les centre d'accueil (ou Maximiser le nombre de bâtiments affectés)

- La condition de ce modèle : le nombre de centres d'accueil est limité. C'est-à-dire que c'est possible qu'il y a des sans-abris qui ne sont pas affectés (car il n'y a pas assez d'espaces), on veut déterminer quels bâtiments doivent être affectés ;
- Cette condition correspond parfaitement à la conditon de notre problème.

1.3 Comparaison avec le problème du p-médian

La comparaison entre notre problème et le problème du p-médian est présenté dans le tableau comme Tableau3.

- La condition de ce modèle :la capacité de chaque centre d'accueil et le nombre de centres d'accueil sont inconnus. C'est-à-dire que tous les sans-abris doivent être affectés à un

Table 3 – *Le tableau de comparaison avec le problème du p-médian*

	Problème du p-médian	Notre problème
Description	***n*** clients numérotés par l'indice ***i*** variant de 1 à ***n***	***n*** bâtiments numérotés par l'indice ***i*** variant de 1 à ***n***
	m nœuds de service numérotés par l'indice ***j*** variant de 1 à ***m***	***m*** centres d'accueil numérotés par l'indice ***j*** variant de 1 à ***m***
	le client numéro ***i*** a le nombre de besoins v_i et la distance avec chaque nœuds de service $d_{i,j}$	le bâtiment numéro ***i*** a le nombre de sans-abris P_i et la distance avec chaque centre d'accueil $d_{i,j}$
Objectif	Minimiser la distance avec les poids (le nombre de besoins) entre les clients et les nœuds de service sélectionnés les plus proches	Minimiser la distance avec les poids (le nombre de sans-abris) entre les bâtiments et les centres d'accueil sélectionnés les plus proches

centre d'accueil mais peut-être il reste des certains centres d'accueil qui ne sont pas utilisés, on veut déterminer quels centres d'accueil doivent être sélectionnés entre ces centres d'accueil ;

- Mais la capacité de chaque centre d'accueil et le nombre de centres d'accueil sont déjà fixés dans notre problème.

1.4 Identification du problème

Selon les comparaisons, on trouve que notre problème est plus proche du **problème de sac à dos multiple**.

Les bâtiments sont les objets, et les centres d'accueil sont les sacs à dos. La distance est presque équivalent au profil de chaque objet, et le nombre de population de chaque bâtiment est équivalent au poids de chaque objet.

Néanmoins, il y a aussi la différence. Dans le problème du sac à dos multiple, le profil de chaque objet est déjà fixé, mais dans notre problème, la distance de chaque bâtiment dépend du centre d'accueil.

2 Modélisation mathématique

Dans cette section, on fait la modélisation mathématique de notre problème, y compris les données, la valeur, les contraintes et les fonctions objectives.

2.1 Données

On considère une instance du problème à n bâtiments et m centre d'accueil. Pour chaque bâtiment $i \in \{1,...,n\}$, on connait :

— La distance pour rejoindre chaque centre d'accueil :

$$d_{i,j} \geq 0 \; \forall (i,j) \in \{1,...,n\} \times \{1,...,m\}$$

— le nombre de sans-abris : $p_i \geq 0 \; \forall i \in \{1,...,n\}$

Pour chaque centre d'accueil $j \in \{1,...,m\}$, on connait :

— sa capacité d'accueil : $W_j \geq 0 \; \forall j \in \{1,...,m\}$

2.2 Variable

C'est l'affectation des sans-abris d'un même bâtiment à un centre d'accueil :

$$x_{i,j} = \begin{cases} 1 & \text{si les sans-abris du bâtiment } i \text{ sont affectés au centre } j \\ 0 & \text{sinon} \end{cases}$$

2.3 Contraintes

— un bâtiment ne peut être affecté qu'à un centre d'accueil :

$$\sum_{j \in \{1,...,m\}} x_{i,j} \leq 1 \; \forall i \in \{1,...,n\}$$

— la capacité d'un centre d'accueil j est limitée :

$$\sum_{i \in \{1,...,n\}} x_{i,j} \times p_i \leq W_j \; \forall j \in \{1,...,m\}$$

2.4 Fonctions objectifs

Selon les besoins de client, on conçoit deux plans de fonctions objectives, et chaque plan contient 2 fonctions objectives.

- on cherche à minimiser la distance totale parcourue par les sans-abris pour rejoindre leur centre d'accueil (f) tout en maximisant le nombre de personnes abritées (g) :

$$f(x_{i \in \{1,...,n\}, j \in \{1,...,m\}}) = \min \sum_{i \in \{1,...,n\}} \sum_{j \in \{1,...,m\}} p_i \times x_{i,j} \times d_{i,j}$$

$$g(x_{i \in \{1,...,n\}, j \in \{1,...,m\}}) = \max \sum_{i \in \{1,...,n\}} \sum_{j \in \{1,...,m\}} x_{i,j} \times p_i$$

- on cherche à minimiser la distance totale parcourue par les sans-abris pour rejoindre leur centre d'accueil (f) tout en maximisant le nombre de bâtiments évacués(h) :

$$f(x_{i \in \{1,...,n\}, j \in \{1,...,m\}}) = \min \sum_{i \in \{1,...,n\}} \sum_{j \in \{1,...,m\}} p_i \times x_{i,j} \times d_{i,j}$$

$$h(x_{i \in \{1,...,n\}, j \in \{1,...,m\}}) = \max \sum_{i \in \{1,...,n\}} \sum_{j \in \{1,...,m\}} x_{i,j}$$

Ce deux plans de fonctions objectives correspondent respectivement aux objectifs décrits dans le chapitre 1 « Objectifs ». C'est le client qui détermine finalement quel plan il prendra selon la solution que l'on fournit.

3 Modélisation logicielle

On utilise le pattern MVC pour notre modèle de programme. Parce que l'on n'a pas besoin d'interface graphique, il n'y a pas la partie « Vue » dans la modélisation. La Figure1 montre

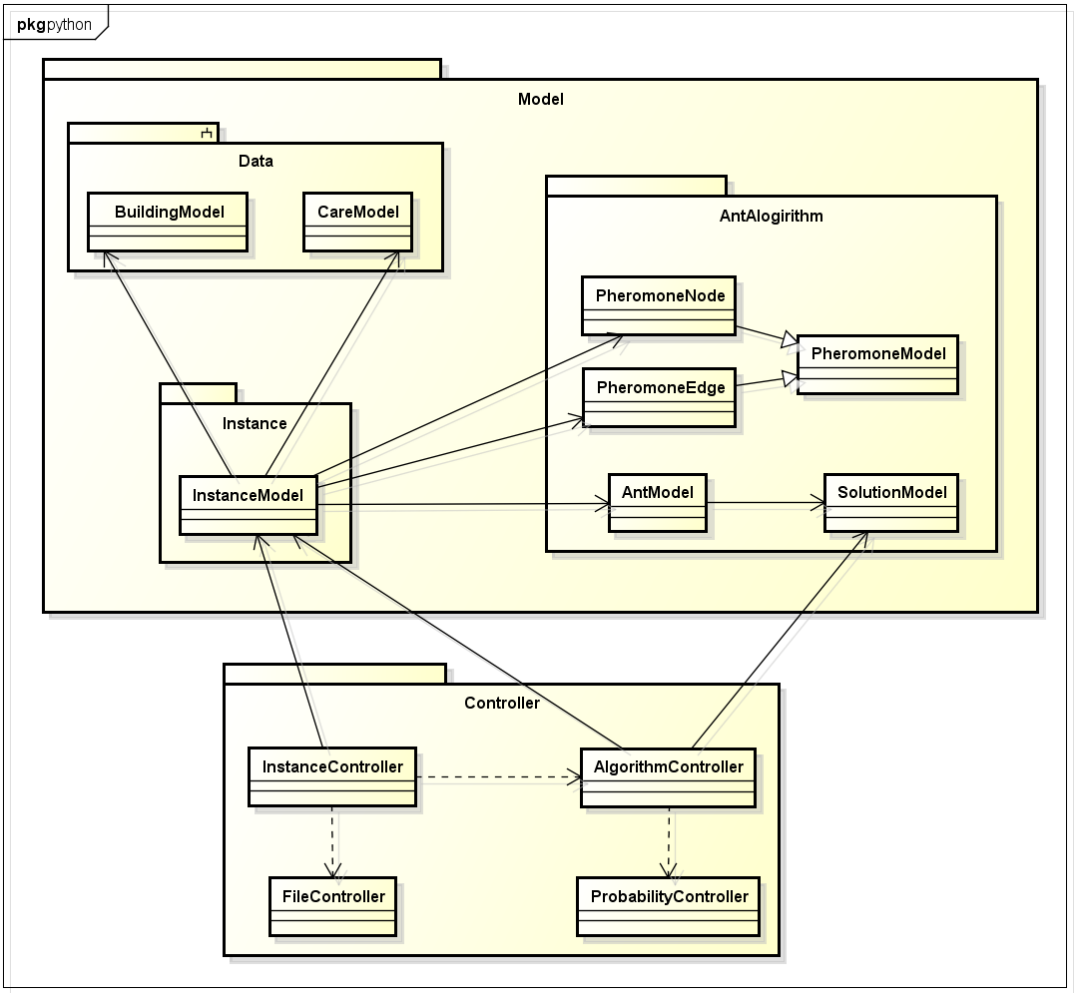


Figure 1 – *Le diagramme de classe(sans attributs et sans méthodes)*

les relations entre des classes dans le diagramme de classe (sans attributs et sans méthodes). Le diagramme de classe entier est présenté dans l'annexe « Aperçu de la diagramme de classe entier ».

3.1 Modèle

La partie de modèle dans le diagramme de classe est comme Figure2.

Dans la partie « Modèle », il y a 3 packages : « Data », « AntAlgorithm » et « Instance ». Chaque package contient des classes de modèle.

- Package « Data »
  1. Classe « BuildingModel » : cette classe est le modèle de bâtiment. Elle contient son identifiant et son nombre de sans-abris.

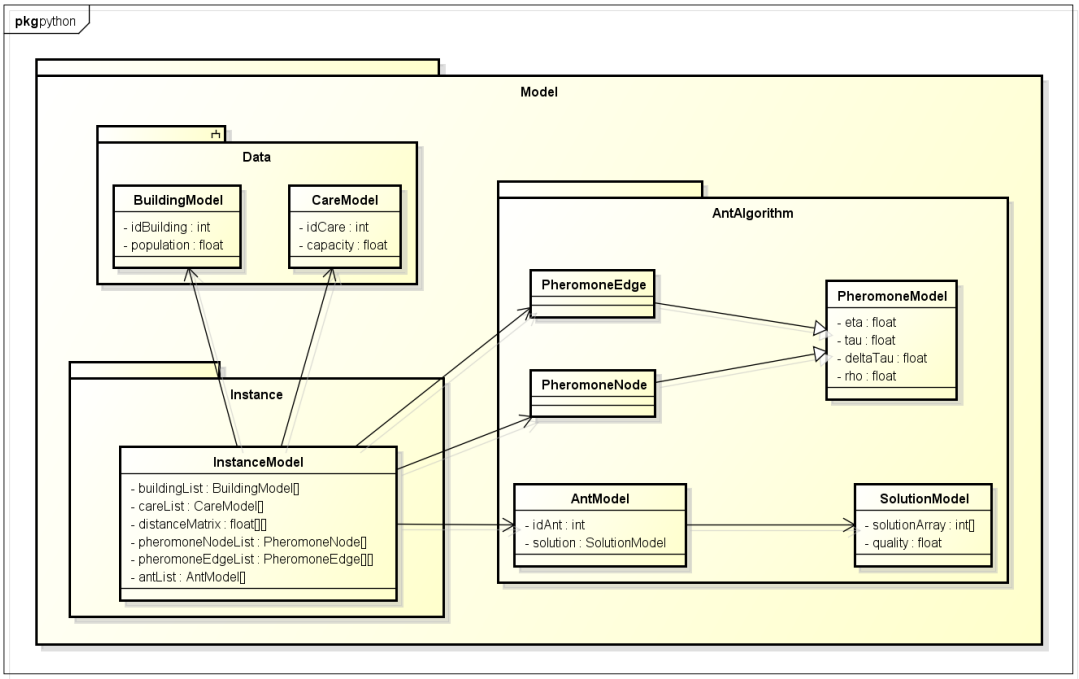


Figure 2 – *La partie « Modèle » détaillée du diagramme de classe*

2. Classe « CareModel » : cette classe est modèle de centre d'accueil. Elle contient son identifiant et sa capacité.

- Package « AntAlgorithm »
  1. Classe « SolutionModel » : cette classe est le modèle de solution construite par chaque fourmi. Elle contient un tableau de solution et la qualité de solution.
  2. Classe « AntModel » : cette classe est le modèle de fourmi. Chaque fourmi a un identifiant et un objet de solution.
  3. Classe « PhermoneModel » : cette classe est le modèle de phéromone. Il contient toutes les variables que l'algorithme de colonie de fourmis demande.
  4. Classe « PhermoneNode » : cette classe est le modèle de phéromone qui est déposée sur les nœuds de bâtiment.
  5. Classe « PhermoneEdge » : cette classe est le modèle de phéromone qui est déposée sur les arcs entre le bâtiment et le care.
- Package « Instance »

  1. Classe « InstanceModel » : cette class est le modèle d'instance pour ce projet, y compris la liste de bâtiments, la liste de cares, la liste de phéromones de nœud, la matrice de phéromones d'arc, la matrice de distances et la liste de fourmis.

Les paramètres d'entrée et les valeurs retournées de chaque classe dans la partie « modèle » sont décrites en détail dans l'annexe « Descriptions détaillées des classes du package « modèle » ».

3.2 Contrôleur

La partie de contrôleur dans le diagramme de classe est comme Figure3.

Dans la partie « Contrôleur », il y a 4 classes : « InstanceController », « FileController », « ProbabilityController » et « AlgorithmController ».

1. Classe « InstanceController » : cette classe est pour construire l'instance de projet et fournir un service de demarrer l'algorithme.
2. Classe « FileController » : cette classe sert à faire les opérations sur les fichiers, y compris la lecture et l'écriture.
3. Classe « ProbabilityController » : cette classe est pour contrôler la probabilité lorsque les fourmis déplacent.
4. Classe « AlgorithmController » : cette classe est le contrôleur d'algorithme, qui réalise à chercher la meilleure solution d'affection.

Les paramètres d'entrée et les valeurs retournées de chaque classe dans la partie « contrôleur » sont décrites en détail dans l'annexe « Descriptions détaillées des classes du package « contrôleur » ».

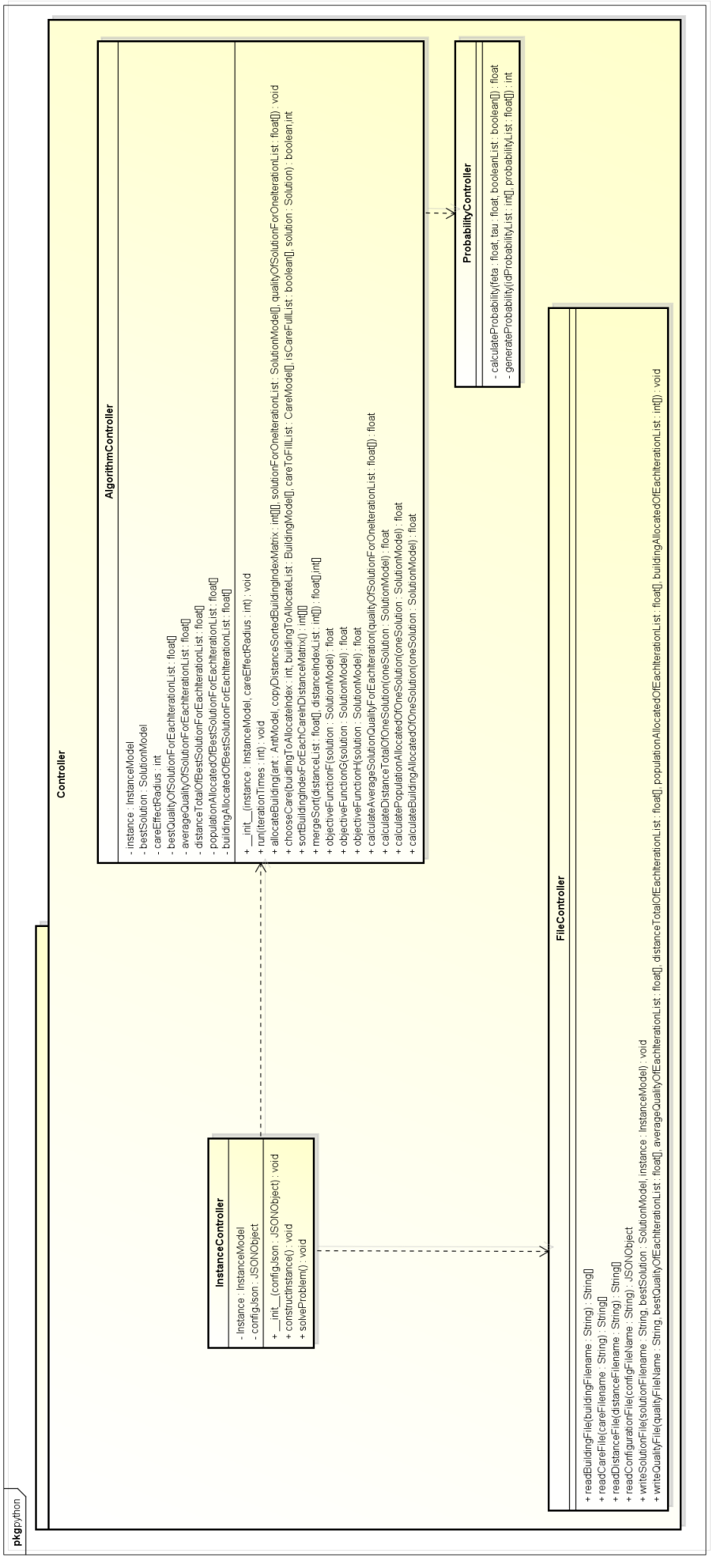


Figure 3 – *La partie « Contrôleur » détaillée du diagramme de classe*

5 Mise en oeuvre

Dans ce chapitre, on explique en détail l'algorithme proposé initiale et l'algorithme amélioré. On explique aussi le détermination des paramètres de l'algorithme.

1 Introduction de l'algorithme proposé

La première version de l'algorithme que l'on propose est une adaptation de l'algorithme de colonies de fourmis (présenté dans le chapitre 3 « Algorithme de colonies de fourmis – métaheuristique ») pour résoudre le problème sac à dos multiple.

1.1 Utilisation des phéromones

On considère notre problème comme un graphe(Figure1), et il y a deux types de phéromones :

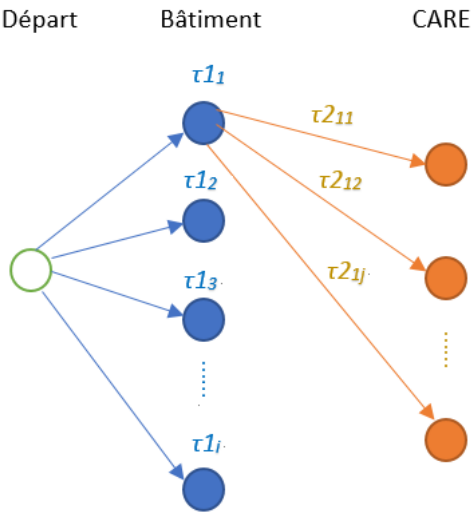


Figure 1 – *Le graphe du problème*

- L'un type de phéromone est déposée sur le nœud de bâtiment : le nœud a plus de phéromones, ce bâtiment a plus de probabilités d'être sélectionné ;
- L'autre type de phéromone est déposée sur l'arc entre le bâtiment et la care(i.e. le centre d'accueil) : l'arc a plus de phéromones, ce bâtiment a plus de probabilité d'être affecté à cette care.

1.2 Variables nécessaires

Selon les formulaires de l'algorithme de colonies de fourmis qui sont présenté dans le chapitre 3, les variables suivantes sont obligatoires :

- $\eta \mathrm{N}ode_i$: la désirabilité de mouvement sur le nœud du bâtiment $\boldsymbol{i}$;
- $\eta \mathrm{E}dge_{i,j}$: la désirabilité de mouvement sur l'arc du bâtiment $\boldsymbol{i}$ à la care $\boldsymbol{j}$;
- $\tau \mathrm{N}ode_i$: la quantité de phéromones existante sur le nœud du bâtiment $\boldsymbol{i}$;
- $\tau \mathrm{E}dge_{i,j}$: la quantité de phéromones existante sur l'arc du bâtiment $\boldsymbol{i}$ à la care $\boldsymbol{j}$;
- $\Delta\tau \mathrm{N}ode_i$: la quantité de phéromones déposées sur le nœud du bâtiment $\boldsymbol{i}$ lors que les fourmis prennent ce nœud ;
- $\Delta\tau \mathrm{E}dge_{i,j}$: la quantité de phéromones déposées sur l'arc entre le bâtiment $\boldsymbol{i}$ et la care $\boldsymbol{j}$ lors que les fourmis prennent cet arc ;
- $\rho \mathrm{N}ode$: le taux d'évaporation de phéromones déposées sur le nœud de bâtiment.
- $\rho \mathrm{E}dge$: le taux d'évaporation de phéromones déposées sur l'arc entre le bâtiment et le care.
- $proba\mathrm{N}ode_i$: la probabilité de sélectionner le bâtiment $\boldsymbol{i}$ pour le pas prochain ;
- $proba\mathrm{E}dge_{i,j}$: la probabilité d'affecter le bâtiment $\boldsymbol{i}$ au care $\boldsymbol{j}$ pour le pas prochain ;

Sauf ces variables, il y a aussi les autres variables sont très importantes :

- $building\mathrm{T}o\mathrm{A}llocate\mathrm{L}ist_i$: la liste de bâtiment à affecter ;
- $care\mathrm{T}o\mathrm{F}ill\mathrm{L}ist_j$: la liste de care à remplir ;
- $is\mathrm{B}uilding\mathrm{S}elected\mathrm{L}ist_i$: la liste de signe de bâtiment, 1 si le bâtiment i est sélectionné, 0 sinon ;
- $is\mathrm{C}are\mathrm{F}ull\mathrm{L}ist_j$: la liste de signe de care, 1 si la care est pleine, 0 sinon ;
- $ant\mathrm{Q}uantity$: le nombre de fourmis ;
- $iteration\mathrm{T}imes$: le nombre d'itérations.

On peut utiliser la classe « InstanceModel » pour encapsuler ces variables dans un objet d'instance.

1.3 différences entre les plans de fonctions objectives

Même si on a deux plans pour les fonctions objectives à résoudre, la différence ne consiste qu'à la valeur de $\eta \mathrm{N}ode_i$ et le choix de fonction objective quand on met à jour $\Delta\tau \mathrm{N}ode_i$:

- Pour le cas de maximiser le nombre de sans-abris affectés :
  - Pour sélectionner le bâtiment :
    - ★ $\eta \mathrm{N}ode_i = p_i$(le nombre de personne dans le bâtiment i), plus le nombre de personnes dans le bâtiment est élevé, plus la probabilité qu'il soit sélectionné est élevé ;
    - ★ On met la phéromone sur le nœud de bâtiment ;
    - ★ Quand on met à jour $\Delta\tau \mathrm{N}ode_i$, on utilise la fonction objective g(x).
  - Pour sélectionner la care :
    - ★ $\eta \mathrm{E}dge_{i,j} = 1/d_{i,j}$(la distance entre le bâtiment i et la care j), si la distance est plus petite, la probabilité qu'il soit sélectionné est plus grande ;
    - ★ On met la phéromone sur l'arc entre le bâtiment et la care ;

★ Quand on met à jour $\Delta\tau Edge_{i,j}$, on utilise la fonction objective f(x).

- Pour le cas de maximiser le nombre de bâtiments affectés :
  - Pour sélectionner le bâtiment :
    - ★ $\eta Node_i = \frac{1}{p_i}$(le nombre de personne dans le bâtiment i). On affecte en priorité les bâtiments qui possède le moins de personnes ;
    - ★ On met la phéromone sur le nœud de bâtiment ;
    - ★ Quand on met à jour $\Delta\tau Node_i$, on utilise la fonction objective h(x).
  - Pour sélectionner la care :
    Tous sont pareils que le cas précédent.

1.4 Cas d'arrêt

Dans cet algorithme, il y a 3 cas d'arrête :

1. Quand tous les bâtiments sont affectés ou quand tous les cares sont pleins, la fourmi finit sa cherche pour une itération.
2. Quand tous les fourmi trouvent leurs propres solutions, une itération finit et le programme passe à l'itération suivante.
3. Quand toutes les itérations finissent, l'algorithme s'arrête.

2 Version initiale de l'algorithme proposé

Cette section présente les procédures de l'algorithme initiale, y compris le procédure d'affectation de bâtiment, le procédure de sélection de care, et le procédure de générateur de probabilité.

2.1 Procédure initial d'affecter les bâtiments

Le rocédure initial d'affecter les bâtiments est décrit dans le pseudocode de fonction « allocateBuilding__Initial() » comme Algorithme1.

Au début, les fourmis doivent calculer la probabilité de mouvement au nœud de bâtiment avec $\tau Node_i$ et $\eta Node_i$ pour choisir un numéro de bâtiment à affecter. Si c'est la première fois d'itération, les fourmis ne calculent pas ces probabilités. Elles choisissent un bâtiment par hasard. Sinon, après qu'elles calculent toutes les probabilités de bâtiments non-affectés, elles utilisent un générateur de probabilité à choisir un bâtiment. Ce générateur est réalisé par la fonction suivante qui s'appelle « GenerateProbability() », et il est pour assurer que les mouvements avec la très petite probabilité sont aussi possible d'avoir lieu.

