Simulation et optimisation de la gestion dynamique de tâches évolutives sur des robots mobiles autonomes.Application sur le traitement UV-C robotisé dans l'horticulture.

Merouane Mazar

HESAM UNIVERSITÉ

Arts et Métiers Sciences et Technologies / lispen

LINEACT cesi

ÉCOLE DOCTORALE SCIENCES ET MÉTIERS DE L'INGÉNIEUR
LINEACT de CESI & LISPEN des Arts et Métiers

THÈSE

présentée par : **Merouane MAZAR**
soutenue le : **10 janvier 2022**

pour obtenir le grade de : **Docteur d'HESAM Université**

préparée à : **École Nationale Supérieure d'Arts et Métiers**

Discipline : **Section CNU 61**

Spécialité : **Génie industriel**

Simulation et optimisation de la gestion dynamique de tâches évolutives sur des robots mobiles autonomes

Application sur le traitement UV-C robotisé dans l'horticulture

THÈSE dirigée par :
LOUIS Anne, LINEACT-CESI

et co-encadrée par :
KLEMENT Nathalie, LISPEN des Arts et Métiers
BETTAYEB Belgacem, LINEACT-CESI

Jury

M. David Lemoine	Professeur, IMT Atlantique Nantes	Rapporteur
M. Faicel Hnaien	MCF HDR, Université de Technologie de Troyes	Rapporteur
M. Michel Tollenaere	Professeur des Universités, Grenoble-INP GI, UGA, Grenoble	Examinateur
M. Khaled Hadj Hamou	Professeur des Universités, INSA de Lyon	Examinateur
M. Mhammed Sahnoun	EC CESI HDR, CESI de Rouen	Examinateur
Mme. Anne Louis	Directeur de recherche CESI, CESI de Rouen	Examinatrice
Mme. Nathalie Klement	MCF, Arts et Métiers de Lille	Examinatrice
M. Belgacem Bettayeb	EC CESI, CESI de Lille	Examinateur

THÈSE

« Cela semble toujours impossible, jusqu'à ce qu'on le fasse »

Nelson Mandela

Remerciements

Je tiens à remercier à travers cette page tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail de thèse.

Je tiens particulièrement à remercier mes encadrants. Tout d'abord, je remercie Belgacem Bettayeb, pour son encadrement quasi quotidien durant ces trois années de thèse. Ensuite, je remercie Nathalie Klement qui m'a toujours aidé et encouragé durant la période de cette thèse. Je remercie également Mhammed Sahnoun qui ne faisait pas partie de la liste de mes encadrants, mais qui a toujours été présent en tant qu'encadrant de ma thèse. Sa place est donc dans ce paragraphe. Ce sont eux qui ont été présents au quotidien pour m'aider à avancer et sans eux cette expérience de trois ans n'aurait pas été la même. Enfin, je tiens à remercier Anne Louis, ma directrice de thèse, qui m'a accueilli dans son équipe de recherche, m'a fait confiance et m'a soutenu pendant ces trois années.

Je remercie ensuite tous mes collègues de l'équipe LINEACT qui m'ont accueilli durant cette thèse et qui m'ont permis de travailler dans une ambiance de travail exceptionnelle : Amine, Mourad, Mejdi, Wael, Souleyman, Imen, Abdelkrim, Nasredine, Nicolas, Elodie, Vincent... Sans oublier bien sûr les doctorants du LISPEN de Lille : Mouad, Eddy, Amélie, Zein, Laurent...

Mes remerciements vont également aux personnes qui ont accepté de rapporter ce manuscrit de thèse. Je remercie tout d'abord Monsieur M. Khaled Hadj Hamou, qui a été souvent présent lors du comité de suivi concernant cette thèse et qui par ses retours m'a beaucoup aidé. Je tiens à remercier M. David Lemoine et M. Faicel Hnaien qui ont accepté de relire cette thèse afin d'apporter leur expertise dans les domaines de la recherche opérationnelle et du génie industriel. Je tiens également à remercier M. Michel Tollenaere pour avoir fait partie du jury de ma thèse.

Finalement mes derniers remerciements vont à ma famille et mes amis qui m'ont soutenu tout au long de cette thèse. Merci à la femme de ma vie Khadidja et à ma petite princesse Mellina. Merci également à mes parents et mes frères (Fateh, Abdellah et Mimou) et ma sœur Chahinez qui sont toujours là pour moi, ainsi que mes grands-parents.

Résumé

L'ordonnancement de tâches sur des systèmes stochastiques complexes est une activité qui ne peut être optimisée avec les méthodes d'optimisation classiques. La principale difficulté réside dans le comportement stochastique et dynamique de ces systèmes, qui peut venir modifier les propriétés et/ou le nombre de tâches et/ou les ressources nécessaires à leur exécution. En effet, il est difficile d'estimer le temps nécessaire à l'exécution d'une tâche et définir un ordonnancement efficient qui prend en compte le comportement et l'évolution du système.

Cette thèse propose de traiter ce problème en utilisant une approche basée sur le couplage entre simulation et optimisation dans plusieurs situations. Nous avons appliqué cette approche dans le contexte de l'Agriculture 4.0, où nous avons automatisé le traitement robotique de maladie du mildiou en horticulture. Nous avons développé un simulateur basé sur des systèmes multi-agents, où nous avons incorporé des moteurs d'optimisation basés sur des méthodes exactes et approchées. Nous avons modélisé le comportement de la maladie dans une serre en se basant sur le formalisme de chaine de Markov pour chaque agent représentant une plante. Les scénarios de simulation sont basés sur différents types de traitements préventifs (conditionnels, prédictifs et calendaires). Les résultats obtenus ont montré l'efficacité de l'approche et ont permis de proposer aux horticulteurs un outil combinant la simulation et l'optimisation pour les aider à définir et paramétrer la politique de traitement robotique appropriée.

Mots-clés : Ordonnancement dynamique, Simulation, Optimisation, Agriculture 4.0, Industrie 4.0, Traitement préventif, Heuristique, Algorithme Génétique, Robot autonome.

Abstract

Task scheduling on complex stochastic systems is an activity that cannot be optimized with classical optimization methods. The main difficulty lies in the stochastic and dynamic behavior of these systems, which can modify the properties and/or the number of tasks and/or the resources required for their execution. Indeed, it is difficult to estimate the time needed to execute a task and to define an efficient scheduling that takes into account the behavior and the evolution of the system.

This thesis proposes to solve this problem using an approach based on the coupling between simulation and optimization in several situations. We have applied this approach in the context of Agriculture 4.0, where we have automated the robotic treatment of mildew disease in horticulture. We developed a simulator based on multi-agent systems, where we incorporated optimization engines based on exact and approximate methods. We modeled the disease behavior in a greenhouse based on the Markov chain formalism for each agent representing a plant. The simulation scenarios are based on different types of preventive treatments (conditional, predictive and calendar). The results obtained showed the efficiency of the approach and allowed to propose to horticulturists a tool combining simulation and optimization to help them to define and parameterize the appropriate robotic treatment.

Keywords : Dynamic scheduling, Simulation, Optimization, Agriculture 4.0, Industry 4.0, Preventive treatment, Heuristics, Genetic algorithm, Autonomous robot.

Table des matières

Liste des tableaux

Table des figures

Introduction

Depuis le Néolithique, l'homme cultive la terre et élève des animaux pour obtenir la nourriture dont il a besoin pour survivre. Cette pratique, appelée agriculture, a évolué selon un processus graduel et à long terme, souvent représenté par 4 ères, caractérisées par des évolutions (ou révolutions) significatives, nommées « Agriculture 1.0 » à « Agriculture 4.0 ». L'Agriculture 1.0 correspond à l'ère de l'agriculture traditionnelle, qui repose principalement sur le travail et la force animale. À ce stade, bien que des outils simples, tels que la faucille et la pelle, soient utilisés dans les activités agricoles, les humains ne peuvent toujours pas se débarrasser du travail manuel lourd, de sorte que la productivité reste à un faible niveau. Jusqu'au XIXe siècle, les machines à vapeur ont été améliorés et largement utilisés pour fournir de nouvelles ressources dans tous les domaines de la vie et de l'industrie, y compris l'agriculture. À l'ère de l'Agriculture 2.0, diverses machines agricoles étaient actionnées manuellement par les agriculteurs et de nombreux produits chimiques étaient utilisés. De toute évidence, l'Agriculture 2.0 a considérablement augmenté l'efficacité et la productivité du travail agricole. Au vingtième siècle, l'Agriculture 3.0 est née avec le développement rapide des ordinateurs et de l'électronique, certains l'appellent aussi « révolution biotechnologique », d'autres « green révolution ». Les programmes informatiques et les techniques robotiques ont permis aux machines agricoles d'effectuer des opérations de manière efficace et intelligente. Une division raisonnable du travail entre les machines agricoles a permis de réduire l'utilisation de produits chimiques, d'améliorer la précision de l'irrigation, etc. Aujourd'hui, l'agriculture connaît sa quatrième révolution, grâce à l'utilisation des technologies actuelles telles que l'Internet des objets, le Big Data, l'intelligence artificielle, le cloud computing, la télédétection, etc. Les applications de ces technologies peuvent améliorer considérablement l'efficacité des activités agricoles en termes de production, de rendement, de qualité des aliments, d'impact environnemental et social.

Parallèlement à ce qui se passe dans l'industrie, l'Agriculture 4.0 se caractérise par l'utilisation « massive » de robots. De plus en plus de chercheurs développent plusieurs types de robots dans le domaine agricole. Plusieurs types de robots ont été développés, que ce soit dans le domaine de la recherche ou sur le marché, tels que les robots de récolte, les robots de traitement, les robots de surveillance et les robots de pulvérisation. Les robots de pulvérisation, qui sont utilisés pour le traitement des maladies des plantes. Ces robots pulvérisent des produits chimiques, tels que des pesticides ou des insecticides, sur les plantes afin de les traiter. En général, les robots de culture sont spécifiques au type de plante, contrairement aux pulvérisateurs, qui peuvent être utilisés sur de nombreux types de plantes. Récemment, les chercheurs ont découvert que l'utilisation des rayons Ultraviolet de type C (UV-C) pouvait éliminer certains champignons tels que le mildiou et l'oïdium. Cette technique de traitement est plus durable car elle réduit la nécessité de pulvériser des pesticides. Cependant, cette technologie présente un risque de brûlures pour l'agriculteur lorsque le traitement par rayons UV-C est effectué manuellement. Par conséquent, l'utilisation d'un robot autonome qui transporte des lampes est nécessaire pour bénéficier de cette technologie de manière sûre et efficace.

Le travail de cette thèse porte sur une application d'un projet européen appelé « UV-Robot ». L'objectif de ce projet est de développer un robot portant des lampes UV-C pour traiter la maladie du mildiou dans les serres. Pour assurer un meilleur fonctionnement du robot, nous devons optimiser la gestion de ses activités de traitement. Comme le comportement de l'apparition et de l'évolution de la maladie suit un processus stochastique. Nous devons étudier l'ordonnancement dynamique des tâches évolutives pour le traitement des serres avec le robot UV-C. Le robot UV-C étant autonome, il devra prendre plusieurs décisions lors de l'exécution de ces tâches. Nous utiliserons des outils d'aide à la décision automatique pour assurer son autonomie et optimiser son travail. Il s'agit d'optimiser l'ordonnancement des tâches de traitement de manière statique et dynamique. La difficulté de traitement de ce système réside dans le comportement de la maladie que le robot doit traiter. Ainsi, pour optimiser l'ordonnancement des tâches de traitement, il est nécessaire de simuler le comportement dynamique de la maladie dans l'environnement de la serre.

Le couplage entre simulation et optimisation nous permet d'étudier ce problème. En effet, la simulation gère un horizon temporel, mais présente une myopie dans l'espace d'état. D'autre part, l'optimisation gère bien l'espace d'état, mais elle a une myopie dans le temps. L'objectif dans le couplage de ces deux approches est de créer un environnement proche du système réel avec la simulation et, ensuite, d'intégrer les algorithmes d'optimisation pour améliorer le comportement du système.

Ce manuscrit de thèse contient quatre chapitres qui sont résumés dans les paragraphes qui suivent :
Dans le premier chapitre (cf. chapitre 1), nous présentons le contexte et le cadre de notre travail, puis un aperçu général sur l'environnement que nous étudions, quelques définitions et une revue de l'existant en ce qui concerne le domaine agricole. Ensuite, nous passons en revue l'état de l'art sur les problèmes d'ordonnancement dynamique et les méthodes de couplage entre simulation et optimisation. Enfin, nous exposons la problématique de recherche de cette thèse, en précisant la question à laquelle nous devons répondre et l'hypothèse proposée.
Le deuxième chapitre (cf. chapitre 2) s'intéresse au développement de simulateur en détaillant chaque module de simulateur. Nous allons commencer par un état de l'art sur les problèmes et méthodes de simulation. Puis, nous proposons la modélisation de la simulation qui est basée sur les systèmes multi-agents. Ensuite, nous montrons comment nous avons modélisé le comportement de la maladie et son intégration dans le simulateur. Enfin, nous détaillons le développement du simulateur avec des schémas explicatifs pour le codage des agents.
Nous commençons le troisième chapitre (cf. chapitre 3) par un état de l'art sur les méthodes d'optimisation, et sur le problème de Bin-Packing qui est très semblable au notre dans certains cas. Pour notre problème d'optimisation de tâches de traitement robotisé, nous proposons une formalisation analogue à celle du problème de Bin-Packing pour modéliser notre problème. Ensuite, nous présentons une partie dédiée aux algorithmes d'optimisation, où nous expliquons le choix et le développement des algorithmes intégrés dans le simulateur.
Le quatrième chapitre (cf. chapitre 4) est consacré à l'étude de différents types de traitement robotisé qui peuvent être utilisés pour maîtriser le niveau de maladie dans une serre. Au début de ce chapitre, et par analogie entre les tâches de traitement de plantes et celles de maintenance d'équipements, nous présentons un bref état de l'art sur les différents modèles développés dans la littérature pour les différents types de maintenance. Puis, nous détaillons les trois modèles de traitement utilisés et la construction de leurs scénarios de simulation. Enfin, nous fournissons et analysons les résultats obtenus pour chaque expérimentation.

Ce manuscrit se termine par une conclusion générale, où nous passons en revue les principales

contributions de la thèse et nous donnons quelques idées comme perspectives pour nos futurs travaux de recherche.

Chapitre 1

Contexte et problématique de la thèse

Contenu

1.1 Introduction

Le domaine de l'industrie a rencontré plusieurs évolutions à travers le temps, en commençant par les premières machines jusqu'aux dernières technologies. La première révolution industrielle a eu lieu à la fin du 18e siècle et au début de 19e siècle, avec l'invention des machines à vapeur. Cette révolution a permis l'apparition de la production mécanique qui a amélioré l'industrie, l'économie et même l'agriculture. L'utilisation massive des machines qui étaient poussées par l'énergie électrique et pétrolière a donné naissance à la deuxième révolution industrielle. La deuxième ère industrielle est passé d'une production artisanale dans des ateliers à une production de masse dans des usines, comme les usines d'automobiles au 19e siècle. Ensuite, l'apparition des cartes électroniques et les technologies informatiques au 20e siècle ont permis d'automatiser les productions industrielles et de révolutionner l'industrie pour une troisième fois. Aujourd'hui, nous vivons une quatrième révolution industrielle appelée « l'industrie 4.0 », qui est une amélioration de la troisième révolution. Le développement des technologies numériques a permis de connecter tous les outils informatiques et robotiques, une connexion qui a facilité le pilotage du monde physique à partir d'un monde virtuel. De plus, internet a permis de passer d'une supervision centralisée vers une supervision distribuée.

Cette dernière révolution est visible non seulement dans le domaine industriel, mais aussi dans le médical, le transport, le design/architecture et l'agriculture. Cela est dû au fait que les nouvelles technologies de l'industrie 4.0 sont aussi utiles et efficaces dans plein d'autres domaines. Comme les travaux de cette thèse s'intéressent à la supervision des robots agricoles, nous allons parler de l'Agriculture 4.0. En effet, l'agriculture 4.0 s'appuie sur les technologies développées à la base pour l'industrie 4.0 mais avec une application agricole, où des technologies telles que les robots, les capteurs, l'analyse de données et l'Internet des objets sont présents dans les fermes [Zhai et al., 2020]. Comme le montre la figure 1.1, un exemple d'« Agriculture 4.0 » (ou d'« Horticulture 4.0 ») se compose d'un robot, d'une tablette de supervision, d'un serveur de simulation et d'un cloud qui sont tous connectés à la serre. Les outils connectés permettent de visualiser l'état des plantes et aident l'agriculteur à prendre de bonnes décisions en peu de temps. La figure 1.1 ne montre que les technologies utilisées dans les travaux de cette thèse, mais il existe d'autres technologies comme les drones, d'autres types de robots, etc.

Ce chapitre présente dans un premier temps le contexte du projet dans lequel se déroulent les travaux de cette thèse, tout en précisant les objectifs et les attentes. Puis, un état de l'art sur les traitements des plantes est présenté pour bien connaître le système étudié. Comme les travaux de thèse utilisent des techniques d'aide à la décision, des définitions sur le problème d'ordonnancement et le couplage entre la simulation et l'optimisation sont présentés. Ensuite, la problématique scientifique de la thèse est exposée. Puis ce chapitre s'achèvera par une conclusion.

1.2 Le projet UV-ROBOT

Cette thèse a été réalisée en collaboration entre le laboratoire LINEACT de CESI et le laboratoire LISPEN des Arts et Métiers. Les travaux de la thèse sont réalisés dans le cadre d'un projet européen Interreg appelé « UV-Robot » (voir logo en Figure 1.2). Plusieurs partenaires originaires de 3 pays (France, Belgique et Angleterre) participent à ce projet, et partagent leurs connaissances pour développer et tester l'UV-Robot. L'objectif du projet est de développer un robot doté de lampes ultraviolets (UV) afin d'améliorer les stratégies de traitement existant et faire un bénéfice pour les agriculteurs,

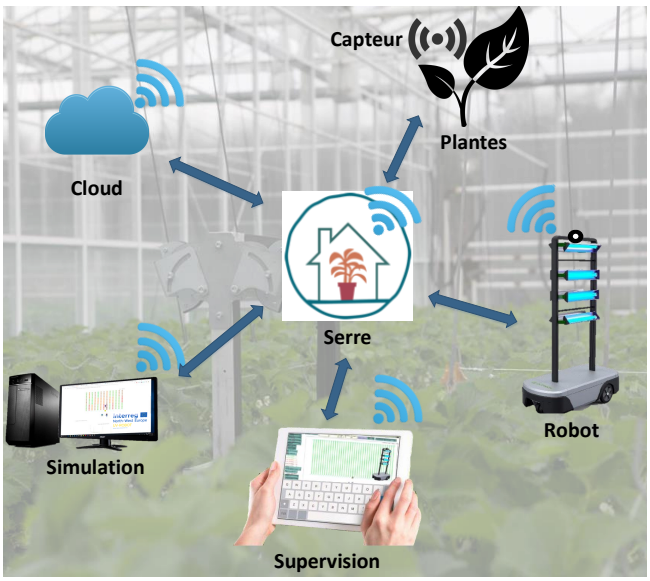

FIGURE 1.1 : Exemple d'Agriculture 4.0

les consommateurs et l'environnement.

L'équipe de chercheurs LINEACT CESI participe dans deux lots de travail (Work-Packages ou WP) du projet. Le premier est le WP Long Term (WPLT) où se focalisent les travaux de la thèse. Son objectif est de développer un simulateur capable d'imiter l'environnement de la serre avec le robot UV et de prédire le comportement des plantes dans la serre. Le deuxième est le WP T1 dont l'objectif est de développer une interface graphique de supervision qui permet aux agriculteurs de simuler les traitements, communiquer avec le robot et suivre l'état des plantes dans la serre. Le simulateur développé dans WPLT sera intégré dans l'interface graphique de WP T1.

Le but de ce projet est de créer un robot, qui porte des lampes ultraviolets de type C (UV-C), afin de traiter les plantes dans les serres contre une maladie appelée Mildiou [1]. C'est un type de champignon qui touche plusieurs plantes qui est largement connu dans différents types de récolte. Dans ce projet, l'application est limitée à cinq types de plantes à traiter : fraise, tomate, concombre, laitue et basilique. Le traitement actuel de mildiou est un traitement basé sur des pesticides. L'utilisation de la technologie UV-Robot vise donc à diminuer la pulvérisation des pesticides, et à protéger les plantes et l'environnement. Comme les robots sont autonomes et prennent de nombreuses décisions sur l'ordonnancement des trajectoires et la prioritisation des missions, il est nécessaire d'utiliser des outils d'aide à la décision qui permettent de planifier les traitements du robot dans les serres.

1.2.1 Le consortium du projet UV-Robot

Plusieurs partenaires travaillent sur ce projet afin de le mener à bien en accomplissant leurs missions. Leur objectif est de réduire l'utilisation des pesticides en utilisant un robot UV-C. Des réunions du projet sont régulièrement organisées, permettant de connaître l'avancement des travaux de chaque

1. Le mildiou est le nom générique d'une série de maladies cryptogamiques affectant de nombreuses espèces de plantes, mais prenant des proportions épidémiques dans certaines cultures de grande importance économique, telles que la vigne, la tomate, la pomme de terre, la laitue ou les courges.

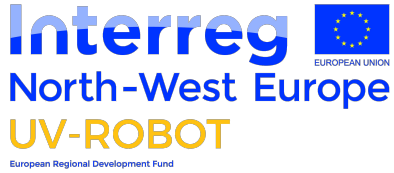

FIGURE 1.2 : Logo Interreg du projet UV-Robot

FIGURE 1.3 : Les logos des partenaires du projet UV-Robot

partenaire, et d'échanger sur les pratiques et idées concernant les stratégies de traitement par UV-C testés par chaque collaborateur. Les décisions prises lors de ces réunions avec les partenaires ont été essentielles et utiles pour adapter/modifier le développement des outils nécessaires et les mettre à jour.

La figure 1.3 présente les logos de tous les partenaires du projet. Il s'agit de 9 partenaires provenant de trois pays européens de la région nord-ouest (France, Belgique et Angleterre), qui sont :

- LINEACT de CESI : prend en charge la partie de la simulation du robot et le développement de l'interface graphique de supervision.
- Octinion : celui qui développe le robot UV-C et ses composants.
- Roboscientific : est le partenaire qui crée et développe le nez intelligent (e-noze) qui détecte les niveaux de maladie.
- Caté, CDDM, NIAB EMR, STC, et Proefstation voor de Groenteteelt : sont des centres de recherche dans le domaine agricole, ils permettent de préparer l'environnement pour tester le robot dans les serres.

1.2.2 Le robot UV-Robot

L'utilisation du robot pour traiter les plantes aux UV-C est essentielle pour éviter le risque de brûlures pour les humains lors de la manipulation des lampes UV-C. L'UV-Robot est développé par Octinion, l'un des partenaires belges du projet. Le robot porte des lampes UV-C des deux côtés comme le montre la figure 1.4. Il est également équipé d'un Nez intelligent (E-nose), qui est développé par Roboscientific, l'un des partenaires anglais. Le nez intelligent devrait permettre de mesurer les différents niveaux de maladie sur les plantes. Au début du projet, le robot avait une autonomie moyenne de 30 minutes avec un temps de charge de la batterie de 4 heures. Puis, après un an et demi de développement, Octinion a communiqué de nouvelles mises à jour du robot : une autonomie

FIGURE 1.4 : Robot mobile développé par Octinion pour le projet UV-Robot

moyenne de 3 heures avec un temps de charge de 2 heures et demi. Ces données ont été rapidement prises en compte car l'outil développé est facilement adaptable aux mises à jour.

Notons que presque la même technologie de robot, avec des lampes UV-C, existe dans d'autres travaux de recherche. Nous pouvons citer des robots équipés de lampes UV-C qui traitent des plantes dans des serres [Le et al., 2020], ou même dans les champs agricoles extérieurs [Cubero et al., 2020]. La particularité du robot utilisé dans les champs est d'avoir une couverture sur les lampes pour permettre aux plantes de bien absorber les lumières UV-C. Avec l'apparition de la pandémie du Coronavirus (COVID-19), beaucoup d'entreprises ont développé des robots équipés avec des lampes UV-C pour désinfecter les hôpitaux, les salles de sport ou les maisons [Ackerman, 2020], [Begić, 2017]. La figure 1.5 montre l'un des prototypes des robots désinfectant proposé par [Guettari et al., 2020].

1.3 Définition, Évaluation et Détection du mildiou

Cette section présente dans une première partie un état de l'art sur la maladie de mildiou, puisque c'est cette maladie qui sera traitée par UV-C avec le robot. Il faut connaître cette maladie et son comportement dans les cultures avant de se lancer dans la recherche de méthodes adéquates pour la modéliser et la maîtriser. Puis, en deuxième partie il y a l'évaluation de cette maladie dans le projet. Enfin, dans la troisième partie il y a les outils de détection de cette maladie qui sont utilisés dans les serres.

1.3.1 Maladie de mildiou

Tout comme l'oïdium, le mildiou est une maladie très répandue qui peut toucher toute plante et est susceptible d'endommager aussi bien ses feuilles, ses tiges, que ses fruits. Le mildiou et l'oïdium sont deux types de champignons de la même famille (phytopathogènes) qui se caractérisent différemment sur les plantes et présentent des symptômes similaires [Peries, 1962], [Zhang et al., 2018]. Le mildiou se caractérise par des tâches huileuses et grasses sous les feuilles. Ce champignon prend des proportions

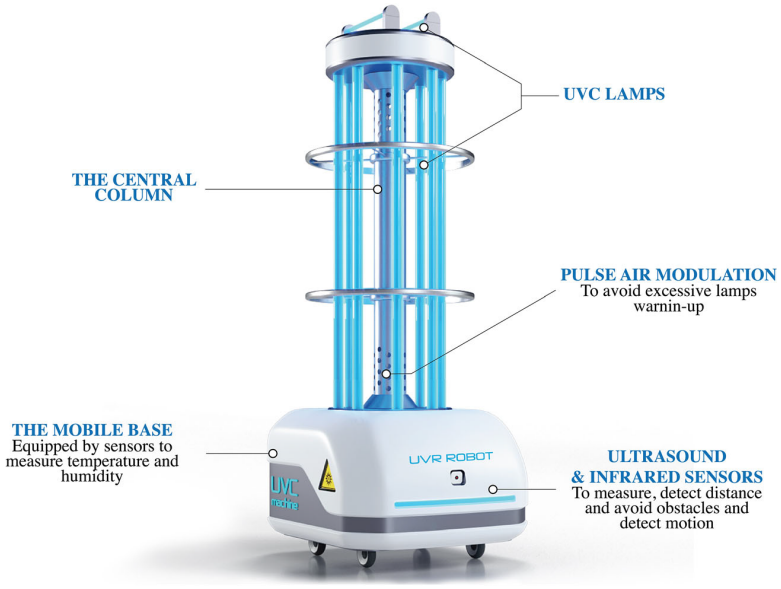

FIGURE 1.5 : Exemple de robot de désinfection par UV-C. Source : [Guettari et al., 2020]

épidémiques dans certaines cultures de grande importance économique, telles que la vigne, la tomate, la pomme de terre, la laitue ou les courges [Li et al., 2017]. L'oïdium, appelé également la maladie du blanc, peut se distinguer par une poudre blanche, comme une farine qui recouvre les plantes. Il s'attaque principalement à certaines espèces d'arbres comme le chêne, l'érable, le cognassier, le pommier ou l'aubépine qui y sont particulièrement sensibles [Jordan and Hunter, 1972], [Janisiewicz et al., 2016].

Pour mesurer le taux de champignon sur les cultures, l'Institut Français de la Vigne (IFV) utilise l'indicateur suivants : MILVIT [2] pour le mildiou, SOV [3] pour l'oïdium et EVA [4] pour le cochylis. Pour pouvoir mettre en place un modèle global de simulation du système, il est nécessaire de rechercher des modèles décrivant le comportement du mildiou sur les plantes. [Claude, 2007] a montré dans son article des graphiques du comportement du mildiou sur la vigne. Ces graphiques sont dans la figure 1.6, où l'évolution du champignon est très importante en 2007 par rapport aux autres années. En effet, la pluie, l'humidité et la température jouent un grand rôle dans l'évolution du mildiou. L'année 2007 a connu une pluviométrie supérieure à la moyenne, ce qui explique ces observations [Claude, 2007]. Ces courbes peuvent être utilisées pour établir un modèle réaliste de l'évolution de la maladie.

1.3.2 Évaluation de mildiou dans le projet

Dans les travaux de cette thèse, le niveau d'infection par la maladie est discrétisé sur six niveaux : 0, 3, 6, 12, 20 et 30, correspondant à des observations visuelles des agriculteurs. En effet, chaque

2. MILVIT est un modèle mildiou initié dès 1988 par le service de la Protection des Végétaux et opérationnel depuis 1993. C'est un modèle descriptif et quantitatif de la phase épidémique du mildiou.

3. SOV est un modèle créé en 1996 par le Service de la protection des végétaux de Languedoc-Roussillon. Il livrerait, à l'échelle d'un territoire, un indice global pertinent pour caractériser le risque oïdium de l'année avant le début de campagne.

4. EVA est un modèle du Service de la Protection des Végétaux, utilisé dans le cadre des avertissements agricole de cet organisme. Ce modèle, mis au point sur Eudémis, se révèle également intéressant sur Cochylis.

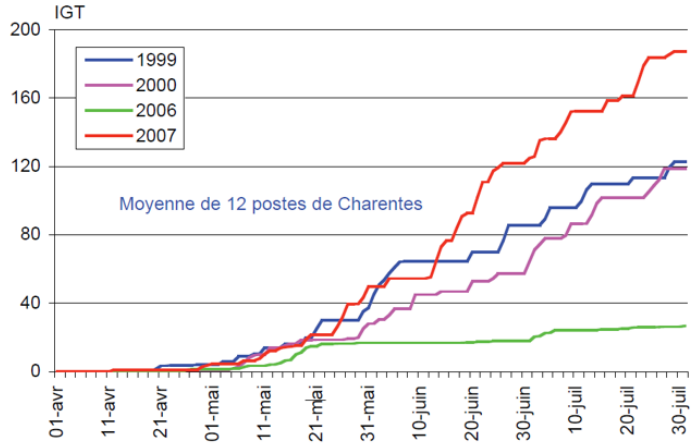

FIGURE 1.6 : Évolution de l'indice de risque Milvit sur 4 années. Source : [Claude, 2007]

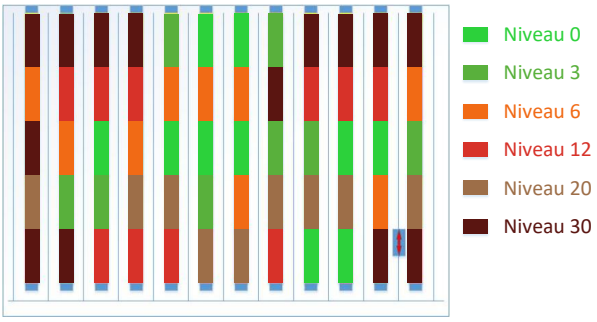

FIGURE 1.7 : Schématisation de la serre et des différentes niveaux d'infection de ses plantes.

niveau correspond a un stade de la maladie reconnaissable visuellement par l'agriculteur. Le niveau 0 représente l'absence d'infection. Ensuite, il y a le niveau 3 qui correspond aux premières apparitions de maladie sur les plantes. Puis, les niveaux intermédiaires 6, 12 et 20. Enfin, le niveau 30 est le plus critique car il représente le cas où le mildiou couvre la totalité des feuilles de la plante. Dans la figure 1.7, une serre est schématisée avec des couleurs qui montrent les différents niveaux de maladies. La figure illustre aussi le robot entre les rangées de la serre en train de faire le traitement.
Pendant le traitement, le robot change sa vitesse en fonction du niveau de maladie, pour que la plante reçoive la dose suffisante de radiation UV-C. Si le niveau de maladie est bas, la vitesse du robot est rapide, et vice-versa. Cependant, lorsque le robot diminue sa vitesse, la consommation de sa batterie augmente, car il y a besoin d'utiliser plus longtemps les lampes UV-C qui consomment plus d'énergie.

1.3.3 Détection de mildiou

Le mildiou est un champignon facile à détecter grâce au regard qui est une détection traditionnelle. Comme le montre la figure 1.8, à gauche il y a le mildiou qui ressemble à des tâches huileuses, et à

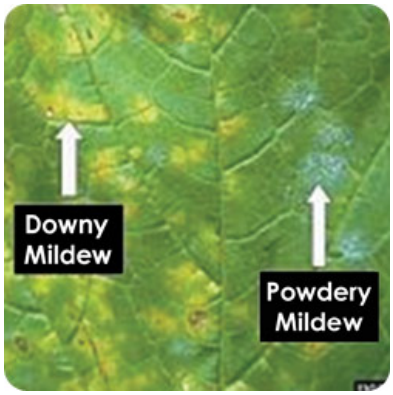

FIGURE 1.8 : Illustration de l'effet de mildiou (à gauche) et de l'oïdium (à droite) sur une feuille de vigne. Source : https ://en.wikipedia.org/wiki/Mildew

droite de la figure il y a l'oïdium qui est comme une poudre de farine.