Après avoir choisi un bâtiment, les fourmis appellent la fonctions suivante qui s'appelle « ChooseCare() » pour déterminer la care à remplir par ce bâtiment. Cette fonction retourne un statut de cares qui indique si toutes les cares sont pleines ou pas, et une solution partielle. Ensuite, les fourmis mettent à jour $\Delta\tau Node_i$, $\tau Node_i$, et $isBatSelectedList_i$. Si la valeur de statut de cares retourné est vraie, c'est-à-dire que toutes les cares sont déjà pleines, l'itération de la fourmi actuelle s'arrête. Sinon, elles font l'itération pour mettre à jour leurs propres solutions partielles.

Dans une itération, lorsqu'une fourmi quitte sa boucle, sa solution finale sera ajoutes dans la liste de solutions de cette itération. Et quand toutes les fourmis finissent leurs cherches, la méthode retournera la liste de solution de cette iteration.

2.2 Procédure initial de choisir les cares

Le procédure initial de choisir les cares est décrit dans le pseudocode de fonction « chooseCare__Initial() » comme Algorithme2.

Cette fonction ressemble à la fonction « allocateBuilding() ». Au début, les fourmis doivent calculer la probabilité de mouvement à l'arc entre le bâtiment et la care non-plaine avec $\tau E dge_{i,j}$ et $\eta E dge_{i,j}$ pour choisir un numéro de care à remplir. Puis elles utilisent un générateur de probabilité à choisir une care, et produit une solution partielle. Ensuite, les fourmis mettent à jour $\Delta\tau E dge_{i,j}$, $\tau E dge_{i,j}$, et $isCareFullList_j$.

Alors les fourmis vérifient si la capacité de ces cares est inférieure à la population minimale entre les bâtiments restant. Si oui, elles pensent que cette care est déjà pleine. Enfin, elles vérifient si toutes les cares sont plaine, et la fonction retourne l'état de toutes les cares et la solution partielle générée par la fourmi actuelle.

2.3 Procédure de générateur de probabilité

Le Procédure de générateur de probabilité est décrit dans le pseudocode de fonction « generateProbability() » comme Algorithme3.

Les paramètres d'entrée de cette fonction sont la liste d'une séquence et la liste de probabilités calculées par la fourmi. La fonction « zip() » est pour encapsuler la séquence et les probabilités en un tuple.

Par exemple, les probabilités de mouvement est (0.1,0.3,0.6), et la séquence est [a,b,c]. Au début, la valeur de « probabilityCompared » est un nombre aléatoire entre 0 et 1, et la valeur de « probabilityCumulative » est égale à 0.0. Puis on calcule la somme de « probabilityCompared » et « probabilityCumulative » en boucle. Donc, la probabilité de sélectionner « a » est égale à la probabilité que probabilityCumulative=[0,0.1], la probabilité de sélectionner « b » est égale à la probabilité que probabilityCumulative=[0.1,0.4], la probabilité de sélectionner « c » est égale à la probabilité que probabilityCumulative=[0.4,1.0]. La fonction retourne l'article sélectionné.

Cette fonction assure que les évènements dont la probabilité est petite peuvent aussi avoir la chance d'être sélectionnées.

2.4 Evaluation les solutions

Pour Evaluer les solutions, on calcule la qualité de chaque solution : « G(x) / (1+F(x)) » ou « H(x) / (1+F(x) » avec « x = solutionArray ». Plus la valeur est grande, plus la solution est forte. Dans chaque itération, parmi toutes les solutions générées par les fourmis, on sélectionne une solution dont la qualité est maximum comme la meilleure solution de l'itération actuelle.

A la fin d'exécution d'algorithme, on sélectionne une solution dont la qualité est plus grande parmi les meilleures solutions de chaque itération comme la meilleure solution finale.

2.5 Analyse du risque

Dans ce projet, la taille de données est très grande, il y aura environ 55000 bâtiments et 187 centres d'accueil selon les données que le client fournit. Et il y aura aussi plusieurs fourmis dans cet algorithme. Donc quand le système s'exécute, le nombre d'itérations atteint à des milliards.

Par exemple, pour chaque fourmi, elle doit passer 50000 bâtiments. Lorsque la fourmi

sélectionne un bâtiment à affecter, elle doit comparer les probabilité de mouvement entre tous les bâtiments pour décider le mouvement prochain. Et lorsque que la fourmi sélectionne une care à remplir pour le bâtiment actuellement sélectionné, elle doit comparer les probabilité de mouvement entre tous les cares.

Donc pour cette algorithme initiale, le nombre d'itération est 55000*55000*187 pour une fourmi et une itération. L'exécution d'une fois prendra beaucoup de temps. La performance de l'algorithme initiale n'est pas bonne.

3 Améliorations de l'algorithme proposé

On a pris beaucoup de mesures et fait beaucoup d'essais pour améliorer l'efficience de l'algorithme. Globalement, les améliorations peut êtres divisées en 4 étapes.

3.1 Etape 1 : Ajouter le rayon d'attraction de care

La première amélioration que l'on fait est d'introduire le concept de rayon d'attraction pour chaque care. Pour un care, on suppose qu'il y a un cercle dont son centre est le care et son rayon est le rayon d'attraction que l'on introduit. Le schéma de cercle de care est comme Figure2). .

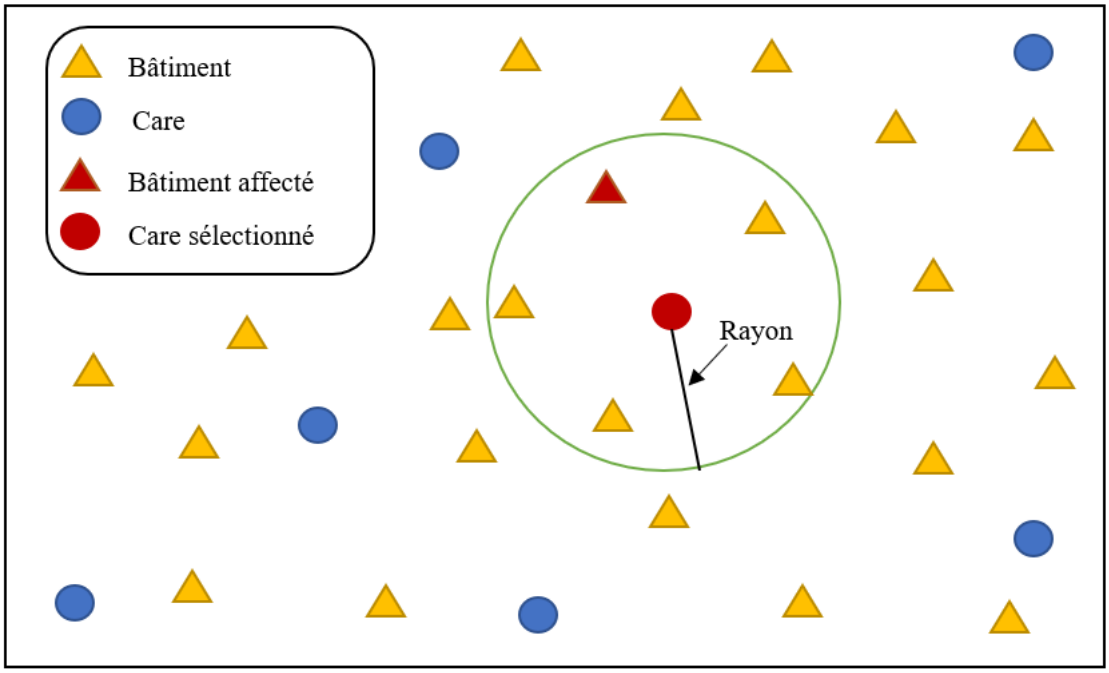


Figure 2 – *Le schéma de cercle de care*

Lorsqu'un bâtiment est affecté à un care j, et la fourmi commence à sélectionner le bâtiment suivant, il suffit de chercher le bâtiment parmi les bâtiments qui sont dans le cercle de care j. Ce n'est plus la peine de parcourir tous les bâtiments. Par exemple, dans la figure 1, on initialise un rayon pour chaque care, si bâtiment rouge est affecté aux cares rouges, on dessine un cercle dont le centre est le care sélectionné avec le rayon initial. Dans ce cercle, il y aura des bâtiments non-affecté (ce sont les triangles jaunes dans le cercle). La fourmi choisit le bâtiment à affecter pour le déplacement suivant parmi ces bâtiments jaunes encadrés par le cercle.

Si tous les bâtiments dans le cercle sont affectés, il faut étendre le rayon du care sélectionne, puis la fourmi choisit un bâtiment dans le nouveau cercle. Le schéma d'extension de cercle est comme Figure3.

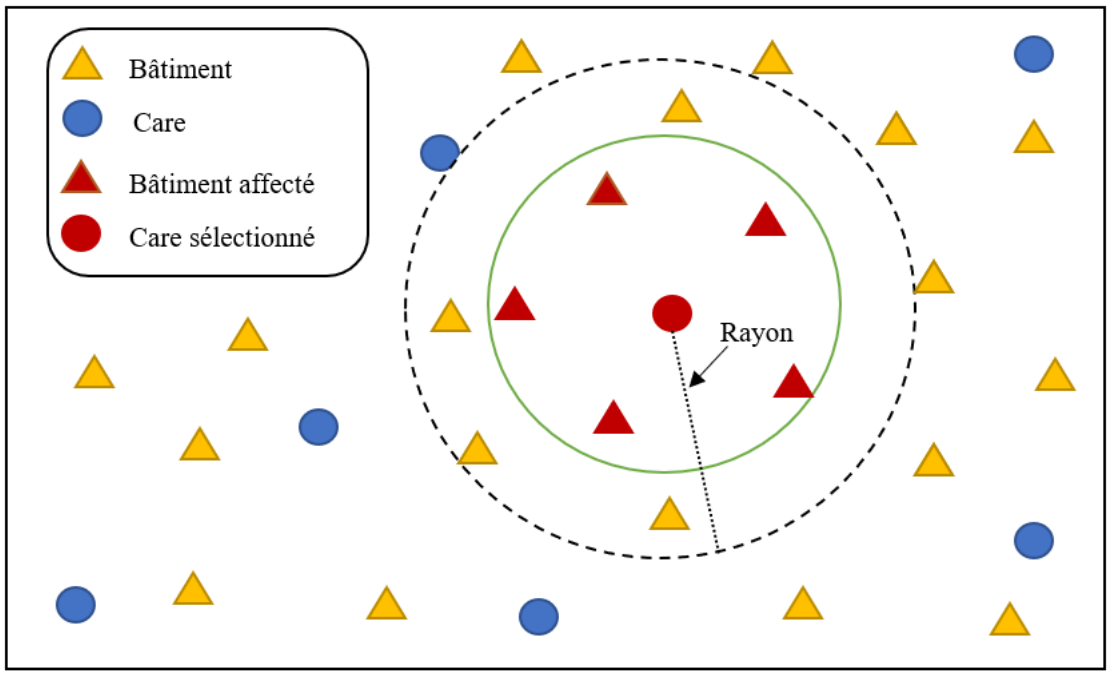


Figure 3 – *Le schéma d'extension de cercle d'attraction*

3.2 Etape 2 : Essayer d'utiliser le multithread

La deuxième idée pour diminuer le temps d'exécution est d'essayer d'utiliser le multithreads. Mais après beaucoup d'expérimentation, on était surprise que le multithreads prenne plus de temps que le thread unique.

Pour le multithread en langage Python, il y a un verrou exclusif global qui s'appelle GIL « Global Interpreter Lock ». Ce verrou est pour synchroniser les ressources et les états entre les threads. Donc seulement un thread peut exécuter en même temps, les autres threads doivent attendre. L'attente de thread prend beaucoup de temps.

Pour le programme qui exécute sur CPU, il calculera le nombre de codes exécutés en ce montent afin que les threads puissent utiliser le temps de CPU en moyenne. Lorsque le nombre de codes exécutés atteint un seuil, le verrou GIL sera forcément libéré, et une compétition de ressource aura lieu entre les threads. Ainsi, la compétition de ressource gaspille beaucoup de temps.

Bref, le multithread en Python n'exécute pas en parallèle, il exécute alternativement. La Figure4 montre l'exécution de multithread en Python.

Parce que la performance de multithread n'est pas bonne pour notre projet, on ne l'utilise pas finalement.

3.3 Etape 3 : Ajouter la liste de candidat de care

La troisième mesure que l'on fait pour augmenter l'efficacité est d'ajouter la liste de candidat pour chaque care. On fixe la taille de la liste de candidat (par exemple 10), et la liste stocke les indices de bâtiment. Il y a deux contraintes lors de remplir la liste de candidat :

- La distance entre le bâtiment à ajouter et le care j est inférieure ou égale au rayon actuel du care j ;
- La taille de la liste de candidat du care j ne dépasse pas encore 10.

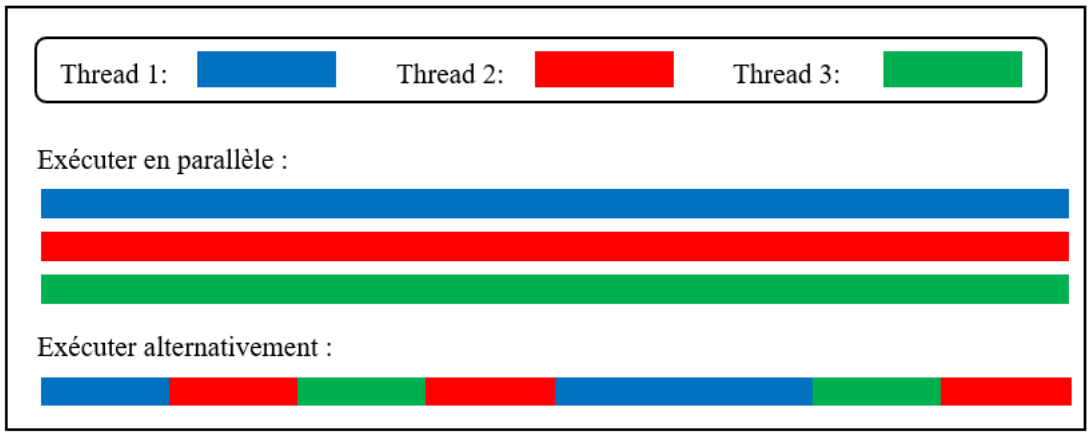


Figure 4 – *L'exécution parallèle et l'exécution alternative pour le multithread*

Ainsi, on réduit de plus le nombre de bâtiments à parcourir quand la fourmi choisit son déplacement suivant. La Figure5 montre la structure de la liste de candidat de care.

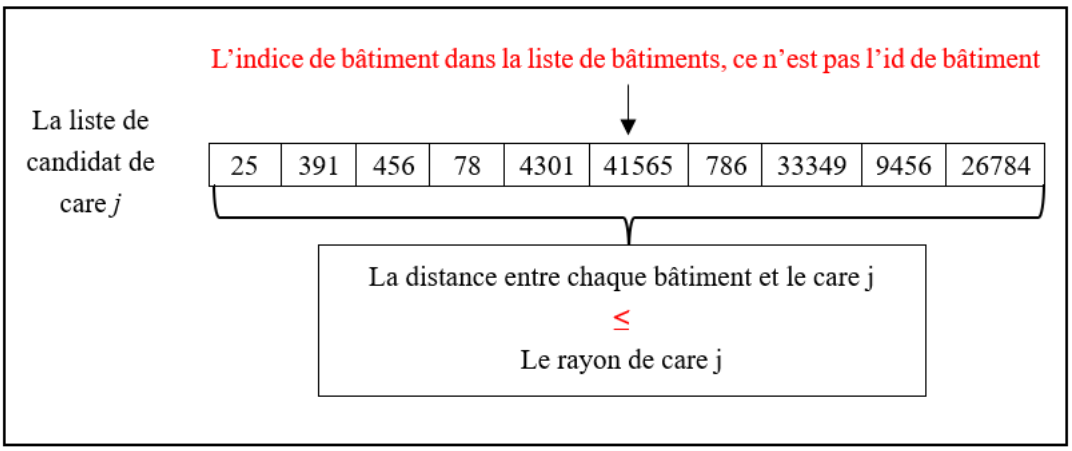


Figure 5 – *La liste de candidat de care j*

Quand la fourmi commence à choisir un autre bâtiment, si la liste de candidat de care j est vide, il faut reremplir la liste de candidat. Et quand on reremplit la liste de candidat.

3.4 Etape 4 : Trier les distances au préalable

La dernière amélioration est d'obtenir la matrice de distance triée par l'ordre croissante l'algorithme commence. Il y a beaucoup d'endroit où on doit chercher le bâtiment la distance est minimum. Auparavant, on fait le tri dans le boucle. Mais si on fait le tri d'avance, il ne suffit de faire le tri qu'une fois.

Parce que l'on a besoin de savoir le numéro de bâtiment et le numéro de care pendant l'exécution d'algorithme, on ne peut pas directement trier la matrice de distance. On trie une matrice qui stocke les indices de bâtiments en référant la matrice de distance.

Par exemple, on a 5 bâtiments et 7 care. La matrice d'indices de bâtiment initiale est comme Figure6

Ensuite, on trie cette matrice en référant la matrice de distance. La matrice d'indices de bâtiment triée est comme Figurereffig15

La matrice d'indices de bâtiment initiale

Care 1	0	1	2	3	4
Care 2	0	1	2	3	4
Care 3	0	1	2	3	4
Care 4	0	1	2	3	4
Care 5	0	1	2	3	4
Care 6	0	1	2	3	4
Care 7	0	1	2	3	4

Les indices de bâtiment dans la liste de bâtiments

Figure 6 – *La matrice d'indices de bâtiment initiale*

La matrice de distance à référer

	Care 1	Care 2	Care 3	Care 4	Care 5	Care 6	Care 7
Bat 0	6	34	12	25	43	28	9
Bat 1	24	15	55	23	22	45	32
Bat 2	36	56	3	19	12	33	26
Bat 3	47	51	30	17	42	38	30
Bat 4	8	35	25	36	24	49	16

La matrice d'indices de bâtiment triée

Care 1	0	4	1	2	3
Care 2	1	0	4	3	2
Care 3	2	0	4	3	1
Care 4	3	2	1	0	4
Care 5	2	1	4	3	0
Care 6	0	2	3	1	4
Care 7	0	4	2	3	1

Les indices de bâtiment dans la liste de bâtiments

Figure 7 – *La matrice d'indices de bâtiment triée*

Dans la matrice de distance, on fait le tri pour chaque care. Les distances entre les bâtiments de 0 à 4 et le care 1 est [6,24,36,47,8], on voir que la distance entre le bâtiment 1 et le care 1 est minimum, et la distance entre le bâtiment 3 et le care 1 est maximum. Donc on obtient une liste d'indices de bâtiment triée par l'ordre croissante pour care 1 : [0,4,1,2,3]. C'est la première ligne de la matrice d'indice de bâtiments triée.

Grace à cette matrice, quand on étend le rayon du care j, il suffit comparer la distance du premier bâtiment non-affecté dans la ligne du care j et le rayon du care j. Si la distance est supérieure au rayon, il faut étendre le rayon.

Pour faire le tri, on a essayé 4 algorithmes de tri efficaces : le tri par insertion, le tri par sélection, le tri à bulles et le tri par fusion. Le tri par fusion prend le moins de temps, donc on utilise le tri par fusion à trier la matrice d'indices de bâtiment.

4 Version améliorée de l'algorithme proposé

Cette section présente les procédures de l'algorithme amélioré, y compris le procédure de démarrage d'algorithme et d'obtention de la meilleure solution finale, le procédure d'affectation de bâtiment, et le procédure de sélection de care.

4.1 Variables globales

Pour l'algorithme amélioration, il y a 4 variables globales :

- descriptionInstance : l'objet de la classe « InstanceModel » construit par la méthode constructInstance() de la classe « InstanceController » ;
- bestSolution : l'objet de la classe « SolutionModel() ». C'est la meilleure solution trouvée finalement ;
- careEffectRadius : un nombre entier définit par l'utilisateur. C'est le rayon d'attraction initial de cercle dont le centre est chaque care ;
- bestQualityOfSolutionForEachIterationList : la liste de qualités de meilleure solution de chaque itération.

4.2 Procédure de démarrer l'algorithme

Le procédure de démarrer l'algorithme est décrit dans le pseudocode de fonction « run() » comme Algorithme4.

La fonction « run() » est l'entrée d'algorithme. Dans chaque itération, elle passe les fourmis à la fonction « allocateBuidling() » pour construire les solutions, et elle sélectionne la meilleure solution de l'itération actuelle. A la fin de fonction, elle synthétise toutes les meilleures solutions et sélectionner la meilleure solution finale selon leurs qualités.

4.3 Procédure amélioré d'affecter les bâtiments

Le procédure amélioré d'affecter les bâtiments est décrit dans le pseudocode de fonction « allocateBuilding__Improved() » comme Algorithme5.

Ce procédure a besoin des variables suivantes :

- ant.solution : l'objet de la classe « SolutionModel() » ;
- buildingToAllocateList = instance.buildingList : la liste de bâtiments à affecter ;
- careToFillList = instance.careList : la liste de care à remplir ;
- isBuildingSelectedList = [False] * len(buildingToAllocateList) : la liste qui marque si le bâtiment est déjà affecté ;
- isCareFullList = [False] * len(careToFillList) : la liste qui marque si le care est déjà plein ;
- radiusList = [self.careEffectRadius] * len(careToFillList) : la liste de rayon d'attraction de care ;
- ant.solution.solutionArray = [-1] * len(buildingToAllocateList) : la solution initiale, si la valeur est -1, ça veut dire que aucun care peut héberger ce bâtiment ;
- candidateListForCare = [[]] : la matrice de candidat de care.

Quand une fourmi commence à construire sa solution, d'abord, elle trie la matrice d'indices de bâtiment en référant la matrice de distance, et elle construit la liste de candidat pour chaque care. Quand elle affecte un bâtiment à un care, elle mémorise l'index de care sélectionné. Pour le pas suivant, la fourmi ne parcourt que les bâtiments dans la liste de candidat de care sélectionné dans le pas précédent.

Si la liste de candidat de care j est vide, il faut reremplir la liste de candidat pour le care j. Avant que la fourmi reremplit la liste de candidat, elle doit comparer la distance minimum à care j avec le rayon actuel de care j. Si la distance minimum est supérieure au rayon actuel de

care j, il faut étendre le rayon pour care j. Puis la fourmi reremplit la liste de candidat avec la contrainte de nouveau rayon.

Lorsque la fourmi finit la construction de solution, elle calcule la qualité de sa solution. Ensuite, elle ajoute respectivement la solution et la qualité de solution dans les listes correspondantes qui sont pour l'itération actuelle.

4.4 Procédure amélioré de choisir les cares

Le procédure amélioré de choisir les cares est décrit dans le pseudocode de fonction « chooseCare__Improved() » comme Algorithme6.

Il n'y a pas beaucoup de changements sur cette fonction par rapport à la version initiale, sauf deux points :

- Quand la fourmi cherche la population minimum, quand le bâtiment sélectionné est le dernier bâtiment qui peut être affecté (c'est possible qu'il reste encore des bâtiments, mais ces bâtiments ne peuvent pas être affectés car leurs populations dépassent toutes les capacités de care. Dans ce cas, on ne les considère plus), la population liste est vide. En ce moment, il faut directement retourner les valeurs pour informer la fonction « allocateBuidling() » que tous les bâtiments sont affectés, afin que la fourmi finit de construire la solution.
- Les valeurs retournées de la fonction dans cette version sont une signe qui marque si tous les cares sont pleins et l'indice de care sélectionné, car la fonction « allocateBuidling() » a besoin de savoir l'indice de care sélectionné dans chaque pas. Dans la version initiale, elle retourne directement la solution partielle.

5 Détermination des paramètres du programme

Pour déterminer les paramètres du problème, il faut faire beaucoup d'expérimentations. Souvent, ils dépendent de l'échantillon.

1. eta : la désirabilité de mouvement.
   (a) pour la phéromone déposée sur le nœud de bâtiment i :
      - si on veut maximiser le nombre de sans-abris hébergé(g(x)) : etaNode = le nombre de sans-abris dans le bâtiment i, car on veut affecter à priorité le bâtiment dont le nombre de sans-abris est le plus. Plus le nombre de sans-abris est grand, plus la probabilité de sélection est grand.
      - si on veut maximiser le nombre de bâtiment (h(x)) : eta = 1/ le nombre de sans-abris dans le bâtiment i, car si on affecte à priorité le bâtiment dont le nombre de sans-abris est le moins, on peut assurer d'affecter autant de bâtiments que possible.

   (b) pour la phéromone déposée sur l'arc entre le bâtiment i et le care j : etaEdge = 1/la distance entre le bâtiment i et le care j, car on sélectionne à priorité le care dont la distance est minimum pour minimiser la distance totale parcourue.
2. tau : la quantité de phéromones existant sur un nœud de bâtiment ou un arc.
   Généralement, la valeur de tau est entre 0.1 et 0.9. Et on doit faire la valeur de tau est plus grande possible, car plus le tau est grand, plus la probabilité de déplacement est grande. Mais elle ne peut pas dépasser 1 pendant toute itération.
3. rho : le taux d'évaporation de phéromone.
   La valeur de rho peut influencer sur la valeur de tau. Généralement, la valeur de rho est entre 0 et 0.1. Et on doit faire la valeur de rho est plus petite possible.