Mais la méthode traditionnelle prend du temps pour les agriculteurs surtout dans les grandes serres ou champs. Grâce à la technologie, les scientifiques ont développé des outils qui permettent de détecter la maladie de mildiou. Il y a la détection par des tests de réaction en chaîne par polymérase (en anglais, Polymerase chain reaction (PCR)) [Thiessen et al., 2016]. D'autres revues, comme dans [Wspanialy and Moussa, 2016], travaillent sur la détection de mildiou avec des caméras qui prend en photo les feuilles des plantes. Puis, savoir si la maladie est présente sur les plantes ou pas à l'aide des algorithmes d'intelligence artificielle. La dernière technologie développée pour détecter le mildiou est le E-nose. C'est un nez intelligent qui absorbe des substances chimiques, puis il peut donner le niveau de mildiou sur les plantes [Gu et al., 2021].

1.4 Les robots agricoles

Depuis la nuit des temps, l'humain n'a cessé d'améliorer le domaine agricole. Il a commencé par utiliser les pierres taillées, puis les outils métalliques, ensuite les animaux pour l'aider à bien travailler sa terre. Ces derniers temps, l'homme utilise les machines d'agriculture qu'il a inventées. De nos jours, dans l'ère du numérique, la robotisation a une place majeure dans la société. Dans [Bonadies et al., 2016], les auteurs font un état de l'art sur les véhicules terrestres non habités utilisés dans le domaine agricole pour en accroître l'efficacité, en particulier en réduisant les besoins de main-d'oeuvre. Ces véhicules sans pilote sont utilisés à diverses fins, notamment pour l'échantillonnage du sol, la gestion de l'irrigation, la pulvérisation de précision, le désherbage mécanique et la récolte des cultures. Les facteurs pris en compte par les auteurs concernent les tendances futures et les problèmes potentiels des véhicules terrestres sans pilote, comme le développement, la gestion et la performance. Cette section présente une étude de l'état de l'art sur les types de robots utilisés en agriculture. Deux types de robots sont principalement utilisés : les robots pour le traitement des maladies, appelés robots pulvérisateurs (sprayers robots), et ceux utilisés pour la récolte, appelés robots récolteurs (harvesting robots). Plusieurs laboratoires travaillent dans ce domaine pour améliorer le secteur agricole en perfectionnant les robots.

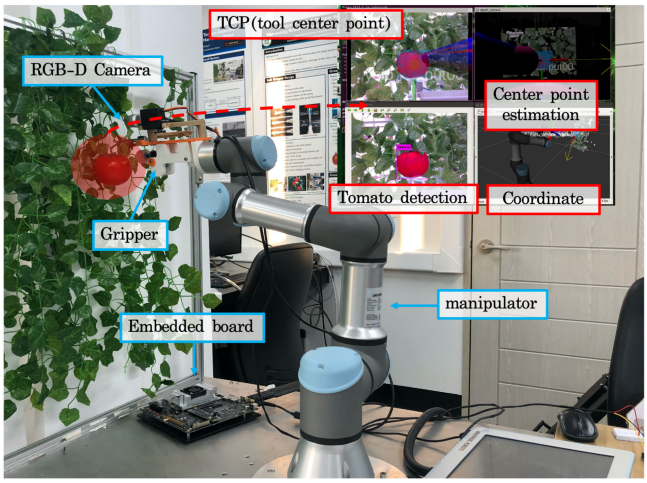

FIGURE 1.9 : Exemple de robot récolteur de tomate. Source : [Jun et al., 2021].

1.4.1 Robots récolteurs

Dans la littérature, de nombreuses recherches parlent des travaux de robotisation de l'agriculture ou des activités agricoles afin d'améliorer les récoltes. Le département du génie agricole de la station d'expérimentation agricole de la Louisiane (Etats-Unis d'Amérique) a mis au point un modèle de transplanteur semi-robotisé dans son laboratoire [Hwang and Sistler, 1985]. Le prototype ne pouvait transplanter qu'à un rythme moyen de six plantes par minute, ce qui représente un cinquième du taux pour un opérateur humain. Les auteurs ont donné une vue globale sur les machines agricoles du passé, du présent et du futur. Ils ont également donné une liste des laboratoires de recherche qui s'intéressent à la robotique agricole.

Les robots récolteurs, que l'on trouve dans littérature sous le nom « harvesting robots » (voir l'exemple dans la figure 1.9), sont utilisés le plus souvent lorsque la récolte est mature. Il existe différentes formes de robots récolteurs, chaque forme étant conçu pour récolter un type de plantes en particulier (différents légumes et fruits).

Les robots automatisés pour les récoltes ont été un sujet de recherche populaire au cours des 30 dernières années. Les besoins en main-d'oeuvre pour la récolte des fruits, légumes et autres cultures représentent une part importante des coûts de production. L'automatisation de la récolte de ces cultures a le potentiel de réduire considérablement les coûts, en partie, en réduisant ou en supprimant les besoins de main-d'oeuvre.

Énormément de recherches sont faites dans le domaine de la robotique agricole. [Sistler, 1987] cite plusieurs axes qui sont étudiés dans ce cadre, notamment l'irrigation régulée par robots afin de minimiser les pertes en eau lors de l'arrosage. Il aborde aussi les possibilités d'amélioration des robots récolteur en intégrant des capteurs fiables et précis pour mesurer le taux d'humidité du grain entrant dans la moissonneuse et la quantité de grain perdue à l'arrière de la machine. Ainsi, la machine pourrait être automatiquement ajustée pour répondre aux diverses conditions de récolte. Aussi, un capteur d'humidité du sol pourrait être monté sur un planteur pour placer les graines à la profondeur optimale par rapport à la quantité d'humidité présente.
[Van Henten et al., 2002] ont mis au point un véhicule de cueillette de concombres qui fait la récolte

et le transport vers une zone de stockage dans le but de remplacer le travail humain par plusieurs robots. Pour un fruit plus volumineux, [Sakai et al., 2008] ont conçu un véhicule autonome de récolte de pastèques pour démontrer la capacité des robots mobiles dans la récolte de cultures lourdes. Pour la récolte des pommes, [De-An et al., 2011] ont développé un robot récolteur entièrement autonome. Une configuration à chenille plutôt qu'à roues a été choisie pour ce robot, et les données GPS ont été utilisées pour la navigation à travers les vergers de pommiers. Un bras de récolte à cinq degrés de liberté a été conçu pour mener à bien la tâche de récolte. Un récolteur de fraises a été développé par [Feng et al., 2012]. Ce robot est constitué d'un bras manipulateur à six degrés de liberté avec des doigts de préhension pneumatiques à ventouse. Ce bras a été montée sur un véhicule à quatre roues motrices pour pouvoir se déplacer dans la serre. Le travail de [Southall et al., 2002] concerne un système de vision artificielle pour un véhicule autonome conçu pour traiter les cultures horticoles. Le véhicule navigue en suivant les rangées de cultures (plantes de chou-fleur individuelles) qui sont plantées dans un réseau raisonnablement régulier. En outre, [Zhang et al., 2002] donne un aperçu du développement mondial des technologies de l'agriculture de précision. Cela comprend plusieurs aspects tels que : la variabilité des ressources naturelles et leur gestion, l'impact des technologies de l'agriculture de précision sur la rentabilité et l'environnement, ainsi que les innovations techniques en matière de capteurs, de commandes et de télédétection.

1.4.2 Robots pulvérisateurs

La pulvérisation est le processus de distribution de produits phytosanitaires sur les cultures agricoles à différents stades du cycle de vie des plantes. Classiquement, cette activité est réalisée en utilisant un tracteur équipé d'une unité de pulvérisation, qui parcourt progressivement tout le champ afin d'effectuer la pulvérisation. La pulvérisation se distingue des autres tâches agricoles par sa redondance. Dans d'autres tâches agricoles (par exemple, le labour, l'ensemencement et la récolte), même si la redondance existe (c'est-à-dire traiter un point sur le terrain plus d'une fois), le coût reste acceptable comparé au gain de temps apporté par la robotisation. Lors de la pulvérisation, toute portion du terrain ne doit être traitée qu'une seule fois, car une distribution excessive des produits pulvérisés détruit la culture [Janani et al., 2016]. En faisant appel à la robotique, il est possible de simplifier cette tâche et éviter la pulvérisation excessive sur certaines parties du champ. Les robots pulvérisateurs (ou sprayers robots dans la littérature) sont des robots qui font la pulvérisation de pesticides à des moments précis de la vie des plantes (Figure 1.10). Ce type de robots est plus standard que les robots de récolte, car ils peuvent être utilisés pour pulvériser différents types de pesticides sur différents types de plantes.

Les robots pulvérisateurs ont suscité l'intérêt des chercheurs dans différents laboratoires dans le monde et plusieurs exemples sont présents dans la littérature. Ainsi, [Janani et al., 2016] ont proposé une stratégie coopérative pour permettre à une équipe de robots d'effectuer une pulvérisation sur un grand champ. L'équipe de [Oberti et al., 2016] a développé un robot agricole pour détecter l'oïdium sur les vignes et appliquer des pesticides pour réduire la maladie sur ces plantes (voir Figure 1.10). Les résultats expérimentaux de ce robot ont révélé une capacité à réduire l'utilisation de pesticides de 65 à 85%.

D'autres travaux de recherche ont été étudiés où un GPS est installé sur les robots pulvérisateurs pour améliorer la précision du déplacement du robot. Par exemple, le travail de [Talbot, 2014], qui a monté une start-up nommée Rowbort, présente le développement d'un robot pulvérisateur autonome

FIGURE 1.10 : Exemple de robot pulvérisateur. Source : [Oberti et al., 2016].

pour l'engrais azoté dans les cultures de maïs. Le robot permet de réduire la quantité de pollution par l'azote dans les cours d'eau après la pluie en appliquant des engrais azotés directement à la base de la plante. Dans [Sarri et al., 2017], les auteurs développent un prototype du système de télémétrie adapté aux viticulteurs pour suivre en temps réel les performances de leurs opérations de pulvérisation et acquérir des données utiles. Leurs résultats montrent que la pression de pulvérisation et le débit mesurés par les capteurs du système de télémétrie étaient similaires aux valeurs théoriques définies pour la régulation du pulvérisateur. [Gonzalez-de Soto et al., 2016] ont développé un véhicule autonome pour la pulvérisation de précision d'herbicides pour lutter contre les mauvaises herbes dans les cultures céréalières, telles que le blé. Les essais de performance du système automatisé dans un champ de blé ont montré que l'herbicide était appliqué sur 95 % des mauvaises herbes du champ. Cette plateforme pourrait également être utilisée pour des applications de pesticides et d'engrais.

1.5 Problème d'ordonnancement

L'ordonnancement joue un rôle central dans de nombreux secteurs industriels et agricoles tels que l'industrie pharmaceutique, pétrochimique, automobile, aérospatiale et agroalimentaire. L'ordonnancement des opérations dans une usine ou une serre consiste à déterminer le nombre de ressources et de tâches à traiter ainsi que l'affectation, le séquençage et la synchronisation des lots dans les unités de traitement, compte tenu des données relatives aux installations de production et à la disponibilité des ressources, afin d'optimiser un objectif donné. Les méthodes d'ordonnancement basées sur l'optimisation peuvent entraîner une augmentation des bénéfices des coûts, par rapport à l'ordonnancement utilisant des méthodes heuristiques. Il est donc important de développer des modèles d'optimisation qui peuvent représenter avec précision les caractéristiques souvent rencontrées dans les installations industrielles ou agricoles.

Dans un environnement dynamique, des perturbations ou de nouvelles informations peuvent rendre l'ordonnancement calculé sous-optimal ou infaisable, nécessitant ainsi un ré-ordonnancement. Pour ce faire, il est nécessaire d'utiliser toutes les informations en temps réel disponibles dans l'usine ou la

serre. En général, un processus de traitement se fait en une séquence d'étapes dont les transitions sont guidées par certaines conditions logiques. Récemment, [Rawlings et al., 2019] ont fait un premier effort pour intégrer des informations dérivées de l'automatisation sur les étapes dans le problème de l'ordonnancement. Dans notre travail, l'objectif sera de déterminer les tâches du robot autonome, notamment les rangées à visiter pour traiter la maladie.

Selon les gammes opératoires des produits, on distingue plusieurs problèmes d'ordonnancement d'atelier, qui correspondent à des organisations physiques différentes : sur une machine, sur plusieurs machines, à cheminement unique (flow-shop), à cheminements multiples (job-shop) et à cheminement libre (open-shop).

1.6 Couplage simulation et optimisation

La simulation permet d'exploiter l'espace du temps en simulant sur de larges horizons d'un système. Elle permet de voir le comportement des entités du système et leur interaction après un certain temps. L'utilisation de scénarios dans un simulateur permet de prévoir les événements clés des systèmes, et donne la possibilité de les éviter ou de les assumer en fonction de leur influence (positive ou négative) sur les systèmes étudiés. Cependant, l'optimisation permet de gérer l'espace d'état en prenant les bonnes décisions. En utilisant les bonnes méthodes et algorithmes, cette approche donne la meilleure solution à de nombreux problèmes de recherche opérationnelle.

Malgré leurs avantages, les deux disciplines présentent des inconvénients. La simulation a une myopie dans l'espace d'état, et l'optimisation a une myopie dans le temps. Le couplage de la simulation et de l'optimisation permet d'exploiter l'espace temporel et de gérer l'espace d'état en même temps. Ce couplage permet de simuler et d'optimiser l'ordonnancement dynamique des tâches du robot. La partie simulation donne la possibilité de suivre l'évolution du comportement stochastique des plantes, et l'optimisation permet de prendre la meilleure décision pour améliorer l'ordonnancement des tâches de traitement. La meilleure stratégie pour garantir le couplage est d'intégrer les algorithmes d'optimisation dans le simulateur et de les déclencher au bon moment.

1.7 Problématique

1.7.1 Thème de recherche

Comme l'indique le titre de cette thèse : simulation et optimisation de la gestion dynamique de tâches évolutives sur des robots mobiles autonomes, le thème de recherche de la thèse est la planification des tâches dynamiques et évolutives pour un robot. L'ordonnancement dynamique est utile lorsque l'état du système change avec le temps. Dans ce cas, la dynamique du système ne dépend pas seulement de l'arrivée des jobs, mais elle inclut le temps d'exécution des tâches qui ont un comportement stochastique.

1.7.2 Problème de recherche

On veut planifier des tâches dynamiques et évolutives pour être effectuées par une ou plusieurs ressources (robots). L'arrivée des tâches dépend de l'apparition de la maladie, et l'évolution stochastique de cette maladie augmente la difficulté de l'ordonnancement.
Comme précisé auparavant, la consommation de l'énergie du robot dépend de niveau de maladie des plantes. Donc, lorsqu'on planifie les tâches de traitement pour le robot, il a besoin de plus d'énergie si la maladie évolue entre-temps, c'est-à-dire entre la date de planification et la date d'exécution de la tâche. Afin de respecter la capacité de la batterie du robot, il faut planifier les tâches de traitement en prenant en compte l'évolution de la maladie.

1.7.3 Question de recherche

Avant de poser la question de recherche, il faut bien comprendre les spécificités de la problématique traité dans cette thèse. Dans la figure 1.11, un exemple d'un ordonnancement des tâches pour une ressource (machine) est illustré. Au début, il y a 5 tâches qui sont planifiées pour la machine, avec une durée totale d'exécution de 26 minutes. Après 9 minutes de traitement, la machine reçoit une nouvelle tâche (Tâche 6) pour l'exécuter. L'emplacement de la nouvelle tâche et son temps d'exécution vont décaler à droite certaines tâches et augmenter le temps total de traitement de la machine. Dans l'ordonnancement dynamique, les temps d'arrivée et de départ des jobs ne sont pas connus au préalable. Afin d'augmenter la précision de l'ordonnancement il faut estimer ces temps à travers la prédiction (par anticipation) ou faire un ré-ordonnancement après l'arrivée des jobs.

La figure 1.12 montre un deuxième exemple d'ordonnancement des tâches pour une ressource « robot » (cet exemple est lié aux travaux de cette thèse). Avant de planifier les tâches pour le robot, il faut prendre en compte l'évolution de la maladie, car elle augmente le temps d'exécution des tâches et la consommation d'énergie du traitement. L'évolution des tâches et leur temps d'exécution dépendent du comportement de maladie qui est montré dans le graphe en haut à droite de la figure 1.12. L'apparition de maladie dans des plantes saines correspond à l'arrivée d'une nouvelle tâche. Dans l'ordonnancement dynamique des tâches évolutives, les dates d'arrivée des tâches plus leurs évolutions sont à gérer pour optimiser la planification des tâches.

Les questions de recherche qui s'imposent ici sont :
QR1 : Comment faire pour planifier les traitements pour le robot sachant que les niveaux de maladies sont dynamiques ?

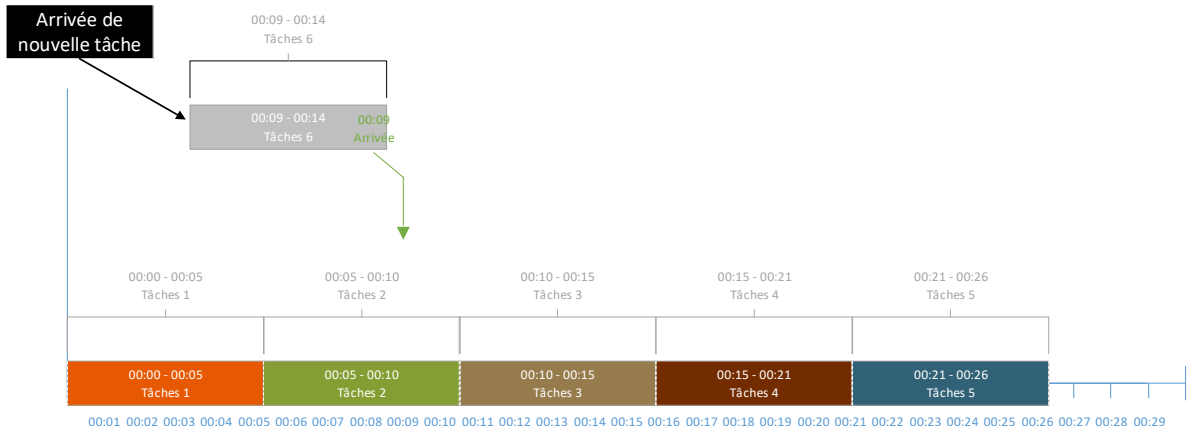

FIGURE 1.11 : Ordonnancement dynamique

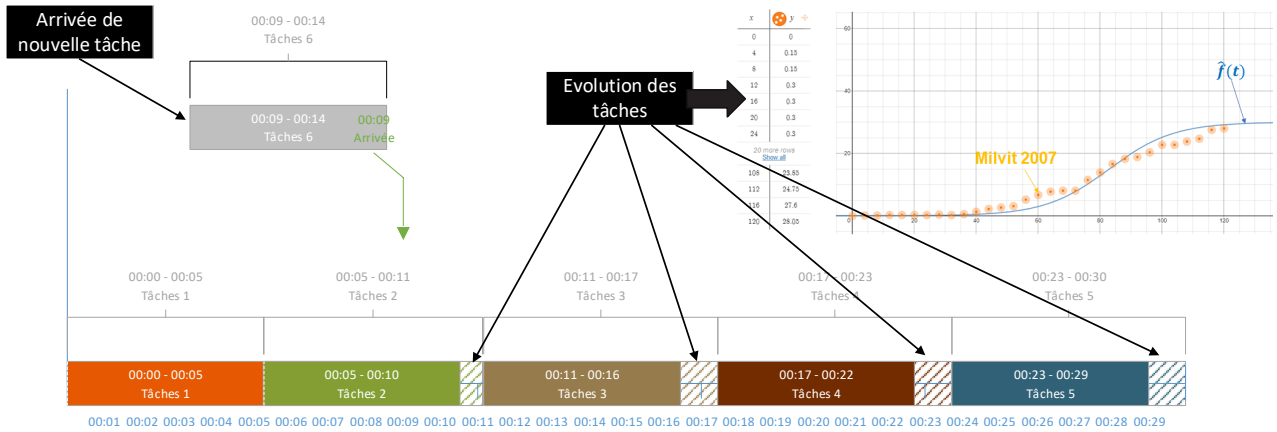

FIGURE 1.12 : Ordonnancement des tâches évolutives

QR2 : Comment faire un ordonnancement optimal des tâches tout en respectant la contrainte de la capacité de batterie ?

1.7.4 Hypothèse de recherche

La résolution de cette problématique demande des outils d'aide à la décision adéquats. L'étude du couplage entre la simulation et l'optimisation permet d'explorer l'espace d'état par l'optimisation et de prédire l'état du système par la simulation. L'optimisation aide à trouver les meilleures solutions pour le problème à chaque instant. La simulation permet de gérer le cas dynamique du système et prévoir le processus d'évolution de la maladie.

1.8 Conclusion

L'objectif de ce travail est de résoudre la problématique d'ordonnancement des tâches évolutives dans le cas d'application qui concerne le traitement de la maladie de mildiou dans les serres agricoles

avec la lumière UV, en utilisant un robot mobile autonome nommé UV-Robot. L'état de l'art sur le domaine agricole et les traitements des plantes permettent de bien connaître le contexte dans lequel se déroulent les travaux de la thèse. Puisque le but est de faire un ordonnancement des tâches de traitement pour le robot UV-C, et que ces tâches ont des durées stochastiques, une définition sur le problème d'ordonnancement a été fait, pour donner une ouverture vers des méthodes de résolutions du problème considéré.

En général, les modèles d'optimisation sont intelligents mais moins flexibles, tandis que les modèles de simulation sont flexibles mais comportent des processus de décision basique (souvent à base de seuils) [Wu et al., 2003]. Le couplage entre la simulation et l'optimisation aide à gérer, à la fois, la myopie de l'optimisation par rapport à la prédiction dans le temps, et la myopie de la simulation en ce qui concerne l'exploration d'espace d'état.

Chapitre 2

Modélisation et simulation multi-agent du traitement robotisé du mildiou

Contenu

2.1 Introduction

Depuis plusieurs décennies, les scientifiques utilisent la simulation, que ce soit pour la prédiction de phénomènes naturels, l'amélioration des calculs au niveau des machines ou, plus généralement, l'étude de systèmes réels ou artificiels. La simulation se situe à l'intersection de plusieurs disciplines scientifiques (mathématique, probabilité, physique et informatique) et est elle-même composée de plusieurs méthodes.

Avant de se lancer dans un projet, on a de plus en plus tendance à faire des calculs pour déterminer les tâches à accomplir et obtenir de bons résultats. Parfois, ces calculs analytiques sont trop complexes et/ou ne sont pas suffisants, d'où l'idée d'utiliser la simulation. Elle permet de prédire le déroulement d'un ou de multiples comportements ou phénomènes d'un système. La simulation permet de prédire le comportement d'un système et d'anticiper les éventuels problèmes qu'il peut générer ou subir. Un simulateur bien construit, avec des données et des contraintes bien définies, permet d'obtenir des résultats très proches de la réalité. Dans la plupart des cas, il est nécessaire de lancer l'algorithme de simulation plusieurs fois pour affiner les paramètres du simulateur et se rapprocher de la bonne solution à chaque étape.

Dans ce chapitre, les méthodes de simulation seront présentées dans la partie état de l'art. Puis, la modélisation de la simulation avec le Système Multi-Agents (SMA)s, et la modélisation du comportement de la maladie sont expliquées. L'interface de simulation est paramétrable afin de mettre en œuvre différents plans d'expériences et de tester plusieurs scénarios montrant l'effet du traitement UV-C sur l'évolution de la maladie. Ensuite, la partie développement du simulateur sera détaillée, où il y aura l'explication de tout le code de la simulation avec des logigrammes. Enfin, ce chapitre se termine par une conclusion.

2.2 État de l'art

La simulation est « le processus de conception d'un modèle d'un système réel et de réalisation d'expériences avec ce modèle dans le but de comprendre le comportement du système ou d'évaluer diverses stratégies (dans les limites imposées par un critère ou un ensemble de critères) pour le fonctionnement du système », [Shannon and Johannes, 1976].

2.2.1 La simulation stochastique ou numérique

Un modèle stochastique est une description de la réalité sous forme d'un ensemble de paramètres numériques et d'un ensemble de relations mathématiques qui décrivent la manière dont certains de ces paramètres, appelés causes, agissent sur d'autres paramètres appelés effets [Drogoul, 1993].

La simulation stochastique permet de déterminer les stratégies optimales dans un contexte statique. Elle nécessite en contre partie une simplification excessivement réductrice des agents et des conditions d'organisation de l'échange économique complexe, spécialement en ce qui touche à leur agencement temporel [O'Driscoll Jr and Rizzo, 2002]. Ce type de simulation a été utilisé pendant des années pour appliquer des traitements analytiques des modèles mathématiques [Troitzsch, 2009].

2.2.2 Simulation à événements discrets

La simulation à événements discrets, Discrete-Event Simulation (DES) en anglais, est une technique dans laquelle le moteur de simulation joue un historique suivant une chronologie d'événements [Zeigler et al., 2000], [Wainer, 2009]. La technique est dite à événements discrets parce que le traitement de chaque événement de la chronologie a lieu à des points discrets d'une chronologie. L'heure virtuelle de la simulation ne nécessite aucune synchronisation avec le temps réel. Cela permet de prédire des phénomènes futurs ou d'étudier des processus complexes qui se déroulent en peu de temps.

Plusieurs modèles et langages de simulation ont été développés pour la DES, comme le langage populaire SIMULA [Dahl and Nygaard, 1966]. Les approches les plus avancées pour formaliser la DES ont ajouté des sémantiques temporelles à des approches de modélisation statique bien connues telles que les automates programmés et les processus semi-markoviens généralisés (GSMP). D'autres formalismes centrés sur les problèmes de simultanéité ont émergé comme les réseaux de Petri, le calcul des systèmes de communication (CCS), et la communication des processus séquentiels (CSP) [Vicino, 2015].

2.2.3 Formalisme de spécification du système à événements discrets (DEVS)

[Zeigler et al., 2000] proposent le formalisme DEVS (Discrete Event System Specification) dans les années 70 pour permettre la formalisation de modèles modulaires et hiérarchiques. Ce formalisme est basé sur la théorie des systèmes ; il permet une représentation formelle de modèles susceptibles de manipulations mathématiques comparables aux équations différentielles pour les systèmes continus. Dans le formalisme DEVS, un modèle est vu comme une boîte noire qui reçoit et émet des messages sur ses ports d'entrées ou de sorties [Federici, 2006].

Le formalisme DEVS manipule les concepts de structure, d'ensemble et de fonction mettant en relation les différents éléments de ces ensembles. L'un des avantages principaux du formalisme DEVS est qu'il spécifie également les mécanismes de simulation de ses modèles. En effet, le simulateur DEVS va exécuter les fonctions de transition des modèles atomiques et gérer la communication entre les modèles à partir d'un arbre de simulation [Bouanan, 2016].

La popularité de DEVS dans le monde de la recherche académique vient principalement du fait qu'il permet d'appréhender un système de manière comportementale et structurelle : DEVS est modulaire et hiérarchique [Garredu, 2013].

2.2.4 Simulation par systèmes multi-agents

Un SMA est un système composé d'un ensemble d'agents (un processus, un robot, un être humain, une plante, etc.), actifs dans un certain environnement et interagissant selon certaines règles. Un agent est une entité caractérisée par le fait qu'elle est, au moins partiellement, autonome, ce qui exclut un contrôle centralisé du système global.
Le domaine des SMAs est actuellement un domaine de recherche qui suscite beaucoup d'intérêt. Ce domaine est né à la fin des années 70 et début des années 80, de l'idée de distribuer les connaissances et le contrôle dans les systèmes d'Intelligence Artificielle. Cette idée a émergé d'une part du besoin de faire face à la complexité croissante de systèmes et a été favorisée d'autre part par l'émergence des modèles et des machines parallèles, rendant possible la mise en oeuvre opérationnelle de la distribution [Hassas, 2003].

[Ferber, 1999] classe les recherches sur les SMAs au travers de deux objectifs majeurs :
— L'analyse théorique et expérimentale des mécanismes d'auto-organisation : en expliquant, modélisant et simulant des phénomènes naturels.
— La réalisation d'artefacts logiciels ou matériels distribués capables d'accomplir des tâches complexes : grâce à la réalisation de systèmes informatiques complexes via les concepts d'agents, de communication, de coopération et de coordination d'actions.

L'intérêt du SMAs dans ce travail est dans le cadre de la simulation qui permet de représenter facilement le comportement des populations tel que c'est le cas des plantes d'une même serre. Ils permettent également de séparer les entités qui interviennent dans le système et de donner le niveau d'intelligence et d'autonomie nécessaire à chaque agent. Dans le système étudier, le robot et le superviseur sont présentés comme des entités actives avec deux niveaux d'intelligence différents. Nous allons essayer d'évoquer les différents concepts liés au domaine avant de considérer les SMAs comme un outil de simulation.

[Tranier, 2007] décrit l'environnement d'un SMA comme étant le contexte dans lequel les agents vont évoluer. Il fournit un support commun aux actions des agents, permettant ainsi l'interaction dans le système, il constitue une source d'information à laquelle les agents peuvent accéder au travers de leur perception.

Pour [Hassas, 2003], un SMA est un ensemble d'entités (physiques ou virtuelles) appelées agents, partageant un environnement commun (physique ou virtuel), qu'elles sont capables de percevoir et sur lequel elles peuvent agir. Les perceptions permettent aux agents d'acquérir des informations sur l'évolution de leur environnement, et leurs actions leur permettent, entre autres, de modifier l'environnement. Les agents interagissent entre eux, directement ou indirectement, et exhibent des comportements corrélés créant ainsi une synergie permettant à l'ensemble des agents de former un collectif organisé.

[Ferber, 1999] appelle SMA un système composé :
— D'un ensemble d'agents ;
— D'un environnement dans lequel sont immergés les deux précédents ensembles ;
— D'un ensemble de relations qui unissent les agents entre eux ;
— D'un ensemble d'opérations permettant aux agents de percevoir, produire, consommer, transformer et manipuler des objets ;
— D'un ensemble d'opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification.

Les SMAs permettent particulièrement bien de modéliser et de simuler des systèmes complexes. Ils permettent d'avoir une vue globale avec des prévisions de l'état du système sur plusieurs scénarios qu'on veut adapter. Cela permet de faire le meilleur choix possible parmi les solutions simulées les plus performantes.

Afin de simuler des systèmes complexes, il faut d'abord les modéliser pour pouvoir reproduire leur comportement. Ensuite, l'exécution du modèle mathématique obtenu permet de prévoir le déroulement des systèmes avant de passer par la pratique, et elle permet de faire les meilleurs choix pendant la simulation, dans plusieurs cas aléatoires.

Plusieurs travaux sur la simulation de systèmes complexes ont été trouvés au cours de la recherche, certains d'entre eux seront présentés dans ce qui suit.
Dans [Dahane et al., 2015] et [Sahnoun et al., 2015], les auteurs ont adopté l'approche multi-agent pour prévoir des comportements différents pour les parcs éoliens offshore et améliorer leurs stratégies

d'exploitation et de maintenance. Ils ont essayé différentes approches pour trouver la meilleure façon d'utiliser chaque boite de vitesses de chaque éolienne. En effet, la boîte de vitesses est considérée comme l'une des composantes qui cause des pannes la plupart du temps, comme l'ont constaté plusieurs autres chercheurs comme [Hameed et al., 2009] et [Byon and Ding, 2010].
La difficulté de l'entretien des parcs éoliens offshores est liée à la météo et à la disponibilité de ressources humaines adéquatement qualifiées, de pièces de rechange, de bateaux appropriés et de grues.
Pour faciliter la tâche de simulation, [Dahane et al., 2015] et [Sahnoun et al., 2015] ont développé un SMA qui est divisé en sept parties inter-connectées. Chaque partie du système est constituée d'un ou plusieurs agents autonomes. Les modèles des SMAs sont organisés de manière hétérogène, ce qui assure la distribution contrôlée des décisions entre les entités au même niveau hiérarchique et utilise une approche multi-agent pour bénéficier de leur comportement dynamique.

Le problème majeur de ces modèles de systèmes hétérarchiques (un système qui favorise l'interrelation et la coopération entre les membres plutôt qu'une structure ascendante) est le comportement myope, qui rend difficile la fourniture de résultats efficaces et de mécanismes d'optimisation. Bien que ce comportement myope puisse parfois s'avérer bénéfique pour l'individu, il est préjudiciable au succès à long terme de l'ensemble du système [Trentesaux, 2009]. Pour faire face à ce comportement myope, le choix de coupler optimisation et simulation semble très pertinent. Pour cela, des algorithmes d'optimisation sont intégrés dans les processus de décision de certains agents. En effet, la combinaison de l'optimisation et de la simulation permet de pallier les défauts des deux et d'offrir de meilleurs résultats en termes de comportement et de fidélité au système réel.