(a) pour la phéromone déposée sur le nœud de bâtiment i :
Quand on met à jour le tau de phéromone de bâtiment i, il s'agit de la multiplication de population. Parce qu'il y a miller de sans-abris dans certains bâtiments, pour assure que la valeur finale de tau est absolument est inférieur à 1, la valeur de rho est 10^{-5}.

(b) pour la phéromone déposée sur l'arc entre le bâtiment i et le care j :
Quand on met à jour le tau de phéromone de l'arc, il s'agit de la division du produit de distance et population. C'est déjà d'assurer que la valeur de tau est inférieure à 1, donc on peut prendre 0.1 pour le rho.

4. Rayon initial : le rayon d'attraction initial pour chaque care.
Si la distance initiale est petit, il peut prendre plus de temps pour étendre le rayon, mais si la distance initiale est grande, il ne peut pas bien diminuer le nombre de bâtiments à parcourir. Parce que les distances sont tous entre 1000 et 8000, on peut faire l'expérimentation sur 1000, 2000,...,5000. Et on prend le rayon avec lequel le programme exécute le plus rapide.
5. Le nombre de fourmi et le nombre d'itération :
Le nombre de fourmi et le nombre d'itération s'influencent. Si le nombre de fourmi est plus, la meilleure solution trouvée dans une itération est plus fiable. Il a besoin de moins d'itération pour trouver la meilleure solution finale. Au contraire, si le nombre de fourmi n'est pas plus, il faut faire plus d'itération pour trouver la meilleure solution.
Dans ce programme, on fixe le nombre de fourmi et on cherche le nombre d'itération par les expérimentations. Le nombre d'itération dépend aussi de la taille d'échantillon. Si la taille d'échantillon est plus petit, le programme a besoin de moins d'itération.
La solution sera presque stable dans certain nombre d'itération. On peut prendre un nombre plus grand possible comme la fois d'itérations, et après que l'exécution de programme finit, on peut dessiner une figure de qualités de solution pour regarder si le courbe atteint la stabilité, et imprime l'id d'itération où la qualité de solution est maximum. On peut considérer cette solution comme la meilleure solution.
On peut référer l'annexe « Cahier de test » pour regarder l'influence de nombre d'itération et la performance(le temps d'exécution et l'utilisation de mémoire) de cet algorithme.

```
/*Cette fonction est la version initiale pour déterminer les bâtiments à affecter*/
Fonction : allocateBuilding__Initial()
Entrées : instance : l'objet de la classe « InstanceModel »
Sorties : la liste de toutes les solutions générées par chaque fourmi
solutionForOneIterationList ← []    //la variable à retourner contient toutes les solutions générées par chaque fourmi
solutionArray ← [−1] ∗ len(buildingToAllocateList)    //le tableau d'une solution générée par une fourmi

/*chaque fourmi commence à chercher la solution, k est le numéro de fourmi*/
pour k de 0 à amountAnts-1 faire
    /*sélectionner le numéro de bâtiment, step est la fois de pas*/
    pour step de 0 à len(buildingToAllocateList)-1 faire
        /*si c'est le premier pas*/
        si step==0 alors
            buidlingToAllocateIndex ← un nombre aléatoire entre 0 et len(buildingToAllocateList) − 1
            //sélectionner un bâtiment par hasard
        sinon
            buildingProbabilityList ← []    //la liste qui stocke les probabilités de mouvement au nœud de bâtiments dans chaque pas
            buildingIndexForProbabilityList ← []    //la liste qui stocke les indices de bâtiments qui correspondent à chaque probabilité dans la liste « buildingProbabilityList »

            /*calculer la probabilité de mouvement sur les nœuds de bâtiments, i est l'indice de bâtiment*/
            pour i de 0 à len(buildingToAllocateList)-1 faire
                /*si le bâtiment i n'est pas encore sélectionné avant*/
                si isBuildingSelectedList[i] != 1 alors
                    probaNode[i] ← τNode[i]∗ηNode[i] / (1+∑_{l∈{0,...,n}} τNode[i]∗ηNode[i]∗(1−isBuildingSelectedList[l]))
                    //calculer la probabilité de sélectionner le bâtiment i pour le pas prochain
                    Ajouter la probabilité probaNode[i] dans la liste « buildingProbabilityList »
                    Ajouter l'indice de bâtiment i dans la liste « buildingIndexForProbabilityList »
                fin
            fin
            buidlingToAllocateIndex ← generateProbability(buildingIndexForProbabilityList, buildingProbabilityList)    //obtenir l'indice de bâtiment à affecter selon le générateur de probabilité
        fin
        isAllCareFull ← chooseCare(buidlingToAllocateIndex, buildingToAllocateList, careToFillList, solutionArray)    //Affecter une care pour le batiment[buidlingToAllocateIndex] après appeler la fonction « chooseCare() »
        ΔτNode[buidlingToAllocateIndex] ← ρ ∗ g(solutionArray)    //mettre à jour ΔτNode[i], la fonction objective g(x) est le problème de maximization
        τNode[buidlingToAllocateIndex] ← (1 − ρNode) ∗ τNode[buidlingToAllocateIndex] + ΔτNode[buidlingToAllocateIndex]
        //mettre à jour τNode[i]
        isBuildingSelectedList[buidlingToAllocateIndex] ← 1    //mettre à jour isBuildingSelectedList[i], le bâtiment[buidlingToAllocateIndex] est déjà sélectionné

        /*si toutes les cares ne peuvent plus héberger les sans-abris*/
        si isAllCareFull == True alors
            break    //quitter de la boucle
        fin
    fin
    Ajouter « solutionArray » dans la liste « solutionForOneIterationList »
fin
retourner solutionForOneIterationList
```

Algorithme 1 : Le procédure initial d'affecter les bâtiments

```
/*Cette fonction est la version intiale pour déterminer la care à choisir pour le
  bâtiment actuel*/
Fonction : chooseCare__Initial()
Entrées : instance +
          buidlingToAllocateIndex,buildingToAllocateList,careToFillList,solutionArray
Sorties : l'état des cares (si toutes les cares sont pleines ou pas), la solution du pas actuel
careProbabilityList ← []     //la liste qui stocke les probabilités de mouvement
  à l'arc entre le bâtiment et la care dans chaque pas
careIndexForProbabilityList ← []     //la liste qui stocke les numéro de cares
  qui correspondent à chaque probabilité dans la liste «
  careProbabilityList »
allowedCareLenght ← len(careToFillList)     //le nombre de cares qui ne sont pas
  plein, la valeur initiales est égale à len(careToFillList)

/*calculer la probabilité de mouvement sur les arcs entres le bâtiment
  actuel et les cares, j est l'indice de care*/
pour j de 0 à len(careToFillList)-1 faire
    /*si le care j n'est pas plein et le nombre de sans-abris dans le
      bâtiment i est inferieur a sa capacité*/
    si populationBuilding[i] < capacityCare[j] and isCareFullList[j] != 1 alors
        probaEdge[i][j] ← τEdge[i][j]*ηEdge[i][j] / (1+∑_{l∈{0,...,m}} τEdge[i][j]*ηEdge[i][j]*(1−isCareFullList[l]))     //calculer
          la probabilité d'affecter le bâtiment i à la care j pour le pas
          prochain
    fin
fin
careToFillIndex ← generateProbability(careIndexProbabilityList,careProbabilityList)
  //obtenir l'indice de care à remplir selon le générateur de probabilité,
  et mettre à jour la solution
solutionArray[careToFillIndex] ← careToFillIndex     //produire la solution
  partielle
ΔτEdge[i][careToFillIndex] ← ρ/f(solutionArray)     //mettre à jour
  ΔτEdge[i][j], la fonction objective f(x) est le problème de minimisation
τEdge[i][careToFillIndex] ←
  (1 − ρEdge) * τEdge[i][careToFillIndex] + ΔτEdge[i][careToFillIndex]     //mettre à
  jour τEdge[i][j]
capacityCare[careToFillIndex] ← capacityCare[careToFillIndex] − populationBuilding[i]
  //mettre à jour la capacité de care[careToFillIndex]
minPopulation ← le nombre minimum de population parmi les bâtiments non affectés

/*si la capacité de cette care est inférieure au nombre minimum parmi les
  bâtiments non affectés*/
si capacityCare[careToFillIndex] < minPopulation alors
    isCareFullList[careToFillIndex] ← 1     //le care[careToFillIndex] est déjà
      plein
    careAllowedLength−     //diminuer la longueur de la liste de cares qui
      peuvent héberger les sans-abris
fin

/*Aucune care ne peut héberger les sans-abris*/
si careAllowedLength == 0 alors
    retourner True, solutionArray
sinon
    retourner False, solutionArray
fin
```

Algorithme 2 : Le procédure initial de choisir les cares

```
/*Cette fonction est un générateur de probabilité*/
Fonction : generateProbability()
Entrées : idProbabilityList, probabilityList
Sorties : un article dans la liste idProbabilityList lequel correspond à une probabilité
probabilityCompared ← random.uniform(0,1)      //la probabilité à comparer est une valeur aléatoire entre 0 et 1
probabilityCumulative ← 0.0      //la valeur initiale de la probabilité accumulant est 0.0

/*manipuler chaque paire de l'article et sa probabilité correspondant*/
pour item, item_probability dans zip(idProbabilityList, probabilityList) faire
    probabilityCumulative ← probabilityCumulative + item_probability//mettre à jour la probabilité accumulant

    /*//si la probabilité à comparer la probabilité accumulant, quitter la boucle*/
    si probabilityCompared < probabilityCumulative alors
        break
    fin
fin
/*retourner l'article correspond à la probabilité actuelle lors de quitter le boucle*/
retourner item
```

Algorithme 3 : Le pseudocode de générateur de probabilité

```
/*Cette fonction est l'entrée de l'algorithme, et synthétise les solutions générées par
  chaque fourmi dans chaque itération, et obtenir la meilleure solution*/
Fonction : run(iterationTimes)
Entrées : la fois d'itérations
Sorties : la liste de toutes les solutions générées par chaque fourmi
1  distanceSortedBuildingIndexMatrix ← trier la matrice d'indices de bâtiment en référant la
     matrice de distance avec le tri par fusion
2  bestSolutionForEachIterationList ← []     //la liste de meilleure solution de
     chaque itération
3
4  iterationCounter ← 0     //le compteur d'itération
5  /*commencer à faire l'itération*/
6  tant que iterationCounter < iterationTimes faire
7      solutionForOneIterationList ← []     //la liste de solutions d'une itération
8      qualityOfSolutionForOneIterationList ← []     //la liste de qualités de
         solution d'une itération
9      pour k de 0 à len(instance.antList)-1 faire
10         copyDistanceSortedBuildingIndexMatrix ← distanceSortedBuildingIndexMatrix
             //copie la matrice d'indices de bâtiment triée
11         allocateBuilding(ant[k],copyDistanceSortedBuildingIndexMatrix,solutionFor –
             OneIterationList,qualityOfSolutionForOneIterationList)     //sélectionner
             un bâtiment à affecter en appelant la fonction « allocateBuilding()
             »
12     fin
13     bestSolutionIndexForOneIteration ← l'indice de solution dont la valeur de qualité est
         maximum     //on cherche la qualité maximum dans la liste «
         qualityOfSolutionForOneIterationList »
14     Ajouter la qualité de la meilleure solution de l'itération actuelle dans la liste «
         bestQualityOfSolutionForEachIterationList »
15     Ajouter la meilleure solution de l'itération actuelle dans la liste «
         bestSolutionForEachIterationList »
16 fin
17 bestSolutionIndex ←l'indice de solution dont la valeur est maximum dans la liste «
     bestQualityOfSolutionForEachIterationList »     //on cherche la qualité maximum
     dans la liste « bestQualityOfSolutionForEachIterationList »
18 bestSolution ← bestSolutionForEachIterationList[bestSolutionIndex]     //obtenir la
     meilleure solution
```

Algorithme 4 : Le procédure de démarrer l'algorithme

```
/*Cette fonction est pour sélectionner les bâtiments à affecter */
Fonction : allocateBuilding__Improved(ant,copyDistanceSortedBuildingIndexMatrix,
  solutionForOneIterationList,qualityOfSolutionForOneIterationList)
Entrées : une fourmi qui va chercher sa solution,
          la matrice copiée d'indices de bâtiment référant la matrice de distance,
          la liste de solutions pour une itération,
          la liste de qualités de solution pour une itération
Sorties : rien
Construire la liste de candidat pour chaque care en utilisant «
  copyDistanceSortedBuildingIndexMatrix »

step ← 0     //le pas de foumi
careToFillIndexOfLastStep ← −1     //l'indice de care qui est sélectionné dans
  le pas précédent
buildingIndexList ← [indexforindexinrange(0,len(buildingToAllocateList))]     //une
  liste qui stocke juste l'indece de bâtiment, elle sert à sélectionner un
  bâtiment au hasard. (par contre la liste "buidlingToAllocateIndex" stocke
  l'id de bâtiment)
tant que step < len(buildingToAllocateList) faire
    /*si c'est le premier pas ou aucun care est sélectionné dans le pas
     précédent*/
    si step == 0 ou careToFillIndexOfLastStep == -1 alors
        randomNumber ← random.randint(0,len(buildingIndexList) − 1)
         //sélectionner un nombre entité aléatoire)
        buidlingToAllocateIndex ←buildingIndexList[randomNumber]
         //sélectionner un bâtiment au hasard
        Enveler buidlingToAllocateIndex de la liste « buildingIndexList »
    sinon
        si la liste de candidat de care[careToFillIndexOfLastStep] n'est pas vide alors
            Calculer la probabilité de sélection de chaque bâtiment qui est dans la liste de
             candidat et qui n'est pas encore affecté, et construire la liste «
             buildingProbabilityList » et la liste « buildingIndexForProbabilityList »
            buidlingToAllocateIndex ←
             generateProbability(buildingIndexForProbabilityList,buildingProbability−
             List)     //sélectionner un bâtiment en appelant la fonction «
             generateProbability() »
            Enveler buidlingToAllocateIndex de la liste « buildingIndexList »
        sinon
            Re-remplire la liste de candidat de care[careToFillIndexOfLastStep]
            continue
        fin
    fin
    isAllCareFull, careToFillIndex ←
     chooseCare(buidlingToAllocateIndex,buildingToAllocateList,careToFillList,isCa−
     reFullList,ant.solution)     //sélectionner un care pour le
     bâtiment[buidlingToAllocateIndex] en appelant la fonction «
     chooseCare() »

    si careToFillIndex != -1 alors
        Mettre la phéromone déposée sur le nœud de bâtiment[buidlingToAllocateIndex]
        isBuildingSelectedList[buidlingToAllocateIndex] ← True     //marquer que le
         bâtiment[buidlingToAllocateIndex] est déjà sélectionné
    fin

    //si tous les cares sont pleins, cette fourmi trouve sa solution dans
     cette itération
    si isAllCareFull == True alors
        break
    fin
    step ← step + 1
fin
ant.solution.quality ← G(ant.solution)/(1 + F(ant.solution))     //calculer la
  qualité de solution (pour l'autre objectif,quality =
  H(ant.solution)/(1+F(ant.solution))
Ajouter la qualité calculée dans la liste « qualityOfSolutionForOneIterationList »
Ajouter la solution dans la liste « solutionForOneIterationList »
```

Algorithme 5 : Le procédure amélioré d'affecter les bâtiments

```
/*Cette méthode est pour sélectionner les cares à remplir*/
Fonction : chooseCare__Improved(buidlingToAllocateIndex, buildingToAllocateList,
  careToFillList, isCareFullList, solution)
Entrées : l'indice de bâtiment sélectionné,
          la liste de bâtiments,
          la liste de care,
          la liste qui marque si le care est plein,
          une solution
Sorties : une variable booléenne qui signifie si tous les cares sont pleins
          l'indice de care sélectionné
1  allowedCareLength ← len(careToFillList)      //le nombre de cares qui sont encore
     disponibles
2  Calculer la probabilité de sélection de chaque care qui n'est pas encore plein avec la
     construction de la liste « careProbabilityList » et de la liste « careIndexForProbabilityList
     »
3
4      //s'il existe un care qui peut héberger le
     bâtiment[buidlingToAllocateIndex]
5  si carеProbabilityList n'est pas vide alors
6  |   careToFillIndex ←
   |     generateProbability(careIndexForProbabilityList,careProbabilityList)
   |     //sélectionner un care en appelant la fonction « generateProbability()
   |     »
7  |   Mettre à jour la phéromone déposée sur l'arc entre le
   |     bâtiment[buidlingToAllocateIndex] et le care [careToFillIndex]
8  |   Mettre à jour la capacité de care[careToFillIndex]      //capacity = capacity -
   |     population
9  |
10 |   populationList ← []      //la liste de population des bâtiments non-affectés
11 |   Remplir la liste « populationList » parmi les bâtiments non-affectés
12 |   si polulationList est vide alors
13 |   |   retourner True, careToFillIndex
14 |   sinon
15 |   |   minPopulation ← min(populationList)      // chercher la population
   |   |     minimum
16 |   |   si la capacité de care[careToFillIndex] < minPopulation alors
17 |   |   |   isCareFullList[careToFillIndex] ← True      //marquer que ce care est
   |   |   |     déjà plein
18 |   |   |   allowedCareLenght ← allowedCareLenght − 1      //le nombre de cares
   |   |   |     disponibles-1
19 |   |   fin
20 |   fin
21 sinon
22 |   //si tous les cares ne peuvent pas héberger le
   |     bâtiment[buidlingToAllocateIndex]
23 |   careToFillIndex ← −1
24 fin
25 si allowedCareLenght == 0 alors
26 |   //si tous les cares sont pleins
27 |   retourner True, careToFillIndex
28 sinon
29 |   //s'il reste des cares disponibles
30 |   retourner False, careToFillIndex
31 fin
```

Algorithme 6 : Le procédure amélioré de choisir les cares

6 Conclusion

Ce projet se passe en 2 semestres, et le diagramme de Gantt pour gestion de projet est dans l'annexes « Aperçu de gestion de projet ». Pendant le semestre 9, on fait la partie de recherche, et on doit faire la partie de développement pendant le semestre 10.

1 Bilan du semestre 9

Toutes les taches de S9 sont listées dans les cartes de Trello comme Figure1.

Figure 1 – *Les tâche de S9 sur Trello*

1. Les tâches finies Pendant la S9
   Les taches suivantes sont déjà finies :
   - Comprendre le sujet et étudier sa faisabilité ;
   - Analyser les besoins de client ;
   - Identifier le problème et faire la modélisation mathématique ;
   - Présenter le modèle pour les membre de mon groupe ;
   - Etudier les algorithmes existants et écrire l'état de l'arc ;
   - Ecrire le cahier de spécification ;
   - Proposer un nouvel algorithme de fourmis basé sur les algorithmes etudiés ;
   - Ecrire le rapport et préparer la soutenance de S9.
2. La tâche en retard
   Dans l'ensemble, ce projet se passe très bien pendant le semestre 9. Toutes les taches sont finies en date prévue selon le diagramme de Gantt. Néanmoins, le cahier de spécification est modifié plusieurs fois, donc cette tâche est un peu en retard de presque une semaine par rapport à la date prévue.
3. Les tâches à faire :
   - Il faut continuer à améliorer l'algorithme proposé pour augmenter son efficience d'exécution et diminuer ses ressource occupées ;
   - Parce que l'on garde la possibilité d'utiliser NVIDIA Jetson TX1 et CUDA pour aider à améliorer l'efficience d'exécution, donc avant que je commence le développement, je dois étudier de l'implémenter sur Jetson TX1 en Python.

2 Planning du semestre 10

Dans le diagramme de Gantt, on a déjà prévu les tâches et leurs durées pour le semestre 10. Ces taches suivantes sont à faire pendant le S10 :

- Concevoir l'interface visuelle du logiciel ;
- Faire la modélisation logiciel telle que la diagramme de classe ;
- Programmer
- Tester et améliorer la performance du logiciel
- Ecrire le rapport et préparer la soutenance de S10.

Les descriptions détaillées des tâches sont dans l'annexes « Découpage des tâches ».

3 Bilan du semestre 10

1. Les tâches finies Pendant la S10
   Les taches suivantes sont déjà finies :
   - Etudier la faisabilité d'utiliser CUDA et NVIDIA Jetson TX1 ;
   - Faire la modélisation logiciel ;
   - Réaliser l'algorithme proposé la S9 ;
   - Améliorer l'algorithme pour élever la performance ;
   - Faire des tests ;
   - Ecrire le cahier de développeur et le cahier de test ;
   - Ecrire le rapport et préparer la soutenance de S10.
2. La tâche en retard
   Parce que la taille d'échantillon de ce projet est très grande, je met beaucoup d'effort à faire l'expérimentation et améliorer l'algorithme. Donc le cycle de test est en retard de presque 2 semaines par rapport à la date prévue. J'ai commencé à faire le test de mi-mars.
3. La tâche à faire :

- Il faut continuer à améliorer l'algorithme proposé pour diminuer le temps d'exécution.

4 Bilan sur la qualité

Pour la qualité d'algorithme, ce n'est pas très bon pour l'instant. Même si on a diminué le temps d'exécution de 30s pour une fourmi et une itération pour un échantillon avec 500 bâtiments et 187 cares (cela prend plus de 2 minutes avant les améliorations), mais le temps aussi très long. Le problèmes consiste à beaucoup d'aspects. Par exemple, parce que le passage de paramètres entre les méthodes est par référence en Python, si on ne veut pas changer la valeur de variable originale, il faut faire la copie profonde. La copie profonde prend beaucoup de temps. Et pour le programme exécutant sur CPU, la performance de multithread est pire que celle de thread unique. Donc on ne peut pas profiter de l'avantage de multithread.

Je pense que l'on peut ajouter la base de données à l'avenir. Parce qu'il y a beaucoup d'occasion où on a besoin de chercher les éléments dans la liste, ça prend beaucoup de temps et occupe beaucoup de mémoire si la taille de liste est grande. Si on utilise la base de données, la vitesse de cherche sera plus vite.

Pour la qualité de programme, je pense que la qualité est bon, car j'ai bien préparé les documents nécessaires tels que le cahier de spécification, le cahier de développeur, le cahier de test. Et pour la modélisation, grâce aux conseils de professeurs, j'ai fait des améliorations. Pour le code, les commentaires sont détaillés et suffisant.

5 Bilan auto-critique

Pendant cette année, je pense que mon PRD se passe bien. Mon projet est bien avancé, et j'ai eu des résultats pendant chaque période. J'ai rencontré beaucoup de problème, mais grace aux aides de mon encadrant M. MONMARCHE, la pluspart de problèmes sont résolus. Et j'ai appris beaucoup pendant resoudre les problème.

Je voudrais remercie aussi M. RAGOT et M. RAMEL de leurs aides sur ma spécification et ma modélisation de projet. Je trouve beaucoup de choses que je n'ai pas fait attention auparavant. Et j'ai appris beaucoup de techniques pratiques.

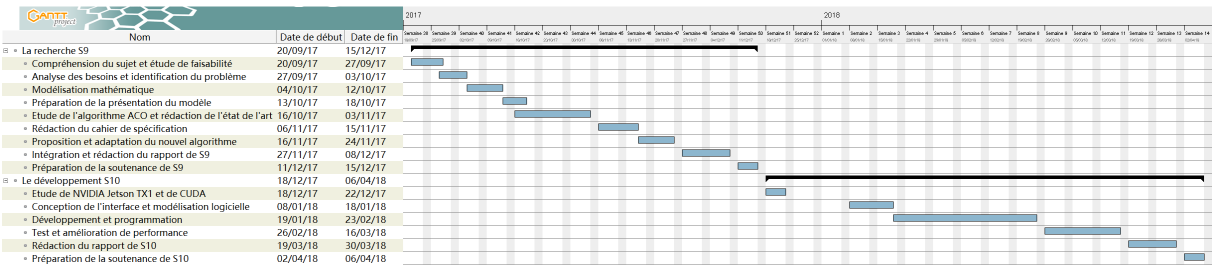
Annexes

A Planification

En respectant le modèle en « cascade », on fait 3 cycles : l'étude de faisabilité, la spécification des besoins, et l'analyse pendant le semestre 9, et on fait les autres 4 cycles pendant le semestre 10 : la conception détaillée, l'implémentation, le test et la mise en place.

1 Aperçu de gestion de projet

Le diagramme de Gantt pour la planification de ce projet est comme Figure1.



Nom	Date de début	Date de fin
La recherche S9	20/09/17	15/12/17
Compréhension du sujet et étude de faisabilité	20/09/17	27/09/17
Analyse des besoins et identification du problème	27/09/17	03/10/17
Modélisation mathématique	04/10/17	12/10/17
Préparation de la présentation du modèle	13/10/17	18/10/17
Etude de l'algorithme ACO et rédaction de l'état de l'art	16/10/17	03/11/17
Rédaction du cahier de spécification	06/11/17	15/11/17
Proposition et adaptation du nouvel algorithme	16/11/17	24/11/17
Intégration et rédaction du rapport de S9	27/11/17	08/12/17
Préparation de la soutenance de S9	11/12/17	15/12/17
Le développement S10	18/12/17	06/04/18
Etude de NVIDIA Jetson TX1 et de CUDA	18/12/17	22/12/17
Conception de l'interface et modélisation logicielle	08/01/18	18/01/18
Développement et programmation	19/01/18	23/02/18
Test et amélioration de performance	26/02/18	16/03/18
Rédaction du rapport de S10	19/03/18	30/03/18
Préparation de la soutenance de S10	02/04/18	06/04/18

Figure 1 – *Le diagramme de Gantt*

La Figure2est pour plus clairement afficher les tâches.