	Technique de simulation utilisée		
Référence	SED	DEVS	SMA
[Dahl and Nygaard, 1966]	X		
[Zeigler et al., 2000]	X		
[Hassas, 2003]			X
[Federici, 2006]		X	
[Tranier, 2007]			X
[Garredu, 2013]		X	
[Zankoul et al., 2015]	X		X
[Vicino, 2015]	X		
[Dahane et al., 2015]			X
[Bouanan, 2016]		X	
[Sahnoun et al., 2019]			X

SED : Simulation à Événements Discrets ; DEVS : Spécification des systèmes à ÉVénements Discrets ; SMA : Simulation multi-agents (SMA)

TABLE 2.1 : Techniques de simulation

Le tableau 2.1 résume les choix de certaines revues sur les techniques de simulation utilisées. La méthode DEVS est utilisée dans le cas où il y a une hiérarchie dans les Système Complexe (SC)s. Cette méthode permet de modéliser, simuler et analyser les systèmes à événements discrets représentés par des fonctions de transition d'état ou les systèmes continus représentés par des équations différentielles. Le formalisme DEVS ne sera pas utilisé dans les systèmes étudiés dans cette thèse, car il ne contient pas de hiérarchies. Les travaux de [Zankoul et al., 2015] permettent de faire la différence

entre l'utilisation de DES ou SMAs, puisqu'ils ont testé les deux modèles. La conclusion du travail de [Zankoul et al., 2015] est que l'utilisation des SMAs est meilleure que DES pour simuler et modéliser plusieurs entités intelligentes (le cas de plusieurs plantes dans le système étudié dans cette thèse). Ces conclusions ont facilité le choix de modéliser le système UV-Robot avec les SMAs qui sera présenté dans la section suivante.

2.3 Modélisation

Dans cette section, le modèle de simulation basé sur les SMAs sera détaillé, ainsi que le modèle de comportement de la maladie utilisé dans le simulateur.

2.3.1 Systèmes multi-agents

Dans le cadre de cette thèse, l'étude est faite sur un environnement composé d'une serre qui contient des plantes, un robot, une station de charge, un superviseur et un horticulteur. Pour représenter cet environnement, un simulateur basé sur les SMAs a été développé avec Netlogo qui est un langage basé sur JAVA et SCALA [Wilensky and Evanston, 1999]. Le simulateur permet de développer l'environnement à simuler avec des lignes de code ce qui le rend plus gérable, personnalisable et configurable. Les SMAs permettent de diviser le système en plusieurs agents et de suivre le comportement de chaque agent [Mazar et al., 2018]. La figure 2.1 montre le modèle SMA du système UV-Robot, qui a été divisé en sept agents : agriculteur, superviseur, robot, lampe UV-C, plante, serre et station de charge. Les agents sont liés les uns aux autres par leurs interactions, qui sont les suivantes :

- Propose des missions : l'agriculteur peut définir et contrôler l'exécution des missions en interagissant avec l'agent Supervision.
- Planifie les missions : le superviseur (agent Supervision) planifie des missions pour le robot.
- Envoie des données : le robot envoie des informations (état de la batterie, emplacement, niveau de maladie des plantes) à la supervision.
- Commande : en fonction de présence, ou pas, de la maladie dans les plantes, le robot allume et éteint les lampes UV-C.
- S'installe sur : chaque lampe UV-C est installé sur un robot.
- Traite : le robot traite les maladies sur les plantes en utilisant les lampes UV-C.
- Se déplace dans : le robot se déplace dans la serre.
- Se recharge : après chaque mission le robot retourne à la station de charge pour se recharger.
- Se développe dans : les plantes sont dans la serre où elles évoluent et elles sont traitées.
- Se pose dans : la station de charge se trouve dans la serre.

2.3.2 Comportement de maladie

L'évolution du niveau d'infection de la plante par le mildiou influence directement sur la dose UV-C à appliquer, c'est-à-dire la durée du traitement. Pour ajuster les doses de traitement UV-C, le robot change sa vitesse en fonction du niveau d'infection de la plante. Lorsque le niveau d'infection est élevé, le robot traite la plante à basse vitesse. Ainsi, la plante reçoit une dose suffisante de rayonnement UV-C. Par conséquent, la consommation d'énergie du robot est proportionnelle à la dose de traitement appliquée. Comme les lampes UV-C représentent la plus grande partie de la consommation d'énergie du

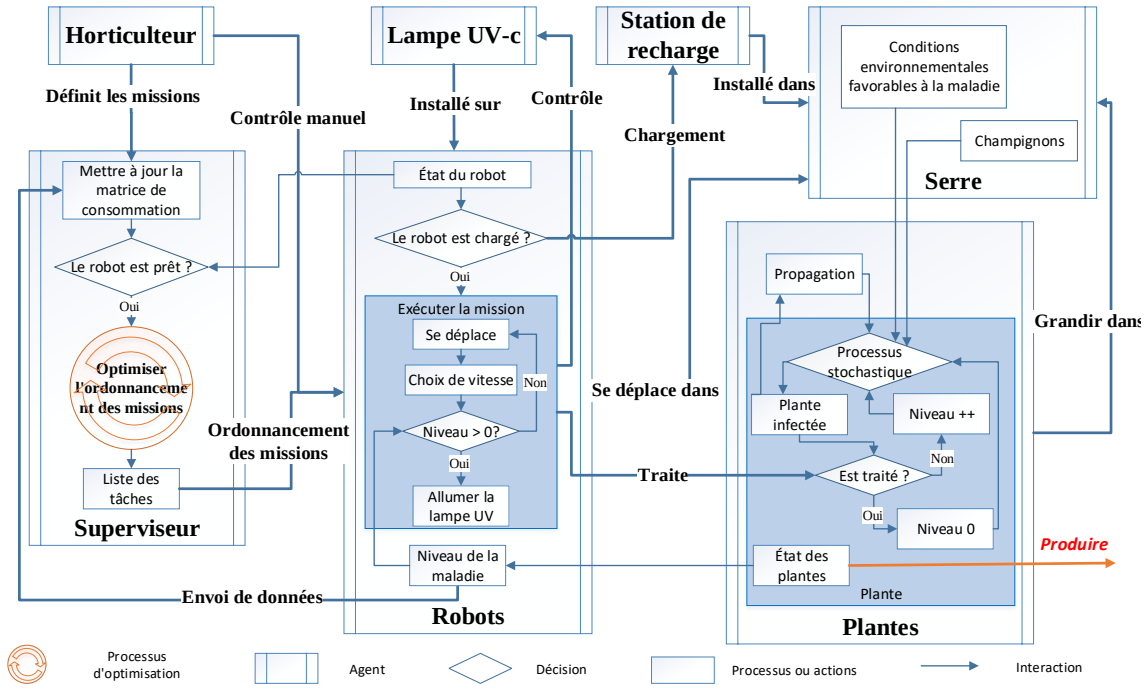

FIGURE 2.1 : Modèle de simulation basé sur les systèmes multi-agents

robot, lorsque le robot se déplace lentement avec des lampes activées, elles consomment plus d'énergie alors que la consommation de déplacement est beaucoup plus faible.

La reproduction du comportement de la maladie dans le simulateur est le point le plus important de l'environnement du système UV-Robot. Car pour étudier le processus stochastique de l'évolution du mildiou dans les serres, la simulation de son comportement doit être proche de la réalité. Le comportement du mildiou a été simulé en deux étapes. D'abord, une estimation avec la fonction logistique à trois paramètres [Magableh, 2006] a été étudiée qui permet d'obtenir des probabilités uniformes pour toutes les plantes. Ensuite, un modèle markovien a été développé pour avoir un comportement local de la maladie spécifique à chaque plante permettant de reproduire un comportement global cible de la serre [Kemeny and Snell, 1976].

Fonction logistique à trois paramètres

Afin de calibrer correctement les algorithmes de résolution du système, le comportement du mildiou doit être simulé pour se rapprocher de la réalité. À cette fin, les données de [White, 2012] sont utilisées, ces données représentent l'évolution du niveau de mildiou au fil du temps dans des serres de tomates en Espagne et aux Pays-Bas. La figure 2.2 montre les courbes de comportement du mildiou en pourcentage de surface foliaire malade au fil des jours.

La courbe rouge (serre d'Espagne) a été choisie pour la suite des travaux, car cette courbe est la plus proche des serres des partenaires du projet UV-Robot. La fonction qui correspond le mieux à cette courbe est une fonction logistique à trois paramètres. Elle ressemble à une sigmoïde, mais elle

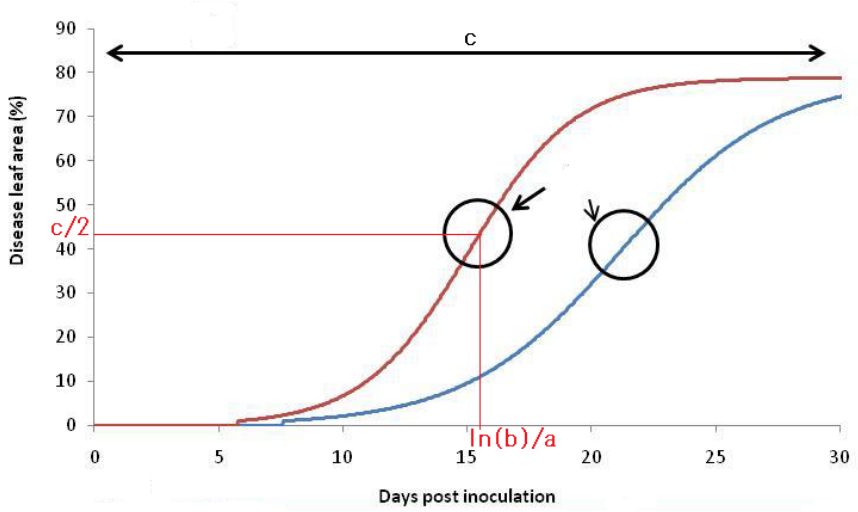

FIGURE 2.2 : Comportement de mildiou de [White, 2012] en Espagne (rouge) et aux Pays-Bas (bleu).

n'est pas symétrique lorsque $x = 0$. La fonction sigmoïde est une fonction logistique à trois paramètres avec $a = 1;\ b = 1\ et\ c = 1$. L'équation (2.1) montre la fonction d'estimation $\hat{f}(t)$, où les paramètres a, b et c sont à définir.

$$\hat{f}(t) = \frac{c}{1 + b \times e^{-a \times t}} \tag{2.1}$$

Le paramètre c est la limite de la fonction à l'infini (la courbe converge sous une asymptote horizontale). Comme le montre la figure 2.2, la valeur expérimentale du niveau de la maladie à l'infini est de 80% du niveau maximum. Le niveau maximal de la maladie dans le cas de cette étude est de 30, donc 80% de ce niveau est égal à 24, ce qui représente la valeur $c = 24$ dans la relation 2.1. De plus, la fonction est symétrique autour de son point d'inflexion avec pour abscisse $\frac{ln(b)}{a}$ et pour ordonnée $\frac{c}{2}$. Pour calculer a et b il suffit de prendre un point dans la courbe rouge du graphique de la figure 2.2. On prend le point $(20; 72)$, 72% qui représente le niveau $21,6$ dans le cas des niveaux de la maladie d'UV-Robot, on a donc besoin du point $(20; 21,6)$ pour l'équation (2.2). Et le point d'inflexion de l'abscisse qui est égal à 15 pour l'équation (2.3). Ensuite, on résout les deux équations.

$$\frac{c}{1 + b \times e^{-20a}} = 21,6 \tag{2.2}$$

$$\frac{ln(b)}{a} = 15 \tag{2.3}$$

Après la résolution des équations (2.2) et (2.3), on trouve $a = 0,44$, $b = 735$ et $c = 24$. Ainsi $\hat{f}(t)$ est donné par l'équation (2.4).

$$\hat{f}(t) = \frac{24}{1 + 735e^{-0,44t}} \tag{2.4}$$

Pour obtenir une simulation du comportement du mildiou similaire à celle représentée par l'équation (2.4), nous procédons à l'essai empirique de plusieurs fonctions de probabilité pour la transition des

taux d'infection des plantes du niveau actuel au niveau supérieur. Plusieurs tests sont effectués à l'aide du simulateur et la fonction de probabilité de transition sélectionnée est donnée dans l'équation (2.5).

$$P = (0,0004 \times level\_mildew \times last\_treatment) + (r \times 0,01) \tag{2.5}$$

Dans l'équation (2.5), *level_mildew* est le niveau actuel de taux d'infection en mildiou de la plante et *last_treatment* est le nombre de jours écoulés depuis son dernier traitement. La propagation des maladies entre plantes voisines est introduite par la variable booléenne r où $r = 1$ s'il y a une plante infectée dans un rayon de 3 mètres et, $r = 0$ sinon.

Il existe 6 niveaux de maladie (0, 3, 6, 12, 20 et 30) définis par les partenaires du projet UV-Robot. L'évolution de la maladie dans le temps n'est pas linéaire et peut être assimilée à la fonction $\hat{f}(t)$. La vitesse d'augmentation de la maladie est donné par la dérivée de la fonction d'évolution $d\hat{f}(t)/dt$. Par exemple, dans la figure 2.2, la vitesse est faible au début (entre le jour 0 et le jour 5) et à la fin (après le jour 25), alors qu'elle est beaucoup plus élevée autour du point d'inflexion (jour 15).

En utilisant la fonction $\hat{f}(t)$, on arrive à reproduire le même comportement de maladie présenté dans [White, 2012]. Cependant, la fonction est la même pour toutes les plantes de la serre. Dans la réalité, chaque plante a son propre comportement et, selon les horticulteurs, les plantes en bordure de la serre ont une évolution plus rapide de la maladie par rapport aux plantes du centre. Pour avoir un comportement de la maladie proche de la réalité, il est nécessaire de modéliser un comportement spécifique pour chaque plante. La solution proposée pour cette étape est de trouver une matrice de transition markovienne pour chaque plante. En calibrant les matrices markoviennes avec plusieurs tests de simulation. Le comportement de la maladie est hautement stochastique, car il est défini par 6 niveaux de maladie, ceux-ci peuvent représenter 6 états différents pour chaque plante. Il est facile de représenter le comportement des plantes lors d'une transition d'un état à un autre. Cela peut être représenté par une machine à état qui est modélisée par des réseaux bayésiens ou plus simplement par une chaîne de Markov.

Matrice de transition Markovienne

L'objectif d'utiliser des matrices de transition Markovienne est d'avoir un modèle d'un comportement dédié pour chaque plante [Mazar et al., 2021]. On veut aussi que le comportement moyen de l'évolution de maladie dans la serre soit similaire à la fonction $\hat{f}(t)$. La figure 2.3 présente le schéma qui représente les six niveaux de maladie par six états et les probabilités de transition entre tous les états.

Par conséquent, la forme des matrices qu'on veut construire est donnée par 2.6.

$$\begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} & P_{15} & P_{16} \\ 0 & P_{22} & P_{23} & P_{24} & P_{25} & P_{26} \\ 0 & 0 & P_{33} & P_{34} & P_{35} & P_{36} \\ 0 & 0 & 0 & P_{44} & P_{45} & P_{46} \\ 0 & 0 & 0 & 0 & P_{55} & P_{56} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

Dans la réalité, la maladie est plus active et plus présente sur les bords de la serre qu'au centre. Ceci est dû au manque d'aération et de luminosité en bordure ainsi qu'à une mauvaise régulation de

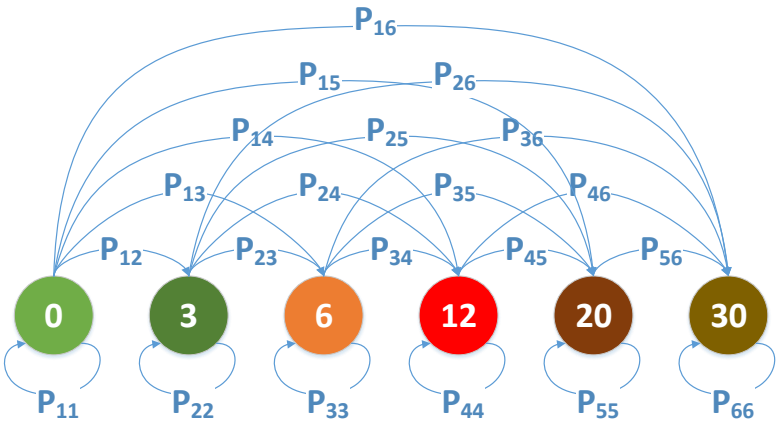

FIGURE 2.3 : Les probabilités de transition sur les niveaux de mildiou

la température et de l'humidité. Afin de représenter ce comportement, des matrices dépendant de la position de la plante ont été développées. La construction des matrices se fait lors de l'initialisation des agents `Plants`, chacune ayant sa propre matrice. Pour chaque matrice, le simulateur prend en compte la distance de la plante concernée par rapport au centre de la serre. Ensuite, il génère une valeur qui augmente la probabilité d'évolution de la maladie pour chaque distance supérieure. Cette valeur est calculée avec la fonction linéaire croissante définie par l'équation (2.7). Pour simplifier, nous avons supposé que la variation du niveau de la maladie en fonction de sa distance par rapport au centre de la serre est une fonction linéaire. La variable x dans la fonction $g(x)$ représente la distance de la plante par rapport au centre de la serre. La première probabilité est le produit de la fonction $g(x)$ par β qui est une variable d'ajustement définie empiriquement afin de faire correspondre le comportement global de la serre entière au comportement observé dans la littérature. L'ajustement de la variable β est très important car toutes les probabilités de la matrice sont dérivées de cette variable.

$$g(x) = 0,1x + 0,5 \tag{2.7}$$

La suite de constructions de la matrice se fait de la façon suivante :
— $P_{12} = g(x) \times \beta$
— $P_{23} = P_{12} \times 3$
— $P_{34} = P_{12} \times 6$
— $P_{45} = P_{12} \times 12$
— $P_{56} = P_{12} \times 20$

Le triangle de la matrice qui se trouve au-dessus des valeurs précédentes se calcule avec l'équation (2.8).

$$P_{ij} = \frac{P_{ij-1}}{10 \times j} \tag{2.8}$$

Les valeurs de la diagonale sont calculées à la fin pour respecter l'équation de la somme des probabilités de chaque ligne égale à un.

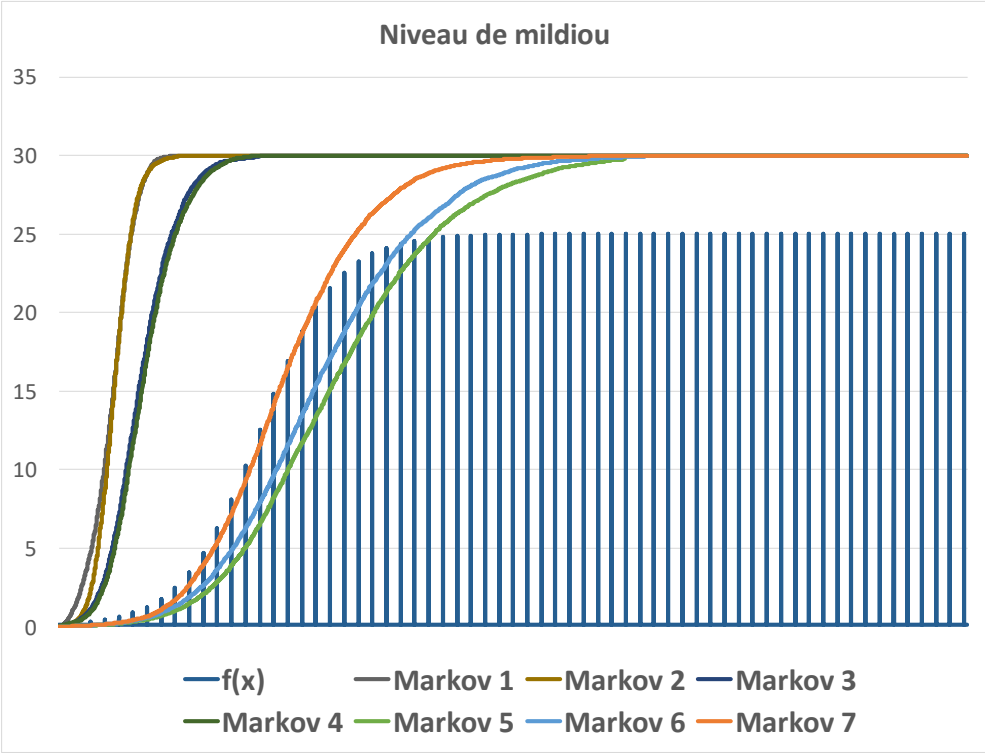

FIGURE 2.4 : Simulation de plusieurs comportements de mildiou avec différents β

Après la construction des matrices de transition markoviennes, plusieurs simulations sont effectuées afin de calibrer β qui permet au comportement individuel des plantes à l'aide des matrices de transition d'approcher le comportement global de la serre défini par la fonction $\hat{f}(t)$. Les résultats de la simulation sont présentés dans la figure 2.4. Le meilleur comportement est la courbe orange (Markov7), avec $\beta = 5,5 \times 10^{-6}$.

La différence entre le niveau maximal de la maladie et le niveau observé par la fonction $\hat{f}$ est due au fait que ce niveau maximal correspond au point de non-retour sur l'état des plantes, qui est effectivement a évité.

Une fois le modèle de simulation et le comportement des agents les plus complexes établis, les étapes de développement du simulateur seront détaillées dans la suite de ce chapitre.

2.4 Développement du simulateur

Le simulateur est codé sous le langage de programmation appelé « NetLogo » qui est un environnement permettant de modéliser et développer les SMAs. La première version de NetLogo a été développée en 1999 par une équipe de l'université Northwestern (Etats-Unis) sous la direction d'Uri Wilensky [Wilensky and Evanston, 1999]. Dans cette partie, il y aura la présentation de tous les algorithmes de simulation sous forme de logigrammes afin de faciliter la compréhension.
Chaque simulateur développé sur Netlogo est composé de deux fenêtres, la première pour l'interface

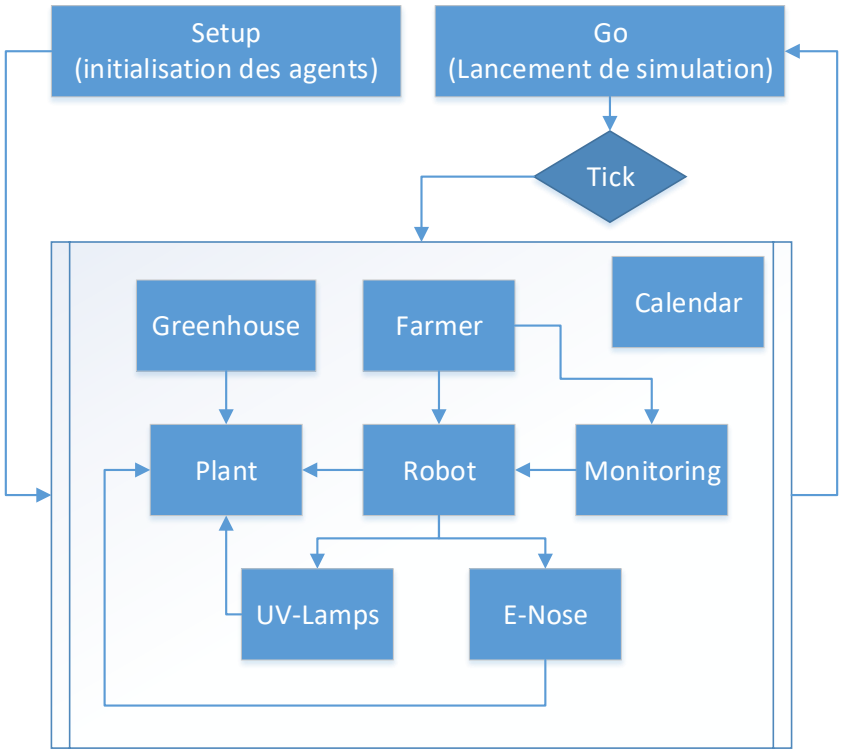

FIGURE 2.5 : Squelette du code principal du simulateur

et la deuxième pour le code. La fenêtre d'interface permet aux utilisateurs de manipuler et d'ajuster les paramètres de simulation et de jouer tous les scénarios souhaités. Dans l'interface, il y a des boutons, des sliders, des Input et des Output..., en plus de l'environnement où s'affiche le modèle simulé. La fenêtre du code permet de développer le programme de simulation et de coder des algorithmes d'optimisation.

2.4.1 Code principal

Le code principal donne une vue globale sur le squelette du simulateur. C'est la partie la plus importante avant de se lancer dans le développement. La figure 2.5 montre le squelette du code principal du simulateur, qui est divisé sur trois parties importantes (`Setup` , `Go` , bloc des agents). La partie `Setup` permet d'initialiser l'interface du simulateur et tous les agents, c'est cette partie qui permet d'effacer une simulation qui tourne et de tout remettre à zéro pour une nouvelle simulation. La partie `Go` permet de lancer la simulation et fait tourner tous les agents en boucle jusqu'à la condition d'arrêt ou un arrêt manuel. On remarque dans la figure 2.5 que les deux parties (`Setup` , et `Go`) sont liées au bloc des agents, car chaque partie touche à certaines fonctions de chaque agent. Il y a aussi dans le bloc des agents des liaisons entre agents puisque certains agents communiquent entre eux comme dans la figure 2.1. Le calendrier permet de donner la notion du temps au simulateur.

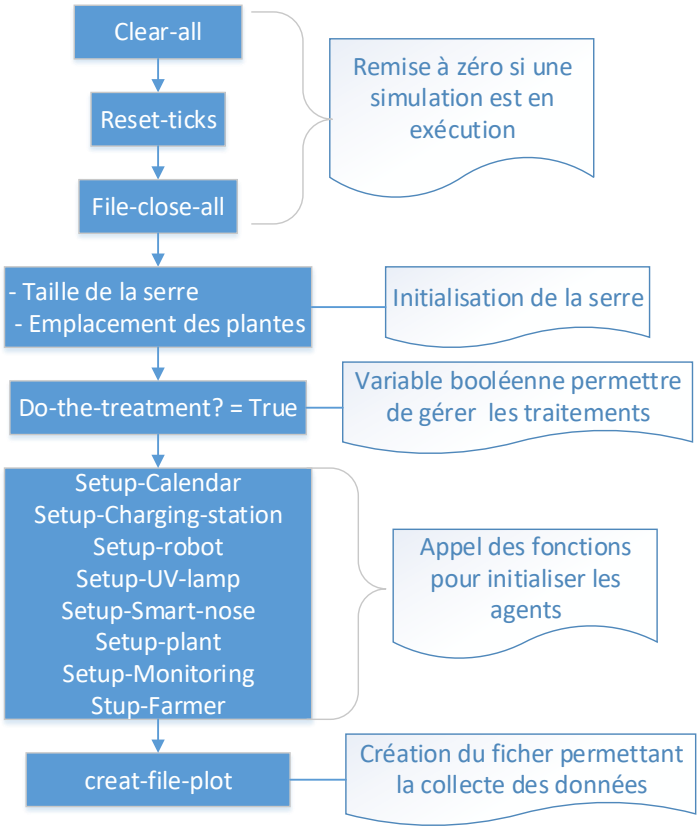

FIGURE 2.6 : Logigramme du code d'initialisation du simulateur

Initialisation

L'initialisation se fait dans le simulateur à travers le bouton `Setup`. Ce bouton permet d'initialiser tous les paramètres de simulation et de créer un nouvel environnement vierge, ou de supprimer un environnement de simulation qui tourne et de remettre tout à zéro afin de démarrer un nouveau scénario.

La figure 2.6 montre le logigramme du code associé au bouton `Setup`. Au début, pour initialiser le simulateur, il faut tous supprimés avec la commande `Clear-all`, puis `Reset-ticks` pour remettre à zéro le temps de simulation, suivi de la commande `File-close-all` qui permet de fermer les fichiers de sauvegarde comme les extensions (.CSV). Ensuite, le code permet d'ajuster la taille de la serre par rapport au nombre de plantes choisi par l'utilisateur. L'emplacement des plantes se fait avec une fonction qui se trouve dans l'agent `Plant`. La variable `Do-the-treatment?`, qui est initialisée à `True`, est une variable globale qui se trouve dans l'interface du simulateur et qui est modifiable manuellement, si l'utilisateur souhaite de désactiver les traitements du robot. Puis, le code d'initialisation fait appel aux fonctions d'initialisation de tous les agents et du calendrier pour le remettre à la date de simulation. Enfin, l'utilisation de la commande `creat-file-plot` permet de créer le fichier de sauvegarde utilisé dans ce simulateur pour collecter les données.

Exécution

L'exécution de la simulation se lance avec un bouton appelé `Go`. Le bouton `Go` est différent de `Setup` parce qu'il reste actif tant que la condition d'arrêt n'est pas atteinte. Cette dernière, peut être liée à l'horizon de simulation, comme c'est le cas ici, ou l'atteinte d'un objectif défini par le développeur. Ce bouton peut être également désactivé manuellement par l'utilisateur pour arrêter la simulation à tout moment.
La figure 2.7 montre le logigramme du code d'exécution pour le bouton `Go`. Comme le montre La Figure la première chose est d'incrémenter le temps. Puis, chaque heure, le code fait appel à la fonction `infection-plants` qui se trouve dans l'agent `Plant` est exécutée pour gérer (simuler) le comportement de mildiou. Ensuite, la fonction `Update-Greenhouse` est exécutée chaque jour avec l'incrémentation d'une variable locale des plantes `Last-treatment` qui correspond au nombre de jours cumulés depuis le dernier traitement pour chaque plante. Ensuite, après vérification de la variable globale `Speed-simul?`, le programme peut lancer les fonctions `Update-Robot`, `Update-plants` et `Update-Farmers` qui sont liées aux agents Robot, Plante et Agriculteur, respectivement. Le traitement ne peut être effectué que dans des fenêtres de temps spécifiques définies par l'utilisateur. La condition `Speed-simul?=vrai` permet de vérifier ces plages horaires. Si elle est vérifiée, le programme principal `Go` exécute la fonction `Update-Monitoring` permettant la mise à jour de l'agent de supervision. Enfin, dans la dernière partie du code du bouton `Go` le programme collecte les données de la simulation chaque minute et incrémente les `Ticks` [1] de simulation.

2.4.2 Temps de simulation

La notion de temps dans le langage NetLogo est définie par les `Ticks`. Avant de commencer le développement du simulateur, il est nécessaire de définir le pas de simulation, ce qui revient à définir l'échelle d'un `Tick` dans NetLogo. Un `Tick` peut être égal à une seconde, comme il peut être égal à une heure, la question est de savoir quelles informations sont collectées à chaque `Tick`. Comme dans le cas du simulateur UV-Robot au début, le pas de simulation a été défini par cinq minutes pour simuler de grands horizons en un temps rapide. Sachant que le temps de traitement d'une rangée se fait en quelques minutes, il est donc impossible d'utiliser un pas d'une heure. D'autre part, le robot a une vitesse qui peut atteindre 1 mètre par seconde. Cela signifie qu'en une minute le robot se déplace de 60 mètres, ce qui ne permet pas de voir avec précision le comportement réel du système. Il est donc plus logique de faire passer le pas de simulation de cinq minutes à une seconde (1 `Tick` = 1 seconde). Bien que la simulation soit un peu plus longue avec le pas d'une seconde, l'avantage est que nous obtenons des calculs plus précis et des données plus fiables.

Pour gérer le temps de simulation, un agent calendrier a été ajouté, ce dernier gère juste la date et l'heure du simulateur et n'influence pas les autres agents. La date du calendrier est automatiquement fixée à la date de lancement de la simulation. Un bouton « Date » est également ajouté dans l'interface du simulateur pour permettre aux utilisateurs de choisir une date manuellement.

1. Tick : est le pas de simulation qui définie la notion de temps sur Netlogo.

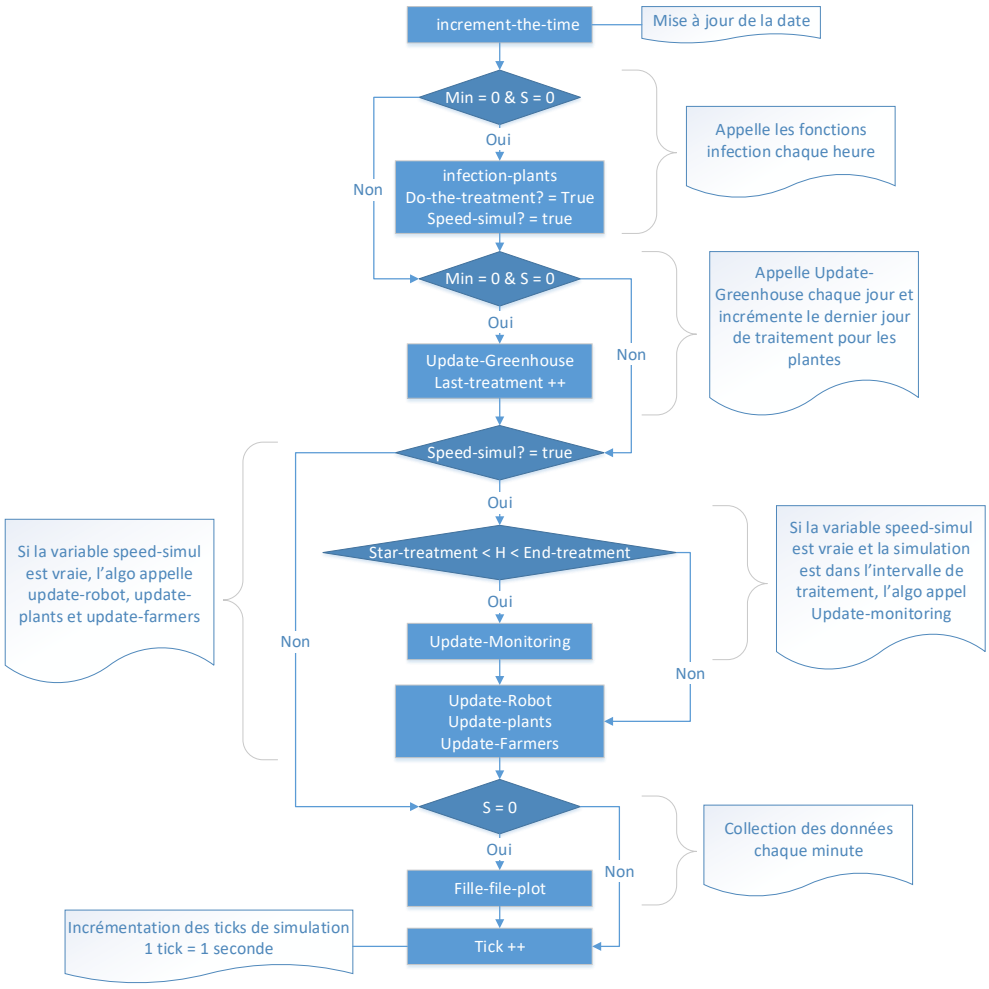

FIGURE 2.7 : Logigramme du code de lancement de la simulation

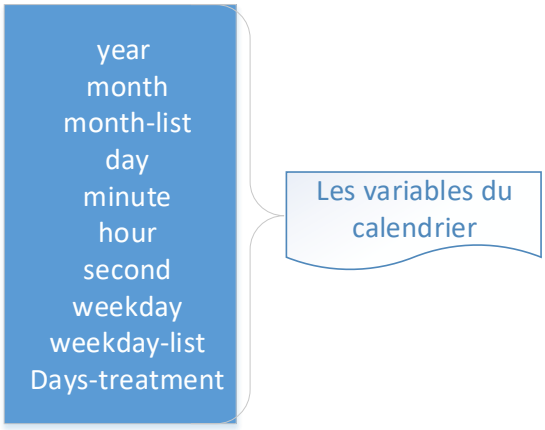

FIGURE 2.8 : Variables du calendrier

Fonctionnement du calendrier

Dans cette partie il y aura l'explication du fonctionnement du code de l'agent calendrier. Les logigramme du code sont divisés sur trois sous-parties : la première pour montrer les variables du calendrier, la deuxième pour expliquer le code de l'initialisation de l'agent qui est appelé dans le code de `Setup` et enfin la partie de la mise à jour de calendrier qui est appelé dans le code `Go`.

* a) Variables du calendrier : La figure 2.8 montre les variables du calendrier, où il y a des variables qui déterminent le temps comme (seconde, minutes, heure, jour, mois et années). Il y a aussi `month-list` qui est une liste des mois, `weekday-list` une liste pour les jours de la semaine. Enfin la variable `Day-treatment` est mise si l'utilisateur souhaite sélectionner les jours de traitement.

* b) Initialisation du calendrier : Dans la figure 2.9 il y a la fonction `setup-Calendar` qui initialise le calendrier. Au début il y a la création du calendrier avec la fonction `create-Calendar`, `create-nom_d'agent` est prédéfinie sur NetLogo. Puis à l'aide du code python qui est ajouté comme extension dans NetLogo, le programme rempli les variables `Day`, `Month` et `Year` par la date de simulation. Cette étape permet de prendre automatiquement la date du jour de simulation. Enfin le programme remplace toutes les variables de la date pour les afficher dans l'interface de simulation.

* c) Mise à jour du calendrier : Dans la mise à jour du calendrier se fait avec la fonction `increment-the-time` qui est appelé par la fonction du bouton `Go` chaque second (ou chaque `Tick` de simulation). Comme le montre le logigramme de la figure 2.10 les secondes s'incrémentent chaque `Tick` sans aucune condition. Ensuite chaque 60 secondes, la variable `second` devient 0 et les minutes s'incrémentent, et pareilles pour les heures, jours, mois et années. Chaque variable s'incrémente lorsque la variable précédente atteint sa limite et devient 0. Pour les mois le programme vérifie juste s'il y a 30 ou 31 jours pour chaque mois, et il vérifie si l'année est bissextile ou non pour le mois de février. La fonction de vérification d'année bissextile est la même de celle du bouton `Date` qui est montré sur la figure 2.11.

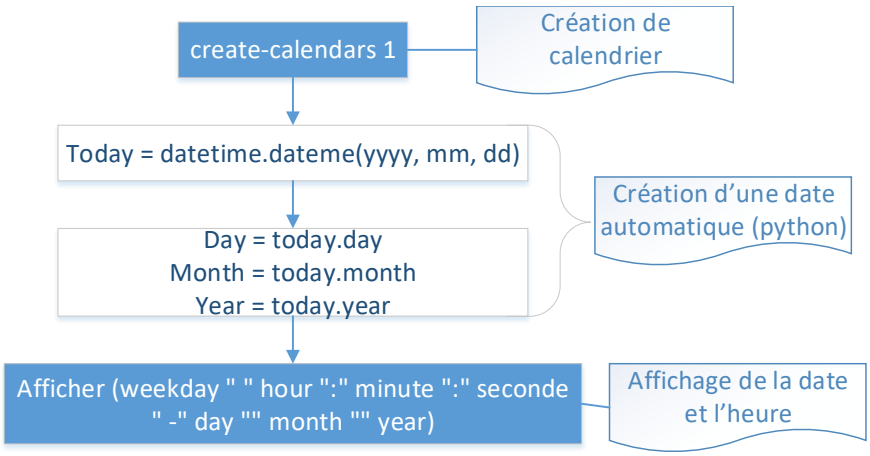

FIGURE 2.9 : Logigramme du code d'initialisation du calendrier

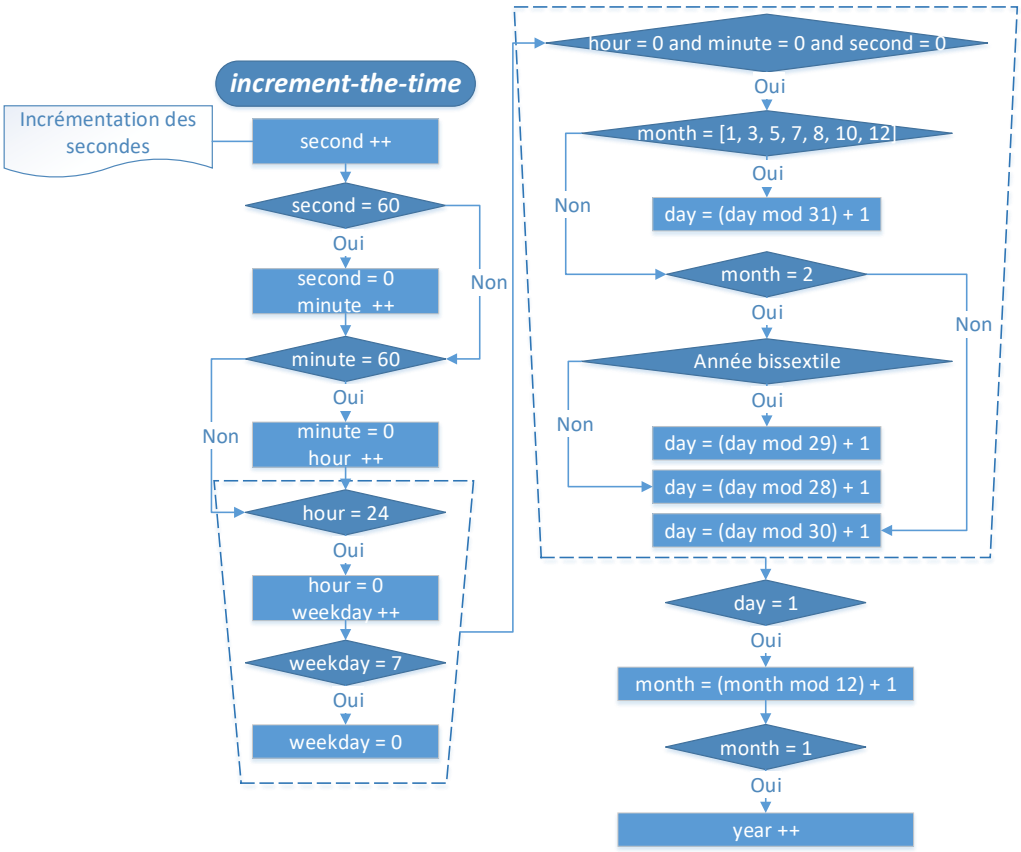

FIGURE 2.10 : Logigramme du code pour le fonctionnement du calendrier

Fonctionnement du bouton de la date

Cette fonction permet de choisir une date manuellement au lieu d'utiliser celle du calendrier automatique. L'utilisateur peut choisir une date quelconque en cliquant sur le bouton `Date` qui se trouve dans l'interface de simulateur. La date est choisie par cet ordre année, puis mois, ensuite jours, afin de donner une possibilité de choix logique par rapport au mois. Par exemple si l'utilisateur choisit l'année 2024, puis le mois de février, le simulateur lui donne une possibilité de choix de 29 jours, car l'année 2024 est bissextile. La figure 2.11 montre le logigramme de fonctionnement du bouton `Date`. Au début le programme initialise des listes pour les jours de semaine, les mois et les années comme celle du calendrier. Lorsque l'utilisateur choisit l'année et le mois le programme vérifie s'il lui donne une liste de jours de 28, 29, 30 ou 31, tout dépend du mois et de l'année. Les listes des mois sont déterminées directement sauf pour le mois de février le programme vérifie si l'année est bissextile ou non. Donc si une année est bissextile la condition [Année modulo 4 = 0 ou (Année modulo 4 = 0 et Année modulo 100 ≠ 0)] soit vérifiée. L'autre partie de programme du bouton `Date` détermine le jour de la semaine de chaque date. Comme le montre le logigramme de la figure 2.11 le programme assigne des codes pour chaque mois tout dépend de type de l'année bissextile ou non. À la fin du code le programme utilise l'équation (2.9) qui donne des résultats entre 1 et 7 pour définir le jour de la semaine, le 1 pour le lundi jusqu'à 7 pour le dimanche.

$$\texttt{Weekday} = \left((\texttt{Day} \ \ \% \ \ 7) + \left(\left(\frac{\texttt{Var} - (\texttt{Var} \ \ \% \ \ 4)}{4} + \texttt{Var} \right) \ \ \% \ \ 7 \right) + \texttt{Code-Month} \right) \ \ \% \ \ 7 \tag{2.9}$$

Tel que :

- `Weekday` : est une valeur du jour de semaine qui varie entre 1 et 7 pour déterminer l'un des jours de semaine du lundi jusqu'à samedi.
- `Day` : est la valeur du jour en du mois choisi par l'utilisateur.
- `Var` : est une valeur de l'année choisi par l'utilisateur moins 2000. Sachant que la liste des années qu'on a sélectionnée sur le simulateur commence de 2018.
- `Code-Month` : est un code de mois sélectionné par le programme. Chaque mois a un code attribué tout dépend de type de l'année bissextile ou non.

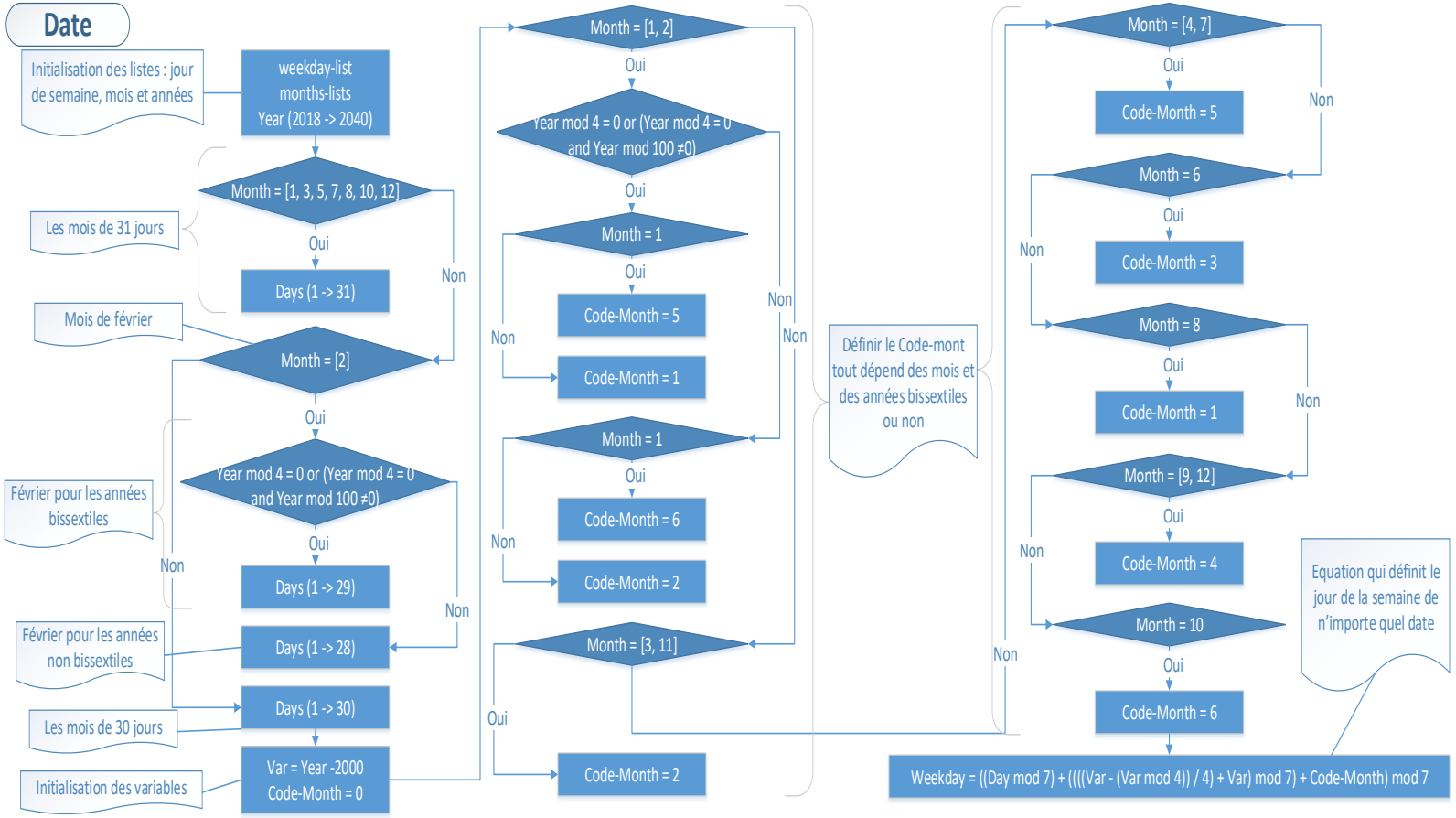

FIGURE 2.11 : Logigramme du code de la fonction `Date`

2.4.3 Codage des agents

Dans cette partie, il y a de brèves explications sur le codage des agents. L'annexe .1.3 contient plus de détails avec des organigrammes du fonctionnement de chaque agent.

Greenhouse

La serre représente tout l'environnement de simulateur. C'est là où tous les autres agents sont installés. C'est un agent simple, il est caractérisé par la taille de l'environnement de la serre et la couleur de ses `patches` qui sont initialisé au début.

Plants

L'agent `Plants` est une classe d'agents qui contient tous les agents `Plant`. Pour modifier une variable sur les plantes le programme appelle l'agent `Plantes`, et pour une modification sur une plante précise il appelle l'agent `Plant` (numéro de la plante). Pour simuler le comportement réel des plantes, il est nécessaire de programmer le comportement des agents `Plant`.

Robot

Le robot a plusieurs fonctions dans le simulateur qui lui permet de faire des traitements, de se déplacer, de se décharger et de se charger. Le type d'agents `Robots` peut contenir plusieurs agents `Robot`, selon le nombre de robots dans la serre indiqué par l'utilisateur.

UV-Lamps

La classe d'agent `UV-Lamps` contient deux sous-classes d'agents, une pour les lampes qui sont à gauche du robot, et l'autre pour ceux du côté droit.

Agriculteur

Dans cette simulation, l'agriculteur n'a pas beaucoup de tâches, il surveille l'état du robot et le niveau de sa batterie. Si le robot manque d'énergie, l'agriculteur s'en occupe pour le recharger.

Station de charge

La station de charge où l'agent `Charging-station` a que le programme d'initialisation. Le programme d'initialisation de la station de charge crée une station dans la serre. Il initialise ensuite son apparence et son emplacement au milieu de la serre. Lorsque le robot atteint le point de la station de charge, sa fonction de charge est activée.

FIGURE 2.12 : Interface du simulateur

Superviseur

L'agent superviseur où `Monitorings` est un agent qui dirige l'ordonnancement des tâches du robot et permet de suivre l'état des plantes. La planification des tâches sera menée par le superviseur en fonction du type de traitement sélectionné par l'utilisateur. Beaucoup de calcul s'exécute dans l'agent `Monitorings`, car il intègre des algorithmes d'optimisation qui seront présentés dans le chapitre 3.

2.4.4 Interface du simulateur

L'interface du simulateur permet aux utilisateurs de jouer sur plusieurs paramètres et choisir les meilleurs scénarios. L'interface du simulateur UV-Robot est présenté dans la figure 2.12 UV-Robot. L'utilisateur peut définir la taille de la serre en choisissant le nombre total de plantes, et le nombre de plantes dans chaque ligne. Il peut aussi définir l'horizon de la simulation, le taux des plantes malades au début de simulation et il peut choisir un algorithme d'optimisation. Si l'algorithme choisi est le génétique, l'utilisateur peut modifier les paramètres de cet algorithme. Il y a d'autres choix comme le scénario du type de traitement. Si le type de traitement choisi est le préventif, il peut choisir la période de traitement et la vitesse du robot pendant le traitement. Enfin, il peut définir les jours et la plage horaire de traitement.

La figure 2.13 présente l'interface du simulateur en 3D avec un zoom sur le robot au milieu des plantes pendant le traitement. Cela donne plus de précision sur les observations dans le simulateur.

2.5 L'utilité du simulateur

L'idée de développer un simulateur est d'étudier un système qui n'existe pas au début des travaux de cette thèse. En effet, le robot qui traite le mildiou avec des lampes UV-C est une nouvelle technologie. Il n'y avait pas le même type de robot dans la littérature au début du projet UV-Robot. Ainsi la

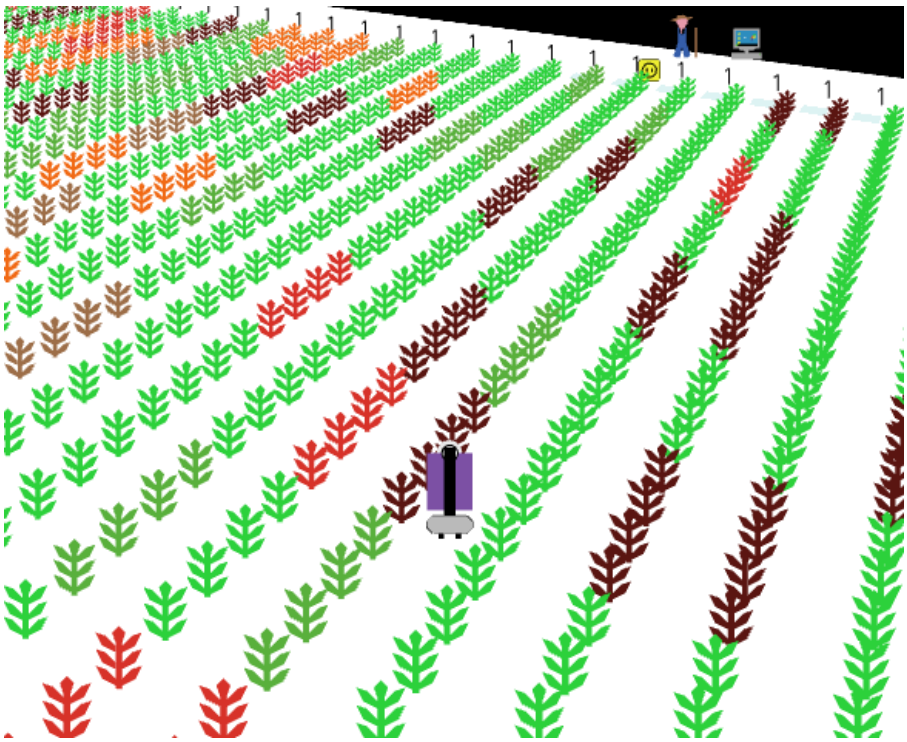

FIGURE 2.13 : Interface du simulateur avec un zoom sur le robot durant un traitement

création d'un simulateur permet de voir l'évolution du système UV-Robot avant son existence. Toutes les informations utilisées lors du développement du simulateur sont réelles comme : l'architecture et la taille des serres, les paramètres des robots, le comportement de la maladie et même l'effet du traitementUV-C sur le mildiou.

Ce simulateur est configurable et flexible, permettant à l'utilisateur de choisir le nombre de plantes, la taille de la serre, le taux de maladie au début de la simulation et le nombre de robots. Il y a également la possibilité d'optimiser avec l'un des algorithmes intégrés dans le simulateur. L'utilisateur peut également définir des stratégies de traitement telles que : les heures et les jours de traitement. Ainsi, le développement du simulateur permet aux horticulteurs de jouer plusieurs scénarios afin de choisir la stratégie de traitement qui convient à chaque serre. Trois scénarios pouvant être assimilés à des types de traitements préventifs sont étudiés dans cette thèse. Les expérimentations et les résultats de ces types de traitement seront présentés dans le prochain suivant.

2.6 Conclusion

Avant de se lancer dans la simulation, il est nécessaire de créer un modèle, qui permet d'avoir une vision sur le déroulement du simulateur. Le point fort de SMAs, qui facilite la partie développement, est de diviser le système en plusieurs agents. Après la modélisation, vient la partie de collecte de données sur le fonctionnement des agents. Cette partie a été réalisée grâce aux échanges avec les partenaires du projet. Il y a eu une recherche sur les données manquantes, comme le comportement des maladies, qui a été validée par des experts du domaine agricole.

Après la modélisation et la collecte des données vient la partie de la définition des paramètres. Au début du développement, nous avons défini certains paramètres pour chaque agent, et nous avons calibré l'étape de simulation par des premiers tests. Ensuite, après chaque nouveau code développé, il y a la vérification, qui permet de savoir si le simulateur fonctionne bien et si les résultats sont cohérents. Enfin, la dernière partie est la validation avec les partenaires pour s'assurer que les résultats de la simulation reflètent la réalité. Une fois que toutes les étapes précédentes ont été confirmées, les simulations peuvent être exécutées avec de larges horizons pour faire des prédictions sur le comportement de l'ensemble du système

L'amélioration du comportement du système nécessite une partie d'optimisation qui est intégrée dans la simulation. Le chapitre suivant (cf. chapitre 3) présente les algorithmes d'optimisation qui sont développés et intégrés dans le simulateur.

Chapitre 3

Modélisation et optimisation de tâches évolutives de traitement

Contenu

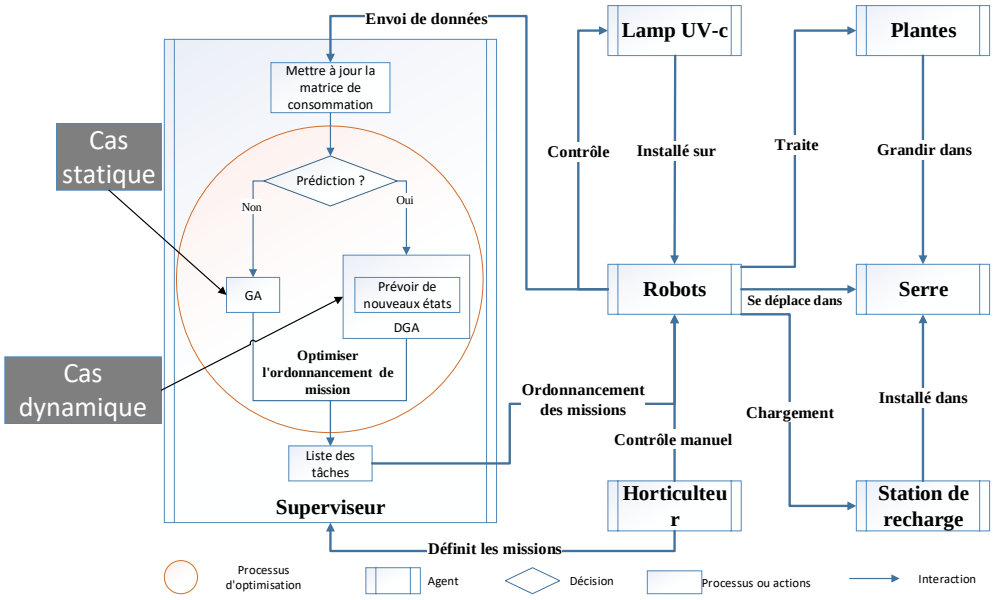

FIGURE 3.1 : Modèle de système multi-agents avec un zoom sur la partie optimisation

3.1 Introduction

Pour résoudre certains problèmes de la recherche opérationnelle, plusieurs méthodes sont utilisées. On peut les diviser en deux grands axes. Le premier axe concerne les méthodes exactes et le deuxième regroupe les méthodes approchées. Le but de toutes ces méthodes est de trouver une ou plusieurs solutions optimisées durant un certain temps. Les méthodes exactes permettent d'obtenir une solution optimale pour un problème. Le temps d'exécution des méthodes exactes dépend de la complexité et la taille de l'instance du problème (pour chaque problème plus grand, le temps d'exécution est plus long). Les méthodes approchées permettent d'obtenir une solution réalisable pour un problème dans un temps réduit.

L'optimisation couplée à la simulation permet d'améliorer le comportement d'un système complexe même pour des décision opérationnelles contenant beaucoup de phénomènes stochastiques. En effet, la simulation à une myopie dans l'espace d'état et l'optimisation a une myopie dans le temps. En outre, la fusion entre simulation et optimisation permet l'exploration d'espace d'état et le temps. Des travaux sur des algorithmes d'optimisation seront étudiés dans ce chapitre. Ces algorithmes sont développés au niveau de l'agent de supervision « `Monitorings` » du système multi-agents décrit dans le chapitre 2. La figure 3.1 montre le modèle du système multi-agents du simulateur avec une vue détaillée sur l'agent « `Monitorings` » où les algorithmes d'optimisation sont intégrés. Les algorithmes d'optimisation sont choisis en fonction du scénario étudié.

Ce chapitre est divisé en plusieurs parties, dont les deux premières sont consacrées à l'état de l'art sur l'ordonnancement dynamique et le bin-packing respectivement. La troisième partie contient l'approche utilisée dans ce travail. Puis, avant d'attaquer la cinquième partie sur les algorithmes d'optimisation vient la quatrième partie sur l'état de l'art des méthodes d'optimisation. Enfin, la conclusion de ce chapitre fera l'objet de la dernière partie.

3.2 Ordonnancement dynamique

L'ordonnancement est le processus qui consiste à trouver la meilleure combinaison pour affecter les tâches aux ressources, ressources qui peuvent être des robots. Généralement, l'objectif est d'optimiser le temps d'exécution de toutes les tâches. L'utilisation de robots, qui sont des entités autonomes, avec des systèmes de supervision automatique, nécessite l'adoption de la théorie de l'ordonnancement. Le niveau de la maladie évolue en permanence, ce qui rend difficile la modification du plan de traitement entre le moment de la prise de décision et son application. Il est donc important d'étudier l'ordonnancement dynamique.
Dans la littérature, il y a deux approches différentes utilisées pour résoudre les problèmes d'ordonnancement dynamique : une approche dynamique basée sur la planification (utilisée dans les lignes de flux de fabrication), et une approche dynamique de meilleur effort [Suresh and Chaudhuri, 1993], [Ouelhadj and Petrovic, 2009] et [Barenji et al., 2017]. Les auteurs ont fait leur étude sur le comportement dynamique lié aux demandes des clients. Ils ont défini les points négatifs du système de fonctionnement des machines de fabrication comme : l'ordonnancement statique [Huff et al., 2021], le manque d'autonomie des machines [Abosuliman and Almagrabi, 2021], le manque de planification en temps réel pour assurer la flexibilité du système face aux demandes dynamiques des clients [Karimi et al., 2021], et le système d'ordonnancement qui ne réagit pas aux perturbations internes du système [Shi et al., 2021]. Les étapes qu'ils ont développées sont :

- La planification dynamique réactive aux demandes des clients,
- L'utilisation des SMA qui peuvent être optimaux en cas de perturbation de l'ordonnancement (panne de machine),
- La planification autonome au niveau des stations,
- La communication entre SMA et système en temps réel,

Les auteurs ont mis au point plusieurs diagrammes afin de définir l'architecture des agents du SMA et ont choisi la plateforme $JACK^{TM}$ [1] pour sa mise en œuvre.

3.2.1 Ordonnancement sur une machine

[Wang et al., 2017] traitent des problèmes d'ordonnancement dynamique d'une seule machine avec l'arrivée continue de nouveaux travaux, où le rejet de travaux est autorisé, le temps de traitement des travaux peut être contrôlé et l'effet de détérioration des jobs [2] est pris en compte. Ils utilisent le front de Pareto pour comparer la performance des algorithmes évolutifs, avec l'objectif d'optimiser le coût d'exploitation et le coût de l'écart mesuré par le total du temps des retards virtuels (optimisation bi-objectif). L'algorithme génétique de tri non dominé (non-dominated sorting genetic algorithm (NSGA-II)) obtient les pires résultats sur les problèmes d'ordonnancement envisagés. En partant des informations historiques, le mécanisme de ré-initialisation de la population (population reinitialization mechanism (PRM)) est capable d'améliorer la performance de NSGA-II, mais n'est pas assez capable de gérer des événements dynamiques. Avec l'aide de mécanisme de génération de descendance (offspring generation mechanism (OGM)), la diversité de la population est encore renforcée et la convergence est

1. $JACK^{TM}$ Intelligent Agents, est une plate-forme d'agents qui inclut un langage de programmation orienté agent, une plate-forme d'exécution d'agents avec une infrastructure telle que le regroupement de messages et un serveur de noms, des outils de développement comprenant un outil de conception, un éditeur de plan graphique et un certain nombre de vues de débogage.
2. Un job est un ensemble de tâches qui sont exécutées pour la fabrication d'un produit.

accélérée. Par conséquent, ils obtiennent les meilleurs résultats avec NSGA-II/PRM+OGM, quoique cet algorithme prend plus de temps d'exécution, à cause de plusieurs opérations supplémentaires qu'il fait.

Dans [Cai et al., 2017], les auteurs travaillent sur l'ordonnancement des tâches dans un réseau de « cloud » (serveur informatique distant qui sert à stocker des données et les exploiter). Ils louent des machines virtuelles (VM) pour réduire le temps d'exécution de leur algorithme. La location des VM est dynamique et n'est pas fixe au début. Les auteurs proposent un modèle mathématique non linéaire, ainsi qu'un algorithme d'ordonnancement dynamique basé sur les retards (Delay-based Dynamic Scheduling (DDS)) composé de trois étapes : 1) Répartition des délais basée sur les sacs (Bag-based deadline division (BBD)), 2) Planification des délais basée sur les sacs (Bag-based delay scheduling (BDS)), 3) Une location de ressources d'un seul type (A single-type based resource renting (SIR)). Le workflow appelle l'algorithme DDS chaque fois qu'une tâche est terminée et retournée par la VM assignée. Afin d'évaluer l'algorithme DDS, les auteurs ont utilisé un simulateur appelé ElastiSim. Cet algorithme utilise la somme de l'espérance de temps d'exécution des tâches et de l'écart-type pour estimer le temps d'exécution stochastique des tâches. Ils comparent la variance du type de distribution gaussien. Ils ont ajouté une variable dans leurs algorithmes afin de comparer plusieurs sous-algorithmes et choisir le meilleur. Ils ont comparé un algorithme appelé DDSRR30 avec un autre algorithme (Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT)). DDSRR30 s'est avéré plus performant car il peut garantir les délais exigés (les deadlines) alors que ces délais sont violés par MOHEFT dans la plupart des cas. DDS est proposé pour minimiser le coût de location des ressources VM tout en respectant les délais du workflow.

D'autres travaux ont traité le cas d'une ressource comme dans [Chetto et al., 1990], [Cowling and Johansson, 2002] ou un processeur de smartphone dans [Li et al., 2015a], [Guo et al., 2018]. Les tâches arrivent de manière dynamique, soit dans un ordinateur qui doit exécuter un ensemble de programmes, ou des applications de smartphone (jeux, images et vidéos ...). Certaines tâches ont des périodes de traitement qui dépendent de processus physique. Ces tâches peuvent avoir des variations d'exécution ou recevoir des nouvelles informations. En outre, elles impactent le déchargement dynamique de la batterie, comme dans le cas de smartphone, par exemple.