Pour le S9, le projet commence le 20/19/2017 par la compréhension du sujet et finit le 15/12/2017 par la soutenence de S9. Pour le S10, le projet commence le 19/12/2017 en continuant à étudier Jetson TX1 et CUDA et le projet finit le 04/06/2018 par la soutenence de S10. En moyen, chaque tâche dure 1 ou 2 semaines, mais pour la programmation et le test, ils prandront environ 4 ou 5 semaines.

2 Découpage des tâches

1. Compréhension du sujet et étude de faisabilité :

Nom	Date de début	Date de fin
La recherche S9	20/09/17	15/12/17
Compréhension du sujet et étude de faisabilité	20/09/17	27/09/17
Analyse des besoins et identification du problème	27/09/17	03/10/17
Modélisation mathématique	04/10/17	12/10/17
Préparation de la présentation du modèle	13/10/17	18/10/17
Etude de l'algorithme ACO et rédaction de l'état de l'art	16/10/17	03/11/17
Rédaction du cahier de spécification	06/11/17	15/11/17
Proposition et adaptation du nouvel algorithme	16/11/17	24/11/17
Intégration et rédaction du rapport de S9	27/11/17	08/12/17
Préparation de la soutenance de S9	11/12/17	15/12/17
Le développement S10	18/12/17	06/04/18
Etude de NVIDIA Jetson TX1 et de CUDA	18/12/17	22/12/17
Conception de l'interface et modélisation logicielle	08/01/18	18/01/18
Développement et programmation	19/01/18	23/02/18
Test et amélioration de performance	26/02/18	16/03/18
Rédaction du rapport de S10	19/03/18	30/03/18
Préparation de la soutenance de S10	02/04/18	06/04/18

Figure 2 – *Les tâches dans le diagramme de Gantt*

- Date de début : le 20 septembre 2017 ;
- Date de fin : le 27 septembre 2017 ;
- Durée : 6 jours ;
- Description : Il faut comprendre le sujet, connaitre l'objectif et étudier la faisabilité technique.

2. Analyse des besoins et identification du problème :
   - Date de début : le 27 septembre 2017 ;
   - Date de fin : le 3 octobre 2017 ;
   - Durée : 5 jours ;
   - Description :
     - Il faut débrouiller les besoins de client, y compris les données que il peut nous fournit et le résultat espéré ;
     - Il faut aussi étudier les problèmes d'optimisation combinatoire pour trouver un problème classique qui ressemble à notre problème.
3. Modélisation mathématique :
   - Date de début : le 4 octobre 2017 ;
   - Date de fin : le 12 octobre 2017 ;
   - Durée : 7 jours ;
   - Description : Il faut faire la modélisation mathématique pour notre projet.
4. Préparation de présentation du modèle :
   - Date de début : le 13 octobre 2017 ;
   - Date de fin : le 18 octobre 2017 ;
   - Durée : 4 jours ;
   - Description :

- Il faut modifier le modèle que on a fait si nécessaire ;
- Il faut préparer une présentation sur le modèle au professeur et les étudiants en DAE qui sont relatives à ce projet.

5. Etude de l'algorithme ACO et rédaction de l'état de l'art :
   - Date de début : le 16 octobre 2017 ;
   - Date de fin : le 03 novembre 2017 ;
   - Durée : 15 jours ;
   - Description :
     - Il faut apprendre l'algorithme de colonies de fourmis et son application sur le problème du sac à dos ;
     - Il faut écrire le rapport de l'état de l'art qui contient les problèmes classiques étudiés, les algorithmes existants, etc.
6. Rédaction du cahier de spécification :
   - Date de début : le 6 novembre 2017 ;
   - Date de fin : le 15 novembre 2017 ;
   - Durée : 8 jours ;
   - Description : Il faut écrire le cahier de spécification qui contient la présentation de structure, les interfaces, les fonctionnalités et les non-fonctionnalités.
7. Proposition et adaptation de nouvel algorithme :
   - Date de début : le 16 novembre 2017 ;
   - Date de fin : le 24 novembre 2017 ;
   - Durée : 7 jours ;
   - Description : Il faut proposer un nouvel algorithme basé sur l'algorithme de colonies de fourmis, qui peut être appliqué à notre projet.
8. Intégration et rédaction du rapport de S9 :
   - Date de début : le 27 novembre 2017 ;
   - Date de fin : le 8 décembre 2017 ;
   - Durée : 8 jours ;
   - Description : Il faut rédiger le rapport de S9 en intégrant tous les documents écrit pendant S9.
9. Préparation de soutenance de S9 :
   - Date de début : le 11 décembre 2017 ;
   - Date de fin : le 15 décembre 2017 ;
   - Durée : 5 jours ;
   - Description : Il faut faire le ppt pour la soutenance de S9 et passer la soutenance.
10. Etude de NVIDIA Jetson TX1 et CUDA :
    - Date de début : le 18 décembre 2017 ;
    - Date de fin : le 22 décembre 2017 ;
    - Durée : 5 jours ;
    - Description : Il faut apprendre et étudier l'utilisation de Jeston TX1 et vérifier si CUDA est compatible de Python.
11. Conception de l'interface et modélisation logicielle :
    - Date de début : le 8 janvier 2018 ;
    - Date de fin : le 18 janvier 2018 ;
    - Durée : 9 jours ;
    - Description :
      - Il faut une interface visuelle pour notre logiciel ;
      - Il faut faire la modélisation logicielle en utilisant UML telle que le diagramme de classe.
12. Développement et programmation :
    - Date de début : le 19 janvier 2018 ;
    - Date de fin : le 23 février 2018 ;

- Durée : 26 jours ;
- Description : Il faut faire la programmation pour réaliser ce logiciel avec l'algorithme que l'on a proposé.

13. Test et amélioration de performance :
    - Date de début : le 26 février 2018 ;
    - Date de fin : le 16 mars 2018 ;
    - Durée : 15 jours ;
    - Description :
        - Il faut vérifier si le résultat que le logiciel trouve correspond aux besoins de client ;
        - Il faut tester le logiciel que l'on réalise, et détecter les exceptions possibles ;
        - Il faut améliorer la performance de logiciel, par exemple, diminuer le temps d'exécutions.
14. Rédaction du rapport de S10 :
    - Date de début : le 19 mars 2018 ;
    - Date de fin : le 30 mars 2018 ;
    - Durée : 10 jours ;
    - Description : Il faut écrire le rapport de S10, ajouter la partie de développement dans le rapport de S9.
15. préparation de soutenance de S10 :
    - Date de début : le 2 avril 2018 ;
    - Date de fin : le 6 avril 2018 ;
    - Durée : 5 jours ;
    - Description : Il faut faire le ppt pour la soutenance de S10 et passer la soutenance.

B Description des interfaces externes du logiciel

1 Interfaces matériel/logiciel

On réserve la possibilité d'utiliser « NVIDIA Jetson TX1 »(Figure1).« NVIDIA Jetson TX1 » est un supercalculateur intégré sur un module. Il sert à améliorer la performance et l'efficience de l'application.

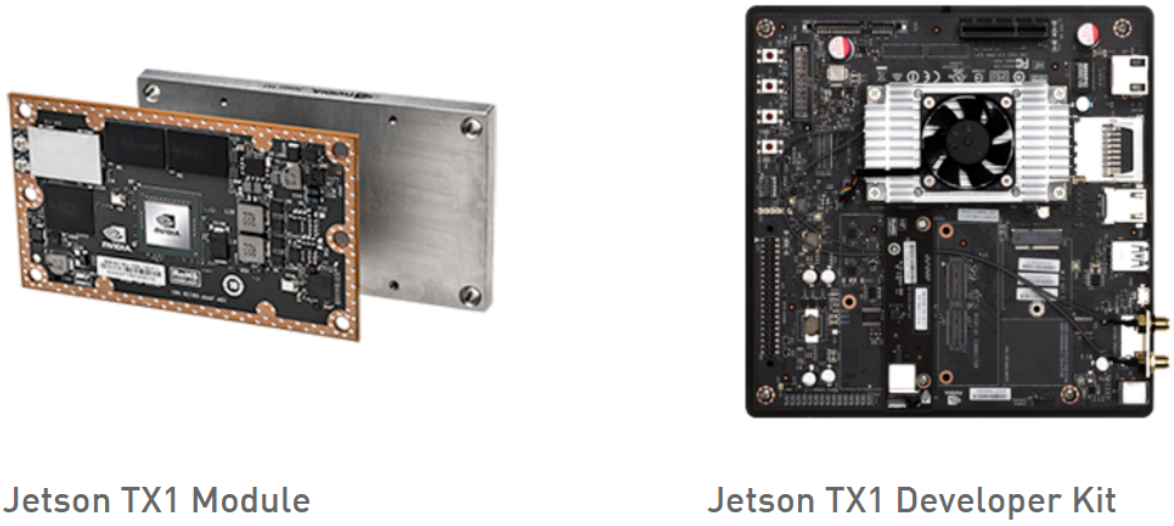


Figure 1 – *NVIDIA Jetson TX1*

NVIDIA Jetson TX1 est développé en l'architecture NAVIDIA Maxwell™ avec 256 cœurs CUDA et CPU 64 bit. Il adopte des techniques novelles sur le deep learning, la vision par ordinateur, le calcul intensif par le GPU et la visualisation graphique, qui le fait être très pratique pour le secteur de l'embedded.[WWW1]

CUDA est une plateforme de calcul parallèle et un modèle d'API, qui nous permet d'utiliser un GPU pour exécuter des calculs généraux. Elle fonctionne généralement avec la langage C ou C++, et c'est plus facile pour les développeurs d'utiliser les ressources de GPU.

Parce que la taille de données dans notre projet sera très grande, et quand le système fait l'analyse, il effectuera beaucoup d'itérations. Donc peut-être on a besoin de les utiliser pour élever la performance du logiciel et diminuer le temps d'exécution.

2 Interfaces homme/machine

Vu l'ergonomies du système, on conçoit des messages d'erreurs en cas de :

- la connexion :
  - le nom de l'utilisateur n'existe pas ;
  - le mot de passe n'est pas correct.
- l'inscription :
  - le nom de l'utilisateur est déjà inscrit ;
  - le mot de passe à nouveau ne correspond pas au mot de passe.
- l'importation des fichiers de données : les fichiers « batiments.txt », « cares.txt » et « distances.txt » que l'utilisateur choisit ne se correspondent pas.

Quand l'utilisateur s'inscrit, les champs suivants sont au moins demandés :

- le nom de l'utilisateur ;
- le mot de passe ;
- le mot de passe à nouveau ;
- le sexe ;
- etc.

Et quand l'utilisateur se connecte, l'utilisateur doit saisir le nom de l'utilisateur et le mot de passe.

Lors que l'utilisateur importe les données, il doit choisir 3 fichiers :

- Le fichier « batiment.txt » : contient les informations de tous les bâtiments. Les colonnes que l'on est le numéro de bâtiment et le nombre de population de chaque bâtiment ;
- Le fichier « cares.txt » : contient les informations de tous les centres d'accueils. Le système extraira deux colonnes : le numéro de centre d'accueil et la capacité de chaque centre d'accueil ;
- Le fichier « distances.txt » : enregistre toutes les distances entre chaque bâtiment et chaque centre d'accueil.

3 Interfaces logiciel/logiciel

Selon les besoins d'utilisateur, ce logiciel n'utilise pas la base de données. Il lit les données à partir des fichiers textes et exporte aussi le résultat au fichier texte. Donc ce logiciel ne dépend d'aucun autre logiciel.

Mais, un logiciel qui s'appelle « ArcGIS » coopère avec notre logiciel. Il fournit des outils contextuels pour la cartographie et le raisonnement spatial afin que l'on puisse comprendrez et exploser les données de manière de la localisation.

Après on obtient la meilleure solution par notre logiciel, le résultat sera enregistré dans un fichier texte. Et ce fichier texte peut être lu par « ArcGIS », « ArcGIS » nous montrera une carte visualisé basée sur le résultat de ce projet comme Figure2.

Dans la Figure2, une couleur représente un ensemble de bâtiments qui doivent être affectés au même care, et le numéro est l'identifiant de care. Cette carte aide l'utilisateur à faire la décision avec le résultat obtenu par notre logiciel.

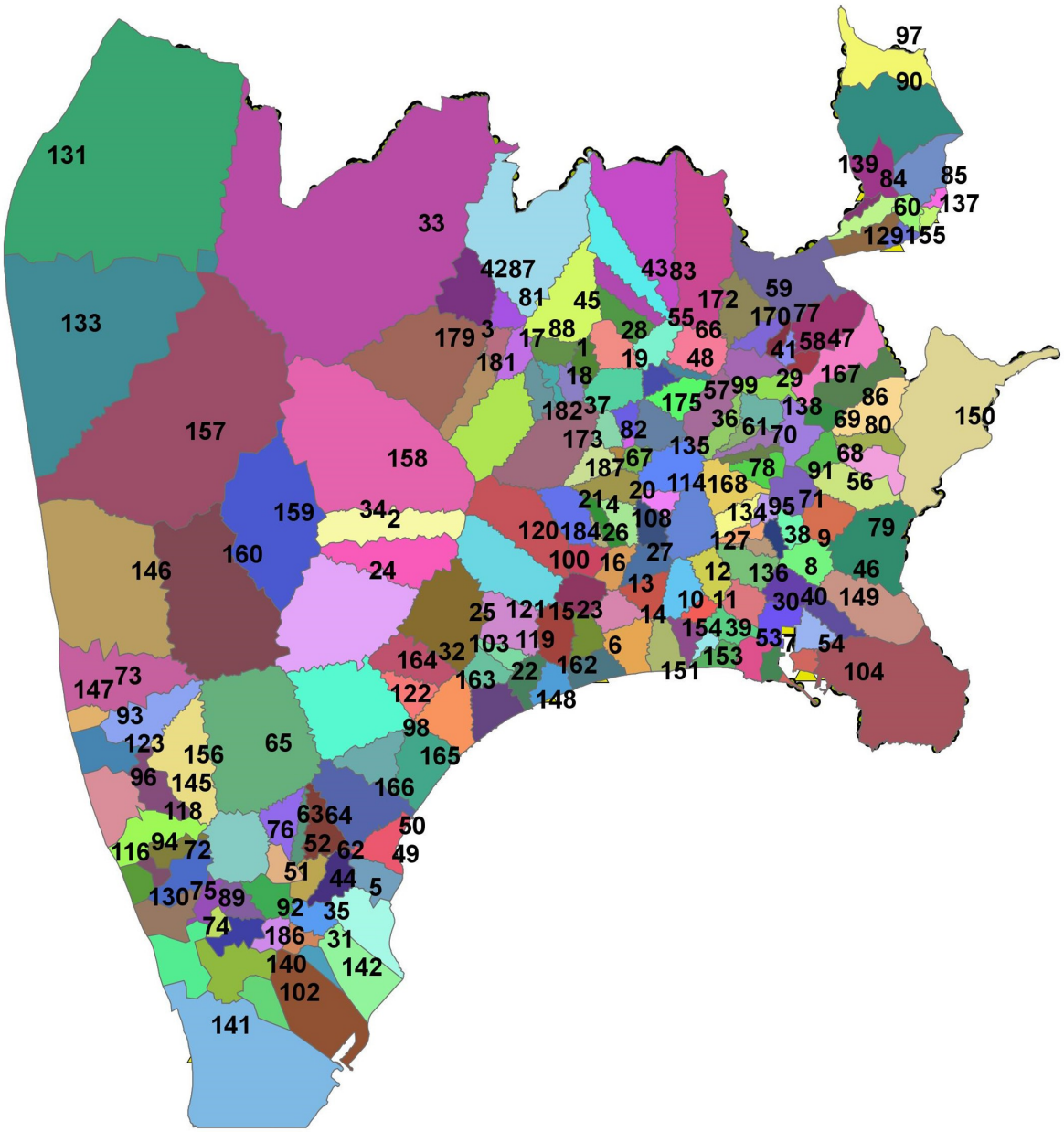


Figure 2 – *La carte dessinée par « ArcGIS »*

C Spécifications fonctionnelles

On conçoit 3 fonctions majeures pour ce projet. Ces 3 fonctions sont promaires et facultatives.

1 Fonction de l'importation

1. Présentation
   - Nom : l'importation ;
   - Rôle : cette fonction sert à l'importer les fichiers de données (bâtiments, cares, distances) au système ;
   - Priorité : c'est la fonction primordiale et facultative.
2. Description
   Avant que le système commence à analyser, l'utilisateur doit importer les fichiers de données au système. Elle correspond au composant « Procédure du chargement de données ». Il y a 3 fichiers à importer : Le fichier de bâtiment, le fichier de care, le fichier de distance entre les bâtiments et les cares(les centres d'accueil). Tous les fichiers doivent être en fichier texte. Après l'importation, le système doit extraire les colonnes de besoin à partir de 3 fichiers et les garder dans les listes ou la matrice.
   - Entrée : les données lues des 3 fichiers texte :
     - dans la fichier « batiments.txt », chaque ligne contient le numéro de bâtiment, le nombre de population de chaque bâtiment, et les coordonnées de chaque bâtiment ;
     - dans la fichier « cares.txt », chaque ligne contient le numéro de centre d'accueil, la capacité de chaque centre d'accueil, et les coordonnées de chaque centre d'accueil ;
     - dans la fichier « distances.txt », chaque ligne contient les distances entre un bâtiment et tous les centres d'accueil.
   - Sortie : les données extraites :
     - la listes de bâtiments qui contient la paire [idBat, population] ;
     - la listes de care qui contient la paire [idCare, capacité] ;
     - la matrice de distance entre chaque bâtiment et chaque care
   - Cette fonction correspond au composant « Procédure du chargement de données ». Il fournit une iterface de service « Importer les fichiers de données » à l'utilisateur

pour importer les 3 fichiers de données. Et il doit fournir les données extraites au composant « Procédure de l'analyse » ;

- Traitement : le traitement de cette fonction est présenté dans l'organigramme de programmation comme Figure1.

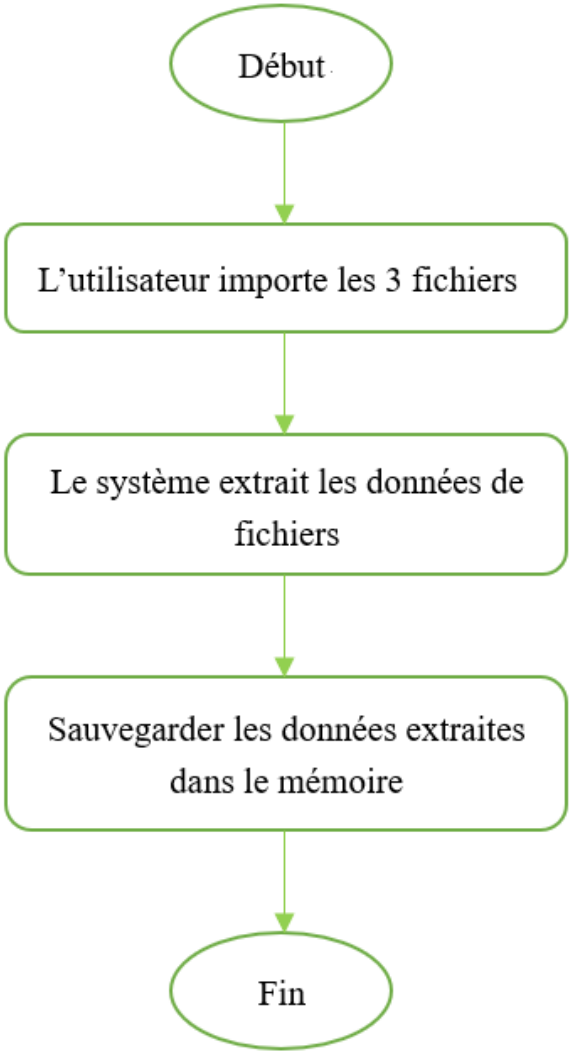


Figure 1 – *L'organigramme de la fonction « importation »*

- Exception : les 3 fichiers que l'utilisateur choisit ne se correspondent pas.

2 Fonction de l'analyse

1. Présentation
   - Nom : l'analyse ;
   - Rôle : cette fonction sert à analyser les données et trouver la meilleure solution d'affectation ;
   - Priorité : c'est la fonction primordiale et facultative.
2. Description
   Cette fonction est la fonction la plus importante. Elle utilise les données extraites par la fonction de l'importation avec l'algorithme de colonies de fourmis à trouver des meilleures solutions d'affectation de sans-abris. Les solutions correspondent respectivement aux objectifs décrits dans le chapitre 1 « Objectifs ».

- Entrée : les données extraites par la fonction de l'importation ;
- Sortie : les solutions correspondant aux objectifs ;
- Cette fonction correspond au composant « Procédure de l'analyse ». Elle a besoin d'obtenir les données extraites dans le composant « Procédure du chargement de données » par l'interface de demande « Demander les données », et elle passera le résultat au composant « Procédure de l'écriture de résultat » ;
- Traitement : le traitement de cette fonction est présenté dans l'organigramme de programmation comme Figure2.

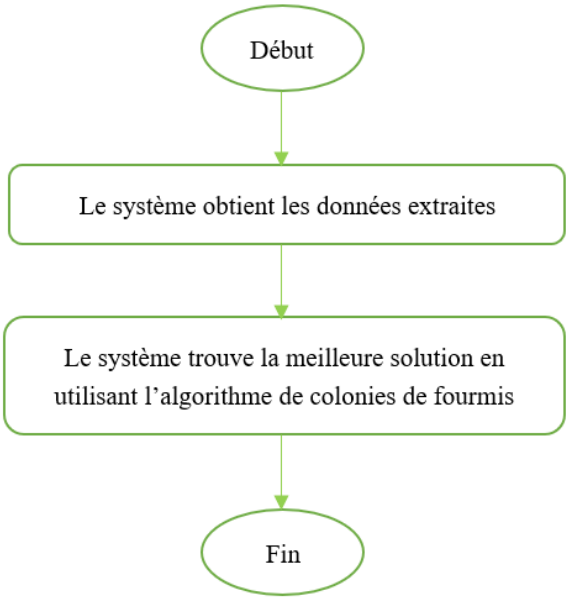


Figure 2 – *L'organigramme de la fonction « analyse »*

- Exception : rien.

3 Fonction de l'écriture au fichier

1. Présentation
   - Nom : l'écriture au fichier ;
   - Rôle : cette fonction sert à écrire la meilleure solution au fichier texte ;
   - Priorité : c'est la fonction primordiale et facultative.
2. Description
   Cette fonction doit réaliser l'écriture au fichier texte pour enregistrer la meilleure solution trouvée par la fonction de l'analyse.
   - Entrée : la meilleure solution trouvée par la fonction de l'analyse ;
   - Sortie : les solutions enregistrées dans un fichiers texte.
   - Cette fonction correspond au composant « Procédure de l'écriture de résultat ». Elle a besoin d'obtenir les solutions trouvées dans le composant « Procédure de l'analyse » par l'interface de demande « Demander le résultat d'analyse », et elle enregistrer au fichier texte par l'interface de service « Ecrire le résultat dans les fichier » ;
   - Traitement : le traitement de cette fonction est présenté dans l'organigramme de programmation comme Figure3.
   - Exception : rien.

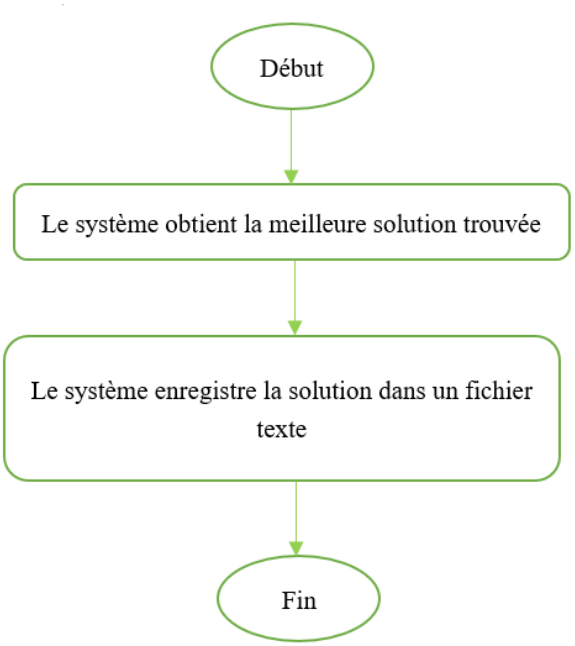


Figure 3 – *L'organigramme de la fonction « écriture au fichier »*

D Spécifications non fonctionnelles

1 Contraintes de développement et conception

- Langages : Python3.6 ;
- Plateforme optionnelle : CUDA ;

2 Contraintes de fonctionnement et d'exploitation

Cette section décrit les contraintes de fonctionnement, qui contiennent 4 enjeux. Ce sont les performances, les capacités, les contrôlabilités et la sécurité.

2.1 Performances

- Du point de vue de l'utilisateur : parce qu'il aura beaucoup d'itérations dans le programme, on prévoit que le temps de réponse souhaité est de 25 secondes en moyenne.
- Du point de vue de l'environnement : ce logiciel ne dépend pas du système d'exploitation, car il est réalisé en Python, qui est un langage multi-plateforme.