3.2.2 Ordonnancement sur plusieurs machines

Il existe aussi des problèmes d'ordonnancement sur plusieurs ressources comme des machines virtuelles ou des multi-processeurs dans un ordinateur qui doivent exécuter un ensemble de tâches [Sahni and Vidyarthi, 2015], [Thakor and Shah, 2011], des nœuds de réseaux qui font des mises à jour [Jin et al., 2014], ou des fours pour chauffer plusieurs produits [Baykasoğlu and Ozsoydan, 2018]. Pour chaque cas, il y a une certaine dynamicité qui est liée soit aux nouvelles demandes des clients, soit aux nouvelles arrivées des commandes urgentes ou leur annulation, soit à la surcharge des tâches sur le processeur, soit au comportement dynamique des commutateurs dans les nœuds.

3.2.3 Flow-shop

Dans un atelier flow-shop, tous les jobs suivent la même séquence sur les machines, ce qui correspond à des lignes de production : un seul flux physique dans l'atelier. Typiquement, les produits passent d'une machine à l'autre avec des convoyeurs. [Ivanov et al., 2016] et [Tang et al., 2016] ont traité le

problème d'ordonnancement dynamique d'ateliers de type flow-shop flexible une généralisation du flow-shop simple, où il y a au moins deux machines identiques qui font le même travail dans chaque étape. Les deux travaux considéraient deux objectifs. Les objectifs dans [Ivanov et al., 2016] sont d'optimiser à la fois 1) le choix des machines pour chaque pièce et 2) les séquences de chargement des pièces sur les machines pour améliorer la productivité. Ils ont travaillé sur deux machines dans une chaîne de production adoptant les technologies de l'Industrie 4.0. Les chaînes d'approvisionnement ont des structures dynamiques qui évoluent avec le temps. Leur approche est basée sur une théorie sous-jacente à l'étude de systèmes dynamiques multi-étapes et multi-périodes, avec des variables continues et indicateurs de performance accumulés au fil du temps. [Tang et al., 2016] ont travaillé sur des machines de fabrication où il y a des événements incertains comme les pannes et l'arrivée dynamique de nouvelles tâches. Leurs objectifs sont : réduction de la consommation d'énergie et minimisation du Makespan (date du fin de la dernière tâche).

[Qin et al., 2018] ont traité un problème de ré-ordonnancement hybride (séquencement et affectation) de ligne de flux flexible ou flow-shop (hybrid flow-shop scheduling (HFS)). Le document examine le problème de l'ordonnancement dynamique du HFS dans les lignes d'assemblage de cartes de circuits imprimés avec un temps de traitement incertain. Pour résoudre ce problème d'ordonnancement, les auteurs ont utilisé l'algorithme de colonie de fourmis (en anglais, Ant colony optimization (ACO)) qui doit non seulement séquencer tous les travaux sur les lignes d'assemblage, mais aussi prendre en compte le problème d'affectation des machines parallèles pour chaque travail. Au début de l'optimisation, l'algorithme planifie toutes les tâches pour obtenir un pré-planning. Durant l'avancement du processus réel, le ré-ordonnancement est nécessaire en raison des changements dans l'environnement de l'atelier. L'algorithme supprime les tâches traitées pour faire un nouveau planning avec les tâches restantes.

3.2.4 Job-shop

Avec la configuration d'atelier de type job-shop, chaque produit suit sa propre séquence sur les machines. Cela correspond à une organisation en ilots de production, avec des flux complexes dans l'atelier. [Kouiss et al., 1997], [Aydin and Öztemel, 2000], [Xiang and Lee, 2008] et [Umar et al., 2015] ont traité le problème d'ordonnancement dynamique avec des ateliers correspondant au job-shop. En général, ils avaient des ressources en parallèle (plusieurs machines qui peuvent faire une même opération) sauf le cas de [Umar et al., 2015] où il n'y avait pas de machines en parallèle. Dans ces articles, les jobs arrivent de manière dynamique, dont les auteurs ne peuvent pas toujours prédire la date d'arrivée. Il y a aussi la contrainte des événements qui se produisent comme les pannes des machines. [Kouiss et al., 1997] ont utilisé la simulation-optimisation avec les SMAs afin de simuler les événements des machines de fabrication et optimiser leur comportement pendant l'arrivée des nouvelles tâches. Le travail de [Aydin and Öztemel, 2000] consiste à faire un apprentissage par renforcement pour créer des agents intelligents afin de planifier cinq tâches dans neuf machines. [Xiang and Lee, 2008] ont utilisé un algorithme de colonie de fourmis avec des SMAs afin de gérer la fabrication des produits dans des machines reconfigurables qui peuvent faire plusieurs opérations. Les objectifs de [Umar et al., 2015] étaient l'optimisation du Makespan et les coûts de pénalité due au retard. Pour résoudre ce problème multi-objectif, les auteurs ont utilisé un algorithme génétique.

3.2.5 Open-Shop

Dans un atelier de production open-shop, il n'y a pas de précédence entre les tâches sur le même job. La différence de l'open-shop avec le flow-shop et le job-shop est que l'ordre de traitement des tâches n'est pas défini. Il y a plusieurs cas d'ateliers dans lesquels l'exécution des tâches n'a pas d'ordre à suivre, et où on ne peut pas traiter plus qu'une tâche à la fois. L'objectif du problème open-shop est de déterminer le temps d'exécution de chaque tâche dans la ressource qu'il lui convient.
L'article de [Gonzalez and Sahni, 1976] est le premier travail qui définit le problème d'ordonnancement dans un atelier open-shop. L'objectif est de minimiser le temps d'exécution des tâches. Les auteurs ont proposé trois exemples d'ateliers open-shop et un algorithme de résolution pour chacun.

3.2.6 Synthèse sur l'ordonnancement dynamique

Le tableau 3.1 présente un résumé d'état de l'art sur le problème d'ordonnancement dynamique. Le tableau classe les articles selon cinq critères (colonnes 2 à 6) : les tâches à traiter, le type de ressource, le(s) paramètre(s) dynamique(s), le type d'atelier et les méthodes de résolution utilisées. Sans prétention d'exhaustivité, la synthèse présentée dans ces tableaux permet aux lecteurs de ce mémoire d'avoir une idée globale sur le problème d'ordonnancement dynamique et d'avoir des idées sur les méthodes de sa résolution.
Cette synthèse permet également de positionner la problématique et notre contribution par rapport à ce qui est fait dans la littérature. L'apparition de la maladie et son évolution étant un phénomène dynamique suivant un processus stochastique, ce problème s'apparente à un problème d'ordonnancement de tâches dynamique et évolutif. Pour gérer cette évolution stochastique et optimiser les résultats, une approche basée sur le couplage entre simulation et optimisation semble présenter une bonne alternative. Une synthèse bibliographique de cette approche est présentée dans la section suivante.

Article	Tâches	Type de Ressource	Paramètre(s) dynamique(s)	Type d'atelier	Méthode de résolution
[Chetto et al., 1990]	Exécution des tâches	Ordinateur	Temps d'exécution	Ordonnancement des tâches apériodiques	Earliest Deadline strategy (EDF)
[Kouiss et al., 1997]	Une seule opération par machine	4 machines de fabrication exécutent une seule opération	Arrivées des jobs (processus de Poisson)	Job-shop	SMA & simulation-optimisation
[Aydin and Öztemel, 2000]	5 jobs	9 machines de fabrication non parallèles	Arrivée des jobs	Job-shop	Apprentissage par renforcement
[Cowling and Johansson, 2002]	20 jobs	Une machine	Arrivée des nouvelles informations sur les produits	Job-shop	Simulation
[Xiang and Lee, 2008]	Jobs	Machines de fabrication parallèle	Arrivée des jobs selon la règle stochastique	Job-shop	SMA & Colonie de fourmis
[Thakor and Shah, 2011]	Tâches exécutables sur l'ordinateur	Multi processeurs	Surcharge des tâches sur le processeur	Ordonnancement dynamique sur un multi processeur	Dynamic Earliest Deadline First strategy (DEDF)
[Jin et al., 2014]	Mises à jour	Nœuds de réseaux	Comportement dynamique des commutateurs	Ordonnancement dynamique des mises à jour du réseau	Simulation
[Li et al., 2015a]	Applications de smartphone	Processeur de smartphone	Variations d'exécution	Ordonnancement dynamique	Energy-aware Dynamic Task Scheduling (EDTS)
[Sahni and Vidyarthi, 2015]	Transfert de données	Machines virtuelles	Temps d'arrivée	Ordonnancement dynamique	Simulation
[Umar et al., 2015]	Fabrications des véhicules	10 machines de fabrication	Arrivée des nouvelles tâches	Job-shop	Algorithme génétique
[Ivanov et al., 2016]	Opérations sur des machines	Deux machines parallèles	Arrivée des tâches	Flow-shop flexible	Optimal programme control (OPC)
[Tang et al., 2016]	Opérations sur des machines	Machines de fabrication parallèles	Déchargement dynamique	Ordonnancement dynamique	Simulation
[Barenji et al., 2017]	Opérations sur des machines	Des machines de fabrication	Arrivée des nouvelles tâches	Flow-shop flexible	Particle swarm optimization
[Cai et al., 2017]	Opérations sur des machines virtuelles louées	Cloud	Location des VM	Ordonnancement dynamique	Delay-based Dynamic Scheduling (DDS)
[Wang et al., 2017]	Opérations sur une machine	Une machine	Arrivée des nouvelles tâches	Ordonnancement dynamique de machine multi-objectif	NSGA-II/PRM+OGM
[Baykasoğlu and Ozsoydan, 2018]	Chauffer plusieurs types de produit	5 fours parallèles	Arrivées de commandes	Ordonnancement dynamique	Algorithme de recherche à démarrage multiple et constructif
[Guo et al., 2018]	Jeux, images, vidéos, ...	Processeur de smartphone	Déchargement dynamique	Ordonnancement dynamique	Simulation
[Qin et al., 2018]	Assemblage de cartes de circuits imprimés	Machines parallèles	Temps de traitement incertain	Flow-shop	Algorithme de colonie de fourmis
Notre problème	**Traitement d'une rangée de plantes**	**Robot qui effectue une mission de traitement**	**Nouvelle maladie & évolution de la maladie**	**Ordonnancement des tâches dynamiques et évolutives**	**Simulation et optimisation**

TABLE 3.1 : Résumé d'état de l'art sur le problème d'ordonnancement dynamique

3.3 Problème de Bin-Packing

Dans le problème de bin-packing, il y a plusieurs objets et plusieurs bacs, l'objectif est de remplir tous les objets dans le minimum de bacs. Il faut aussi respecter la contrainte de capacité des bacs, car chaque objet a un poids et chaque bac a une capacité de chargement.

Le problème abordé dans [Epstein and Levy, 2010] est l'emballage de tous les articles demandés dans un nombre minimum de bacs unitaires. Leur objectif est de regrouper tous les éléments demandés dans un nombre minimum de cellules. Dans cet article, les auteurs traitent un problème de bin-packing dynamique multidimensionnel (des objets de différentes tailles et dimensions). Ils appliquent l'algorithme First-Fit (premier correspondant) afin d'obtenir une solution de placement des objets entrants (les objets qui seront livrés) dans des bacs. La dynamicité de ce problème est dans les périodes d'arrivée des objets, et non pas par rapport à leurs tailles, qui sont fixes.

L'objectif de [Ren and Tang, 2016] est de minimiser le temps d'utilisation total de tous les bacs dans le processus d'emballage. L'emplacement de chaque article peut être déterminé au moment de son arrivée puisque l'heure de départ de l'article est connue. Les auteurs appliquent l'algorithme First-Fit pour emballer les articles dans chaque catégorie séparément. La difficulté de leur problème est liée à la dynamicité de l'heure d'arrivée des articles.

D'autres travaux sur le problème de bin-packing dynamique, où la dynamicité est par rapport aux heures d'arrivées et heures de départ des tâches, ont été étudiés. Parmi les premiers travaux, on peut citer le travail de [Coffman et al., 1983] qui ont fait une généralisation naturelle du problème classique de bin-packing. Ils ont utilisé la méthode First-Fit afin de gérer les temps d'arrivées et de départs des articles qui sont dynamiques. [Leinberger et al., 1999], [Chan et al., 2009] et [Li et al., 2015b] ont proposé des modèles ayant pour but de minimiser le coût total des bacs utilisés au fil du temps. Ils ont utilisé un algorithme hybride basé sur la méthode First-Fit. Le traitement se fait sur la répartition des demandes découlant des systèmes de jeu dans le cloud. [Leinberger et al., 1999] ont intégré la simulation pour améliorer la performance de l'algorithme First-Fit et traiter le problème de bin-packing en ligne (les tâches entrent et sortent d'une manière dynamique). Dans [Chan et al., 2009], les auteurs ont rajouté les méthodes Best-Fit et Worst-Fit afin d'étudier les performances de chaque algorithme. Les résultats ont montré que le First-Fit est un peu meilleur que Worst-Fit malgré qu'ils étaient assez proches et que tous les deux sont meilleurs que le Best First. Le travail de [Berndt et al., 2015] montre quatre cas de problème d'emballage : Bin-Packing en ligne, Bin Packing en ligne relaxé, Bin-Packing dynamique et Bin-Packing entièrement dynamique. Dans le problème de Bin-Packing entièrement dynamique, les articles arrivent et partent en ligne et le reconditionnement des articles déjà emballés est autorisé. L'objectif est de minimiser à la fois le nombre de bacs utilisés et la quantité de réemballage.

Le problème traité par [Hermenier et al., 2008] est un peu particulier car les bacs peuvent être saturés lorsqu'une configuration est faite, une configuration est une disposition donnée des rangements. Ils ont mis une solution pour reconfigurer les emplacements des objets dans les bacs. Comme la reconfiguration peut prendre plusieurs itérations, l'objectif des auteurs est de minimiser le nombre d'itérations afin de choisir la reconfiguration la plus optimisée.

Le tableau 3.2 présente un résumé de l'état de l'art sur le problème de Bin-Packing. Il classe les articles selon 5 colonnes : les tâches à traiter, le type de ressource, le processus stochastique, le problème dans la littérature et les méthodes de résolution utilisées. La synthèse de ce tableau permet de faire une approche Bin-Packing pour le problème étudié, et d'utiliser les heuristiques proposées dans ce travail. Cette approche sera plus détaillée dans la section suivante.

Article	Tâches	Type de Ressource	Paramètre(s) stochastique(s)	Problème de la littérature	Méthode de résolution
[Coffman et al., 1983]	Items	Sacs (Bins)	Temps d'arrivée et de départ	Bin-packing dynamique	First-Fit
[Leinberger et al., 1999]	Job dans le CPU	CPU	Arrivée des tâches	Bin-packing	First-Fit & simulation
[Hermenier et al., 2008]	Machines virtuelles	Nœuds	Taille des tâches	Bin-packing dynamique	Heuristique
[Chan et al., 2009]	Items	Bins	Temps d'arrivée et départ des items	Bin-packing dynamique	First-Fit
[Epstein and Levy, 2010]	Items	Bins	Dimensions des objets	Bin-packing multi-dimensionnel	First-Fit
[Berndt et al., 2015]	Items	Bins	Temps d'arrivée et de départ	Bin-packing dynamique	First-Fit
[Li et al., 2015b]	Gestion des jeux dans le Cloud	Cloud	Temps d'arrivée et de départ	Bin-packing dynamique	First-Fit
[Ren and Tang, 2016]	Items	Bins	Temps d'arrivée	Bin-packing dynamique	First-Fit & Best-Fit
Notre problème	**Traitement d'une rangée de plantes**	**Robot qui effectue une mission de traitement**	**Nouvelle maladie & évolution de la maladie**	**Bin-packing dynamique**	**SMA & Simulation-optimisation**

TABLE 3.2 : Résumé d'état de l'art sur le problème bin-packing dynamique

3.4 Modélisation du problème

Afin de résoudre le problème d'ordonnancement des tâches d'UV-Robot, nous avons d'abord étudié un environnement statique où l'évolution des niveaux de maladie est stable. Ensuite, le problème a été rapproché du problème de Bin-Packing, bien connu en recherche opérationnelle. Nous avons développé un modèle mathématique afin de résoudre le problème avec des méthodes exactes. Dans cette partie il y a l'explication de l'approche dans un premier temps, ensuite la présentation du modèle, puis la discussion de quelques méthodes utilisées [Mazar et al., 2019].

Dans la figure 3.2, un schéma montre une serre avec N rangées où le robot doit traiter les plantes. Les couleurs indiquent les différents niveaux de maladies. Le point de départ du robot se trouve à la station de charge, il commence ses missions après avoir chargé sa batterie. Comme mentionné précédemment, ce problème est similaire au problème de bin-packing. Dans ce problème, les tâches de traitement et de déplacement du robot sont considérées comme les objets (items) du bin-packing, et les missions de traitement comme les bacs (bins) de bin-packing. Par conséquent, chaque mission doit être remplie par plusieurs tâches de traitement et de déplacement.

On trouve également dans la figure 3.2 les valeurs de consommation d'énergie du robot pour effectuer chaque tâche, notées par w_{ij}. Si $i = j$, w_{jj} correspond à une tâche de traitement où le robot doit consommer une énergie de w_{jj} pour traiter la ligne j. Sinon, pour $i \neq j$ le robot doit consommer une énergie de w_{ij} pour se déplacer de la rangée i à la rangée j. La station de charge est représentée par w_{00}. En effet, les tailles des tâches sont les w_{ij} et la capacité de la mission est l'énergie du robot.

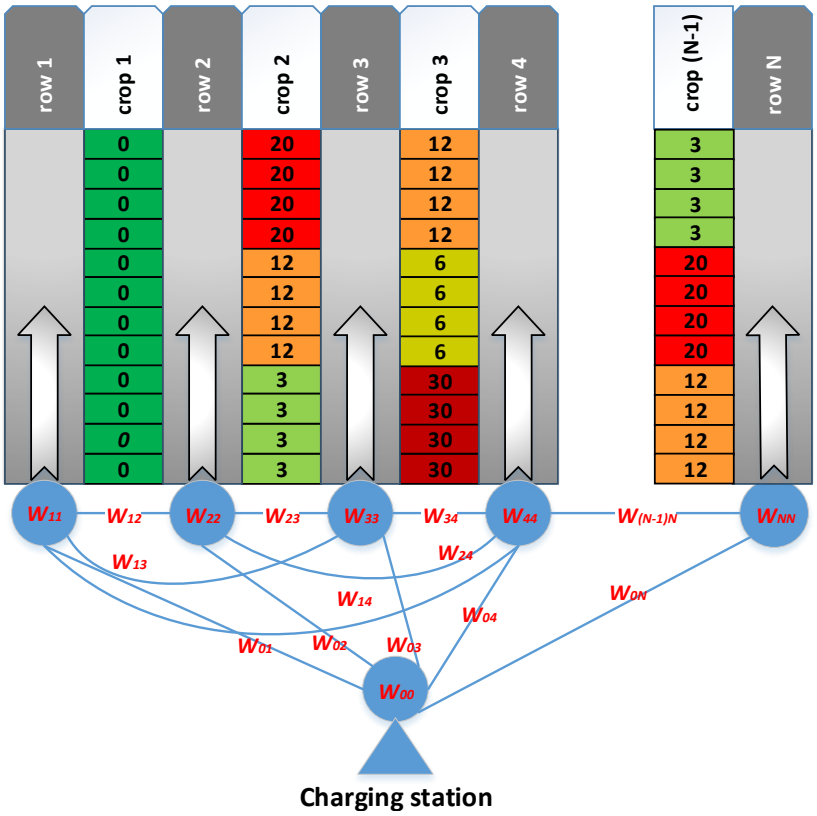

FIGURE 3.2 : Schéma de la serre avec les tâches assignées au robot

Les valeurs de w_{ij} sont représentées dans la matrice W de l'équation (3.1). Dans la diagonale de W se trouvent les valeurs de consommation des tâches de traitement, et hors diagonale de W se trouvent les valeurs de consommation des tâches de déplacement d'une rangée à une autre.

$$W = \begin{pmatrix} w_{00} & w_{01} & \cdots & w_{0N} \\ w_{10} & w_{11} & \cdots & w_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N0} & w_{N1} & \cdots & w_{NN} \end{pmatrix} \tag{3.1}$$

Si on prend un exemple d'une serre de quatre rangées, où la mission du robot a une capacité de traiter les rangées 1, 3 et 4, la matrice de décision (X) suivante montre un exemple de choix de ces 3 rangées parmi 4 et leurs déplacements. Un élément X_{ij} égal à 1 représente une tâche à effectuer par le robot. Si $i = j$, il s'agit d'une tâche de traitement de la rangée i ou j, sinon il s'agit d'une tâche de déplacement de la rangée i vers j.

$$X = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Nous avons modélisé notre problème sous forme d'un PLNE comme suit :

$$Minimiser : \quad Z(Y) = \sum_{k \in K} y^k \tag{3.2}$$

Sous les contraintes :

$$\sum_{i=0}^{N} \sum_{j=0}^{N} w_{ij} x_{ij}^k \leq C y^k \qquad \forall k \in \{1 \cdots K\} \tag{3.3}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{N} x_{ij}^k = x_{jj}^k \qquad \forall j \in \{0, .., N\} \quad \forall k \in \{1, .., K\} \tag{3.4}$$

$$\sum_{\substack{j=0 \\ j \neq i}}^{N} x_{ij}^k = x_{ii}^k \qquad \forall i \in \{0, .., N\} \quad \forall k \in \{1, .., K\} \tag{3.5}$$

$$y^k \geq y^{k+1} \qquad \forall k \in \{1, .., K\} \tag{3.6}$$

$$\sum_{i=1}^{N} x_{ii}^k \geq y^k \qquad \forall k \in \{1, .., K\} \tag{3.7}$$

$$x_{ii}^k \leq x_{00}^k \qquad \forall i \in \{1, .., N\} \quad \forall k \in \{1, .., K\} \tag{3.8}$$

$$\sum_{i=1}^{N} \sum_{j=0}^{i-1} x_{ij}^k = y^k \qquad \forall k \in \{1, .., K\} \tag{3.9}$$

$$\sum_{k=1}^{K} x_{ii}^k \geq \frac{w_{ii}}{\max\limits_{j=0,..,N} w_{jj}} \qquad \forall i \in \{0, .., N\} \tag{3.10}$$

Les définitions de différentes notations utilisées sont :

- C : Capacité d'alimentation (en unités de puissance) de la batterie du robot au début de la mission
- N : Taille de la matrice carrée W $(N \times N)$.
- K : Nombre de missions.
- w_{ij} : Consommation d'énergie (en unités de puissance) de la tâche ij
- x_{ij}^k : Variable de décision binaire permettant d'assigner les tâches aux missions

  $x_{ij}^k = \begin{cases} 1 & \text{si le robot se déplace directement de la rangée i vers la rangée j pendant la mission k} \\ 0 & \text{sinon.} \end{cases}$

  Si $i = j$ le robot se déplace dans la rangée pour traiter les plantes
- y^k : Variable de décision binaire permettant de planifier les missions $y^k = \begin{cases} 1 & \text{si la mission k est planifiée} \\ 0 & \text{sinon.} \end{cases}$

La fonction objectif (3.2) est de minimiser le nombre total de missions du robot pour traiter toute la serre.
La contrainte (3.3) : la consommation totale d'énergie pour effectuer les tâches de chaque mission k ne doit pas dépasser la capacité de la batterie du robot.
Les contraintes (3.4) et (3.5) : si une rangée est visitée, cela signifie que le robot doit provenir d'une autre rangée et doit partir vers une autre, respectivement.
La contrainte (3.6) : signifie que la mission numéro $k+1$ ne peut pas être planifié si la $k^{\text{ème}}$ mission n'est pas déjà planifié.
La contrainte (3.7) : si une mission est planifiée, elle contient au moins une rangée à visiter.
La contrainte (3.8) : aucune rangée i ne peut être planifiée si la $k^{\text{ème}}$ mission n'est pas planifié ($x_{00}^{k} = 0$).
La contrainte (3.9) : lors de chaque mission, le robot retourne à la station de charge.
La contrainte (3.10) : cette contrainte force le programme à planifier des traitements si la maladie existe ($w_{ii} > 0$).

3.5 Descriptif des méthodes d'optimisation

Dans cette section, il y a l'explication des différentes classes de méthodes d'optimisation qui ont été utilisées dans des problèmes similaires au problème étudié.

3.5.1 Méthodes exactes

Pour résoudre un problème d'optimisation avec une méthode exacte, il faut d'abord construire un modèle mathématique qui formalise une ou plusieurs fonctions objectif à optimiser (minimiser ou maximiser) et des contraintes à respecter. Le modèle mathématique peut être écrit sous forme de : Programme Linéaire (PL), Programme Linéaire en Nombres Entiers (PLNE), programme mixte, en anglais Mixed Integer Programming (MIP), Integer Programming (IP) ou Integer Linear Programming (ILP). Le type de programme dépend de sa linéarité, de son nombre d'objectifs, de ses contraintes et de ses types de variables. Le tableau 3.3 résume les différents modèles mathématiques en fonction de leurs caractéristiques [Klement, 2014].

	Variables entières	Variables réelles
Problème linéaire	PLNE ou ILP	PL ou MILP
Problème non linéaire	IP	MIP

Table 3.3 : Les différents types de modèles

Algorithme du simplexe

Le simplexe permet de résoudre les programmes linéaires qui contiennent des variables réelles [Dantzig, 1990]. Il utilise plusieurs itérations depuis une solution réalisable en passant par des solutions voisines jusqu'à ce qu'il atteigne la solution optimale.

Les méthodes suivantes, quant à elles, peuvent être utilisées dans le cas de variables entières (programme linéaire en nombre entier).

Génération de colonnes

Cette méthode, qui est une amélioration du simplexe, consiste à décomposer les contraintes du programme afin de faire une relaxation. La génération de colonnes permet de résoudre des grands problèmes de la recherche opérationnelle (RO) [Desaulniers et al., 2006].

Algorithme de Benders

C'est une méthode qui est inverse à la génération de colonnes, où elle fait la génération de ligne. Cet algorithme génère des contraintes au fur et à mesure de sa progression vers la solution [Benders, 1962].

Branch-and-bound

C'est un algorithme de séparation et évaluation qui cherche une solution optimale en parcourant toutes les solutions possibles. Mais cette méthode permet de stopper des parcours en déduisant que les solutions d'après ne sont pas optimales ou réalisables. On peut imaginer cette méthode comme une méthode arborescente qui cherche une solution optimale dans un nœud. Lorsqu'elle parcourt les nœuds de l'arbre elle s'arrête dans certains qui sont meilleurs par rapport à leur descendants pour éviter d'aller jusqu'à la racine.

Cutting planes

La méthode des plans sécants permet de trouver une solution optimale en nombre entier en rajoutant des contraintes au problème. En effet, tant qu'on a une solution de programme linéaire x* qui n'est pas entier, cette méthode crée une contrainte qui conserve la solution optimale en nombre entier, mais x* viole cette contrainte.

Branch-and-cut

Cette méthode est la fusion des algorithmes de Branch-and-bound et Cutting planes [Padberg and Rinaldi, 1991]. Elle applique l'algorithme Cutting planes dans chaque nœud de Branch-and-bound.

Branch-and-price

La méthode branch-and-price combine l'algorithme du branch-and-bound et la génération de colonnes après relaxation du problème [Johnson, 1989]. La génération de colonnes est appliquée à chaque nœud du branch-and-bound.

3.5.2 Méthodes approchées

Les méthodes approchées permettent de résoudre les grands problèmes de la RO afin de donner une solution réalisable dans un temps réduit.

Les heuristiques

Les heuristiques sont des algorithmes inventés pour permettre de trouver une solution réalisable dans un temps court. Le principe des heuristiques est de programmer un algorithme facile à implémenter qui respecte les contraintes du problème.

Parmi les heuristiques, les plus connues sont celles dites gloutonnes, qui permettent de résoudre des problèmes de la RO de façons approchée, voire exacte. Un algorithme glouton, ou greedy algorithm, est un algorithme qui construit la solution de façon séquentielle et ne revient pas en arrière une fois qu'il a pris une décision. Par exemple, [Johnson, 1973] a développé trois heuristiques gloutonnes (Next-Fit, First-Fit, et Best-Fit) qui résolvent le problème de bin-packing très rapidement.

L'algorithme Next-Fit consiste à remplir les boîtes une par une en affectant les objets dans l'ordre. Si l'objet courant ne rentre pas dans une boîte, on ferme cette boîte pour continuer les affectations dans la prochaine boite. L'algorithme First-Fit prend un objet pour l'affecter dans la première boite disponible qui peut prendre cet objet. Les boîtes restent ouvertes jusqu'à la fin de l'algorithme. L'algorithme Best-Fit affecte l'objet courant dans la meilleure boite disponible qui minimise l'espace restant après l'affectation de l'objet.

Les métaheuristiques

Les métaheuristiques sont des algorithmes d'optimisation rapide appliqués sur des problèmes difficiles. Le développement de ces algorithmes est inspiré des phénomènes et des comportements de la vie réelle, comme l'algorithme génétique, la colonie de fourmis et le recuit simulé. Les métaheuristiques sont meilleures que les heuristiques dans la plupart des cas, car elles permettent souvent d'éviter une recherche locale pour se rapprocher vers un optimum global.

Recuit simulé

C'est une métaheuristique qui s'inspire du recuit, qui est une méthode de traitement d'un matériau permettant de modifier son état (propriétés magnétiques, par exemple), par des cycles successifs de chauffage et de refroidissement lent et contrôlé. L'algorithme de recuit simulé exploite l'analogie, en considérant des notions de température et de niveau d'énergie du système (représentant la valeur de la solution courante).

Algorithme génétique

L'algorithme génétique s'inspire de la génétique biologique de l'ADN, où la solution du problème est codée sous forme de chromosomes et la recherche de la solution est inspirée à partir du processus de sélection naturelle. Il permet de résoudre des grands problèmes d'optimisation afin de chercher une solution très performante dans un temps réduit. Les étapes de l'algorithme génétique sont :

- Sélection : sélection des parents depuis une population de départ pour engendrer des enfants
- Croisement et mutation des individus sélectionnés
- Évaluation des performances des enfants par rapport à la fonction objectif
- Sélection naturelle parmi tous les individus (déjà existants et nouvellement créés) pour former la prochaine génération

Colonie de fourmis

L'algorithme de colonie de fourmis est une métaheuristique inspirée de comportement des fourmis dans leur colonie. Cet algorithme cherche une meilleure solution en relâchant plusieurs fourmis au début d'une manière aléatoire. Chaque fourmi laisse dans son trajet une quantité de substance chimique (appelée phéromone) émettant une odeur et se vaporisant avec le temps. Les fourmis suivent les portions de chemin qui accumulent la plus grande quantité de phéromone (plus forte odeur) afin de trouver l'optimum. Pour ne pas tomber dans un optimum local, on garde une possibilité de changer de trajectoire aléatoirement pour quelques fourmis afin de trouver un meilleur chemin.

3.6 Méthodes d'optimisation proposées

Afin d'optimiser et d'analyser le système UV-Robot dans le cas statique, nous avons développé une heuristique gloutonne et une métaheuristique qui est l'algorithme génétique [Mazar et al., 2020c]. Pour évaluer leurs performances, ces méthodes approximatives ont été comparées à la solution exacte obtenue par le solveur « FICO® Xpress ».

3.6.1 Heuristique

Cette heuristique est un algorithme glouton proche du Best-Fit, où il sélectionne d'abord les rangées qui ont une consommation d'énergie maximale. Lorsqu'une rangée ne peut être sélectionnée dans une mission, l'algorithme cherche une autre rangée qui nécessite une quantité d'énergie inférieure. Il passe en revue toutes les rangées jusqu'à ce qu'il ait fini de remplir la mission. Lorsqu'une mission est terminée, il passe à la suivante.

Afin de générer les missions nécessaires pour traiter toutes les serres, cet algorithme est répété plusieurs fois où la matrice W est mise à jour sans tenir compte des tâches déjà attribuées aux missions précédentes. La figure 3.3 montre un exemple de la construction de deux missions en utilisant l'heuristique. Nous pouvons observer que la « Mission 1 » contient les plus grandes tâches (17 kW et 18 kW). La tâche de 15 kW ne peut pas être affectée à cette mission car sa capacité dépasse la capacité énergétique de la batterie du robot (45 kW), mais la tâche de 10 kW peut être traitée dans cette mission. Les autres tâches sont affectées à la « Mission 2 » à l'aide du même algorithme.

L'heuristique proposée (Algorithme 1) est un algorithme basé sur les gloutons qui affecte les tâches de traitement aux missions du robot de manière interactive. Au début de chaque itération, l'algorithme initialise le vecteur des consommations énergétiques des tâches Vc, qui correspond à la diagonale de la matrice W (ligne 2), la charge de la batterie E et la liste des tâches `TASKS=[]` sont initialisées comme vide (ligne 3 et 4). Pour être sûr que le robot puisse retourner à la station de charge, une puissance de sécurité ($Max_k w_{k0}$) sera retirée, elle correspond à la puissance nécessaire pour parcourir la distance maximale entre la station de charge et la rangée la plus éloignée (ligne 6). La règle d'affectation pour sélectionner les tâches d'une mission est la suivante : la première tâche ayant la plus forte consommation d'énergie qui peut être ajoutée (lignes 8 et 9), elle doit être inférieure ou égale à la capacité énergétique restante moins la puissance nécessaire pour se déplacer vers sa rangée depuis la rangée de la dernière tâche ajoutée. Si la puissance est suffisante, le traitement de la rangée sélectionnée est ajouté à la mission (ligne 11) et l'énergie correspondante est retirée de E (ligne 12), et ce processus se poursuit jusqu'à ce que toutes les rangées restantes soient testées.

Algorithm 1 Algorithme heuristique d'affectation de tâches de traitement

Inputs :

W	Mise à jour de la matrice de consommation d'énergie
E^0	Niveau d'énergie initial de la batterie du robot

Outputs :

TASKS	Liste des tâches assignées au robot

Variables :

Vc	Consommation d'énergie des tâches de traitement
CT	Consommation totale d'énergie pour chaque tâche
E	Niveau d'énergie restant de la batterie du robot

Initialisation :
  $Vc \leftarrow \boldsymbol{Diag}(W)$
  $E \leftarrow E^0$
  TASKS $\leftarrow [\,]$
  $i \leftarrow 0$
  $E \leftarrow E - \boldsymbol{Max}_k w_{k0}$
while $E > 0$ **do**
  $j \leftarrow \boldsymbol{ArgMax}(Vc)$
  $CT \leftarrow w_{ij} + Vc(j)$
  if $CT < E$ **then**
    TASKS $\leftarrow$ TASKS $\cup \{j\}$
    $E \leftarrow E - CT$
    $i \leftarrow j$
  end if
  $Vc(j) \leftarrow 0$
end while
$\boldsymbol{Sort}$(TASKS)
return TASKS

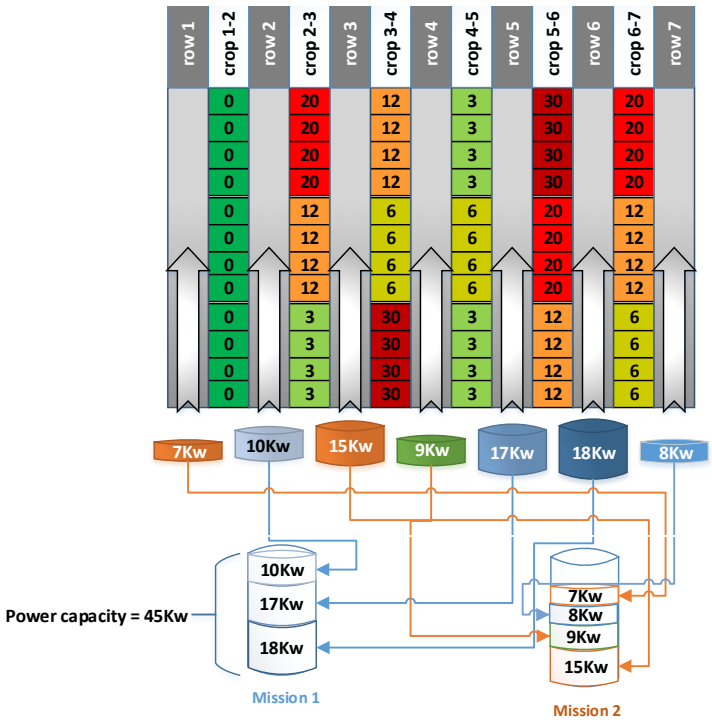

FIGURE 3.3 : Mécanisme de l'heuristique pour définir les missions du robot

3.6.2 Métaheuristique

Le chromosome de l'algorithme génétique est codé comme une matrice contenant plusieurs missions. Chaque ligne de la matrice est une mission et chaque colonne représente une rangée de la serre. Si le robot doit traiter la rangée j lors de la mission i, la valeur de la case (i, j) est égale à un, sinon elle est égale à zéro. Le fonctionnement de l'algorithme génétique est donné dans l'algorithme 2. L'algorithme commence par générer S_P matrices d'individus qui constituent la population initiale. Tant que la condition d'arrêt (nombre de générations) n'est pas satisfaite, une nouvelle génération est créée. Dans chaque génération, des opérateurs génétiques ordinaires sont utilisés pour constituer chaque population. Après chaque croisement des deux parents, l'algorithme prend les enfants obtenus pour effectuer une mutation. Après la mutation, chaque enfant est évalué à travers le calcul de sa fonction « fitnesse ». S'il ne satisfait pas les contraintes, il ne sera pas sélectionné dans la nouvelle génération. À la fin, l'algorithme renvoie le meilleur individu, qui contient le nombre minimum de missions.

Parmi les principaux opérateurs de l'algorithme génétique, le croisement est représenté sur la figure 3.4. Le croisement utilisé dans cet algorithme se fait en un seul point choisi au hasard.

Après le croisement, les enfants obtenus peuvent être mutés avec une probabilité donnée. Quatre possibilités sont définies : i) seul le premier enfant est muté, avec une probabilité de 0,3 ; ii) seul le deuxième enfant est muté, avec une probabilité de 0,3 ; iii) les deux enfants sont mutés, avec une probabilité de 0,3 ; et iv) aucun enfant n'est muté, avec une probabilité de 0,1. La figure 3.5 montre un exemple de mutation d'un enfant. La mutation se fait en choisissant deux lignes aléatoires. L'algorithme échange, point par point, avec une probabilité de 0,5, les éléments des deux lignes.

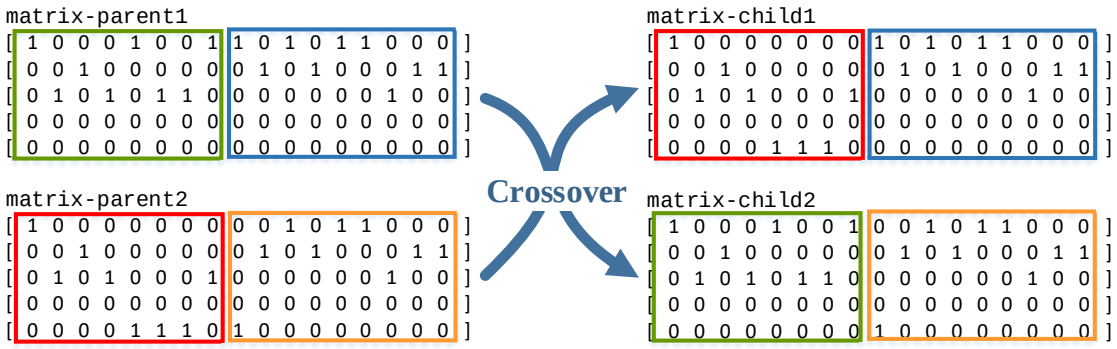

FIGURE 3.4 : Méthode de croisement

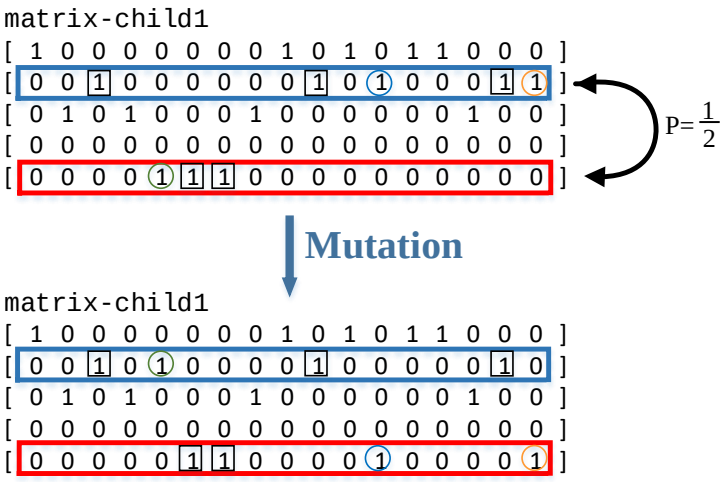

FIGURE 3.5 : Méthode de mutation

Algorithm 2 Algorithme génétique d'affectation de tâches de traitement

Inputs :
$W[Nbr_R, Nbr_R]$: Matrice de consommation de tâches
Vc : diagonale de W
E : Energie de robot
Nbr_G : Nombre de génération
S_P : Taille de la population
Initialize :
$S_{Ind} \leftarrow$ ***round_sup***$($***Sum***$(Vc)/E) + 2$ * La taille des chromosomes est aussi le nbre de missions
$Matrix_{Ind}[S_{Ind} \times Nbr_G, Nbr_R] \leftarrow zero$
for $i = 0$ to Nbr_G **do**
  for $j = 0$ to Nbr_R **do**
    if $j + 1$ *mod* $S_{Ind} = 0$ **then**
      $Proba1 \leftarrow 1$ * Assurer l'affectation des tâches restantes à la dernière mission
    else
      $Proba1 \leftarrow$ ***Random***$(0, 1)$
    end if
    $Proba2 \leftarrow$ ***Random***$(0, 1)$
    if $Proba2 < Proba1$ **then**
      $Matrix_{Ind}[i, j] \leftarrow 1$ * Affecter la tâche de la rangée j à la mission i
    end if
  end for
end for
$Generation \leftarrow 0$
while $Generation < Nbr_G$ **do**
  $Generation\_Size \leftarrow 0$
  while $Generation\_Size < S_P$ **do**
    $Father1 \leftarrow$ ***Random_one_individual***$(Matrix_{Ind})$
    $Father2 \leftarrow$ ***Random_one_individual***$(Matrix_{Ind})$
    $(Child1, Child2) \leftarrow$ ***Crossover***$(Father1, Father2)$
    $Child1 \leftarrow$ ***mutation***$(Child1)$
    $Child2 \leftarrow$ ***mutation***$(Child2)$
    if ***Respect_All_constraints***$(Child1)$= True **then**
      $New\_Matrix_{Ind} \leftarrow Child1$
      $Generation\_Size++$
    end if
    if ***Respect_All_constraints***$(Child2)$= True **then**
      $New\_Matrix_{Ind} \leftarrow Child2$
      $Generation\_Size++$
    end if
    $Matrix_{Ind} \leftarrow$***10%Best***$(Matrix_{Ind})+$***90%Best***(New_Matrix_{Ind})
  end while
  $Generation++$
end while
return ***Best_Individual***

3.6.3 Algorithme génétique dynamique

L'algorithme génétique dynamique proposé (en anglais, Dynamic genetic algorithm (DGA)) est une amélioration de Genetic algorithm (GA) dans le cas dynamique [Mazar et al., 2020a]. En effet, comme le précédent, GA est limité dans le cas dynamique, lorsque le comportement de la maladie évolue continuellement. Les solutions ne sont plus à jour et la simulation montre un GAP entre le résultat obtenu et le résultat attendu. Le GA est adapté pour prendre en compte le comportement de la maladie tout en conservant certaines opérations. La valeur ajoutée du DGA est l'utilisation de la fonction $\hat{f}(t)$ (c.f. 2.1) pour prédire l'évolution du mildiou. Lorsque l'algorithme remplit les missions par les tâches de traitement, il prend en compte une consommation d'énergie supplémentaire obtenue à partir d'une estimation des niveaux de maladie calculée par $\hat{f}(t)$. La figure 3.6 montre un exemple de solution basée sur l'algorithme DGA. Le niveau de maladie prédit augmente avec le temps en raison du temps d'attente avant l'exécution de chaque mission. Le temps d'attente d'une mission est le temps d'exécution de la mission précédente, plus le temps de charge de la batterie.

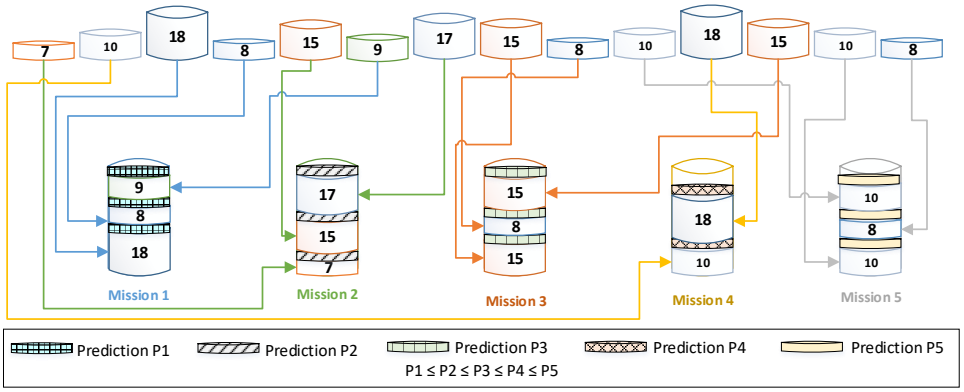

FIGURE 3.6 : Représentation de l'accomplissement dynamique des tâches de traitement dans les missions

La figure 3.7 montre la différence entre DGA et GA consommation d'énergie. Le DGA alloue toujours une partie de la capacité de la batterie à la consommation supplémentaire estimée, due à l'évolution estimée du niveau d'infection sur chaque plante, pour chaque mission. A l'inverse, le GA ne prend pas en compte l'évolution de la maladie. La figure 3.7 montre également les temps d'exécution pour les missions avec un temps de traitement de 3 heures et un temps de charge de 2 heures et 30 minutes. La fonction $\hat{f}(t)$ est utilisée pour le calcul de prédiction P_i comme indiqué dans l'équation (3.11). En effet, P_i représente la quantité d'augmentation supplémentaire du niveau d'infection au moment d'exécution de la $i^{\text{ème}}$ mission. Pour ce faire, le niveau global de la maladie est estimé à l'aide de la fonction $\hat{f}(t)$, puis la valeur mesurée enregistrée lors de la dernière action de mesure (à l'aide de E-nose ou d'une estimation humaine) est retirée.

$$Pi = \frac{\hat{f}\,(3 + 5,5(i-1) + t_m)\,\alpha i}{Nbr_{plants}/4} - \mathcal{M} \tag{3.11}$$

α est un paramètre empirique défini grâce à la simulation, $\mathcal{M}$ est le dernier taux de mildiou mesuré pour la plante concernée, et t_m est le temps estimé correspondant à la valeur du dernier taux de mildiou mesuré ($\mathcal{M}$).

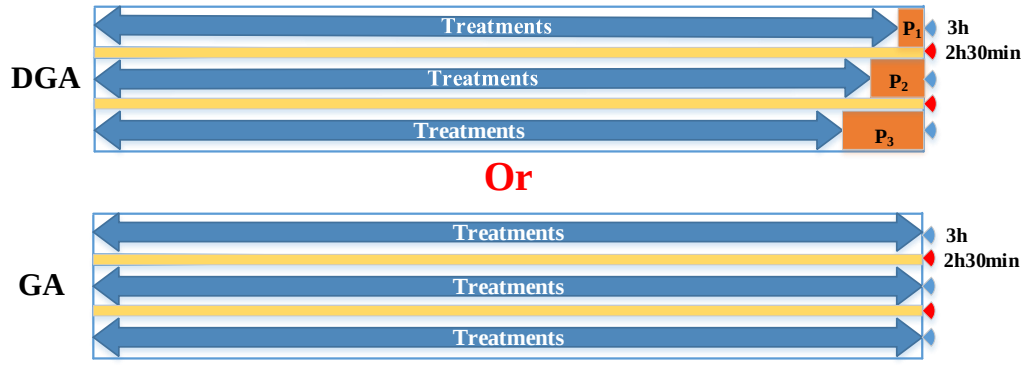

FIGURE 3.7 : Comparaison de l'attribution dynamique et statique du traitement

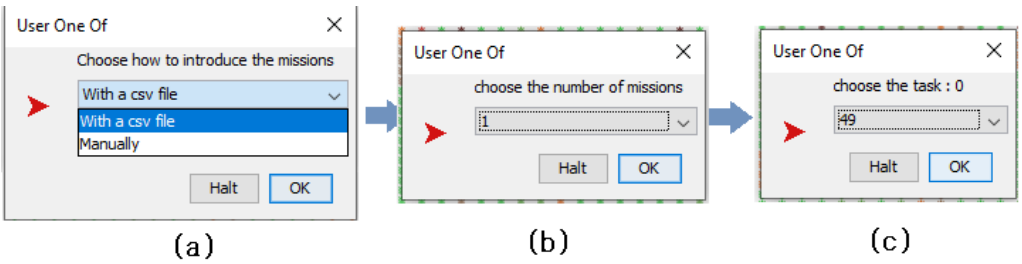

FIGURE 3.8 : Intégration de la solution obtenue par le solveur dans l'interface de simulation

3.6.4 Méthode exacte

La programmation linéaire en nombres entiers développée dans la section 3.4, a été résolue à l'aide d'un solveur commercial 'FICO ® Xpress' pour donner une solution optimale à certaines instances du problème. Afin de réduire le nombre de variables de décision, le nombre maximum de missions est défini, pour chaque instance, par le nombre de missions donné par l'algorithme heuristique.

L'utilisation de la méthode exacte se fait en dehors du simulateur. Ce sont les seuls calculs qui ne sont pas effectués à l'intérieur du simulateur. Le simulateur présenté dans la figure 2.12 commence par une étape de paramétrage avant de lancer l'initialisation et la simulation. Au début de la simulation, si l'utilisateur choisit la méthode exacte, le simulateur génère un fichier « .CSV » qui contient les données de la serre. Ce fichier « .CSV » est utilisé dans le solveur. Lorsque le solveur termine ses calculs, il indique le nombre de missions et les rangées à traiter dans chaque mission. Enfin, l'utilisateur récupère les missions et les intègre dans le simulateur à l'aide d'une interface dédiée que nous avons développée.

La figure 3.8 présente les étapes de l'interface qui permet d'intégrer la solution obtenue par le solveur dans l'interface de simulation. Au début, dans (a) l'utilisateur choisit l'entrée par fichier (.CSV) ou manuellement. Si le choix est manuel, il passe à l'étape (b) pour choisir le nombre de missions. Puis, il passe à l'étape (c) où il ajoute les rangées à traiter pour toutes les missions.

3.7 Conclusion

Ce chapitre a permis de voir la méthode utilisée pour le couplage entre simulation et optimisation, et les algorithmes d'optimisation développés. Tout d'abord, nous avons assimilé le problème de l'optimisation des missions du robot-UV à un problème de bin-packing. Nous avons pu établir un modèle mathématique sous la forme d'un programme linéaire. Ce modèle donne la possibilité de tester des méthodes exactes pour comparer les résultats qui seront présentés dans le dernier chapitre de cette thèse (cf. chapitre 4). La résolution par la méthode exacte atteint ses limites pour des instances de grande ou moyenne taille (une serre de 100 rangées avec un taux de maladie de 50%). Pour surmonter cette limitation, nous avons proposé d'autres algorithmes d'optimisation tels que l'heuristique gloutonne et les métaheuristiques GA et DGA. Le codage de ces algorithmes est intégré dans l'agent de supervision « `Monitorings` », car c'est lui qui envoie les missions au robot. Ces algorithmes d'optimisation ont été testés dans plusieurs situations qui seront discutées dans le prochain chapitre (cf. chapitre 4).

Chapitre 4

Simulation et optimisation des types de traitement robotisé

Contenu

4.1 Introduction

Dans ce chapitre, il y a une explication des différents types de traitements qui peuvent être utilisés dans le système étudié dans cette thèse. Les traitements des plantes sont considérés comme des tâches de maintenance des machines dans l'industrie. En général, le terme de type de traitement est utilisé pour les maladies [Wu et al., 2021], [Rao et al., 2009]. Chaque type de traitement étudié nécessite un paramétrage spécifique et des algorithmes d'optimisation dans le simulateur. La maintenance est utilisée pour améliorer la durée de vie des machines ou pour réparer les machines en panne qui ont un ou plusieurs composants défectueux. Dans ce cas, le traitement est utilisé pour améliorer la durée de vie des plantes, mais les plantes mortes ne peuvent pas être traitées.

Dans une première version du simulateur, il y avait un seul type de traitement. Puis, nous avons voulu étudier d'autres scénarios pour tester d'autres alternatives pour faire des traitements. En outre, nous avons choisit de diviser nos scénarios par types, nommés pareil que ceux de la maintenance, ce qui permet de faciliter l'étude du système et comparer les différentes stratégies.

Le plan d'expérience relatif à tous les scénarios étudiés sera présenté dans ce chapitre. Plusieurs simulations ont été réalisées depuis la première version du simulateur. Il y a également la présentation de tous les tests effectués et le plan expérimental de chacun d'entre eux.

Ce chapitre est divisé en cinq sections, y compris la conclusion. Au début, nous présenterons un état de l'art sur les différents types de maintenance identifiés dans le secteur industriel et sur le couplage entre la simulation et l'optimisation. Puis, dans les trois sections suivantes, nous expliquerons le fonctionnement des scénarios de chaque type de traitement (prédictif, conditionnel et calendaire), avec leur plan expérimental et les résultats des simulations.

4.2 États de l'art

Cette section consiste à présenter un état de l'art sur les problèmes de maintenance qui seront pris en compte pour définir les types de traitement. Ces derniers représentent les scénarios qui seront expérimentés et testés sur le simulateur. Une autre partie de l'état de l'art sur le couplage entre simulation et optimisation sera présentée. Cette approche représente la base du travail de ce papier, qui permet d'améliorer la prise de décision du problème d'ordonnancement du traitement de tâches évolutives sur le robot.

4.2.1 Problèmes de maintenance

La compétition entre les entreprises de fabrication et de service a créé une énorme pression sur le monde industriel. Ainsi, pour bien se positionner par rapport à ses concurrents, une entreprise doit avant tout fournir des produits de haute qualité. De plus, les gestionnaires sont obligés d'optimiser l'efficacité et les coûts de maintenance et augmenter la disponibilité de leurs outils de production. Les types de maintenance disponibles aujourd'hui incluent la maintenance corrective, effectuée après l'occurrence de défaillance d'équipement [Baglee and Jantunen, 2014], la maintenance calendaire [Wang, 2012], la maintenance préventive conditionnelle [Selim and Gurel, 2007], la maintenance centrée sur la fiabilité [Ribrant, 2006] et le type la maintenance prédictive [Shcherbakov et al., 2020].

Plusieurs travaux sur la maintenance utilisent un ou plusieurs types de maintenance pour augmen-

ter la durée de vie des machines. Le déclenchement de la maintenance dépend des événements liés aux pannes des machines et à la dégradation des composants. Par exemple, [Hevesh, 1967] a étudié trois types de maintenance pour un radar dans lequel tous les éléments défaillants pouvaient être détectés immédiatement :

- Maintenance immédiate : les éléments défaillants sont détectés, localisés et remplacés immédiatement
- Maintenance cyclique : les éléments défaillants sont détectés, localisés et remplacés périodiquement
- Maintenance différée : les éléments défaillants sont détectés et localisés périodiquement, et remplacés lorsque leur nombre a dépassé un niveau prédéfini.

Pour évaluer ces types de maintenance, l'auteur a calculé pour chaque type les temps moyens de défaillance et la disponibilité de l'équipement.

[Dhillon, 2002] et [Ben-Daya et al., 2009] ont présenté plusieurs modèles pour différents types de maintenance. Leurs approches sont également basées sur des outils d'optimisation et de simulation pour résoudre les problèmes de maintenance. Les auteurs prennent en compte aussi les erreurs humaines dans la maintenance. Dans les articles examinés dans ce travail, les différents types de maintenance sont très liés au contexte du problème étudié. Par exemple, dans [Sahnoun et al., 2019], les auteurs ont défini trois types de maintenance pour les éoliennes offshore : corrective, préventive conditionnelle et préventive systémique.

Au début des travaux de thèse, il y avait une étude sur un type de traitement des maladies des plantes. Puis, nous avons remarqué qu'il y a la possibilité d'étudier d'autres types de traitements. Le choix des types de traitement se fait comme celui des types d'entretien. La figure 4.1 montre un schéma utilisé pour positionner les différents types de traitement d'une manière similaire à ce qui se fait en maintenance industrielle. Chaque type de traitement choisi représente un scénario dans le simulateur, sauf le type correctif. En réalité, le traitement correctif n'est pas possible pour les plantes, car si une plante meurt, elle ne peut pas être ressuscitée. Pour cette raison, nous ne prenons pas en compte ce type de traitement dans le simulateur, même s'il est théoriquement possible d'imaginer un traitement « miracle » ou de considérer le remplacement de la plante « morte » par une autre comme une forme de traitement. Pour le cas préventif, trois traitements possibles sont étudiés : Conditionnel, Prédictif et Calendaire.

La maintenance corrective est une stratégie, introduite en 1957, dans laquelle l'effort de prévention des pannes d'équipement est élargi pour être appliqué à l'amélioration de l'équipement de façon à ce que les pannes puissent être évitées. Son objectif est d'éliminer les défaillances (amélioration de la fiabilité) et de faciliter la maintenance de l'équipement. La principale différence entre maintenance corrective et préventive est qu'un problème doit exister avant que des mesures correctives soient prises. L'objectif de la maintenance corrective est le suivant : améliorer la fiabilité, la maintenabilité, la sécurité et les faiblesses de conception (matériaux, formes) de l'équipement. Les stratégies de maintenance corrective visent à réduire les détériorations, les défaillances afin que les équipements ne nécessitent pas d'intervention rapide [Ben-Daya et al., 2009].
Une maintenance corrective se fait dans le cas d'une défaillance d'un composant d'une machine ou de son arrêt complet. Bien qu'il existe de nombreux modèles traitant ce type de maintenance (maintenance imparfaite, maintenance opportuniste...), on ne trouve pas ce genre de modèles pour le traitement correctif des maladies.

Le concept de maintenance préventive, introduit en 1951, est une sorte de contrôle physique de

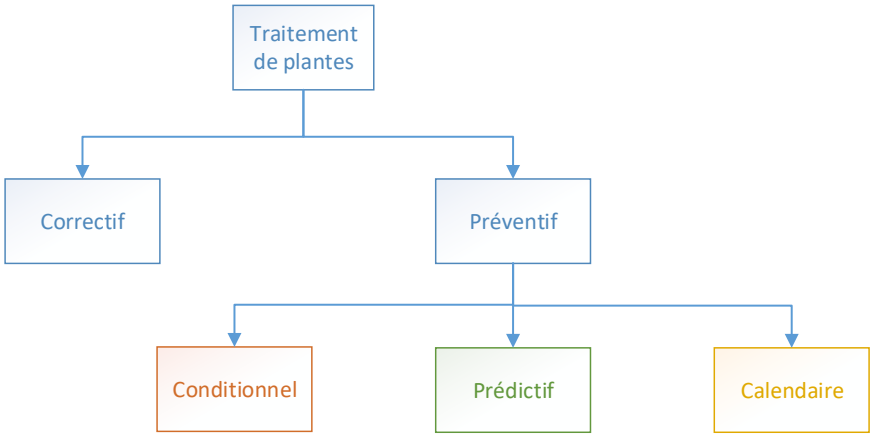

FIGURE 4.1 : Les types de traitement

l'équipement afin de prévenir les pannes et de prolonger la durée de vie de l'équipement. La maintenance préventive comprend des activités de maintenance qui sont entreprises après une période de temps ou une quantité d'utilisation de la machine spécifiée. Les activités de maintenance basées sur le temps sont généralement mieux acceptées par la production. Ce type de maintenance repose sur l'estimation de la probabilité que l'équipement tombe en panne ou que ses performances se détériorent. Les travaux préventifs entrepris peuvent inclure la lubrification de l'équipement, le nettoyage, le remplacement de pièces, le serrage et le réglage. Ce type de maintenance inclut également les inspections qui permettent de détecter l'état d'usure des équipements et qui peut déclencher une maintenance préventive [Ben-Daya et al., 2009].

L'entretien préventif périodique permet de réduire considérablement la fréquence des pannes accidentelles afin d'assurer le fonctionnement des équipements dont les caractéristiques opérationnelles se dégradent avec l'âge. Une telle politique de maintenance propose de remplacer l'équipement de manière préventive après un âge prédéterminé ou à des moments précis indépendamment de l'âge de l'équipement. D'un point de vue économique, il est intéressant de remplacer l'équipement juste avant l'apparition de la défaillance de l'équipement. Cela n'est possible que si la dégradation de l'équipement peut être suivie, ainsi que l'évolution de la situation, et que, d'autre part, si les paramètres de l'environnement opérationnel sont contrôlés de manière pour éviter toute défaillance de l'équipement due à ces paramètres.
Contrairement au remplacement préventif périodique, les actions de maintenance conditionnelle sont étroitement liées à l'état de l'équipement. Ainsi, le remplacement préventif n'est effectué que lorsqu'un seuil d'alarme est atteint. Ceci permet de réduire le nombre de remplacements d'équipements tout en garantissant une plus grande disponibilité des équipements [Ben-Daya et al., 2009].

Dans un système, il y a plusieurs événements qui influencent le déclenchement d'une maintenance préventive conditionnelle. Après la réunion de quelques événements qui valide certaines conditions, cette maintenance peut être lancée. On peut trouver des exemples dans la littérature qui utilisent divers types de conditions : seuil de maladie, états des composants, dégradation de la qualité du produit etc...

La maintenance prédictive est initiée en réponse à un état spécifique de l'équipement ou à une détérioration des performances. Les techniques de diagnostic sont déployées pour mesurer l'état phy-

sique de l'équipement, comme la température, le bruit, les vibrations, la lubrification et la corrosion. Lorsqu'un ou plusieurs de ces indicateurs atteignent un niveau de détérioration prédéterminé, des initiatives de maintenance sont entreprises pour restaurer l'équipement dans l'état souhaité. Cela signifie que l'équipement n'est mis hors service que lorsqu'il existe des preuves directes que la détérioration a eu lieu. L'avantage supplémentaire vient de la nécessité d'effectuer la maintenance lorsqu'elle est imminente et non après l'écoulement d'une période de temps déterminée [Ben-Daya et al., 2009].

La maintenance préventive calendaire (systématique) est une approche systématique pour définir un programme de maintenance planifiée composé de tâches rentables tout en préservant les fonctions critiques de l'usine.

La maintenance calendaire est un ordre de travail récurrent qui est planifié lorsqu'un intervalle de temps spécifié est atteint dans le système de gestion de la maintenance assistée par ordinateur (GMAO). En utilisant ce type de maintenance préventive, on peut spécifier le calendrier des temps d'arrêt et informer l'équipe de gestion de la maintenance et les autres services qui seront affectés par la maintenance [Holmberg et al., 1992].

La maintenance calendaire fait référence au remplacement ou au renouvellement d'un élément pour restaurer sa fiabilité à une heure fixe, quel que soit son état. L'intervalle de temps peut être fixé à toutes les semaines ou tous les mois. Cette stratégie est très efficace pour les équipements ou les pièces qui ont tendance à durer un certain temps qui est connu et stable. Cette méthode est utilisée par la plupart des entreprises car elle rend la maintenance planifiée relativement facile. Cependant, avec cette méthode, il y a un risque d'entretenir l'équipement trop souvent ou l'inverse. Cette stratégie est souvent appliquée aux équipements actifs à courte durée de vie telle que les pompes, les moteurs de portails et les extincteurs. Un exemple pourrait être d'inspecter les extincteurs tous les 6 mois.

Dans les sections suivantes, une démonstration fera expliquer la mise au point des différents types de traitement robotique du mildiou avec le UV-C, qui ont été utilisés dans les travaux de cette thèse. Chaque type de traitement correspond à un type de maintenance qui a été défini dans cette partie. Un type de traitement peut contenir plusieurs scénarios dans le simulateur.

4.2.2 Couplage simulation et optimisation

Un simulateur est un outil qui permet de reproduire des systèmes réels afin de voir leurs évolutions dans le temps. Quand on utilise un simulateur intégrant un certain nombre de décisions durant le déroulement de simulation, ces décisions sont basées sur un ou plusieurs paramètres du système. La simulation-optimisation permet d'optimiser ces paramètres pour améliorer le fonctionnement du système simulé.
Cette méthode est utilisée par plusieurs chercheurs pour représenter, analyser et améliorer un SC [1], en se basant sur différentes techniques. Généralement, une métaheuristique adaptée au système est choisie, comme l'Algorithme Génétique (en anglais, GA), l'ACO ou le Recuit simulé (en anglais, Simulated annealing (SA)).
L'approche du couplage simulation et optimisation est apparue pour la première fois dans les années 90' [Carson and Maria, 1997]. Elle prend plusieurs formes qui seront détaillées par la suite. Dans [Farzanegan and Vahidipour, 2009], les auteurs ont étudié l'intégration de l'algorithme d'optimisation GA avec un simulateur de circuit de meulage préexistant, appelé simulateur de circuits de fraisage à

1. C'est un système où il est impossible de définir exactement son état prochain (à cause des aléa, nombres d'agent, dynamique interne, ...)

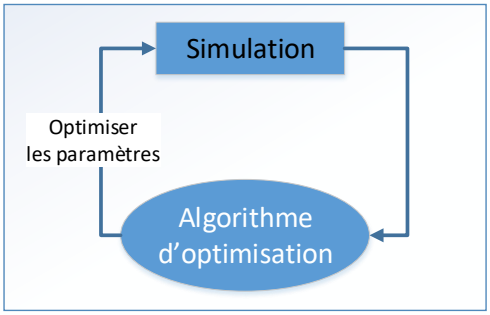

FIGURE 4.2 : Premier type de couplage simulation-optimisation : Optimisation globale du simulateur

billes (en anglais, « ball milling circuits simulator », ou BMCS), dans l'environnement MATLAB. Dans [Lim et al., 2009], les chercheurs ont proposé un SMA qui exploite la dimension efficace de l'agilité technique, en mettant particulièrement l'accent sur la réactivité des programmes de production qui rencontrent des perturbations. Leur idée est d'utiliser une procédure d'enchères itérative, pilotée par un mécanisme d'optimisation de recuit simulé, jusqu'à ce que les coûts de production totaux soient minimisés. Le mécanisme global d'optimisation est alors supporté par des multiples exécutions de l'algorithme de recuit simulé.