2.2 Capacités

Pour assurer le résultat est synchrone, c'est-à-dire pour éviter l'incohérence entre les deux résultats sur les mêmes fichiers avant et après modifier certain fichier, on limite que le nombre max de terminaux est de 1.

- Le nombre de fichiers log stockés est limité à 30 (pour un mois).
- La taille max des données traitées est de 60000*300 (60000 bâtiments, 300 centres d'accueil)

2.3 Contrôlabilité

- Pour contrôler l'accès de l'utilisateur, on peut concevoir des fichiers log qui sont nommés par la date actuelle. Lorsqu'un utilisateur se connecte et fait des opérations, le système enregistrera la date, le temps, l'identifiant et le nom de l'utilisateur, les opérations, les fichiers qu'il a consultés, etc.
- Pour contrôler la fiabilité de résultat d'analyse, on peut concevoir une sous interface qui affiche toutes les solutions générer par chaque fourmi. Cela permet l'utilisateur de manuellement vérifier le résultat s'il veut.

2.4 Sécurité

Le système demande à l'utilisateur de se connecter avant de l'utiliser, mais parce qu'il n'y a qu'un type d'utilisateur, l'utilisateur peut faire toutes les opérations une fois il se connecte au système avec succès.

E Cahier du développeur

1 Aperçu de la diagramme de classe entier

La diagramme de classe entier est comme Figure1.

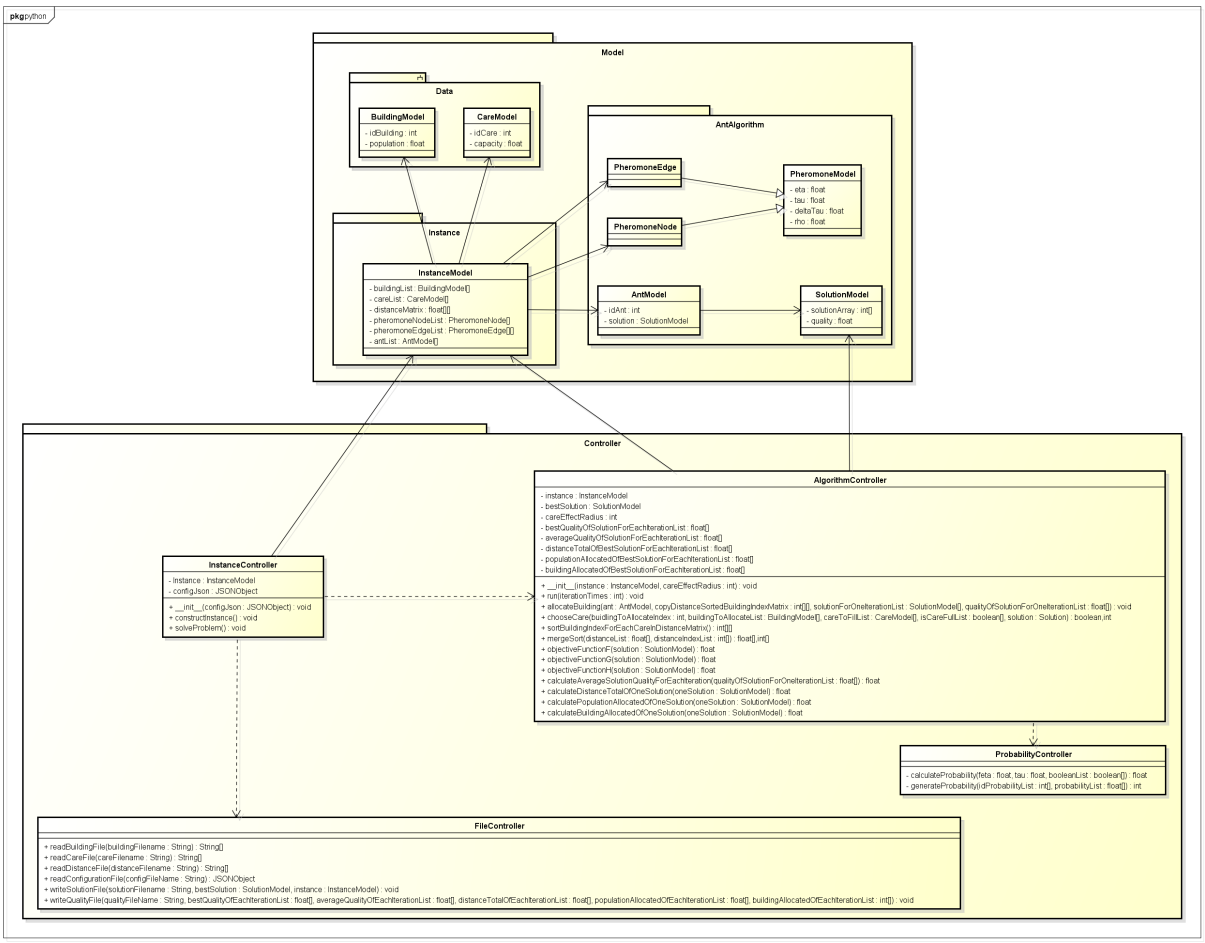


Figure 1 – *Le diagramme de classe entier*

2 Descriptions détaillées des classes du package « modèle »

- Package « Data »
  Il y a deux classes dans le package « Data » : « BuildingModel » et « CareModel ».
  1. Classe « BuildingModel » : Elle contient son identifiant et son nombre de sans-abris.
     - ⋆ Attributs :
       (a) idBuilding : (int) l'identifiant de bâtiment ;
       (b) population : (float) le nombre de sans-abris dans le bâtiment.
  2. Classe « CareModel » : cette classe est modèle de centre d'accueil. Elle contient son identifiant et sa capacité.
     - ⋆ Attributs :
       (a) idCare : (int) l'identifiant de centre d'accueil ;
       (b) capacity : (float) la capacité de centre d'accueil.
- Package « AntAlgorithm »
  Il y a cinq classes dans le package « AntAlgorithm » : « SolutionModel », « AntModel », « PhermoneModel », « PheromoneNode » et « PheromoneEdge ». Les classes « PheromoneNode » et « PheromoneEdge » héritent de la classe « PhermoneModel ».
  1. Classe « SolutionModel » : cette classe est le modèle de solution construite par chaque fourmi. Elle contient un tableau de solution et la qualité de solution.
     - ⋆ Attributs :
       (a) solutionArray : (int[]) le tableau d'une solution ;
       (b) quality : (float) la qualité de solution.
  2. Classe « AntModel » : cette classe est le modèle de fourmi. Chaque fourmi a un identifiant et un objet de solution.
     - ⋆ Attributs :
       (a) idAnt : (int) l'identifiant de fourmi ;
       (b) solution : (SolutionModel) la solution trouvée de fourmi.
  3. Classe « PhermoneModel » : cette classe est le modèle de phéromone. Il contient toutes les variables que l'algorithme de colonie de fourmis demande.
     - ⋆ Attributs :
       (a) eta : (float) la désirabilité de mouvement ;
       (b) tau : (float) la quantité de phéromones existant ;
       (c) deltaTau : (float) la quantité de phéromones déposés lors que la fourmi prend un nœud ou un arc ;
       (d) rho : (float) le taux d'évaporation de phéromones.
  4. Classe « PhermoneNode » : cette classe est le modèle de phéromone qui est déposée sur les nœuds de bâtiment.
     - ⋆ Attributs :
       Elle hérite tous les attributs de classe parent « PhermoneModel ».
  5. Classe « PhermoneEdge » : cette classe est le modèle de phéromone qui est déposée sur les arcs entre le bâtiment et le care.
     - ⋆ Attributs :
       Elle hérite tous les attributs de classe parent « PhermoneModel ».
- Package « Instance »
  Il y a deux classes dans le package « Instance » : « InstanceModel ».
  1. Classe « InstanceModel » : cette class est le modèle d'instance pour ce projet, y compris la liste de bâtiments, la liste de cares, la liste de phéromones de nœud, la matrice de phéromones d'arc, la matrice de distances et la liste de fourmis.
     - ⋆ Attributs :
       (a) buildingList : (BuildingModel[]) la liste de bâtiments ;
       (b) careList : (CareList[]) la liste de cares ;

(c) distanceMatrix : (float[][]) la matrice de distances entre chaque bâtiment et chaque care ;
(d) pheromoneNodeList : (PheromoneNode[]) la liste de phéromones déposée sur le nœud de bâtiment ;
(e) pheromoneEdgeMatrix : (PheromoneEdge[][]) la matrice de phéromones déposée sur l'arc entre le bâtiment et le case ;
(f) antList : (AntModel[]) la liste de fourmis.

3 Descriptions détaillées des classes du package « contrôleur »

1. Classe « InstanceController » : cette classe est pour construire l'instance de projet et fournir un service de demarrer l'algorithme.
   - ★ Attributs :
     (a) instance : (InstanceModel) l'instance pour ce projet ;
     (b) configJson : (l'objet de JSON) les valeurs des paramètres configurées par l'utilisateur.
   - ★ Méthodes :
     (a) __init()__ : cette méthode est le constructeur de la classe « InstanceController ».
       - ♣ Paramètre :
         - configJson : (l'objet de JSON) les valeurs des paramètres lues du fichier « configParameters.json ».
       - ♣ Valeur retournée : rien.
     (b) constructInstance() : cette méthode est pour construire l'instance de projet.
       - ♣ Paramètre : rien.
       - ♣ Valeur retournée : rien.
     (c) solveProblem() : cette méthode fournit la service de résoudre le problème.
       - ♣ Paramètre : rien.
       - ♣ Valeur retournée : rien.
2. Classe « FileController » : cette classe sert à faire les opérations sur les fichiers, y compris la lecture et l'écriture.
   - ★ Attribut : rien.
   - ★ Méthodes :
     (a) readBuildingFile() : cette méthode est pour lire le fichier de bâtiment.
       - ♣ Paramètre :
         - buildingFilename : (String) le nom du fichier de bâtiment.
       - ♣ Valeur retournée :
         - buildingContentList : (String[]) la liste de lignes du fichier de bâtiment.
     (b) readCareFile() : cette méthode est pour lire le fichier de centre d'accueil(care).
       - ♣ Paramètre :
         - careFilename : (String) le nom du fichier care.
       - ♣ Valeur retournée :
         - careContentList : (String[]) la liste de lignes du fichier de care.
     (c) readDistanceFile() : cette méthode est pour lire le fichier de distance.
       - ♣ Paramètre :
         - distanceFilename : (String) le nom du fichier distance.
       - ♣ Valeur retournée :
         - distanceContentList : (String[]) la liste de lignes du fichier de distance.
     (d) readConfigurationFile() : cette méthode est pour lire le fichier de configuration des paramètres définies par l'utilisateur.
       - ♣ Paramètre :
         - configFileName : (String) le nom du fichier configuration des paramètres.

- ♣ Valeur retournée :
  - configJson : (l'objet de JSON) les valeurs des paramètres configurées par l'utilisateur

(e) writeSolutionFile() : cette méthode est pour écrire la meilleure solution dans le fichier de solution.
- ♣ Paramètres :
  - solutionFilename : (String) le nom du fichier de solution ;
  - bestSolution : (SolutionModel) la meilleure solution ;
  - instance : (InstanceModel) l'instance pour le programme.
- ♣ Valeur retournée : rien.

(f) writeQualityFile() : cette méthode est pour écrire les qualités des meilleures solutions de chaque itération et les qualités moyennes des solutions de chaque itération, la distance totale et le nombre de sans-abris hébergés ainsi que le nombre de bâtiments affectés de chaque itération dans le fichier de qualité.
- ♣ Paramètres :
  - bestQualityOfEachIterationList : (float[]) la liste de qualités des meilleures solutions de chaque itération ;
  - averageQualityOfEachIterationList : (float[]) la liste de qualités moyennes de solutions de chaque itération ;
  - distanceTotalOfEachIterationList : (float[]) la liste de distance totale de la meilleure solution de chaque itération ;
  - populationAllocatedOfEachIterationList : (float[]) la liste de sans-abris totaux hébergés de la meilleure solution de chaque itération ;
  - buildingAllocatedOfEachIterationList : (float[]) la liste de nombre de bâtiments affectés de la meilleure solution de chaque itération.
- ♣ Valeur retournée : rien.

3. Classe « ProbabilityController » : cette classe est pour contrôler la probabilité lorsque les fourmis déplacent.
   - ★ Attribut : rien.
   - ★ Méthodes :

   (a) calculateProbability() : cette méthode est pour calculer la probabilité de déplacement.
   - ♣ Paramètres :
     - (float) la désirabilité de déplacement ;
     - tau : (float) la quantité de phéromones existant sur un nœud de bâtiment ou un arc entre un bâtiment et un care ;
     - booleanList : (boolean[]) soit la liste qui marque si le bâtiment est déjà affecté, soit la liste qui marque si le care est plein.
   - ♣ Valeur retournée :
     - probability : (float) la probabilité de déplacement calculée.

   (b) generateProbability() : cette méthode est pour sélectionner un événement qui correspond à une probabilité au hasard.
   - ♣ Paramètres :
     - idProbabilityList : (int[]) la séquence qui contient les identifiants de bâtiment ou de care ;
     - probabilityList : (float[]) la liste de probabilité de déplacement.
   - ♣ Valeur retournée :
     - item : (int) l'identifiant de l'article(bâtiment ou care) sélectionné au hasard selon les probabilités.

4. Classe « AlgorithmController » : cette classe est le contrôleur d'algorithme, qui réalise à chercher la meilleure solution d'affection.
   - ★ Attributs :

(a) instance : (InstanceModel) l'instance préparée par la classe « InstanceControleur », y compris la liste de bâtiments, la liste de cares, la liste de phéromones sur les nœuds de bâtiment, la matrice de phéromones sur les arcs entre le bâtiment et le care, la liste de fourmis ;
(b) bestSolution : (SolutionModel) la meilleure solution trouvé finalement ;
(c) careEffectRadius : (int) le rayon d'attraction initial de cercle dont le centre est chaque care ;
(d) bestQualityOfSolutionForEachIterationList : (float[]) la liste de qualités de meilleure solution de chaque itération ;
(e) averageQualityOfSolutionForEachIterationList : (float[]) la liste de qualités moyenne des soltuons de chaque itération ;
(f) distanceTotalOfBestSolutionForEachIterationList : (float[]) la liste de distance totale de la meilleure solution de chaque itération ;
(g) populationAllocatedOfBestSolutionForEachIterationList : (float[]) la liste de sans-abris totaux hébergés de la meilleure solution de chaque itération ;
(h) buildingAllocatedOfBestSolutionForEachIterationList : (float[]) la liste de nombre de bâtiments affectés de la meilleure solution de chaque itération.

★ Méthodes :

(a) __init()__ : cette méthode est le constructeur de la classe « AlgorithmController ».
  ♣ Paramètres :
    - instance : (l'objet de la classe InstanceModel) l'instance de programme, qui est préparée par la classe « InstanceControleur » ;
    - careEffectRadius : (int) le rayon d'attraction initial de chaque care, qui est définit par l'utilisateur dans le fichier « configParameters.json ».
  ♣ Valeur retournée : rien.
(b) run() : cette méthode est l'entrée de l'algorithme, et synthétise les solutions générée par chaque fourmi dans chaque itération, et obtenir la meilleure solution.
  ♣ Paramètre :
    - iterationTimes : (int) le nombre d'itérations.
  ♣ Valeur retournée : rien.
(c) allocateBuilding() : cette méthode est pour sélectionner les bâtiments à affecter.
  ♣ Paramètres :
    - ant : (l'objet de la classe AntModel) un fourmi qui va chercher sa solution ;
    - copyDistanceSortedBuildingIndexMatrix : (int[][]) la matrice copiée d'indices de bâtiment référée la matrice de distance ;
    - solutionForOneIterationList : (SolutionModel[]) la liste de solutions pour une itération ;
    - qualityOfSolutionForOneIterationList : (float[]) la liste de qualités de solution pour une itération.
  ♣ Valeur retournée : rien.
(d) chooseCare() : cette méthode est pour sélectionner les cares à remplir.
  ♣ Paramètres :
    - buidlingToAllocateIndex : (int) l'indice de bâtiment sélectionné ;
    - buildingToAllocateList : (BuildingModel[]) la liste de bâtiments ;
    - careToFillList : (CareModel[]) la liste de care ;
    - isCareFullList : (Boolean[]) la liste qui marque si le care est plein ;
    - solution : (l'objet de la classe SolutionModel) une solution partielle.
  ♣ Valeurs retournées :
    - isAllCareFull : (boolean) une variable booléen qui signifie si tous les cares sont pleins ;
    - careToFillIndex : (int) l'indice de care sélectionné.

(e) sortBuildingIndexForEachCareInDistanceMatrix() : cette méthode est pour trier les indices de bâtiments pour chaque care en référant la matrice de distance.
- ♣ Paramètre : rien.
- ♣ Valeur retournée :
  - distanceSortedBuildingIndexMatrix : (int[][]) la matrice de indices de bâtiments triés.

(f) mergeSort() : cette méthode est pour trier une liste d'indice de bâtiments en référant la matrice de distance avec le tri par fusion.
- ♣ Paramètres :
  - distanceList : (float[]) la liste de distance à référer ;
  - distanceIndexList : (int[]) la liste d'indice de bâtiment à trier.
- ♣ Valeurs retournées :
  - result : (float[]) la liste de distances triées ;
  - resultIndex : (int[]) la liste d'indices de bâtiments triées.

(g) objectiveFunctionF() : cette méthode est pour réaliser la fonction objective f(x).
- ♣ Paramètre :
  - solution : (SolutionModel) une solution.
- ♣ Valeur retournée :
  - fx : (float) la valeur calculée de f(x).

(h) objectiveFunctionG() : cette méthode est pour réaliser la fonction objective g(x).
- ♣ Paramètre :
  - solution : (SolutionModel) une solution.
- ♣ Valeur retournée :
  - gx : (float) la valeur calculée de g(x).

(i) objectiveFunctionH() : cette méthode est pour réaliser la fonction objective h(x).
- ♣ Paramètre :
  - solution : (SolutionModel) une solution.
- ♣ Valeur retournée :
  - hx : (float) la valeur calculée de h(x).

(j) calculateAverageSolutionQualityForEachIteration() : cette méthode est pour calculer la qualité moyenne des solutions générées par chaque fourmi dans une itération.
- ♣ Paramètre :
  - qualityOfSolutionForOneIterationList : (float[]) la liste de qualités de chaque solution d'une itération.
- ♣ Valeur retournée :
  - average : (float) la qualité moyenne des solutions.

(k) calculateDistanceTotalOfOneSolution() : cette méthode est pour calculer la distance totale d'une solution.
- ♣ Paramètre :
  - oneSolution : (SolutionModel) une solution.
- ♣ Valeur retournée :
  - distanceTotal : (float) la distance totale calculée d'une solution.

(l) calculatePopulationAllocatedOfOneSolution() : cette méthode est pour calculer le nombre de sans-abris hébergés d'une solution.
- ♣ Paramètre :
  - oneSolution : (SolutionModel) une solution.
- ♣ Valeur retournée :
  - populationAllocated : (float) le nombre de sans-abris hébergés d'une solution.

(m) calculateBuildingAllocatedOfOneSolution() : cette méthode est pour calculer le nombre de bâtiments affectés d'une solution.

♣ Paramètre :
- oneSolution : (SolutionModel) une solution.

♣ Valeur retournée :
- buildingAllocated : (int) le nombre de bâtiments affectés d'une solution.

4 Structure des fichiers utilisés

Pour ce projet, on a besoin d'utiliser 6 fichiers : le fichier de configuration des paramètres, le fichier de bâtiment, le fichier de care, le fichier de distance, le fichier de meilleure solution, le fichier de qualité des solutions.

Le fichier de configuration des paramètres est en format JSON, et les autres fichiers sont en les fichiers de texte.

4.1 Fichiers d'entrée

1. Le fichier de configuration des paramètres (fichier JSON)
   Le structure de fichier de bâtiment est comme Figure2.
   Il y a 6 éléments dans ce fichier JSON :

```
{
    "antQuantity": 50,
    "iterationTimes": 10,
    "careEffectRadius": 3000,
    "inputFiles": {
        "buildingFileName": "files/Rq22_51760B_TriCrOID_TriNSACr4.txt",
        "careFileName": "files/Rq33_187CareMoveID188.txt",
        "distanceFileName": "files/LOD9679120_IdNet_NSACr3.txt"
    },
    "outputFiles": {
        "solutionFileName": "files/bestSolution.txt",
        "qualityFileName": "files/quality.txt"
    },
    "pheromone": {
        "rhoNode": 0.000001,
        "tauNode": 0.8,
        "rhoEdge": 0.1,
        "tauEdge": 0.9
    }
}
```

Figure 2 – *La structure de configuration des paramètres*

- « antQuantity » : le nombre de fourmis ;
- « iterationTimes » : la fois d'itération ;
- « careEffectRadius » : le rayon d'attraction initial de care ;
- « inputFiles » : les noms de fichiers d'entrée :
  - « buildingFileName » : le nom du fichier de bâtiment ;

- « careFileName » : le nom du fichier de care ;
- « distanceFileName » : le nom du fichier de distance ;

- « outFiles » : les noms de fichiers de sortie :
  - « solutionFileName » : le nom du fichier de solution ;
  - « QualityFileName » : le nom du fichier de qualité ;
- « pheromone » : les paramètres sur la phéromone de l'algorithme de colonie de fourmis :
  - « rhoNode » : la taux d'évaporation de phéromone déposé sur les nœuds de bâtiment ;
  - « tauNode » : la quantité initiale de phéromone déposé sur les nœuds de bâtiment ;
  - « rhoEdge » : la taux d'évaporation de phéromone déposé sur les arcs entre les bâtiment et les cares ;
  - « tauEdge » : la quantité initiale de phéromone déposé sur les arcs entre les bâtiment et les cares.

2. Le fichier de bâtiment (fichier texte)
   Le structure de fichier de bâtiment est comme Figure3.
   Il y a 4 colonnes dans ce fichier :

```
1  #""ObjectID"""  X_Centr Y_Centr "NSA"
2  53437    1043175.81   6299445.70   24.45
3  3209     1044803.30   6300964.75   29.76
4  48993    1043153.68   6298401.79   29.84
5  2800     1044833.11   6299050.67   36.14
```

Figure 3 – *Le structure de fichier de bâtiment*

- Colonne 1 : l'identifiant de bâtiment ;
- Colonne 2 : la coordonnée de x ;
- Colonne 3 : la coordonnée de y ;
- Colonne 4 : le nombre de sans-abris du bâtiment (population).

3. Le fichier de care (fichier texte)
   Le structure de fichier de care est comme Figure4.
   Il y a 4 colonnes dans ce fichier :

```
1  #"ObjectID" "Centr_X"    "Centr_Y"    "Cap_CARE"
2  290 1042149.20   6297687.27   500
3  291 1045806.22   6297698.00   1641
4  292 1043681.37   6300153.48   150
5  293 1043769.51   6300232.59   563
```

Figure 4 – *Le structure de fichier de care*

- Colonne 1 : l'identifiant de care ;
- Colonne 2 : la coordonnée de x ;
- Colonne 3 : la coordonnée de y ;
- Colonne 4 : la capacité du care.

4. Le fichier de distance (fichier texte)
   Le structure de fichier de distance est comme Figure5.
   Il y a 4 colonnes dans ce fichier :

1	."ORI_ID" "DEST_ID" "TOTAL_METE" "NSA"
2	53437 290 2447.52 24.45
3	53437 291 3752.35 24.45
4	53437 292 1189.58 24.45
5	53437 293 1723.30 24.45

Figure 5 – *Le structure de fichier de distance*

- Colonne 1 : l'identifiant de bâtiment ;
- Colonne 2 : l'identifiant de care ;
- Colonne 3 : la distance entre le bâtiment et le care ;
- Colonne 4 : le nombre de sans-abris du bâtiment.

4.2 Fichiers de sortie

1. Le fichier de solution (fichier texte)
   Le structure de fichier de solution est comme Figure6.
   Il y a 2 colonnes dans ce fichier :

1	ID_BAT	ID_CARE
2	53437	308
3	3209	305
4	48993	308
5	2800	298

Figure 6 – *Le structure de fichier de solution*

- Colonne 1 : l'identifiant de bâtiment ;
- Colonne 2 : l'identifiant de care où le bâtiment est affecté.

2. Le fichier de qualité (fichier texte)
   Le structure de fichier de qualité est comme Figure7.
   Il y a 6 colonnes dans ce fichier :
   - Colonne 1 : l'identifiant d'itérations ;
   - Colonne 2 : la qualité de meilleure solution ;
   - Colonne 3 : la qualité moyenne des solutions ;
   - Colonne 4 : la distance totale parcourue ;
   - Colonne 5 : la nombre de sans-abris hébergés ;
   - Colonne 6 : le nombre de bâtiments affectés.

```
1  ID_Iteration    Best_Quality    Average_Quality Distance_Total  Population_Quantity Building_Quantity
2  1   0.00031646655985713163  0.00029773789516677655  178124.4    13387.31    49
3  2   0.0003142726713454576   0.0003005503078383915   172504.30000000002  13387.31    49
4  3   0.00031990892857623537  0.00030123933245248543  179689.18   14112.09    49
5  4   0.0003113589639050869   0.0002990594558674062   192196.09999999998  13387.31    49
```

Figure 7 – *Le structure de fichier de qualité*

5 Structure de projet

La fichier de configuration des paramètres est dans le répertoire « configuration ». Les autres fichiers d'entrée et de sortie doivent être dans le répertoire « files ». Le structure de code respecte complètement au diagramme de classe. La structure de projet est comme Figure8.