[Powell, 2008] a publié plusieurs articles qui adaptent la méthode de simulation-optimisation en utilisant la programmation dynamique approximative pour résoudre différents problèmes d'optimisation. La programmation dynamique approximative est utilisée pour produire des fonctions de décision optimisées dans le temps. Elle permet au simulateur d'apprendre et de s'adapter dans le temps avec les meilleurs comportements. Dans certains des articles, Powell et ses coauteurs se basent sur le problème du transport aérien des avions militaires des États Unis d'Amérique, comme dans [Wu et al., 2003] et [Powell, 2005].

Dans [Wu et al., 2003] les auteurs ont présenté leur méthode DRTP (en anglais, « dynamic resource transformation problem »). [Baker et al., 2002] et [Crino et al., 2004] ont mis au point des méthodes basées sur les modèles d'optimisation comme NRMO (en anglais, « NPS/RAND Mobility Optimizer ») créé par l'École Supérieure Navale (NPS) et l'entreprise RAND Corporation. [Ryer, 1999] ont présenté des modèles de simulation comme le MASS (Système d'aide à l'analyse de la mobilité). [Ören et al., 2014] ont développé une méthode basée sur la programmation stochastique en multi étapes SSDM (en anglais, stochastic sealift deployment model). Les différents chercheurs qui utilisent les méthodes NRMO, MASS, SSDM et DRTP ont le même problème à résoudre, c'est le problème du transport aérien des avions militaires des États Unis. L'objectif est de transporter les marchandises et les passagers. Ils ont utilisé les mêmes données TPFDD (Time-Phased Force Deployment Data) qui constituent une liste très détaillée des cargaisons et des troupes. Ces données sont requises par les plans d'urgence pour un théâtre d'opération donné en 1993. Ces listes sont présentées dans [Baker et al., 2002].

Nous distinguons trois types de couplages entre la simulation et l'optimisation. Dans le premier type de couplage, représenté dans Figure 4.2, le simulateur est considéré comme étant la fonction d'évaluation pour l'algorithme d'optimisation. L'optimisation permet de modifier les paramètres d'entrées du simulateur jusqu'à l'obtention les paramètres de la solution optimisée [Sahnoun et al., 2016].

La figure 4.3, représente le deuxième type de couplage. A chaque instant de prise de décision durant

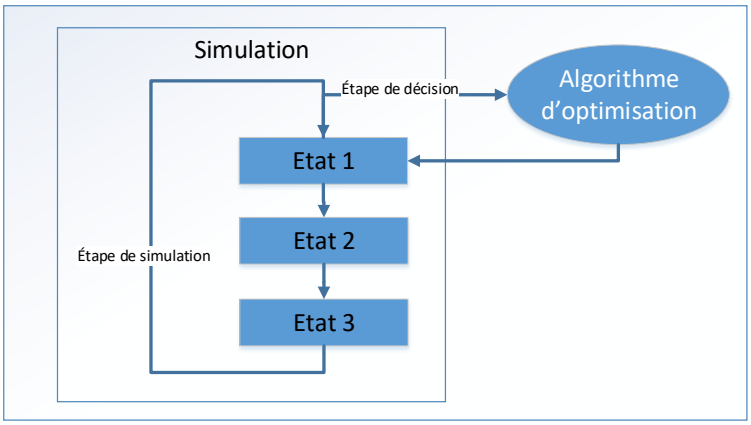

FIGURE 4.3 : Deuxième type de couplage simulation-optimisation : Optimisation locale du simulateur

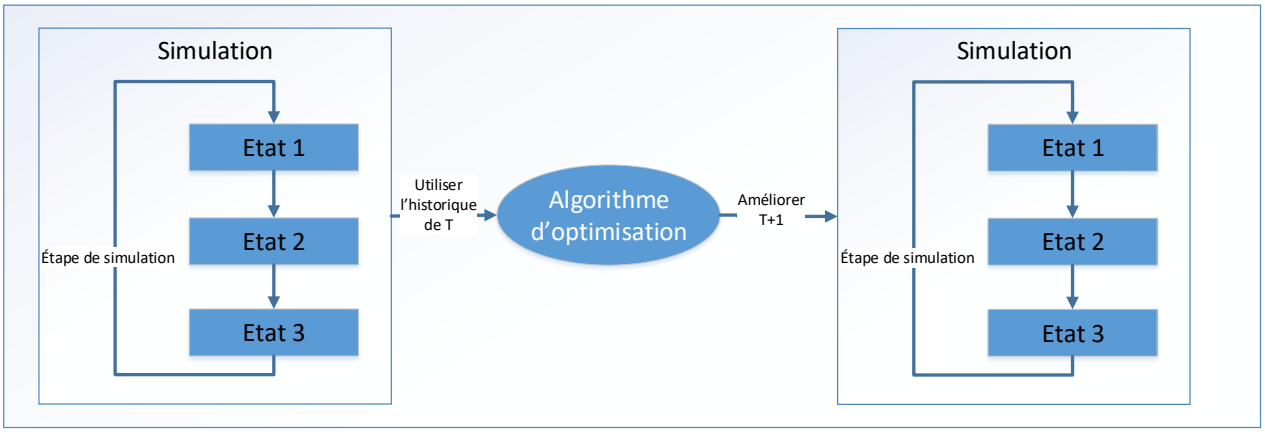

FIGURE 4.4 : Troisième type de couplage simulation-optimisation : Sim-optimisation

la simulation, l'optimisation améliore le comportement d'un état du système [Fu, 2002].

Dans le troisième type de couplage (Figure 4.4), l'optimisation prend une décision à l'instant T en considérant l'impact probable à l'instant T+1 [Campi and Garatti, 2018].
La Sim-optimisation permet d'estimer l'état prochain en fonction de la décision actuelle en se basant sur l'historique du système [Powell, 2005], [Powell, 2008] et [Wu et al., 2003].

4.3 Traitement préventif conditionnel

4.3.1 Fonctionnement

Au début du développement du simulateur nous avons créé un seul type de traitement. Comme les types de traitement avec cette nouvelle technologie qu'est l'UV-C n'étaient pas prévus, nous avons commencé avec un scénario tout simple. Ce scénario consiste à faire un traitement robotisé d'une serre si la maladie existe et dépasse un seuil. Sinon le robot est dans la station de charge pour attendre un ordre de lancement de l'agent monitoring. Les premiers tests sont effectués pour des serres de 50

rangées avec une heursitique basée sur un algorithme glouton. Cette heuristique choisit des rangées pour que le robot traite la serre en commençant par les rangées où il y a les plantes les plus infectées jusqu'au moins infectées.

La condition de lancement de ce traitement est le dépassement d'un seuil de niveau global de la maladie dans la serre. La particularité de ce traitement est l'évolution semi-dynamique de la maladie, c'est-à-dire que sa mise à jour se fait toutes les 24 heures. Pour les premiers tests, nous avons choisi de simplifier le système étudié pour voir comment les algorithmes d'optimisation fonctionnent. L'évolution de la maladie suit un phénomène aléatoire défini dans le chapitre 2. Afin de simuler plusieurs instances avec des plantes différentes, un paramètre de vitesse d'évolution de la maladie représenté par un coefficient multiplicateur de la probabilité d'évolution a été ajouté. Par exemple, si l'utilisateur choisit un coefficient de 0,5, la maladie augmente d'un niveau sur la moitié des plantes de la serre en 24 heures. L'évolution de la maladie est arrêtée pendant 24 heures et n'évolue qu'à la fin de cette période. On obtient donc une boucle d'évolution de la maladie mise à jour toutes les 24 heures. Cela peut correspondre dans la vie réelle, à une mesure de la maladie qui n'est enregistrée qu'une fois toutes les 24 heures.

Une simulation de plusieurs scénarios a été réalisée sur le traitement préventif conditionnel, où trois tailles de serres ont été testées : 50 rangées, 70 rangées et 100 rangées. Sachant que dans le projet UV-Robot, les partenaires disposent de grandes serres (taille de 100 rangs). Le choix des serres de 50 rangs est d'avoir des résultats optimaux rapidement lors de l'optimisation avec la méthode exacte. Dans un deuxième temps, en faisant varier le paramètre de probabilité d'évolution de la maladie. Les probabilités utilisées sont 0,5, 0,7 et 1, où 1 signifie que la maladie est présente sur toutes les plantes de la serre. En combinant les scénarios pour les trois tailles de serre avec les trois paramètres de probabilité, on obtient 9 scénarios à simuler. Le scénario le plus exécrable est celui avec une serre de 100 rangs et une probabilité d'évolution égale à 1. L'exécution des algorithmes d'optimisation pour ce scénario prend plus de calcul, le temps de résolution augmente surtout pour la méthode exacte, où nous l'avons arrêté plusieurs fois après 24 heures d'exécution considérant que l'algorithme ne converge pas dans un temps raisonnable.

Dans tous ces scénarios, un seul robot effectue l'ensemble du traitement de la serre. Le robot modifie sa vitesse pendant le traitement à chaque niveau de maladie. Le traitement est effectué des deux côtés de chaque couloir en même temps. La vitesse du robot est choisie en fonction du niveau de maladie la plus élevée parmi les deux plantes situées de chaque côté du robot. Par exemple, si les plantes du côté gauche sont plus touchées dans une rangée, le robot adapte sa vitesse pour le côté gauche et vice-versa. Comme les vitesses du robot changent, la consommation d'énergie ne sera pas linéaire dans le temps. Le retour du robot se fait avec sa vitesse maximale en désactivant les lampes UV-C. Le robot consomme plus d'énergie pour traiter les niveaux d'infection les plus élevés car il utilise les lampes UV-C plus longtemps.

La figure 4.5 montre un schéma qui donne une vue globale du fonctionnement du traitement conditionnel dans le système UV-Robot. Comme présenté en haut à gauche de la figure 4.5, un suivi de l'évolution de maladie permet le déclenchement de ce traitement. Si le seuil est atteint, un algorithme d'optimisation détermine la liste des tâches au robot. Le robot commence les missions des traitements tout en respectant l'ordre des rangées à traiter. Le robot change la vitesse pendant le traitement afin de donner les doses d'UV-C suffisantes pour éradiquer la maladie. Puisque ce traitement est dans un cas semi-dynamique, la maladie continue à évoluer avec une mise à jour chaque 24 heures.

Pour optimiser le traitement conditionnel, trois méthodes sont testées : l'heuristique, l'algorithme

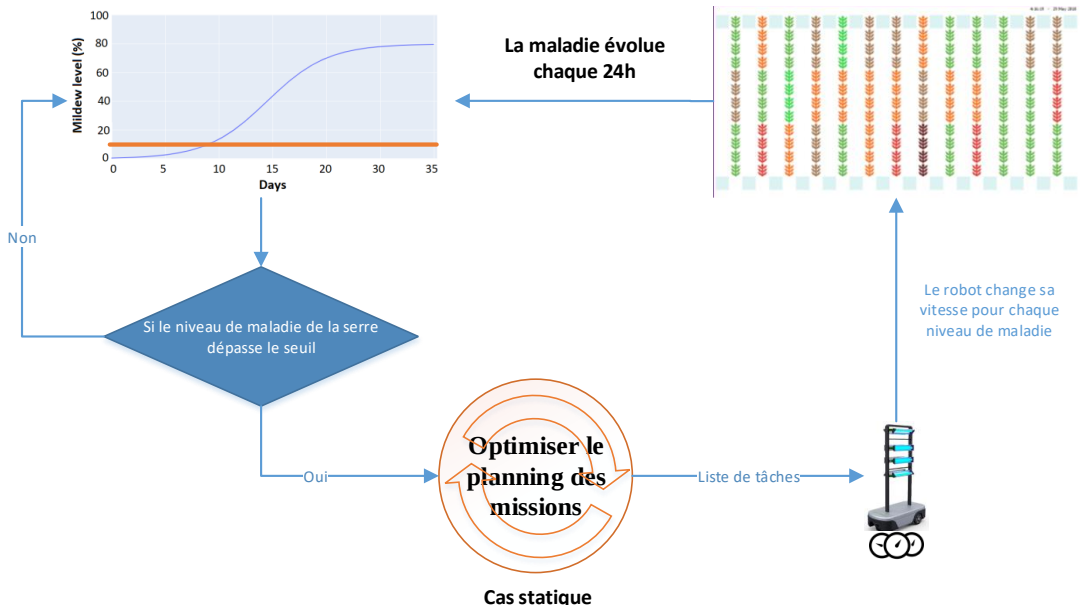

FIGURE 4.5 : Schéma du traitement préventif conditionnel

génétique et la méthode exacte présentées dans le chapitre 3. L'objectif est de minimiser le nombre de missions du robot pour minimiser le temps total de traitement de la serre. L'évolution de la maladie, qui est stable pendant 24 heures, permet de tester ces algorithmes, et d'éviter le problème du cas totalement dynamique, notamment pour la méthode exacte.

4.3.2 Expérimentation

Le plan d'expérimentation et les résultats numériques de la simulation du traitement conditionnel seront présentés. Dans les résultats graphiques, nous présentons la consommation d'énergie du robot pour une simulation. Ensuite, il y a les graphiques pour le niveau moyen de mildiou dans la serre, où deux cas ont été testés. Le premier est un cas statique, où il n'y a pas d'évolution de la maladie et le second est un cas semi-dynamique, où la maladie évolue et son niveau est mis à jour seulement toutes les 24 heures.

Cas statique

Dans ce cas, toutes les méthodes d'optimisation sont comparées à la méthode exacte en calculant le GAP [2].
Le plan d'expérimentation est défini en fonction des paramètres suivants :

- Évolution de maladie : statique (la maladie n'évolue pas)
- Nombre de robots : 1
- Autonomie du robot : 30 min de traitement pour 4h de chargement
- Vitesse du robot : le robot adapte sa vitesse pour chaque niveau de maladie

2. Le terme est souvent qualifié d'écart absolu, qui est l'amplitude de la différence entre la meilleure solution connue et la meilleure borne, ou d'écart relatif, qui est l'écart absolu divisé par la meilleure borne.

— Taille de serre : 50, 75 et 100 rangées
— Taux moyen initial d'infection : 50%, 75% et 100%
— Nombre de scénarios : 9
— Nombre de simulations par scénario : 30
— Nombre de plantes par rangée : 100
— Horizon de simulation : jusqu'au traitement total de la serre. (plus ou moins 1 semaine)

Cas semi-dynamique

Les algorithmes d'optimisation GA et Heuristic algorithm (HA) ont été testés dans ce cas pour voir lequel est le plus efficace lorsque la maladie évolue. À cette fin, le GAP n'a pas été calculé car la maladie évolue de manière semi-continue et stochastique toutes les 24 heures. Sachant que le GAP est calculé à partir d'une solution exacte et que la partie dynamique n'est pas intégrée dans le modèle mathématique.
Le plan d'expérimentation est défini en fonction des paramètres suivants :
— Évolution de maladie : semi-dynamique (évolution chaque 24 heures)
— Nombre de robots : 1
— Autonomie du robot : 30 min de traitement pour 4h de chargement
— Vitesse du robot : le robot adapte sa vitesse pour chaque niveau de maladie
— Taille de serre : 100 rangées
— Taux moyen initial d'infection : 50%
— Nombre de scénarios : 1
— 100 plantes par rangée
— Horizon de simulation : jusqu'au traitement total de la serre. (plus ou moins 1 semaine)

4.3.3 Résultats

Cas statique

Nous supposons dans ce cas que le taux de la maladie n'évolue pas, qu'il diminue seulement avec l'effet du traitement, qu'il n'y a pas non plus d'apparition de nouvelle maladie. D'un point de vue pratique, cela n'est possible que si le taux d'évolution de la maladie est très faible et que le traitement est effectué assez rapidement. Les résultats des trois algorithmes d'optimisation (GA, HA et Exact method (EM)) seront comparés.

La figure 4.6 montre deux courbes du niveau de charge de la batterie du robot pendant le traitement d'une serre composée de 50 rangées de plantes avec une probabilité d'apparition de la maladie P égale à 0,5. Dans les deux courbes, relatives aux méthodes de planification de missions (méthode exacte (en anglais, EM) et heuristique (en anglais, HA), le robot utilise une batterie ayant une capacité énergétique de 960 Wh lui permettant d'exécuter une mission d'une durée environnant 30 minutes, avant de devoir la recharger pendant environ 4 heures. Le traitement de toutes les rangées infectées est réalisé en 7 missions avec EM et en 8 missions en HA.
La figure 4.6 montre également qu'il n'y a quasiment pas de différence entre les deux méthodes dans les cinq premières missions. Cependant, dans les deux dernières missions, HA ne permet pas au robot d'utiliser toute l'énergie disponible sur sa batterie, ce qui a amené à l'ajout d'une mission. Ceci peut s'expliquer par le fait que, dans les deux dernières missions, HA ne trouve aucune mission pouvant

être exécutée en utilisant l'énergie restante. Dans le même temps, la solution optimale utilise toute l'énergie disponible lors de chaque mission. Le traitement total avec EM consomme environ 2% moins d'énergie que la méthode heuristique et termine le traitement 3 heures et 40 minutes plus tôt. Sur la base de ces observations, nous pouvons conclure que l'heuristique peut être une bonne alternative grâce à son temps d'exécution et la qualité de sa solution.

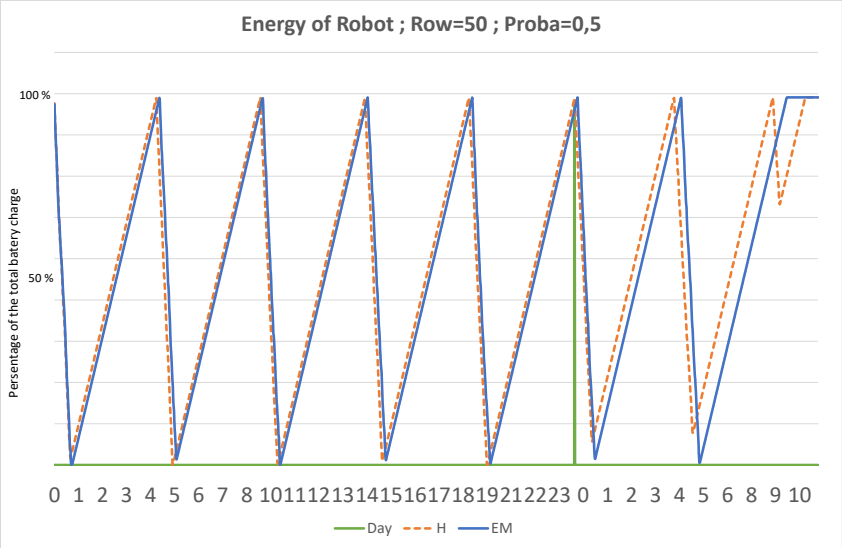

FIGURE 4.6 : Consommation d'énergie du robot pendant un traitement de la serre

Après validation des résultats obtenus par les experts du domaine (horticulteur partenaire du projet), HA, GA et EM ont été testés pour plusieurs tailles de serres avec différentes probabilités d'apparition de la maladie. Les trois méthodes sont comparées pour chaque scénario. Le tableau 4.1 résume les résultats de 810 simulations pour 9 scénarios de serre différents, où 30 simulations sont effectuées pour chacun. Nous choisissons empiriquement de réaliser 30 expériences afin de pouvoir étudier la signification stochastique des résultats. La moyenne et l'écart-type de la solution obtenue par les algorithmes GA et HA par rapport à la solution optimale obtenue par l'EM sont présentés dans les deuxième et troisième colonnes, respectivement. La colonne '# Non Convergence' représente le nombre de simulations, sur 30, dans lesquelles l'EM n'a pas convergé après un certain temps. Nous considérons qu'il n'y a pas convergence si le temps CPU [3] dépasse 8 heures sans trouver la solution optimale. En fait, en raison de la NP-complétude du problème, la méthode exacte peut ne pas converger rapidement avec des instances relativement grandes ($R = 75$ & $P = 1$; $R = 100$ & $P = 0,75$; $R = 100$ & $P = 1$). Pour les deux algorithmes donnant des solutions approximatives, les écarts sont proches de zéro, ce qui signifie que les résultats obtenus ne sont pas très éloignés des solutions optimales. La comparaison des écarts montre que le GA donne de meilleures solutions dans trois cas (présentés en rouge dans la troisième colonne). La faible valeur de l'écart-type (SD) à chaque scénario démontre la stabilité des résultats obtenus pour chaque méthode.

3. Un processeur ou unité centrale de traitement (en anglais, central processing unit (CPU))

TABLE 4.1 : Moyenne GAP et écart type pour les trois algorithmes pour différentes valeurs de R et P (le rouge → GA fonctionne mieux que HA)

$(R; P)$	Heuristique $\overline{GAP}/\sigma_{GAP}$	Algorithme génétique $\overline{GAP}/\sigma_{GAP}$	Méthode exacte # Non Convergence
$(50; 0,5)$	0,056/0,062	0,044/0,061	0
$(50; 0,75)$	0,022/0,038	0,022/0,039	0
$(50; 1)$	0,016/0,22	0,016/0,21	1
$(75; 0,5)$	0,054/0,045	0,054/0,042	0
$(75; 0,75)$	0,049/0,021	0,049/0,025	0
$(75; 1)$	0,021/0,022	0,021/0,021	14
$(100; 0,5)$	0,041/0,033	0,037/0,034	2
$(100; 0,75)$	0,051/0,016	0,038/0,015	13
$(100; 1)$	0,023/0,015	0,023/0,014	16

Le tableau 4.2 présente la moyenne et l'écart type du temps CPU pour chaque algorithme utilisé. Les résultats montrent que le temps CPU augmente avec la taille de l'instance, ainsi que l'écart type. Par exemple, dans le cas d'une grande instance ($R = 100$ & $P = 1$), le temps CPU moyen est de 10426 secondes (2 heures, 55 minutes et 24 secondes) pour la méthode EM, 11,075 secondes pour le GA et 0,084 seconde pour l'heuristique. Pour la plus petite instance (R=50, P=0.5), ces temps moyens sont respectivement 7,75 secondes, 1,448 seconde et 0,016 seconde. Pour toutes les instances testées, il est clair que HA est plus rapide que GA, qui est plus rapide que EM. Les valeurs de l'écart type démontrent que les méthodes sont stables et que les temps CPU enregistrés varient dans une petite plage. Il est à noter que ce temps peut être influencé par d'autres programmes exécutés en même temps par l'ordinateur, comme un antivirus ou d'autres services cachés du système d'exploitation.

TABLE 4.2 : Temps CPU moyen et écart type pour les trois algorithmes pour différentes valeurs de R et P

$(R; P)$	Heuristique $\overline{TCPU}/\sigma_{TCPU}$ [s]	Algorithme génétique $\overline{TCPU}/\sigma_{TCPU}$ [s]	Méthode exacte $\overline{TCPU}/\sigma_{TCPU}$ [s]
$(50; 0,5)$	0,016/0,015	1,448/0,09	7,75/2,13
$(50; 0,75)$	0,010/0,014	3,73/0,19	18/7,1
$(50; 1)$	0,016/0,009	6,153/0,89	73/49
$(75; 0,5)$	0,046/0,097	3,172/0,75	30/64,7
$(75; 0,75)$	0,029/0,004	6,563/0,39	217/94
$(75; 1)$	0,038/0,013	9,994/1,9	1677/884
$(100; 0,5)$	0,049/0,006	5,253/0,74	82/119
$(100; 0,75)$	0,058/0,016	11,751/2,27	431,286/21334
$(100; 1)$	0,084/0,034	11,075/2,9	10426/81203

Pour comprendre l'évolution de la maladie pendant le traitement, la figure 4.7 montre l'évolution du niveau moyen de la maladie dans la serre en utilisant chaque algorithme dans le cas d'une grande instance ($R = 100$ & $P = 0.5$). Le traitement total de la serre prend plus de deux jours pour tous les algorithmes. L'utilisation d'EM termine le premier en 14 missions (courbe bleue), tandis que GA

termine le traitement en 15 missions (courbe en pointillés rouges) et HA en 16 missions (courbe en tirets orange). Chaque ligne verticale verte indique le début d'un jour calendaire. Les périodes de 4 heures où le niveau de la maladie est constant correspondent aux cycles de charge. Les périodes de temps où le niveau de la maladie diminue correspondent aux cycles de traitement. Le taux de diminution du niveau moyen de la maladie est faible dans les premières missions (missions 1 à 4) car le robot traite de nombreuses rangées d'un seul côté, alors qu'une rangée est considérée comme traitée uniquement lorsque le traitement est effectué des deux côtés.

Pour résumer cette partie des expérimentations, nous pouvons conclure que les algorithmes HA et GA proposés présentent des performances intéressantes en termes de temps de calcul et de qualité de solution (GAP). De plus, GA a l'avantage d'améliorer la qualité de la solution par rapport à HA, mais il consomme beaucoup plus de temps CPU.

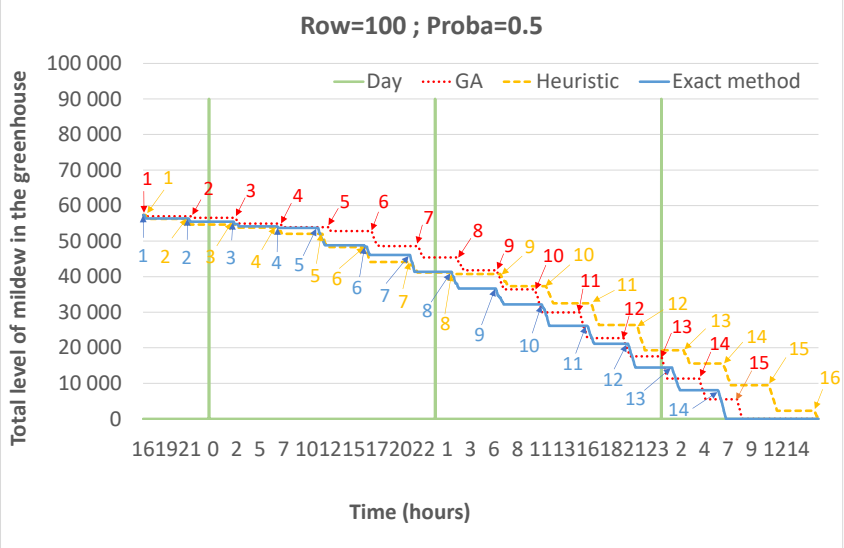

FIGURE 4.7 : Moyenne du niveau global mildiou dans la serre dans un cas statique avec 3 algorithmes (EM, GA, Heuristique)

Cas semi-dynamique

En pratique, comme expliqué dans le chapitre 3, la maladie évolue de manière continue dans le temps. Mais pour des raisons de simplification, nous l'avons considérée comme un processus discret qui n'évolue qu'une fois toutes les 24 heures.
Afin de tester les performances des trois algorithmes proposés (HA, GA et EM), nous considérons une des instances testées précédemment dans le cas de l'environnement statique ($R = 100$ & $P = 0,5$). Cette instance est sélectionnée, car c'est la plus grande instance que la méthode exacte EM peut résoudre en un temps raisonnable. Les résultats sont rapportés sur la figure 4.8, qui montre l'évolution du niveau total de maladie dans la serre, pour les trois algorithmes, jusqu'à ce qu'elle soit totalement traitée. Comme on peut le voir, le niveau de maladie est mis à jour (augmente) à la fin de chaque journée (ligne verte).

Notons que l'algorithme HA est exécuté au début de chaque mission, tandis que le GA et l'EM sont exécutés à la fin de toutes les missions programmées pour un problème de Bin-Packing équivalent.

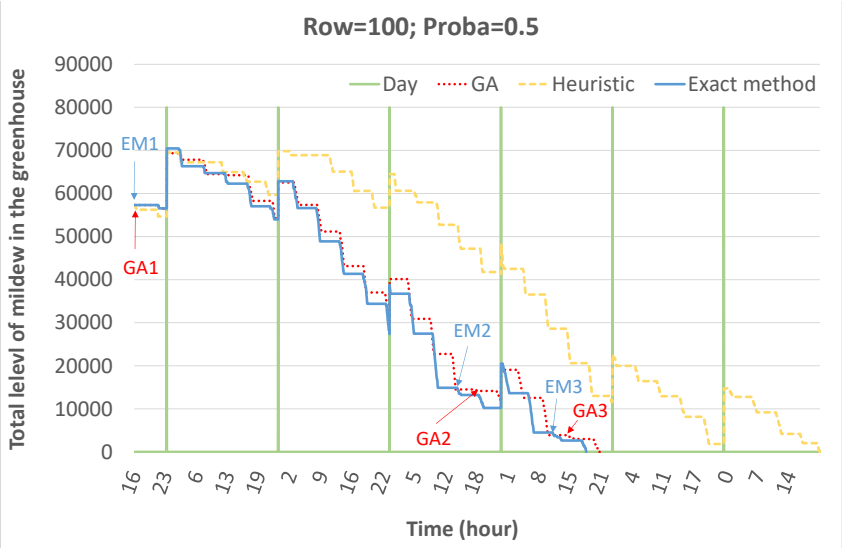

FIGURE 4.8 : Moyenne de niveau de mildiou dans la serre dans un cas semi-dynamique avec 3 algorithmes (EM, GA, Heuristique)

Parce que HA génère des tâches d'ordonnancement d'une seule mission, le simulateur attend la fin de la mission en cours pour mettre à jour le niveau de maladie et se lance pour définir la mission suivante en un temps de calcul négligeable. Concernant le GA et l'EM, comme les deux méthodes génèrent un ensemble de missions, elles sont lancées à la fin de toutes les missions planifiées. Dans ce cas, l'unique façon de comparer HA avec les deux autres méthodes est de l'exécuter jusqu'à traiter toute la serre. Les moments d'exécution de GA et de EM sont mentionnés dans la figure 4.8 par GA_i et EM_i respectivement, où $i \in \{1, 2, 3\}$ correspondent au nombre d'exécutions de l'algorithme.

Le temps de traitement total de la serre utilisant l'EM et le GA est d'environ 4 jours (3 jours dans le cas statique), alors qu'il est d'environ 6 jours pour l'HA. GA augmente le temps de traitement total de 4 heures (une seule mission supplémentaire) par rapport à EM (cf figure 4.8), ce qui représente un résultat intéressant lorsqu'on regarde son faible temps de calcul (5 secondes pour GA et 82 secondes pour EM). Ce temps augmente fortement avec la taille du problème, (Table 4.2). De plus, la solution donnée par GA peut être obtenue en temps quasi-réel.

En conclusion, l'efficacité de l'HA diminue dans le cas de l'environnement semi-dynamique, tandis que le GA présente toujours des résultats très intéressants, proches de la solution optimale fournie par l'EM. Par contre, il se peut que le robot tombe en panne d'énergie pendant le traitement si la maladie évolue rapidement dans les rangées qui seront traitées à la fin d'une mission. Ceci est dû au fait que le robot adapte sa vitesse au niveau de la maladie à traiter au fur et à mesure qu'il passe devant la plante, ce qui augmente sa consommation d'énergie pour la même mission. Cela conduit à une défaillance de la batterie (fin de charge pendant le traitement).

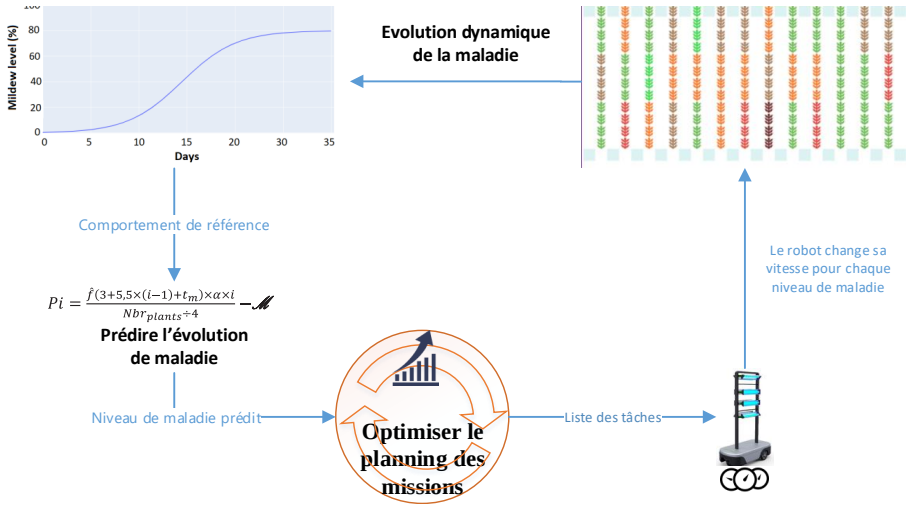

FIGURE 4.9 : Schéma du traitement préventif prédictif

4.4 Traitement préventif prédictif

4.4.1 Fonctionnement

Après avoir étudié le premier type de traitement, nous avons obtenu un retour positif du couplage de la simulation et de l'optimisation. Ceci démontre l'efficacité de l'approche. Malgré les premiers résultats prometteurs, plusieurs limites ont été observées sur cette approche comme l'aspect semi-dynamique et le manque de réactivité et de précision. Afin d'améliorer ces aspects, un comportement plus réaliste de la maladie a été ajouté dans le simulateur (cf. chapitre 2). Cela rend les scénarios de traitement entièrement dynamiques. Le traitement préventif prédictif est similaire au traitement conditionnel, sauf que dans le prédictif, la maladie évolue en temps réel en suivant un modèle comportemental connu.