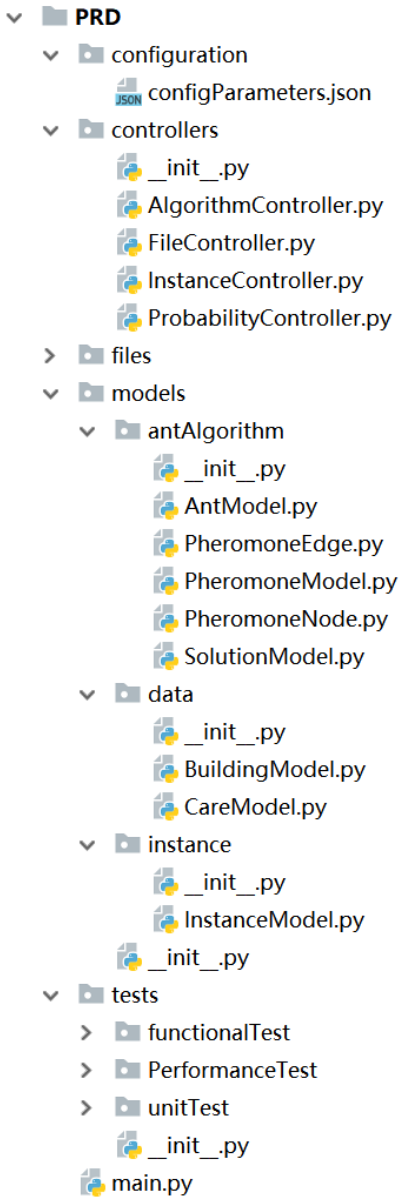


Figure 8 – *Le structure de projet*

F Document d'installation des librairies

Dans ce projet, le développement ne dépend d'aucune librairie externe. Mais le test dépend de 3 libraires :

1. matplotlib : pour dessiner la figure de points de bâtiments et points de cares ;
   - ★ Commande d'installation :
     pip install matplotlib
2. line__profiler : pour test le temps d'exécution d'une méthode ou d'une fonction ligne par ligne ;
   - ★ Commande d'installation :
     pip install line__profiler
   - ★ Commande d'utilisation :
     kernprof -l -v [le chemin de script Python à tester]
3. memory__profiler : pour test l'occupation de mémoire d'une méthode ou d'une fonction ligne par ligne.
   - ★ Commande d'installation :
     pip install memory__profiler
     pip install psutil
   - ★ Commande d'utilisation :
     python -m memory__profiler [le chemin de script Python à tester]

Pour utiliser ligne__profiler et memory__profiler dans le console, il faut ajouter le chemin de notre projet dans la variable d'environnement « PYTHONPATH » pour le système Windows.

G Document d'utilisation

Quand on réalise le logiciel, on a pensé à la modification des paramètres importantes par l'utilisateur ou par l'autre développeur. Donc on conçoit un fichier de configuration de paramètres en format JSON dans le répertoire « configuration ». Si on veut modifier les valeurs de paramètre, il suffit modifier dans le fichier de configuration au lieu de changer le code. Ainsi, il est plus facile pour le développement et la maintenance.

L'utilisation de ce logiciel est très simple. Quand l'utilisateur utilise ce logiciel, il faut assurer que les 3 fichiers(bâtiment, care, distance) respecte absolument aux leurs structures correspondantes. Les structures de ces 3 fichiers sont décrites dans l'annexe « Structure des fichiers utilisés ».

Pour indiquer les chemins de fichiers et modifier les valeurs de paramètre, il suffit de mofifier le fichier « configParamters.json » sans modifier le code. La meilleure solution est stockés dans a fichier « solution.txt » par défaut.

H Cahier de test

1 Introduction

1.1 Objectif de test

Le test logiciel est pour assurer la justesse, l'intégrité, la sécurité et la performance de logiciel. Pour ce projet, les tests servent à vérifier si l'algorithme proposé est correct et pratique, et vérifier si les modules de programme réalisé peuvent bien fonctionner. Par exemple, on peut savoir si le module de sélection de care peut bien sélectionner un centre d'accueil pour un bâtiment par le test. En plus, on peut constater si la sortie réelle correspond bien à la sortie attendue.

1.2 Pré-requis

1.2.1 Typologie de test

Les tests pour ce projet contiennent les 3 types décrits dans le Tableau1.

Table 1 – *Les Types de test mis en pratique*

ID	Type de test	Description
1	Test unitaire	Le test unitaire est pour vérifier l'entrée et la sortie de chaque méthode dans les classes en utilisant des cas de test.
2	Test fonctionnel	Le test fonctionnel est pour comparer le résultat obtenu avec le résultat attendu.
3	Test de performance	Le test de performance est pour constater le temps d'exécution et l'occupation de mémoire de certaines méthodes afin d'améliorer la performance de programme et d'algorithme.

1.2.2 Eléments à tester

Le Tableau2 montre les éléments à tester. Les éléments sont séparés en deux types : les classe de contrôleur et les fichiers obtenus.

Table 2 – *Les élément à tester*

ID	Type d'élément	Nom d'élément	Description
1	<Classe>	FileController	Le contrôleur est le contrôleur de fichier, y compris la lecture et l'écriture
2	<Classe>	InstanceController	Le contrôleur d'instance de projet, y compris les bâtiments, les centres d'accueil, les distances entre les bâtiments et les centres d'accueil, les fourmis et les deux types de phéromone.
3	<Classe>	ProbabilityController	Le contrôler de la probabilité de bâtiments ou de cares lorsque les fourmis déplacent.
4	<Classe>	AlgorithmController	Le contrôleur d'algorithme, qui réalise à chercher la meilleure solution d'affection.
5	<Fichier>	solution.txt	Le fichier de sortie qui stocke la meilleure solution
6	<Fichier>	quality.txt	Le fichier de sortie qui stocke les qualités des meilleures solutions et les qualités moyenne des solutions, la distance totale, le nombre total de sans-abris hébergés et le nombre total de bâtiments affectés de chaque itération.

2 Tests unitaires

Le test unitaire est pour tester chaque fonction ou chaque méthode de la classe. On donne des données d'entrée à la méthode à tester, et on a un résultat attendu d'avance. On constate si le résultat de sortie de méthode après l'exécution avec les données d'entrée est bien correspondant à notre résultat attendu. Si les deux résultats se correspondent bien, ça signifie que la méthode est correcte. Sinon, on doit traiter les défauts de la méthode. La Figure1 montre un petit exemple de test unitaire pour ce projet.

La langage Python fournit une librairie qui s'appelle « unittest » pour facilement faire le test unitaire. Il suffit que la classe de test hérite la classe « unittest.TestCase » pour faire le test unitaire avec des cas de test. Cette libraire nous fournit beaucoup de méthodes d'assertion, par exemples, la méthode « assertTrue() » est pour vérifier si l'expression est vrai, la méthode « assertEqual() » est pour vérifier si les deux variables sont égales, la méthode « assertListEqual() » est pour vérifier si les deux listes sont pareilles, etc. Si la réponse d'assertion est positive, le test réussit. Sinon, il générera l'exception. Enfin, il suffit une phrase « unittest.main() » pour démarrer le test. Une fois cette phrase exécute, toutes les méthodes dans cette classe seront testées.

La Figure2 montre le résultat de test unitaire avec succès.

La figure2 signifie que l'on a testé deux méthodes dans cette classe, et tous les tests réussit. Si le test échoue, le résultat est comme Figure3.

Dans la figure3, il affiche que deux méthodes sont testées, mais un test génère l'exception car le résultat de sortie n'est pas égal au résultat prévu.

```python
class calculateProbability(unittest.TestCase):

    def test_CalculateProbability(self):
        pc = ProbabilityController()
        tau = 0.8
        eta = 1/4000
        booleanList = [False, True, True, False, True]
        sum = 0
        for i, booleanElement in enumerate(booleanList):
            sum += tau * eta * (1 - booleanElement)
        probability = (tau * eta) / (1 + sum)
        self.assertEqual(pc.calculateProbability(eta, tau, booleanList), probability)


if __name__ == '__main__':
    unittest.main()
```

Figure 1 – *L'exemple de test unitaire avec la librairie « unittest »*

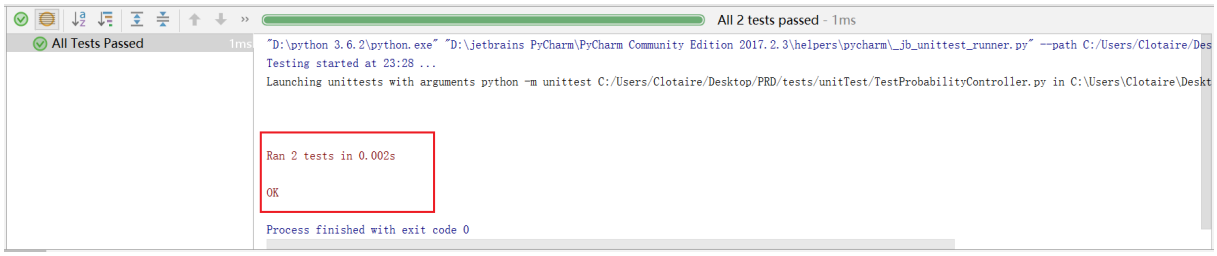


Figure 2 – *Le résultat de test unitaire avec succès*

Figure 3 – *Le résultat de test unitaire échoué*

2.1 Tests unitaires de la classe « FileController »

Les tests unitaires des méthodes de la classe « FileController » sont listés dans le Tableau3.

Table 3 – *Les tests unitaires des méthodes de la classe « FileController »*

Test unitaire de la classe « FileController »		
ID	Article	Contenu
1	Nom de méthode à tester	readBuildingFile()
	Description	Cette méthode est pour lire le fichier de bâtiment.
	Données d'entrée	Le nom fichier texte de bâtiment avec 51 lignes (la première ligne est le titre)
	Sortie attendue	La taille de la liste qui enregistre chaque ligne du fichier de bâtiment (sauf la première liste) est 50
	Résultat réel	50
2	Nom de méthode à tester	readCareFile()
	Description	Cette méthode est pour lire le fichier de care (centre d'accueil).
	Données d'entrée	Le nom fichier texte de care avec 21 lignes (la première ligne est le titre)
	Sortie attendue	La taille de la liste qui enregistre chaque ligne du fichier de care (sauf la première liste) est 20
	Résultat réel	20
3	Nom de méthode à tester	readDistanceFile()
	Description	Cette méthode est pour lire le fichier de distance.
	Données d'entrée	Le nom fichier texte de distance avec 501 lignes (la première ligne est le titre)
	Sortie attendue	La taille de la liste qui enregistre chaque ligne du fichier de distance (sauf la première liste) est 500
	Résultat réel	500
4	Nom de méthode à tester	readConfigurationFile()
	Description	Cette méthode est pour lire le fichier de configuration de paramètre.
	Données d'entrée	Le nom du fichier configuration des paramètres.
	Sortie attendue	Un objet de JSON qui contient tous les éléments de paramètres demandés par le programme
	Résultat réel	Vrai
5	Nom de méthode à tester	writeSolutionFile()
	Description	Cette méthode est pour écrire la meilleure solution dans le fichier de solution.
	Données d'entrée	1) Le nom du fichier de solution
		2) La meilleure solution : [4,5,6,-1,3]
		3) L'instance : les ids de bâtiments sont : [0,1,2,3,4], les identifiants de cares sont : [0,1,2,3,4,5,6]
	Sortie attendue	Un fichier texte qui stocke la meilleure solution, ce fichier a 6 lignes (la première ligne est le titre) et 2 colonne, le nombre de ligne est égal à la taille de meilleure solution+1.
		La première colonne est l'identifiant de bâtiment, la deuxième colonne est l'identifiant de care
	Résultat réel	OK
	Nom de méthode à tester	writeQualityFile()

6	Description	Cette méthode est pour écrire les qualités des meilleures solutions de chaque itération et les qualités moyennes des solutions de chaque itération, la distance totale et le nombre de sans-abris hébergés ainsi que le nombre de bâtiments affectés de chaque itération dans le fichier de qualité
	Données d'entrée	1) La liste de qualités des meilleures solutions de chaque itération : [0.1477832512315271,0.09865470852017937]
		2) La liste de qualités moyennes de solutions de chaque itération : [0.07858062847465425, 0.05374583753472586]
		3) La liste de distance totale de la meilleure solution de chaque itération : [100,200]
		4) La liste de sans-abris totaux hébergés de la meilleure solution de chaque itération : [25,30]
		La liste de nombre de bâtiments affectés de la meilleure solution de chaque itération : [4,5]
	Sortie attendue	Un fichier texte qui stocke les données d'entrée, ce fichier a 3 lignes (la première ligne est le titre) et 6 colonne, le nombre de ligne est égal à la taille de la liste de qualités des meilleures solutions+1, chaque ligne est pour une itération.
		La première colonne est l'id d'itération, a deuxième colonne est la qualité de meilleure solution, la troisième colonne est la qualité moyenne des solutions, la quatrième colonne est la distance totale de la meilleure solution, la cinquième colonne est le nombre de sans-abris hébergés de la meilleure solution, la dernière colonne est le nombre de bâtiments affectés de la meilleure solution.
	Résultat réel	OK

2.2 Tests unitaires de la classe « ProbabilityController »

Les tests unitaires des méthodes de la classe « ProbabilityController » sont listés dans le Tableau]4.

Table 4 – *Les tests unitaires des méthodes de la classe « ProbabilityController »*

Test unitaire de la classe « ProbabilityController »		
ID	Article	Contenu
1	Nom de méthode à tester	calculateProbability()
	Description	Cette méthode est pour calculer la probabilité de déplacement.
	Données d'entrée	1) La désirabilité de déplacement : 1/4000
		2) La quantité de phéromones existant : 0.8
		3) Une liste booléenne : [False, True, True, False, True]
	Sortie attendue	0.00019992
	Résultat réel	0.00019992
	Nom de méthode à tester	generateProbability()
	Description	Cette méthode est pour sélectionner un événement qui correspond à une probabilité au hasard
		1) La séquence qui contient les identifiants : [0,1,2,3]

2	Données d'entrée	2) La liste de probabilité de déplacement : [0.15,0.26,0.58,0.73]
	Sortie attendue	Après 100 itérations, les occurrences de 0, 1, 2 et 3 ne sont pas tous 100. (Parce qu'elle utilise le nombre aléatoire, on ne peut pas prévoir un résultat concret).
	Résultat réel	Vrai

2.3 Tests unitaires de la classe « InstanceController »

Les tests unitaires des méthodes de la classe « InstanceController » sont listés dans le Tableau5.

Table 5 – *Les tests unitaires des méthodes de la classe « InstanceController »*

Test unitaire de la classe «InstanceController »		
ID	Article	Contenu
1	Nom de méthode à tester	constructInstance()
	Description	Cette classe est pour construire l'instance de projet
	Données d'entrée	Un objet de JSON qui contient tous les éléments de paramètres demandés par le programme avec :
		1) Le nombre de fourmis : 10
		2) Le nom fichier texte de bâtiment avec 51 lignes (la première ligne est le titre)
		3) Le nom fichier texte de care avec 21 lignes (la première ligne est le titre)
		4) Le nom fichier texte de distance avec 501 lignes (la première ligne est le titre)
	Sortie attendue	Une instance est construite, dont la taille de son attribut « buildingList » est 50 et la taille de son attribut « careList » est 20 et la taille de son attribut « distanceMatrix» est 50 et le nombre de colonne de « distanceMatrix» est 20 et la taille de son attribut « antList » est 10
	Résultat réel	Vrai
2	Nom de méthode à tester	solveProblem()
	Description	Cette méthode fournit le service de résoudre le problème, elle n'a pas fonction réelle.
	Données d'entrée	Un objet de JSON qui contient tous les éléments de paramètres demandés par le programme avec :
		1) La fois d'itération : 200
		2) Le rayon d'attraction initial pour les cares : 3000m
		3) Le nom du fichier texte sortie de solution
		4) Le nom du fichier texte sortie de qualité
	Sortie attendue	Elle peut appeler la méthode « run() » de la classe « AlgorithmController » et les méthodes «writeSolutionFile() » et « writeQualityFile() » avec succès, et afficher des messages.
	Résultat réel	OK

2.4 Tests unitaires de la classe « AlgorithmController »

Pour faire les tests unitaires pour cette classe, on crée une instance globale pour toutes les méthodes de test d'avance : les valeurs de ses attributs sont :

- instance.distanceMatrix = [[10, 20, 30, 15, 5, 20, 30],[30, 20, 15, 20, 15, 10, 5],[20, 30, 10, 25, 5, 20, 10],[40, 10, 5, 30, 25, 15, 10],[30, 20, 30, 5, 10, 15, 20]];
- instance.buildingList.idBuilding = [0,1,2,3,4]
  instance.buildingList.population = [9, 5, 16, 27, 30];
- instance.careList.idCare = [0,1,2,3,4,5,6]
  instance.careList.capacity = [8, 15, 6, 34, 13, 10, 18];
- instance.pheromoneNode.eta = instance.buildingList.population
  instance.pheromoneNode.rho = [0.000001]*5
  instance.pheromoneNode.tau = 0.8;
- instance.pheromoneEdge.eta = 1/instance.distanceMatrix
  instance.pheromoneEdge.rho = [[0.1] * 7] *5
  instance.pheromoneEdge.tau = 0.9;
- instance.antList = [AntModel(), AntModel(), AntModel(), AntModel(), AntModel()]
  instance.antList.solution.solutionArray = [-1,-1,-1,-1,-1]

Les tests unitaires des méthodes de la classe « AlgorithmController » sont listés dans le Tableau6.

Table 6 – *Les tests unitaires des méthodes de la classe « AlgorithmController »*

Test unitaire de la classe « AlgorithmController »		
ID	Article	Contenu
1	Nom de méthode à tester	mergeSort()
	Description	Cette méthode est pour trier une liste d'indice de bâtiments en référant la matrice de distance avec la trie par fusion
	Données d'entrée	1) La liste de distance de care à référer (une colonne de la matrice de distance) : [2,56,3,22]
		2) La liste d'indice de bâtiment à trier : [0,1,2,3]
	Sortie attendue	[0,2,3,1]
	Résultat réel	[0,2,3,1]
2	Nom de méthode à tester	sortBuildingIndexForEachCareInDistanceMatrix()
	Description	Cette méthode est pour trier les indices de bâtiments pour chaque care en référant la matrice de distance, elle dépend de la méthode « merge_sort() »
	Données d'entrée	1) La matrice de distance à référer : [[2,4,5,3,1],[56,29,3,43,11], [3,54,67,13,4], [22,51,78,32,45]]
		2) La matrice d'indice à trier : [[0,1,2,3],[0,1,2,3],[0,1,2,3],[0,1,2,3],[0,1,2,3]]
	Sortie attendue	[[0,2,3,1],[0,1,3,2],[1,0,2,3],[0,2,3,1],[0,2,1,3]]
	Résultat réel	[[0,2,3,1],[0,1,3,2],[1,0,2,3],[0,2,3,1],[0,2,1,3]]
3	Nom de méthode à tester	objectiveFunctionF()
	Description	Cette méthode est pour réaliser la fonction objective f(x)
	Données d'entrée	1) Une solution : [4,6,4,2,3]
		2) L'instance dessus
	Sortie attendue	435
	Résultat réel	435

4	Nom de méthode à tester	objectiveFunctionG()
	Description	Cette méthode est pour réaliser la fonction objective g(x)
	Données d'entrée	1) Une solution : [4,6,4,2,3]
		2) L'instance dessus
	Sortie attendue	87
	Résultat réel	87
5	Nom de méthode à tester	objectiveFunctionH()
	Description	Cette méthode est pour réaliser la fonction objective h(x)
	Données d'entrée	1) Une solution : [4,6,4,2,3]
		2) L'instance dessus
	Sortie attendue	5
	Résultat réel	5
6	Nom de méthode à tester	calculateAverageSolutionQualityForEachIteration()
	Description	Cette méthode est pour calculer la qualité moyenne des solutions générées par chaque fourmi dans une itération
	Données d'entrée	1) la liste de qualités de chaque solution d'une itération : [0.5, 0.4, 0.6,0.3, 0.2]
		2) L'instance dessus
	Sortie attendue	0.4
	Résultat réel	0.4
7	Nom de méthode à tester	chooseCare()
	Description	Cette méthode est pour sélectionner les cares à remplir
	Données d'entrée	1) La solution initiale : [-1, -1, -1, -1, -1]
		2) La liste qui marque si le care est plein : [False, False, False, False, False, False, False]
		3) L'instance dessus
		4) La liste copiée de bâtiments à affecter : buildingToAllocateList = copy.deepcopy(instance.buildingList)
		5) La liste copiée de care à remplir : careToFillList = copy.deepcopy(instance.careList)
	Sortie attendue	Après 100 itération, pour le bâtiment dont l'indice est 4, il est affecté au care 3 à l'équilibre
	Résultat réel	Vrai
8	Nom de méthode à tester	allocateBuilding()
	Description	Cette méthode est pour sélectionner les bâtiments à affecter
	Données d'entrée	1) L'instance dessus
		2) La matrice d'indices de bâtiments triées : générée par la méthode « sortBuildingIndexForEachCareInDistanceMatrix() »
		3) La liste de solutions initiale pour une itération : []
		4) La liste de qualités de solution initiale pour une itération : []
	Sortie attendue	Après l'exécution d'une fois, la taille de liste de qualité est 1 et la valeur n'est pas 0, et la solution n'est plus [-1,-1,-1,-1,-1]
	Résultat réel	Vrai
	Nom de méthode à tester	run()

9	Description	Cette méthode est l'entrée de l'algorithme, et synthétise les solutions génèrée par chaque fourmi dans chaque itération, et obtenir la meilleure solution
	Données d'entrée	1) L'instance dessus
		2) Le nombre d'itérations : 300
	Sortie attendue	Après 300 itération, afficher la meilleure solution, la population totale hébergé, la distance totale, la qualité de solutions (parce qu'il y a beaucoup de contraintes pour déterminer une solution, c'est difficile de prévoir la meilleure solution d'avance)
	Résultat réel	[4, 6, -1, -1, 3], 44, 15, 2.75
10	Nom de méthode à tester	calculateDistanceTotalOfOneSolution()
	Description	Cette méthode est pour calculer la distance totale d'une solution
	Données d'entrée	1) L'instance dessus
		2) Une solution : [4,6,-1,-1,3]
	Sortie attendue	15
	Résultat réel	15
11	Nom de méthode à tester	calculatePopulationAllocatedOfOneSolution()
	Description	Cette méthode est pour calculer le nombre de sans-abris hébergés d'une solution
	Données d'entrée	1) L'instance dessus
		2) Une solution : [4,6,-1,-1,3]
	Sortie attendue	44
	Résultat réel	44
12	Nom de méthode à tester	calculateBuildingAllocatedOfOneSolution()
	Description	Cette méthode est pour calculer le nombre de bâtiments affectés d'une solution.
	Données d'entrée	1) L'instance dessus
		2) Une solution : [4,6,-1,-1,3]
	Sortie attendue	3
	Résultat réel	3

3 Tests fonctionnels

Le test fonctionnel est pour vérifier si les fonctionnalités du logiciel sont réalisées et correspondent aux besoins de client. Pour ce projet, ses fonctionnalités principales sont la lecture des 3 fichiers (bâtiment, care, distance), la recherche la meilleure solution d'affectation, et l'écriture des solutions au fichier. Les fonctionnalités sur le fichier sont déjà testé pendant le test unitaire, donc cette chapitre décrit principalement le test fonctionnel pour la recherche la meilleure solution d'affectation.

On a deux parties à faire : déterminer le nombre d'itération, et évaluer la solution trouvée. On suppose que le nombre de fourmis est 50. On prend 3 échantillons pour faire le test :

- 20 bâtiments et 5 cares ;
- 50 bâtiments et 20 cares ;
- 500 bâtiments et 187 cares.

3.1 Test de l'influence de nombre d'itérations

Le nombre d'itération dépend de la taille d'échantillon et le nombre de fourmis. On dessiner une figure dont l'axe x est l'itération et l'axe y est la qualité de solution. Il y a deux courbes dans la figure, la courbe bleue représente la qualité de meilleure solution d'une itération, et la courbe rouge représente la qualité moyenne de solutions d'une itération. Si la différence entre les deux courbes est plus petite, cela signifie que la meilleure solution actuelle est plus fiable.

1. Pour 20 bâtiments et 5 cares :
   On fait 100 itération, la figure de qualités est comme Figure4.
   On peut voir que le courbe bleu est stable à partir de début. Et l'identifiant d'itération

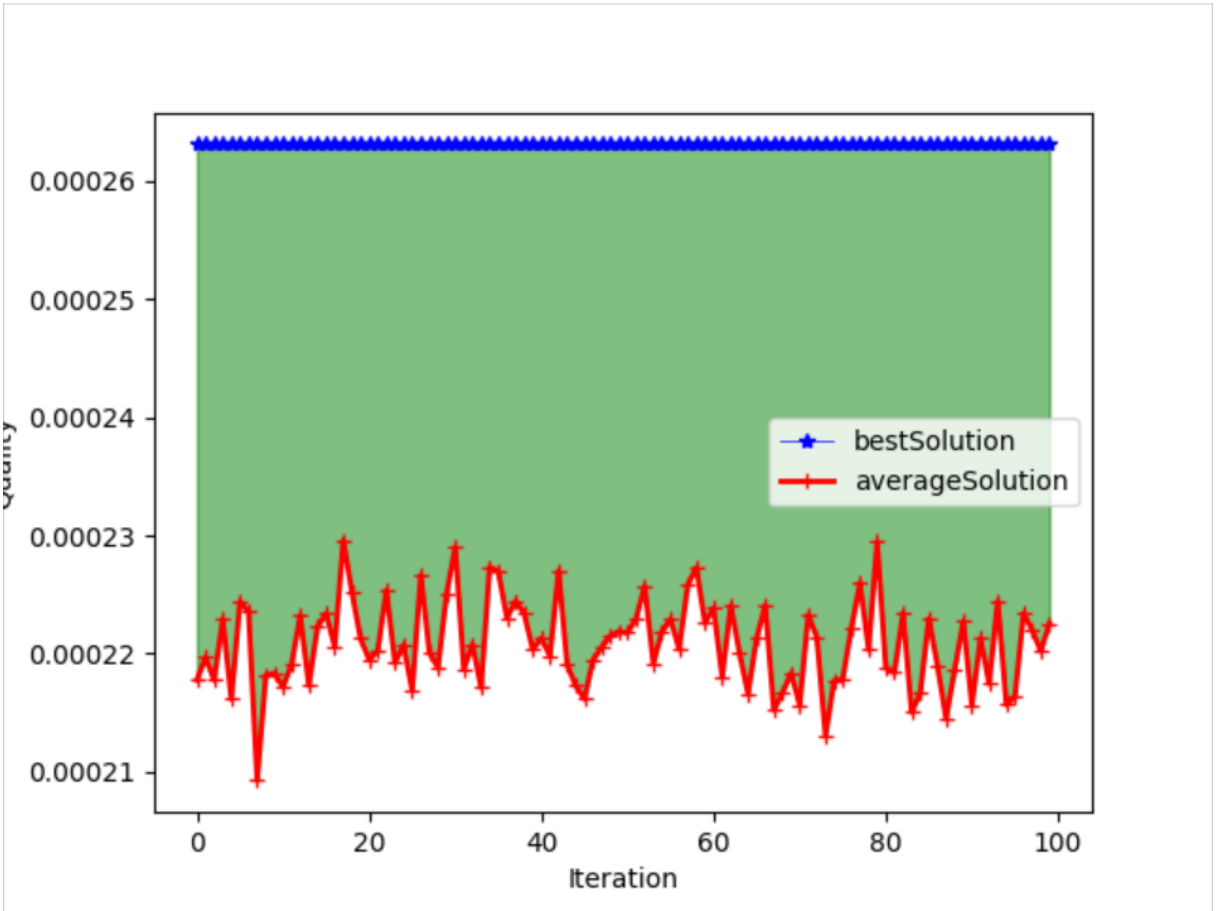


Figure 4 – *La figure de qualité pour 20 bâtiments et 5 cares avec 100 itérations*

imprimé de la meilleure solution est 0, donc pour cet échantillon, le nombre d'itération n'est pas important.

2. Pour 50 bâtiments et 20 care :
   On fait 1000 itérations, la figure de qualité est comme Figure5.
   Selon la figure5, on peut voir que la valeur de courbe bleu est entre 0.000330 et 0.000310, et la valeur de courbe rouge est déjà stable. L'id d'itération de la meilleure solution imprimé est 377, donc pour cet échantillon, on peut mettre le nombre d'itération en 500. (On mettre un nombre un peu plus grand pour éviter le cas accidentel).
3. Pour 500 bâtiments et 187 cares :
   On fait 70 itération, la figure de qualités est comme Figure6.
   Selon la figure6, on peut voir que quand le programme exécute jusqu'à 70 itérations, le courbe n'est pas encore stable, et l'id d'itération de la meilleure solution imprimé est

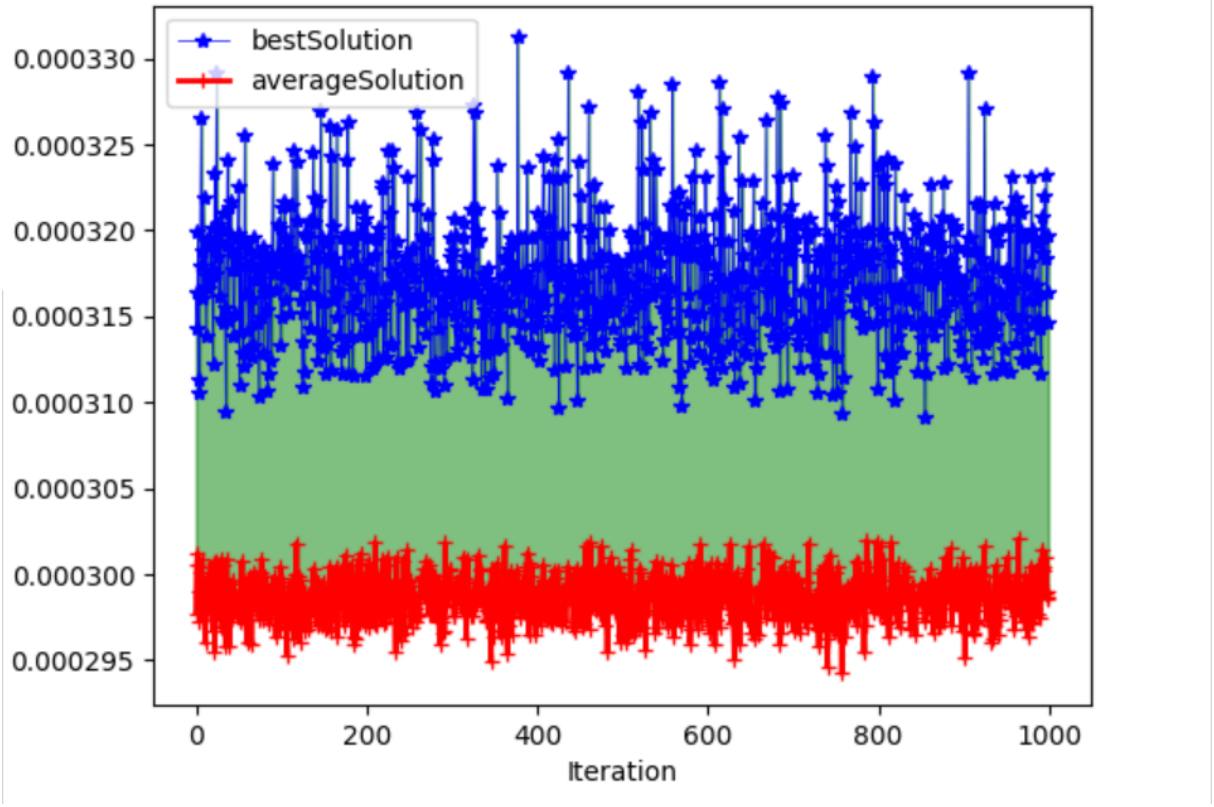


Figure 5 – *La figure de qualité pour 50 bâtiments et 20 cares avec 1000 itérations*

70. Donc pour dermatite le nombre d'itérations pour cet échantillon. Il faut faire plus d'expérimentation et prendre plus d'itération.

3.2 Vérification de la solution

D'abord, on peut dessiner les points de bâtiments et care. La Figure7 montre que l'échantillon de 50 bâtiments et 10 care.

Ensuite, on peut référer le fichier de solution pour évaluer la meilleure solution trouvée.

En fait, pour l'échantillon dont la tailles est petite, on peut facilement et directement savoir si la solution est bonne ou pas selon les distances et les capacités de care. Parce que la solution dépend du plusieurs contraintes, on peut vérifier la justesse de solution, mais il faut peut-être plus de conaissance sur l'aménagement pour vérifier si la solution est raisonnable.

4 Tests de performance

Le test de performance est pour améliorer la performance de logiciel par évaluer le temps d'exécution, l'occupation de mémoire et l'utilisation de CPU, etc.

La langage Python fournit un outil qui s'appelle « profile » pour détecter la performance de programme. Pour ce projet, le test de performance que on fait consiste à deux parties : le test sur le temps d'exécution, et l'occupation de mémoire. Il suffit ajouter l'annotation « @profile » avant le nom de méthode à tester. On teste principalement la performance de choisir les

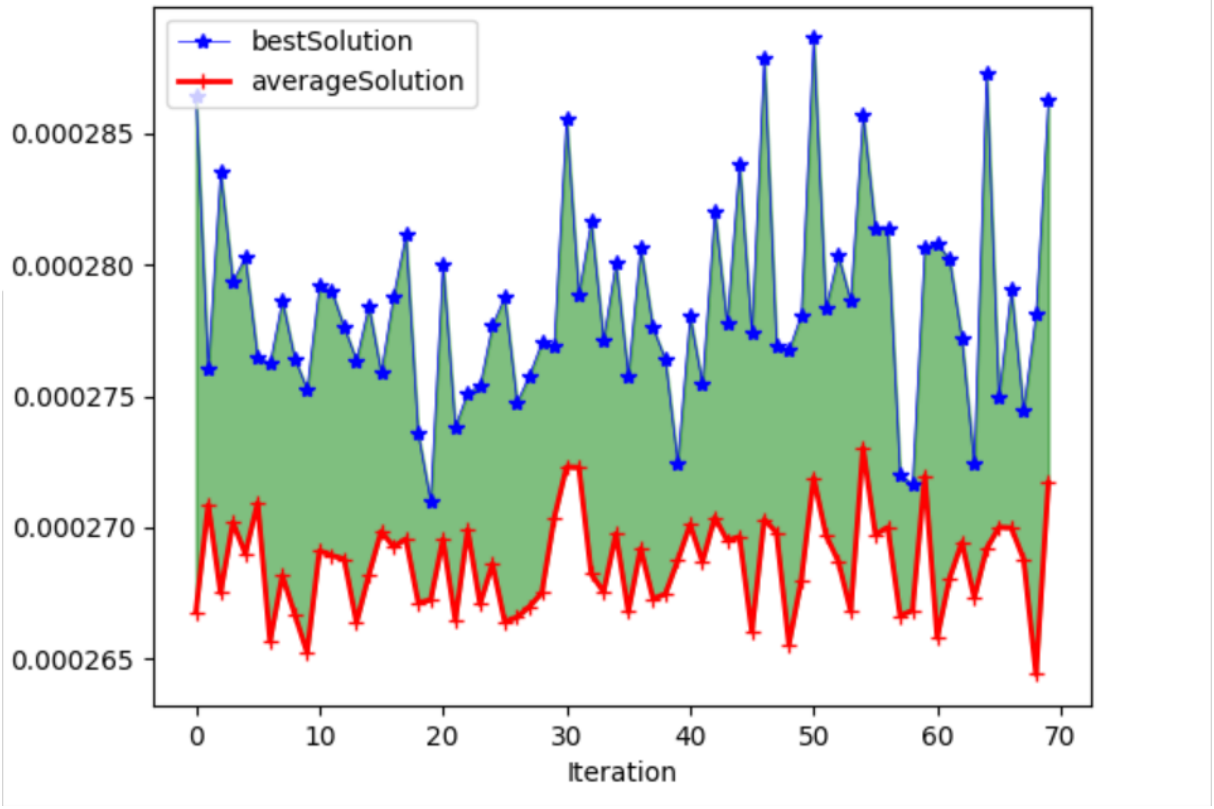


Figure 6 – *La figure de qualité pour 500 bâtiments et 187 cares avec 70 itérations*

cares et la performance d'affecter les bâtiments. Donc on ajoute l'annotation avant la méthode « allocateBuilding() » et « chooseCare() » dans la classe « AlgorithmController ».

4.1 Test du temps d'exécution

Pour tester le temps d'exécution, on peut utiliser la librairie « line_profiler » pour le Python. La commande d'installation et la commande d'utilisation sont présentées dans l'annexe « Document d'installation des librairies ». Cette libraire peut mesurer le temps d'exécution de chaque ligne de méthode, et afficher le résultat ligne par ligne. La Figure8 et la Figure9 montrent respectivement le résultat de test du temps d'exécution.

L'unité de temps est 3.95062e-07 sec. Dans la figure 4.1 et la figure 4.2, la troisième colonne est le temps d'exécution de la ligne, et la cinquième colonne est le pourcentage de temps.

Selon les résultats, on peut voir que les parties suivantes prennent la plupart du temps :

- Copier la liste de bâtiments et copier la liste de cares(0.6% et 0.9%);
- Mettre à jour le paramètre de phéromone « deltaTau »(0.5%);
- Calculer la qualité de solution(0.4%);
- La plupart du temps est pris par la fonction « print() » pour afficher le déroulement du programme (presque 96%).

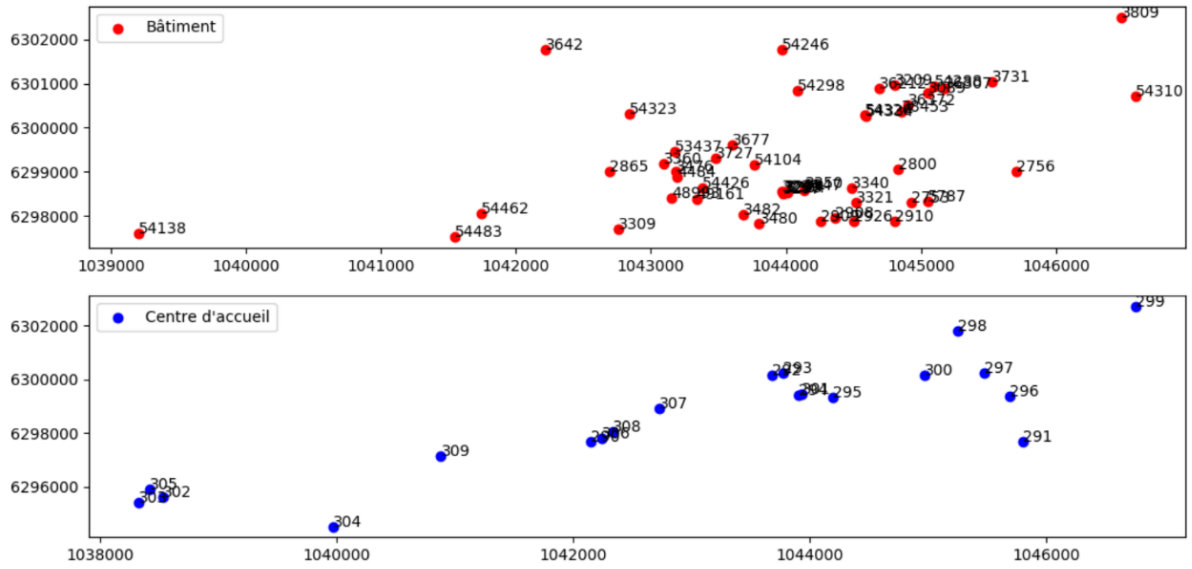


Figure 7 – *La figure d'échantillon de 50 bâtiments et 10 care*

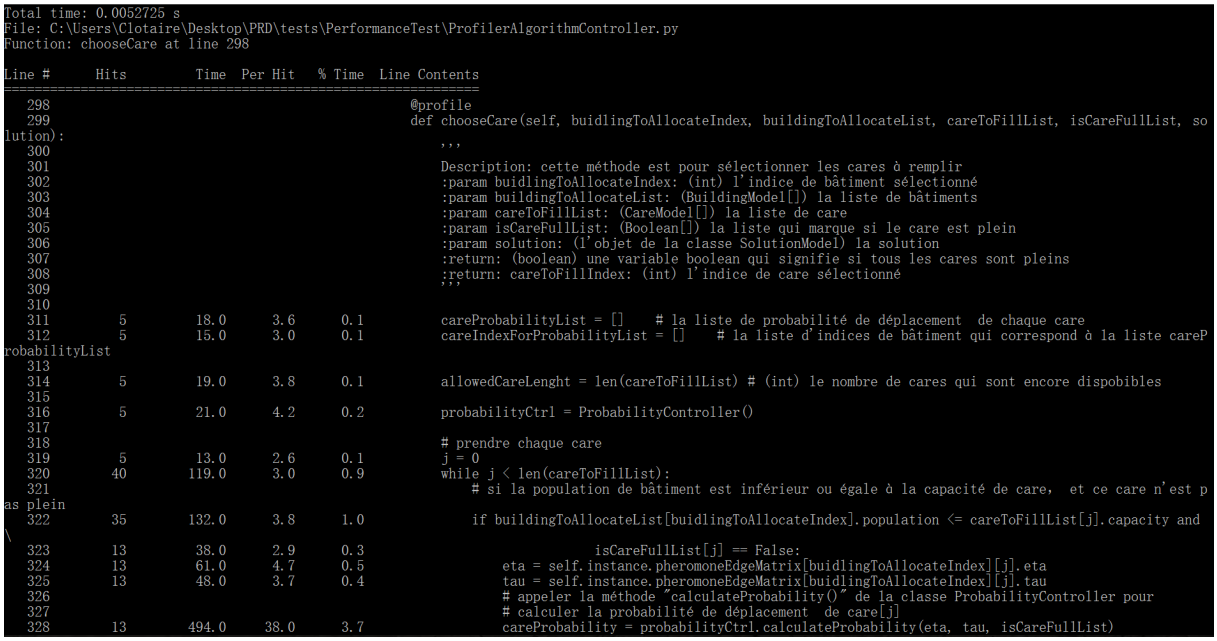
```
Total time: 0.0694286 s
File: C:\Users\Clotaire\Desktop\PRD\tests\PerformanceTest\ProfilerAlgorithmController.py
Function: allocateBuilding at line 107

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   107                                           @profile
   108                                           def allocateBuilding(self, ant,copyDistanceSortedBuildingIndexMatrix, solutionForOneIterationList,qualit
yOfSolutionForOneIterationList):
   109                                               '''
   110                                               Description: cette méthode est pour sélectionner les bâtiments à affecter
   111                                               :param ant: (l'objet de la classe AntModel) un fourmi qui va chercher sa solution
   112                                               :param copyDistanceSortedBuildingIndexMatrix: (int[][]) la matrice copiée d'indices de bâtiment réfé
rée la matrice de distance
   113                                               :param solutionForOneIterationList: (SolutionModel[]) la liste de solutions pour une itération
   114                                               :param qualityOfSolutionForOneIterationList: (float[]) la liste de qualités de solution pour une ité
ration
   115                                               ;return: rien
   116                                               '''
   117
   118          1         23.0     23.0      0.0      ant.solution = SolutionModel()
   119
   120          1       1109.0   1109.0      0.6      buildingToAllocateList = copy.deepcopy(self.instance.buildingList)  # (BuildingModel[]) la liste de
bâtiments
   121          1       1618.0   1618.0      0.9      careToFillList = copy.deepcopy(self.instance.careList)  # (CareModel[]) la liste de care
   122
   123                                               # (Boolean[]) la liste qui sert à marquer si le bâtiment est déjà affecté, les valeurs initiales son
t "False"
   124          1          8.0      8.0      0.0      isBuildingSelectedList = [False] * len(buildingToAllocateList)
   125                                               # (Boolean[]) la liste qui sert à marquer si le care est déjà plein, les valeurs initiales sont "Fal
se"
   126          1          6.0      6.0      0.0      isCareFullList = [False] * len(careToFillList)
   127                                               # (int[]) la liste de rayon d'attraction de care, les valeurs initiales sont égales au rayon initial
e
   128          1          6.0      6.0      0.0      radiusList = [self.careEffectRadius] * len(careToFillList)
   129                                               # (int[]) un solution, les éléments sont les indices de care, les valeurs initiales sont -1
   130                                               # Si la valeur est -1, ça veut dire que aucun care peut héberger ce bâtiment
   131          1          7.0      7.0      0.0      ant.solution.solutionArray = [-1] * len(buildingToAllocateList)
   132
   133          1          6.0      6.0      0.0      probabilityCtrl = ProbabilityController()
   134
```

Figure 8 – *Le test du temps d'exécution lors d'affecter les bâtiments*

4.2 Test d'occupation de mémoire

Pour tester l'occupation de mémoire, on peut utiliser la librairie « memory__profiler » pour le Python.La commande d'installation et la commande d'utilisation sont présentées dans l'annexe « Document d'installation des librairies ». Cette libraire peut mesurer l'utilisation de mémoire de chaque ligne de méthode, et afficher le résultat ligne par ligne. La Figure10et la Figure11montrent respectivement le résultat de test de l'occupation de mémoire.

Dans la figure 4.3 et la figure 4.4, la deuxième colonne est le mémoire utilisé de la ligne, et la troisième colonne est l'incrément de mémoire utilisé.



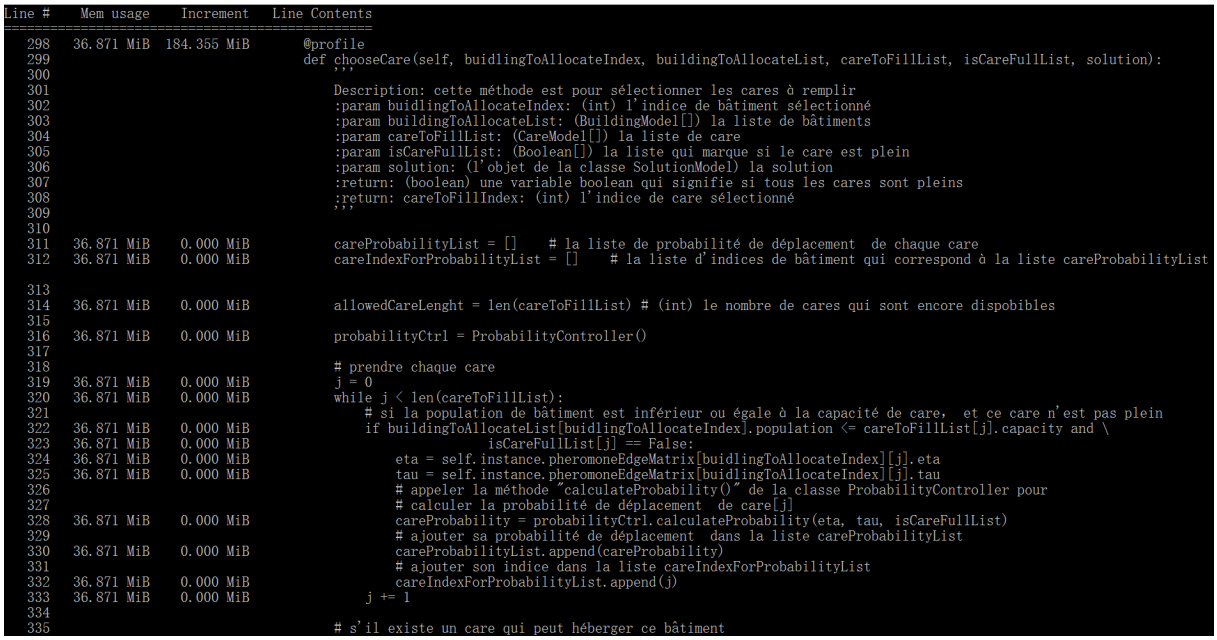
```
Total time: 0.0052725 s
File: C:\Users\Clotaire\Desktop\PRD\tests\PerformanceTest\ProfilerAlgorithmController.py
Function: chooseCare at line 298

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   298                                           @profile
   299                                           def chooseCare(self, buidlingToAllocateIndex, buildingToAllocateList, careToFillList, isCareFullList, so
lution):
   300                                               '''
   301                                               Description: cette méthode est pour sélectionner les cares à remplir
   302                                               :param buidlingToAllocateIndex: (int) l'indice de bâtiment sélectionné
   303                                               :param buildingToAllocateList: (BuildingModel[]) la liste de bâtiments
   304                                               :param careToFillList: (CareModel[]) la liste de care
   305                                               :param isCareFullList: (Boolean[]) la liste qui marque si le care est plein
   306                                               :param solution: (l'objet de la classe SolutionModel) la solution
   307                                               :return: (boolean) une variable boolean qui signifie si tous les cares sont pleins
   308                                               ;return: careToFillIndex: (int) l'indice de care sélectionné
   309                                               '''
   310
   311         5         18.0      3.6      0.1          careProbabilityList = []    # la liste de probabilité de déplacement  de chaque care
   312         5         15.0      3.0      0.1          careIndexForProbabilityList = []    # la liste d'indices de bâtiment qui correspond à la liste careP
robabilityList
   313
   314         5         19.0      3.8      0.1          allowedCareLenght = len(careToFillList) # (int) le nombre de cares qui sont encore dispobibles
   315
   316         5         21.0      4.2      0.2          probabilityCtrl = ProbabilityController()
   317
   318                                               # prendre chaque care
   319         5         13.0      2.6      0.1          j = 0
   320        40        119.0      3.0      0.9          while j < len(careToFillList):
   321                                                       # si la population de bâtiment est inférieur ou égale à la capacité de care,  et ce care n'est p
as plein
   322        35        132.0      3.8      1.0                  if buildingToAllocateList[buidlingToAllocateIndex].population <= careToFillList[j].capacity and
\
   323        13         38.0      2.9      0.3                                  isCareFullList[j] == False:
   324        13         61.0      4.7      0.5                      eta = self.instance.pheromoneEdgeMatrix[buidlingToAllocateIndex][j].eta
   325        13         48.0      3.7      0.4                      tau = self.instance.pheromoneEdgeMatrix[buidlingToAllocateIndex][j].tau
   326                                                               # appeler la méthode "calculateProbability()" de la classe ProbabilityController pour
   327                                                               # calculer la probabilité de déplacement  de care[j]
   328        13        494.0     38.0      3.7                      careProbability = probabilityCtrl.calculateProbability(eta, tau, isCareFullList)
```

Figure 9 – *Le test du temps d'exécution lors de choisir les cares*

```
Line #    Mem usage    Increment   Line Contents
================================================
   107   36.871 MiB   36.871 MiB       @profile
   108                                 def allocateBuilding(self, ant,copyDistanceSortedBuildingIndexMatrix, solutionForOneIterationList,qualityOfSolutionFor
OneIterationList):
   109                                     '''
   110                                     Description: cette méthode est pour sélectionner les bâtiments à affecter
   111                                     :param ant: (l'objet de la classe AntModel) un fourmi qui va chercher sa solution
   112                                     :param copyDistanceSortedBuildingIndexMatrix: (int[][]) la matrice copiée d'indices de bâtiment référée la matrice
 de distance
   113                                     :param solutionForOneIterationList: (SolutionModel[]) la liste de solutions pour une itération
   114                                     :param qualityOfSolutionForOneIterationList: (float[]) la liste de qualités de solution pour une itération
   115                                     ;return: rien
   116                                     '''
   117
   118   36.871 MiB    0.000 MiB           ant.solution = SolutionModel()
   119
   120   36.871 MiB    0.000 MiB           buildingToAllocateList = copy.deepcopy(self.instance.buildingList)  # (BuildingModel[]) la liste de bâtiments
   121   36.871 MiB    0.000 MiB           careToFillList = copy.deepcopy(self.instance.careList)  # (CareModel[]) la liste de care
   122
   123                                     # (Boolean[]) la liste qui sert à marquer si le bâtiment est déjà affectè, les valeurs initiales sont "False"
   124   36.871 MiB    0.000 MiB           isBuildingSelectedList = [False] * len(buildingToAllocateList)
   125                                     # (Boolean[]) la liste qui sert à marquer si le care est déjà plein, les valeurs initiales sont "False"
   126   36.871 MiB    0.000 MiB           isCareFullList = [False] * len(careToFillList)
   127                                     # (int[]) la liste de rayon d'attraction de care, les valeurs initiales sont égales au rayon initiale
   128   36.871 MiB    0.000 MiB           radiusList = [self.careEffectRadius] * len(careToFillList)
   129                                     # (int[]) un solution, les éléments sont les indices de care, les valeurs initiales sont -1
   130                                     # Si la valeur est -1, ça veut dire que aucun care peut héberger ce bâtiment
   131   36.871 MiB    0.000 MiB           ant.solution.solutionArray = [-1] * len(buildingToAllocateList)
   132
   133   36.871 MiB    0.000 MiB           probabilityCtrl = ProbabilityController()
   134
   135                                     # construire la liste de candidat pour chaque care (le candidat est l'indice de bâtiment)
   136   36.871 MiB    0.000 MiB           print("Start to initialize candidate list for care...")
   137   36.871 MiB    0.000 MiB           candidateListForCare = [[]]
   138   36.871 MiB    0.000 MiB           j = 0
   139   36.871 MiB    0.000 MiB           while j < len(careToFillList):
   140                                         # prendre une colonne dans la matrice de distance originale
   141                                         # i.e. prendre la liste de distance entre le care actuel et chaque bâtiment
   142   36.871 MiB    0.000 MiB             originalDistanceColumn = [originalColumn[j] for originalColumn in self.instance.distanceMatrix]
   143                                         # prendre une ligne dans la matrice copiée d'indice de bâtiment
```

Figure 10 – *Le test de l'occupation de mémoire lors d'affecter les bâtiments*

Selon les résultats, on peut voir que quand la méthode « allocateBuilding() » est appelée, l'utilisation de mémoire incrémente de 36.871MB, et quand la méthode de « chooseCare() » est appelé, l'utilisation de mémoire incrémente de 184.355MB, les autres ligne ne fait pas l'incrémentation de mémoire utilisé.



```
Line #    Mem usage    Increment   Line Contents
================================================
   298   36.871 MiB  184.355 MiB       @profile
   299                                 def chooseCare(self, buidlingToAllocateIndex, buildingToAllocateList, careToFillList, isCareFullList, solution):
   300                                     '''
   301                                     Description: cette méthode est pour sélectionner les cares à remplir
   302                                     :param buidlingToAllocateIndex: (int) l'indice de bâtiment sélectionné
   303                                     :param buildingToAllocateList: (BuildingModel[]) la liste de bâtiments
   304                                     :param careToFillList: (CareModel[]) la liste de care
   305                                     :param isCareFullList: (Boolean[]) la liste qui marque si le care est plein
   306                                     :param solution: (l'objet de la classe SolutionModel) la solution
   307                                     :return: (boolean) une variable boolean qui signifie si tous les cares sont pleins
   308                                     ;return: careToFillIndex: (int) l'indice de care sélectionné
   309                                     '''
   310
   311   36.871 MiB    0.000 MiB           careProbabilityList = []    # la liste de probabilité de déplacement  de chaque care
   312   36.871 MiB    0.000 MiB           careIndexForProbabilityList = []    # la liste d'indices de bâtiment qui correspond à la liste careProbabilityList
   313
   314   36.871 MiB    0.000 MiB           allowedCareLenght = len(careToFillList) # (int) le nombre de cares qui sont encore dispobibles
   315
   316   36.871 MiB    0.000 MiB           probabilityCtrl = ProbabilityController()
   317
   318                                     # prendre chaque care
   319   36.871 MiB    0.000 MiB           j = 0
   320   36.871 MiB    0.000 MiB           while j < len(careToFillList):
   321                                         # si la population de bâtiment est inférieur ou égale à la capacité de care,  et ce care n'est pas plein
   322   36.871 MiB    0.000 MiB               if buildingToAllocateList[buidlingToAllocateIndex].population <= careToFillList[j].capacity and \
   323   36.871 MiB    0.000 MiB                       isCareFullList[j] == False:
   324   36.871 MiB    0.000 MiB                   eta = self.instance.pheromoneEdgeMatrix[buidlingToAllocateIndex][j].eta
   325   36.871 MiB    0.000 MiB                   tau = self.instance.pheromoneEdgeMatrix[buidlingToAllocateIndex][j].tau
   326                                             # appeler la méthode "calculateProbability()" de la classe ProbabilityController pour
   327                                             # calculer la probabilité de déplacement  de care[j]
   328   36.871 MiB    0.000 MiB                   careProbability = probabilityCtrl.calculateProbability(eta, tau, isCareFullList)
   329                                             # ajouter sa probabilité de déplacement  dans la liste careProbabilityList
   330   36.871 MiB    0.000 MiB                   careProbabilityList.append(careProbability)
   331                                             # ajouter son indice dans la liste careIndexForProbabilityList
   332   36.871 MiB    0.000 MiB                   careIndexForProbabilityList.append(j)
   333   36.871 MiB    0.000 MiB               j += 1
   334
   335                                     # s'il existe un care qui peut héberger ce bâtiment
```

Figure 11 – *Le test de l'occupation de mémoire lors de choisir les cares*

I Glossaire

- **Care** : le centre d'accueil qui héberge les sans-abris
- **KP** : le problème du sac à dos(Knapsack Problem)
- **MKP** : le problème du sac à dos multiple(Multiple Knapsack Problem)
- **MDKP** : le problème du sac à dos multidimensionnel(MultiDimensional Knapsack Problem)
- **MOKP** : le problème du sac à dos multi-objectif(Multi-Objective Knapsack Problem)
- **MCKP** : le problème du sac à dos multichoix (Multi-Choice Knapsack Problem)
- **MTHM** : la méthode heuristique de Martello et Toth(Martello and Toth heuristique Method)
- **ACO** : l'algorithme d'optimisation de colonies de fourmis(ant colony optimization algorithme)

Comptes rendus hebdomadaires

Compte rendu n°1 du 26/09/2017

Cette semaine, d'abord, j'ai compris le sujet. L'objectif de ce projet est de trouver une solution dont la distance totale parcouru par toutes les sans-abris secourus est la plus courte possible sous la condition que le nombre total de sans-abris secourus est le plus possible.

Ensuite, j'ai étudié la faisabilité de ce projet. On compte utiliser l'algorithme métaheuristique dans ce projet, surtout l'algorithme de colonies de fourmis. Cet algorithme vise spécifiquement au problème de l'optimisation de chemin. Donc ça sera pratique pour ce projet.

Compte rendu n°2 du 03/10/2017

Après que j'ai étudié des problèmes de l'optimisation combinatoire, je trouve que ce projet ressemble aux 2 problèmes classiques : le problème du bin packing et le problème du sac à dos. Donc je compare bien le problème de notre projet avec ces problèmes classiques. Mais parce que le but de ce projet n'est pas de forcement affecter toutes les sans-abris, je pense que notre problème est plus proche du problème du sac à dos.

Compte rendu n°3 du 10/10/2017

Cette semaine, je continue faire la modélisation mathématique après que je l'ai discuté ensemble avec M. Monmarche pendant la réunion la semaine dernière. Selon l'objectif de ce projet, il a deux fonctions objectives. Et dans le problème du sac à dos classique, il n'y a qu'un sac, mais il y a plusieurs centres d'accueil (équivalent au sac) dans notre problème. Donc je vais bien étudier l'état de l'art sur le problème du sac à dos pour chercher la façon de résoudre le problème du sac à dos avec plusieurs sacs.

Compte rendu n°4 du 17/10/2017

Cette semaine, selon ce que j'ai eu discuté ensemble avec M. Monmarche, M. Serrhini et les étudiants en DAE, j'ai fait aussi la comparaison entre notre problème et le problème du p-médian. Mais je pense quand même que notre problème est plus proche du problème du sac à dos multiple. Et j'ai fait la modification sur la modélisation mathématique selon les conseils de M. Monmarche.

Ensuite, j'ai étudié l'état de l'art et je trouve beaucoup de variantes du problème du sac à dos, y compris le problème du sac à dos multidimensionnelle, le problème du sac à dos

multi-objectifs, le problème du sac à dos multiple et le problème du sac à dos multichoix. Après que j'ai étudié chaque variante, je pense que le problème du sac à dos multiple correspond mieux à notre problème. La différence unique consiste à ce que le profil de l'objets dans le problème du sac à dos multiple est fixé, cela ne dépend pas du sac. Mais la distance (équivalent au projet) de chaque bâtiment n'est pas fixé, elle différer selon le centre d'accueil.

Enfin, je prépare le ppt pour la présentation sur le modèle de notre problème pour cette semaine.

Compte rendu n°5 du 24/10/2017

Cette semaine, j'ai bien étudié et appris l'algorithme. Je connais comment cet algorithme marche. La clé de cet algorithme est la phéromone, qui représente la désirabilité de mouvement. Les fourmis prennent leurs décisions de mouvement selon la phéromone. Je commence à écrire mon rapport de l'état de l'art.

Compte rendu n°6 du 14/11/2017

J'ai fini mon rapport de l'état de l'art pendant les vacances de Toussaint. Selon les conseils que M. Ragot m'a donnés sur mon cahier de spécification avant les vacances, je suis en train d'écrire mon cahier de spécification, et je le presque finis. Je vais le finir au-dedans de cette semaine.

J'ai une nouvelle compréhension sur la phéromone de l'algorithme de colonies de fourmis : on peut concevoir deux types de phéromone, l'un est pour choisir les bâtiments, et l'autre est pour choisir les cares. Je vais commencer à proposer un nouvel algorithme qui peut appliquer à notre problème.

Compte rendu n°7 du 21/11/2017

Cette semaine, j'ai proposé un nouvel algorithme de colonies de fourmis. Selon ce que j'ai discuté ensemble avec M. Monmarche, j'ai écrit les pseudocodes de l'algorithme.

Compte rendu n°8 du 28/11/2017

J'ai modifié l'algorithme que je proposé selon les conseils que M. Monmarche m'a donnés la semaine dernière, j'ajoute une fonction qui sert à un générateur de probabilité. Les évènements peuvent avoir lieu selon sa probabilité. Cette fonction assure que les évènements dont la probabilité est petite ont aussi la possibilité d'avoir lieu. Et je vais commencer à rédiger mon rapport de S9 en intégrant ensemble tous les documents que j'ai écrits.

Compte rendu n°9 du 05/12/2017

J'ai modifié mon cahier de spécification selon les conseils que M. Ragot m'a donnés la semaine dernière, et j'ai presque fini mon rapport de S9. Parce que je l'ai écrit dans Word, je vais le mettre dans LaTex et bien l'éditer.

Je préparerai la soutenance de S9 et je ferai le ppt pour la soutenance pour la semaine prochaine.

Compte rendu n°10 du 21/12/2017

J'ai étudié si on peut utiliser CUDA avec le langage Python. CUDA est un framework qui sert au calcul parallèle avec GPU. Il peut augmenter beaucoup d'efficacité d'exécution du programme. Mais la plupart d'exemples sont en langage C/C++. Il n'y a pas beaucoup d'exemple pour la langage Python.

Compte rendu n°11 du 11/01/2018

Cette semaine, j'ai fait la modélisation logiciel pour ce projet. J'ai écrit le diagramme de classe. Et j'ai commencé à faire la programmation pour réaliser l'algorithme proposé de S9.

Compte rendu n°12 du 25/01/2018

Pendant ces deux semaines, j'ai fini la première version de programme. Mais j'ai trouvé que ça prend trop de temps pour une fourmi et une itération. J'ai testé le temps ligne par ligne dans le code, j'ai trouvé que le problème consiste à ce qu'il faut parcourir 55000 bâtiments pour chaque déplacement et calculer 55000 probabilités chaque fois.

Compte rendu n°13 du 02/02/2018

Cette semaine, grâce au conseil de M. MONMARCHE, j'ai ajouté une nouvelle variable dans le programme : le rayon d'attraction de care. Il limite le nombre de bâtiments à parcourir pendant chaque déplacement. C'est utile pour diminuer le temps d'exécution.

Compte rendu n°14 du 12/02/2018

Même si j'ai ajouté le rayon d'attraction de care, mais le programme exécute aussi trop long. Donc j'ai ajouté une autre variable dans le programme : la liste de candidat de care. Elle est utilisée avec le rayon ensemble. Cette amélioration diminue de plus le temps.

Compte rendu n°15 du 19/02/2018

Parce qu'il y a beaucoup de fourmis qui fait la même action. Donc j'ai essayé d'utiliser le multithread. J'ai voulu faire ces fourmis déplacent en parallèle. Mais après que je l'ai mis en pratique, j'étais très surprise qu'il prenne plus de temps. C'est à cause du mécanise de Python. Il y a un verrou exclusif qui s'appelle « GIL ». Il assure que seulement un thread peut utiliser les ressources en même temps.

Compte rendu n°16 du 22/02/2018

Cette semaine, j'ai étudié comment déterminer les valeurs de variables, tels que le nombre d'itération, le nombre de fourmis, la valeur initiale de tau, la valeur initiale de rho, etc. Avec les aides de M. MONMARCHE, j'ai fait beaucoup d'expérimentation pour déterminer leurs valeurs.

Compte rendu n°17 du 08/03/2018

Avant les vacances, j'ai pris un rendez-vous avec M. RAMEL, et il m'a donné des conseils sur mon modélisation. Donc cette semaine, je refais mon diagramme de classe, et je reconstruis le code.

Et pour améliorer le temps d'exécution, j'ai fait le tri de distance avec des algorithme efficace et classique. J'ai essayé 4 algorithmes : le tri par sélection, le tri par insertion, le tri à bulle et le tri par fusion. Je compte respectivement le temps de tri, je trouve que le tri par fusion est le meilleur.

Compte rendu n°18 du 22/03/2018

Pendant ces deux semaines, j'ai fait les tests de programme. J'ai fait les tests unitaires pour tester chaque méthode, j'ai fait aussi les tests de performance avec l'outil « profile ». J'ai commencé à écrire le cahier du développeur et le cahier de test.

Compte rendu n°19 du 29/03/2018

Cette semaine, j'ai fini le cahier du développeur et le cahier de test. Je rédige mon rapport finale en intégrant tous les cahiers que j'ai écrit.

Et je prepare la soutenance de S10 pour la semaine prochaine.

Webographie

[WWW1] NVIDIA Corporation. *Embedded Systems Developer Kits, Modules, & SDKs | NVIDIA Jetson|NVIDIA*. 2017. URL : %5Curl%7Bhttp://www.nvidia.com/object/embedded-systems-dev-kits-modules.html%7D.

[WWW2] Wikipédia. *Problème de bin packing — Wikipédia, l'encyclopédie libre*. [En ligne ; Page disponible le 23-mai-2016]. 2016. URL : %5Curl%7Bhttp://fr.wikipedia.org/w/index.php?title=Probl%C3%A8me_de_bin_packing&oldid=126444210%7D.

[WWW3] Wikipédia. *Problème du sac à dos — Wikipédia, l'encyclopédie libre*. [En ligne ; Page disponible le 24-octobre-2017]. 2017. URL : %5Curl%7Bhttp://fr.wikipedia.org/w/index.php?title=Probl%C3%A8me_du_sac_%C3%A0_dos&oldid=141853415%7D.

Bibliographie

[1] Ines Alaya, Christine Solnon et Khaled Ghedira. « Algorithme fourmi avec différentes stratégies phéromonales pour le sac à dos multidimensionnel ». In : *rapport de recherche, Institut Supérieur de Gestion de Tunis* (2005).

[2] Samir Balbal. « Utilisation de l'intelligence artificielle pour résoudre le problème du sac à dos ». In : (2015).

[3] Stefka Fidanova. « Ant colony optimization and multiple knapsack problem ». In : *Handbook of research on nature inspired computing for economics and management* (2007), p. 498–509.

[4] Abdulrahman Al-khedhairi. « Simulated annealing metaheuristic for solving p-median problem ». In : *Int. J. Contemp. Math. Sciences* 3.28 (2008), p. 1357–1365.

[5] Mohamed Esseghir Lalami, Didier Elbaz, Moussa Elkihel et Vincent Boyer. « Une heuristique pour le problème du sac à dos multiple en variables 0-1 ». In : ().

[6] Alexander E Mohr. « Bit allocation in sub-linear time and the multiple-choice knapsack problem ». In : *Data Compression Conference, 2002. Proceedings. DCC 2002.* IEEE. 2002, p. 352–361.

[7] Ronghua Shang, Licheng Jiao, Wenping Ma et Wei Zhang. « Clonal Selection Algorithm for Multi-Objective 0/1 Knapsack Problems ». In : *JOURNAL-XIAN JIAOTONG UNIVERSITY* 42.2 (2008), p. 156.

Projet Recherche & Développement

Métaheuristiques bio-inspirées pour l'optimisation de l'affectation interne de la population de Nice aux centres d'accueil en cas de séisme

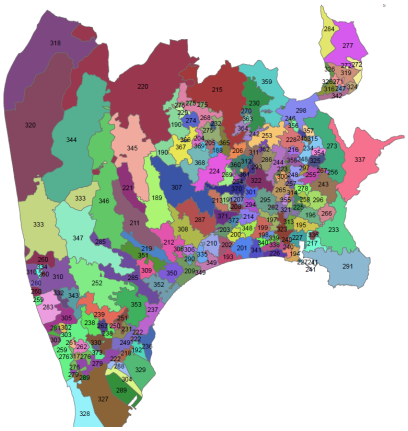
Peng BI

En collaboration avecPolytech Tours

Encadrement : Nicolas MONMARCHE

Objectifs

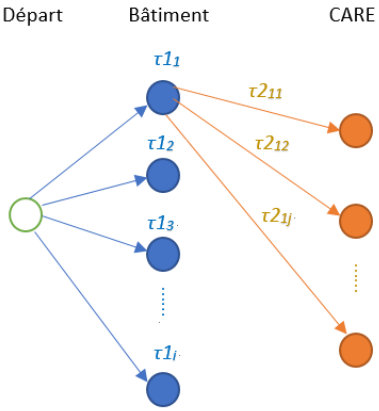
Lors d'un séisme à Nice, le gouvernement doit affecter les sans-abris immédiatement.Ce projet vise à proposer un algorithme métaheuristique pour développer un logiciel qui aide le gouvernement à optimiser l'affectation des sans-abris et à faire une meilleure décision en cas de séisme.



Plan d'évacuation à Nice

Mise en œuvre

Sur la recherche, il faut proposer un algorithme basé sur l'algorithme de colonies de fourmis pour resourdre le problème d'affectation. Sur le développement, le logiciel à réaliser doit trouver la meilleure solution d'affectation selons les fichiers de données que l'utilisateur fournit. Le résultat sera enregistré dans un fichier texte.



Graphe de l'algorithme de colonies de fourmis

Résultats attendus

Le résultat attendu est la solution d'affectation respectant les objectifs. C'est un tableau, dont les colonnes sont:

— le numéro de bâtiment
— le numéro de centre d'accueil qui héberge les sans-abris du bâtiment

	ID_BAT	ID_CARE
2	53437	308
3	3209	305
4	48993	308
5	2800	298

Fichier de solution

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Projet Recherche & Développement

Métaheuristiques bio-inspirées pour l'optimisation de l'affectation interne de la population de Nice aux centres d'accueil en cas de séisme

Peng BI

En collaboration avec Polytech Tours

Encadrement : Nicolas MONMARCHE

Objectifs

Lors d'un séisme à Nice, le gouvernement doit affecter les sans-abris immédiatement.Ce projet vise à proposer un algorithme métaheuristique pour développer un logiciel qui aide le gouvernement à optimiser l'affectation des sans-abris et à faire une meilleure décision en cas de séisme.

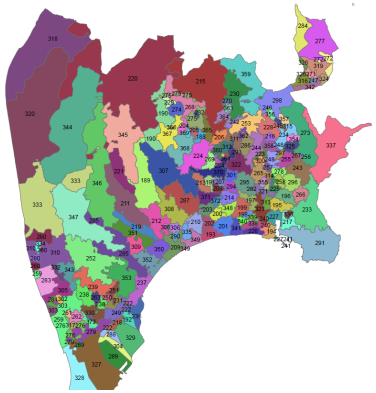
Mise en œuvre

Sur la recherche, il faut proposer un algorithme basé sur l'algorithme de colonies de fourmis pour resourdre le problème d'affectation. Sur le développement, le logiciel à réaliser doit trouver la meilleure solution d'affectation selons les fichiers de données que l'utilisateur fournit. Le résultat sera enregistré dans un fichier texte.

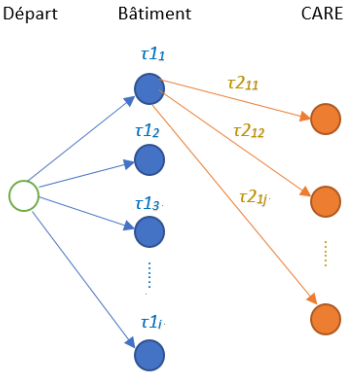
Résultats attendus

Le résultat attendu est la solution d'affectation respectant les objectifs. C'est un tableau, dont les colonnes sont:

- le numéro de bâtiment
- le numéro de centre d'accueil qui héberge les sans-abris du bâtiment



Plan d'évacuation à Nice



Graphe de l'algorithme de colonies de fourmis

	ID_BAT	ID_CARE
1		
2	53437	308
3	3209	305
4	48993	308
5	2800	298

Fichier de solution

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Métaheuristiques bio-inspirées pour l'optimisation de l'affectation interne de la population de Nice aux centres d'accueil en cas de séisme

Résumé

Le projet de recherche et développement, comme le projet de fin d'étude, est le dernier cycle dans la vie d'étudiant, et c'est très important et obligatoire pour que les étudiants poursuivent leurs diplômes d'ingénieur. Il peut intégrer toutes les connaissances et les techniques que les étudiants acquissent de leurs formations, et mettre tous leurs acquis à la pratique.

Ce projet vise à développer un logiciel pour optimiser l'affectation des sans-abris en cas de séisme à Nice avec l'aide de l'algorithme de colonies de fourmis. Selon les données que le client fournit, ce logiciel peut trouver la meilleure solution d'affectation avec la distance parcourue la plus courte et le nombre maximum de sans-abirs affectés ou de bâtiments affectés.

Ce rapport contient 6 partie, y compris l'introduction du projet, les descriptions générales, l'état de l'art, l'analyse et la conception, la mise en oeuvre, la conclusion sur les taches finies et les taches prévue. Il contient aussi la planification, les descriptions sur les interfaces, sur les fonctionnalités et sur les non-fonctionnalités, le cahier du développeur, le document d'installation des librairies, le document d'utilisation et le cahier de test dans l'annexes.

Mots-clés

séisme, affectation, problème du sac à dos multiple, algorithme de colonies de fourmis, sans-abris

Abstract

The research and development project, as the graduation project, is the last phase in student life, and it is very important and obligatory for students to pursue their engineering degrees. It can integrate all the knowledge and skills that students acquire from their training, and put all their gains into practice.

This project aims at developing a software to optimize the assignment of homeless people in case of an earthquake in Nice with the help of the ant colony algorithm. According to the data which client provides, this software can find the best allocation solution with the shortest travalled distance and the maximum amount of allocated homeless people or allocated buildings.

This report contains 6 parts, including introduction of project, general description, state of the art, analyses and design, implementation, and conclusion about finished tasks and anticipated tasks. It also contains planning, description about interfaces, about fonctionality and about non-fonctionality, the developer manual, the libraries installation document, the user manual, and the test book in the appendix.

Keywords

earthquake, allocation, multiple knapsack problem, ant colony algorithm, homeless people

Entreprise
Polytech Tours

Étudiant
Peng BI (DI5)

Tuteur entreprise
Nicolas MONMARCHE (Professeur)

Tuteur académique
Nicolas MONMARCHE