La figure 4.9 montre un schéma explicatif du fonctionnement du traitement préventif prédictif. En haut à gauche, il y a le comportement d'évolution de la maladie qui est dynamique. Ensuite, avec la fonction de prédiction, un algorithme d'optimisation sera exécuté dans le cas dynamique. Ce dernier planifie des listes de tâches pour le robot, qui traite la serre. Le robot effectue le traitement de la serre comme dans le traitement conditionnel. Pendant le traitement par le robot, la maladie continue d'évoluer et de se propager dans la serre.

Pour gérer l'évolution dynamique de la maladie, l'algorithme génétique dynamique DGA a été développé (cf. chapitre 3). L'intégration de la fonction de prédiction à l'intérieur des individus permet à l'algorithme de planifier les tâches évolutives du système. Après le développement de cet algorithme, ses paramètres sont calibrés grâce à la simulation. La performance de cet algorithme a été testée à travers plusieurs scénarios de traitement prédictif en simulant et en comparant les résultats de l'algorithme génétique et de l'algorithme génétique dynamique.

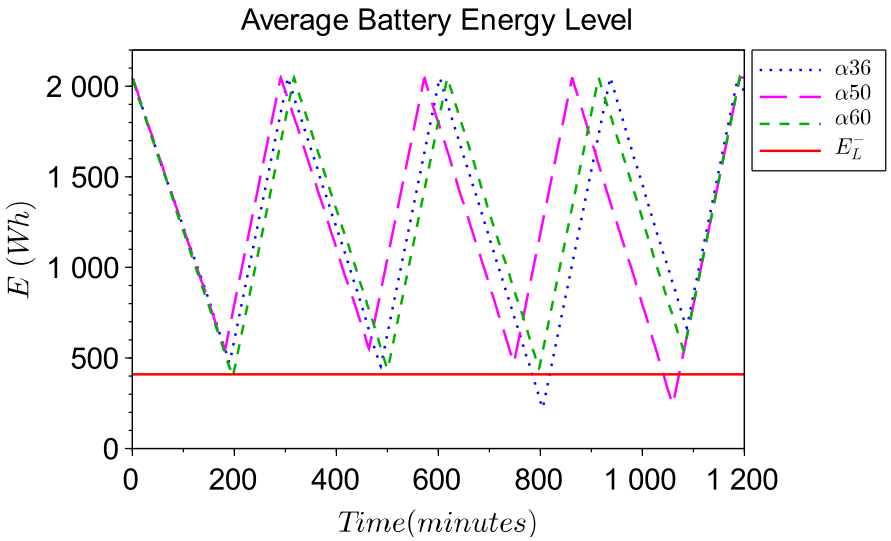

FIGURE 4.10 : Consommation d'énergie du robot pour calibrer le paramètre α de DGA

4.4.2 Expérimentation

Dans ce type de traitement un nouveau type de robot est pris en compte, il a une autonomie de 2 heures et 30 pour une charge d'environ 3 heures. Afin d'augmenter la durée de vie de la batterie du robot, le taux de consommation recommandé est de 80 % [Mazar et al., 2020b]. Dans un premier temps, nous allons montrer des résultats d'état de batterie pour plusieurs tests avec l'algorithme génétique dynamique (DGA), afin de calibrer ses paramètres. Ensuite, nous allons comparer le DGA et le GA dans le cas dynamique pour voir le quel est plus performant.
Le plan d'expérimentation est défini en fonction des paramètres suivants :

- Évolution de maladie : dynamique
- Nombre de robots : 1
- Autonomie du robot : 3 heures de traitement pour 2 heures et 30 min de chargement
- Vitesse du robot : le robot adapte sa vitesse pour chaque niveau de maladie
- Taille des serres : 100 rangées de 100 m chacune
- Pourcentage de plantes malades au début de simulation : 50 %.
- Algorithme d'optimisation : utilisation de l'algorithme GA ou de l'algorithme DGA
- Nombre de scénarios : 2
- 100 plantes par rangée
- Horizon de simulation : jusqu'au traitement total de la serre. (1 journée)

4.4.3 Résultats

Dans cette section, nous allons discuter les résultats obtenus par chaque algorithme (GA et DGA). La maladie augmente continuellement dans toutes les simulations suivantes. La figure 4.10 représente l'évolution du niveau de batterie du robot en fonction du temps. Trois courbes avec différentes valeurs du paramètre α $(36, 50$ et $60)$ sont comparées. Les périodes à valeurs décroissantes dans les courbes correspondent au temps de traitement, et chaque période à valeurs croissantes est relative au temps de

charge de la batterie. Afin d'augmenter la durée de vie de la batterie, la consommation de la batterie est limitée à 80 % de la capacité totale, comme le recommandent les experts [Mei et al., 2005]. Dans la figure 4.10, la ligne E_L^- représente 20% de la charge de la batterie. Le paramètre choisi est $\alpha = 60$ (DGA60) pour les prochaines simulations car sa courbe respecte le niveau d'énergie minimum et la contrainte de capacité.

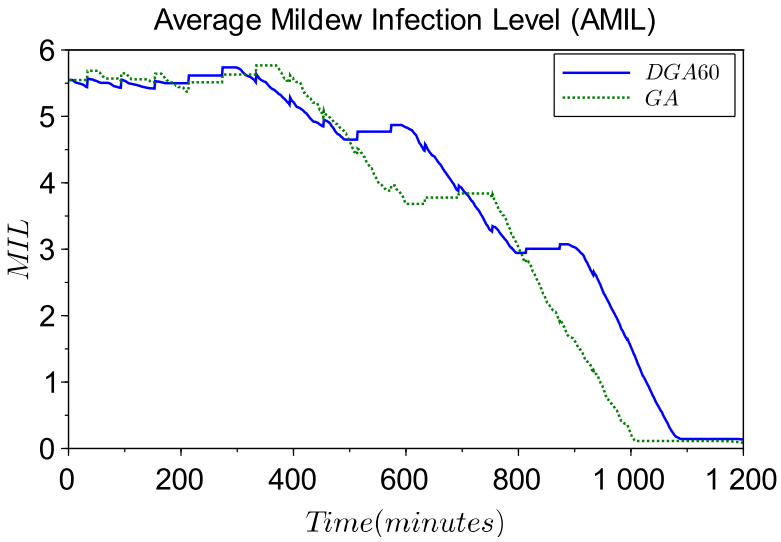

FIGURE 4.11 : Résultats graphique de la moyenne du niveau de mildiou dans une serre avec le GA et le DGA

La figure 4.11 montre l'évolution du niveau de la maladie au fil du temps. Il y a deux courbes sur la figure : la courbe bleue pleine représente l'évolution du niveau de mildiou lorsque la planification du traitement prédictif est basée sur le DGA60 et la courbe verte pointillée lorsque la planification est basée sur le GA. On peut clairement observer que l'utilisation du GA permet au robot de terminer le traitement du mildiou en 1000 minutes, alors qu'il faut 1100 minutes en utilisant le DGA. Cependant, on remarque que l'utilisation du GA est risquée et ne permet pas d'utiliser le robot de manière autonome. En effet, lors de l'exécution du scénario utilisant GA, le producteur redémarre manuellement le robot plusieurs fois car il n'a pas assez d'énergie dans la batterie pour retourner à la station de charge. En fait, une consommation d'énergie supplémentaire peut se produire lorsque le niveau réel de maladie est plus élevé que prévu. Nous notons également que, dans les deux cas, le niveau de maladie augmente sur certains laps de temps qui correspondent aux périodes de charge du robot.

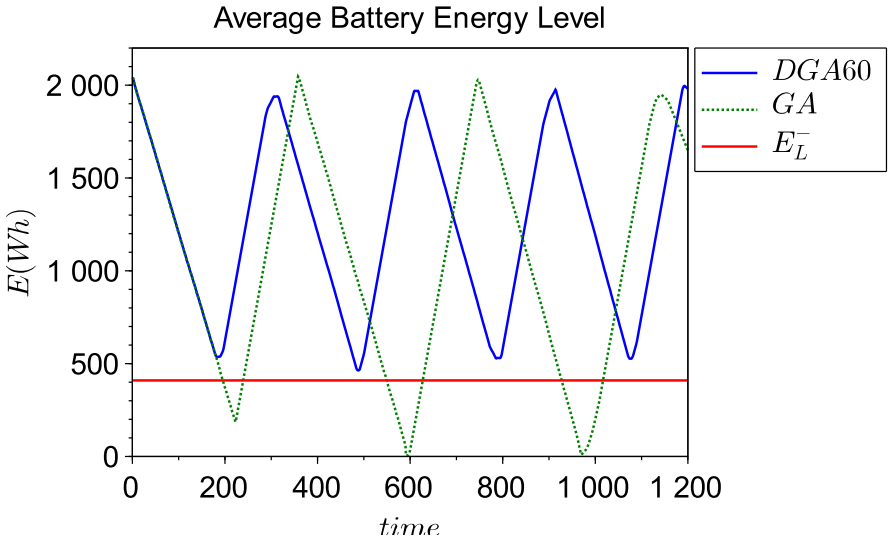

FIGURE 4.12 : Résultats graphique de la consommation d'énergie du robot avec le GA et le DGA

Sur la figure 4.12, sont dessinés les deux courbes relatives à DGA60 et GA qui tracent l'évolution du niveau d'énergie de la batterie en fonction du temps. Les deux courbes correspondent à la moyenne de 30 simulations pour chaque algorithme. Le scénario de simulation avec une planification du traitement prédictif basée sur le GA ne respecte pas la contrainte de capacité de la batterie, car il y a plusieurs périodes où le robot utilise plus de 80% de sa capacité de batterie. Comme le montre la figure 4.12, ce niveau de consommation atteint plusieurs fois 100% de capacité de batterie, ce qui nécessite l'intervention de l'horticulteur pour amener le robot à sa station de charge.

Le temps moyen d'une simulation avec GA est de 16 minutes et 41 secondes, et celui de DGA60 est de 21 minutes et 36 secondes. Cette augmentation du temps de calcul est due au calcul supplémentaire de la prédiction du niveau de mildiou.

En conclusion, on peut dire que même si l'utilisation de GA permet d'avoir un nombre minimal de missions de traitements et de temps de calcul, cela reste une option risquée car cet algorithme ne peut assurer la totale autonomie du robot et nécessite l'intervention d'horticulteur en plusieurs fois. D'autre part, l'utilisation du DGA60 s'est avérée plus efficace car elle respecte la contrainte de l'utilisation de seulement 80% de la capacité totale de la batterie, ce qui permet d'assurer une autonomie totale du traitement robotique. De plus, l'utilisation du DGA60 donne des scénarios et des résultats plus réalistes, ce qui permet d'avancer un pas de plus vers une éventuelle mise en œuvre réelle.

4.5 Traitement préventif calendaire (systématique)

4.5.1 Fonctionnement

L'idée d'ajouter le traitement calendaire dans le simulateur vient du besoin, exprimé par les partenaires du projet UV-Robot, de faire des traitements simples et rapides en alternance avec les traitements chimiques. Le but du traitement calendaire est de réduire les doses de traitement chimique dans le cas où le traitement UV ne fonctionne pas. L'objectif de la simulation de ce type de traitement

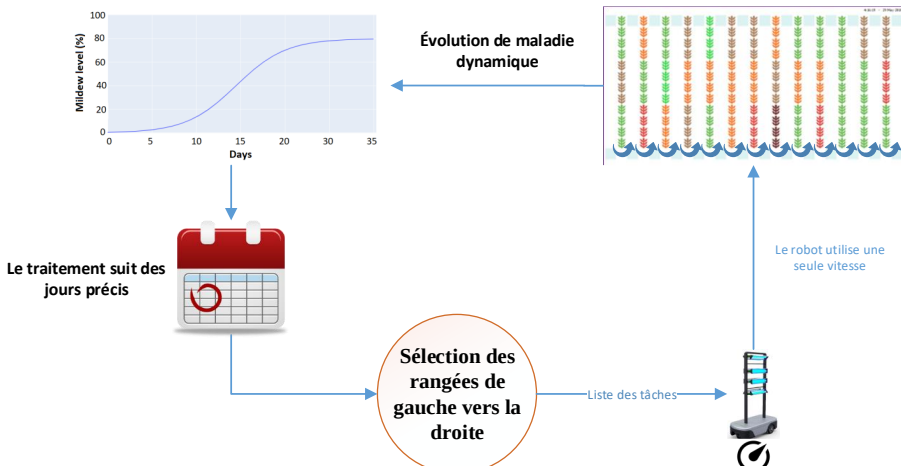

FIGURE 4.13 : Schéma du traitement préventif calendaire

est de rechercher le meilleur scénario de traitement préventif calendaire en définissant la meilleure fréquence et le meilleur dosage de traitement (défini par la vitesse du robot).

Le déroulement du traitement calendaire se fait par fréquence, avec une vitesse constante, de la première rangée jusqu'à la dernière rangée. Le robot commence le traitement par les premières rangées jusqu'à ce qu'il épuise son énergie, puis il retourne à la station de charge pour se charger et continuer vers les rangées qui suivent. Pendant ce type de traitement, certaines plantes ne reçoivent pas assez de doses UV-C pour éradiquer la maladie. En effet, si la plante est infectée à un niveau nécessitant une dose supérieure, le traitement préventif ne fait que diminuer son niveau d'infection.

La figure 4.13, présente un schéma qui illustre la stratégie du traitement calendaire. En haut à gauche se trouve le comportement de la maladie qui évolue en permanence. Ensuite, en bas, il y a un calendrier qui est utilisé pour choisir les fréquences ou les jours de traitement. Ensuite, un algorithme qui choisit les rangées à traiter, et qui envoie la liste des tâches aux robots. Ces derniers, définissent leur vitesse de traitement et lancent un cycle de traitement rangée par rangée.

Nous avons développé un algorithme simple pour ce type traitement. Il sert à définir les missions de traitement aux robots, tout en respectant la contrainte de la capacité d'énergie. Cet algorithme divise aussi les rangées de traitement entre les robots, pour éviter le double traitement de quelques zones. La sauvegarde de l'information de la dernière rangée traitée se fait dans cet algorithme aussi.

4.5.2 Expérimentation

En ce qui concerne le traitement du calendaire, des scénarios sont proposés en fonction de la vitesse, de la fréquence de traitement et du nombre de robots. Dans ce modèle, nous considérons une nouvelle contrainte qui impose que le traitement se fasse la nuit. En effet, selon les premiers tests effectués par les horticulteurs partenaires du projet avec le robot UV, la maladie diminue considérablement si les plantes prennent une dose de UV-C dans l'obscurité. Pour tenir compte de cette contrainte, une plage horaire a été ajouté sur le simulateur pour gérer les traitements de nuit. En effet, le traitement UV-C est plus efficace dans l'obscurité (la nuit), car après l'exposition des plantes à la lumière UV-C, il est

nécessaire de les faire suivre de périodes d'obscurité [Janisiewicz et al., 2016]. Nous avons donc décidé que le traitement se fait uniquement la nuit.
Le plan d'expérimentation est défini en fonction des paramètres suivants :

- Évolution de maladie : dynamique
- Nombre de robots : 1 ou 2
- Autonomie du robot : 3 heures de traitement pour 2 heures et 30 min de chargement
- Vitesse du robot : défini au début de chaque simulation et prend les valeurs suivantes selon le scénario (0,1 ; 0,15 ; 0,25 ; 0,5 et 1) [m/s]
- Taille des serres : 100 rangées de 100 m chacune
- Pourcentage de plantes malades au début de simulation : 0%.
- Scénario : chaque scénario est la combinaison de la fréquence de traitement PPi (un traitement chaque i jours tel que $i \in [1, 2, 3, 4, 5]$), la vitesse du robot pendant le traitement (0,1 ; 0,15 ; 0,25 ; 0,5 et 1) [m/s] et le nombre de robots (1 ou 2)
- Nombre de scénarios : 50
- Densité des plantations : 100 plantes par rangée
- Horizon de simulation : 1 mois

4.5.3 Résultats

Les résultats du traitement calendaire seront présentés sous forme de graphiques pour analyser l'évolution temporelle du niveau de la maladie et avec des boîtes à moustaches pour l'analyse statistique de l'effet de certains paramètres. Les résultats des différents scénarios avec des vitesses différentes sont similaires en ce qui concerne le schéma général du comportement de la maladie. Les courbes augmentent avec la progression de la maladie, et diminuent au cours des périodes de traitement robotique. Cependant, il existe des différences sur les valeurs maximales atteintes dans chaque scénario après un mois de simulation. Pour chaque figure, il y a dix courbes des niveaux moyens de la maladie, cinq pour le cas à un robot et cinq pour le cas à deux robots. Pour chaque cas (1 ou 2 robots) il y a cinq fréquences de traitement (PP1, PP2 PP3, PP4 et PP5), par exemple pour PP3 il y a un traitement de toute la serre tous les trois jours. Nous discuterons brièvement les résultats de chaque graphique et conclurons sur tous les résultats à la fin de la section.

La figure 4.14 montre dix courbes de l'évolution du niveau de la maladie, liées aux résultats des simulations du traitement calendaire avec la vitesse de 1 m/s sur un horizon d'un mois. Chaque courbe représente la variation du niveau moyen de mildiou par plante dans la serre en fonction du temps. Il y a cinq résultats (ligne pleine) pour les scénarios à un seul robot avec différentes fréquences de traitement, et cinq autres (ligne pointillée) pour les scénarios à deux robots avec différentes fréquences de traitement. On remarque que le comportement de la maladie avec un traitement préventif calendaire à une fréquence allant de 1 fois/jour (PP1) à 1 fois/ 4 jours (PP4) est presque le même pour les scénarios utilisant un ou deux robots. Cependant, pour le scénario avec une fréquence PP5 et un robot, la maladie dépasse la moyenne de 7 à la fin du mois, alors qu'elle est inférieure à 6,5 pour le scénario avec une fréquence PP5 et deux robots.

La figure 4.15 montre la représentation en boîtes à moustaches des résultats de la figure 4.14. Nous remarquons dans les boîtes à moustaches avec les fréquences PP1 et PP2, pour les deux scénarios (1 et 2 robots), que le niveau maximal de la moyenne du niveau de maladie ne dépasse pas 1,5 après un mois de simulation avec ce type de traitement. Ces résultats montrent qu'avec les fréquences PP1 et PP2

on maîtrise mieux l'évolution de maladie dans la serre. En résumé, avec une dose d'UV-C équivalente à la vitesse de traitement de 1m/s, l'utilisation d'un seul robot à une fréquence d'un traitement tous les 2 jours semble suffisant pour maîtriser la maladie de mildiou dans la serre.

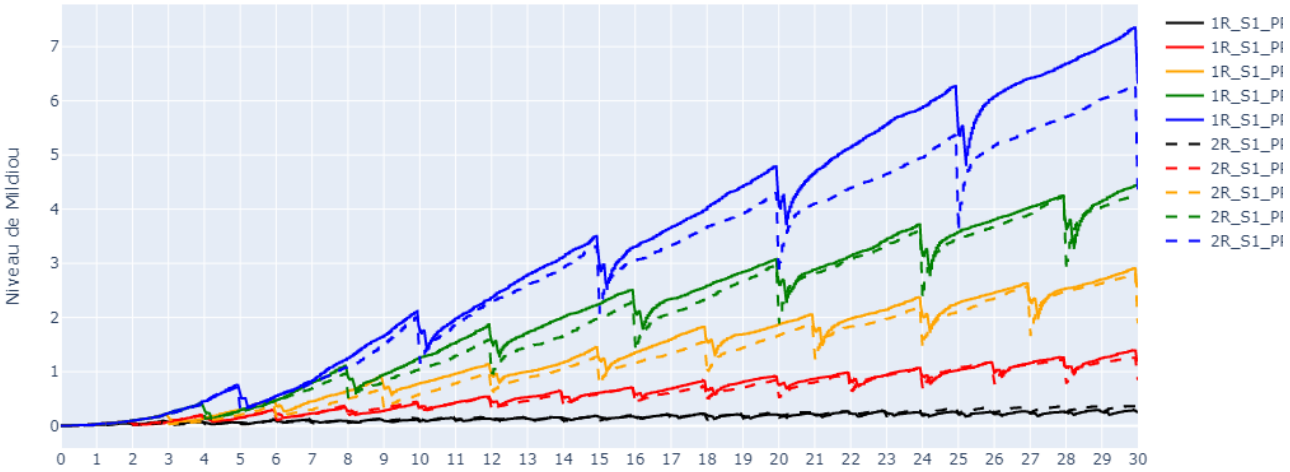

FIGURE 4.14 : Niveau moyen de la maladie dans la serre en fonction du temps (vitesse = 1 m/s)

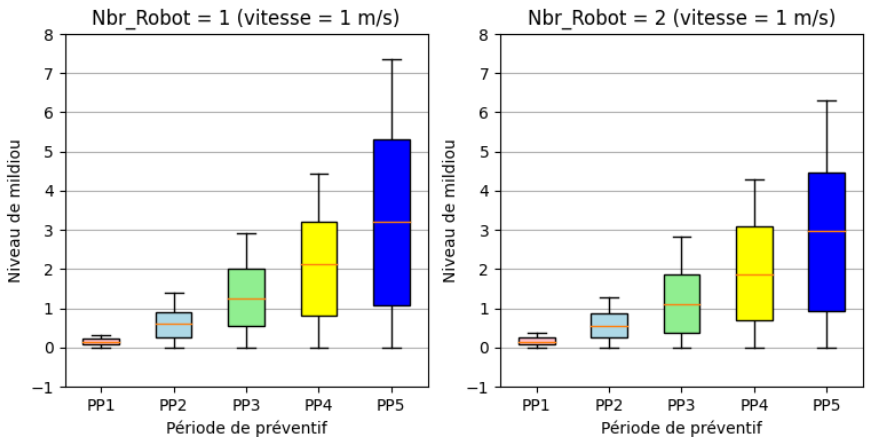

FIGURE 4.15 : Boîte à moustaches pour le traitement calendaire avec 1 et 2 robots (vitesse = 1 m/s)

La figure 4.16 présente le même nombre de graphiques que le cas précédent pour les simulations du traitement calendaire avec la vitesse de 0,5 m/s sur un horizon d'un mois. Nous avons adopté les mêmes notations que précédemment. Nous remarquons que le comportement de la maladie pour toutes les fréquences est différent entre les deux cas (1 ou 2 robots). L'utilisation de deux robots donne presque les mêmes résultats en utilisant un robot avec une fréquence de plus, nous voyons que les courbes de (2R_S0.5_PP5, 2R_S0.5_PP4, 2R_S0.5_PP3 et 2R_S0.5_PP2) devraient avoir les mêmes valeurs maximales que (1R_S0.5_PP4, 2R_S0.5_PP3, 2R_S0.5_PP2 et 2R_S0.5_PP1) respectivement. Ainsi, avec cette vitesse, les robots gèrent mieux le comportement de la maladie que le traitement avec

la vitesse de 1 m/s après un mois de simulation.

La figure 4.17 montre des boîtes à moustaches qui représentent différemment les mêmes résultats de la figure 4.16. Nous remarquons dans les boîtes à moustaches que les niveaux de maladie diminuent avec la vitesse de 0,5 m/s par rapport à la vitesse de 1 m/s (cf. figure 4.15). Le scénario le plus défavorable avec la vitesse de 0,5 m/s est dans 1R_S0.5_PP5 où le niveau maximal de la maladie est inférieur à 4. Le meilleur scénario est dans 2R_S0.5_PP1 où le niveau de la maladie est presque nul pendant un mois, cependant ce scénario est le plus coûteux puisque le traitement est effectué chaque jour avec deux robots.

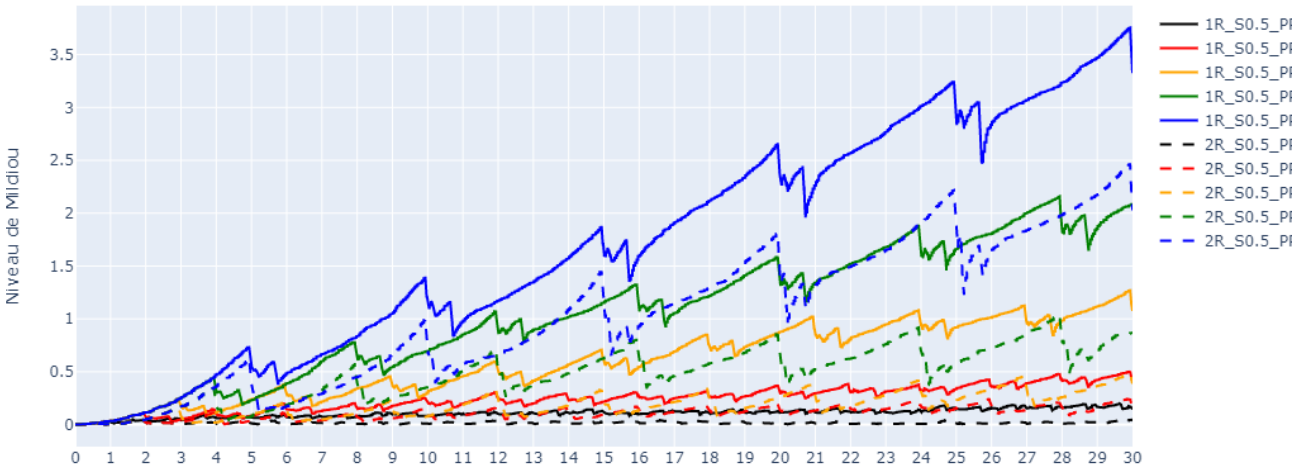

FIGURE 4.16 : Niveau moyen de la maladie dans la serre en fonction du temps (vitesse = 0,5 m/s)

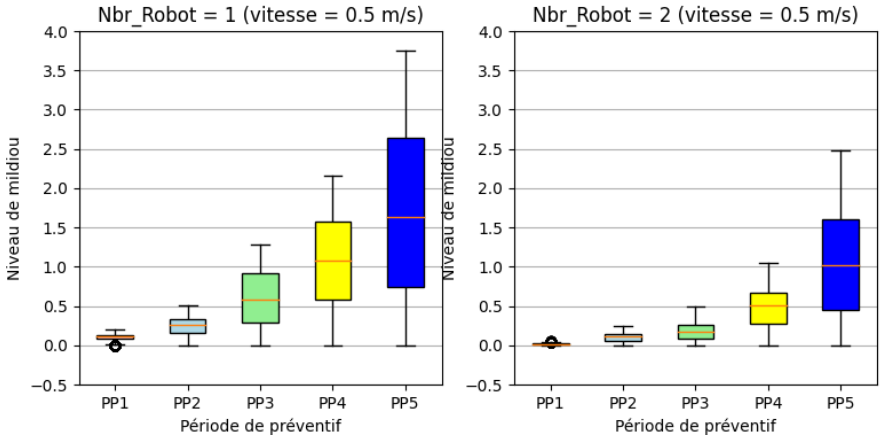

FIGURE 4.17 : Boîte à moustaches pour le traitement calendaire avec 1 et 2 robots (vitesse = 0,5 m/s)

La figure 4.18 présente le même nombre de graphiques que le cas précédent pour les simulations du traitement calendaire avec la vitesse de 0,25 m/s sur un horizon d'un mois. Nous avons adopté les mêmes notations que précédemment. On remarque sur ces graphiques que le niveau de la maladie

diminue plus rapidement pendant les périodes de traitement. Ceci est dû au fait que la dose d'UV-C est plus élevée avec cette vitesse par rapport aux deux précédentes. La différence dans l'évolution de la maladie entre les traitements avec un ou deux robots n'est pas aussi importante que le traitement avec la vitesse de 0,5 m/s.

La figure 4.19 montre des boîtes à moustaches qui représentent les graphes de la figure 4.18. Nous pouvons clairement voir que la valeur maximale pour tous les traitements avec la vitesse de 0,25 m/s est inférieure à 2,5. Donc la maladie moyenne dans la serre est inférieure à 3, ce qui correspond au premier niveau de la maladie du mildiou dans le système étudié. La maladie dans la serre est mieux contrôlée en utilisant une vitesse de 0,25 m/s pendant un traitement calendaire. On remarque également que les traitements de PP1 et PP2 avec un robot sont identiques, car avec cette vitesse le robot ne peut pas traiter toute la serre en une nuit. Comme dans PP1, un traitement est fait chaque jour, le robot traite une moitié de la serre le premier jour et continue l'autre moitié le jour suivant. Dans le cas du PP2, c'est la même chose, mais lorsque le robot termine la deuxième moitié le deuxième jour, les plantes traitées le premier jour doivent être traitées le troisième jour. Le robot effectuera donc un traitement par jour dans le cas du PP2.

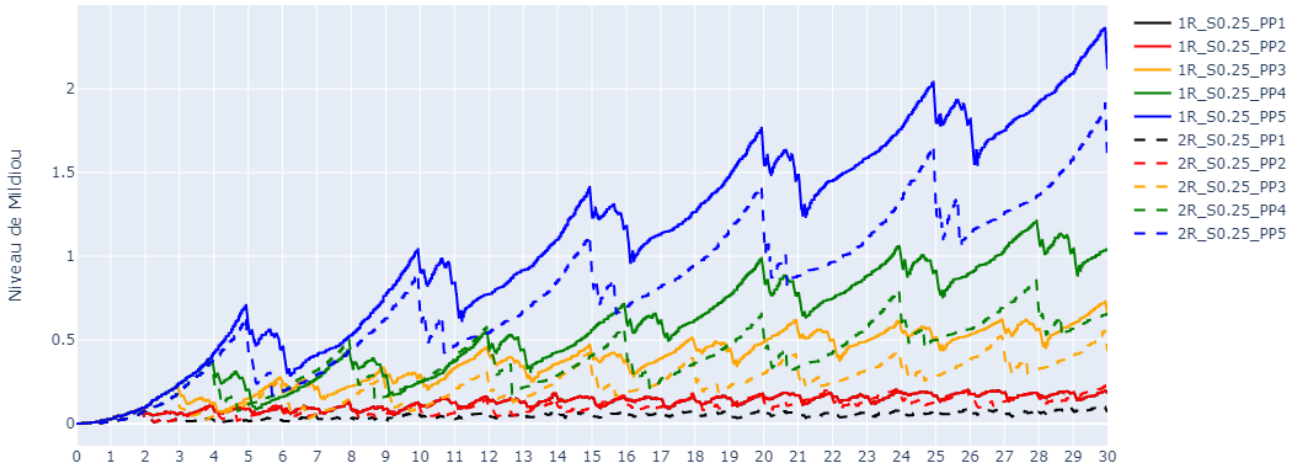

FIGURE 4.18 : Niveau moyen de la maladie dans la serre en fonction du temps (vitesse = 0,25 m/s)

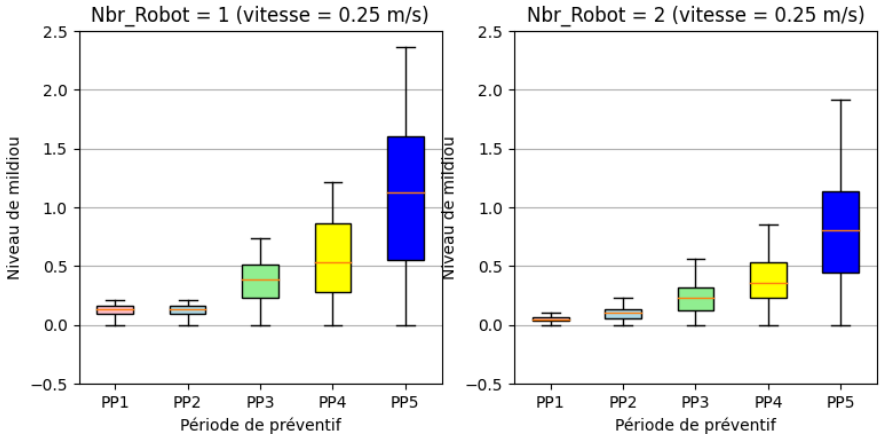

FIGURE 4.19 : Boîte à moustaches pour le traitement calendaire avec 1 et 2 robots (vitesse = 0,25 m/s)

La figure 4.20 présente le même nombre de graphiques que le cas précédent pour les simulations du traitement calendaire avec la vitesse de 0,15 m/s sur un horizon d'un mois. Nous avons adopté les mêmes notations que précédemment. Nous pouvons voir sur ces graphiques que chaque fois que la vitesse diminue, les traitements diminuent le niveau moyen de la maladie. Et les graphiques sont beaucoup plus proches les uns des autres malgré la différence entre la fréquence des traitements.

La figure 4.21 montre des boîtes à moustaches qui représentent les graphes de la figure 4.20. Dans ces boîtes à moustaches on voit que les valeurs maximales sont inférieures au niveau 1,6. La maladie est mieux contrôlée dans ce cas aussi puisque les plantes reçoivent plus de dose d'UV-C. La même remarque que pour le traitement avec un robot et une vitesse de 0,25 m/s, mais cette fois il y a PP1, PP2 et PP3 qui sont identiques. Car il faut trois jours à un robot pour traiter l'ensemble de la serre avec la vitesse 0,15 m/s.

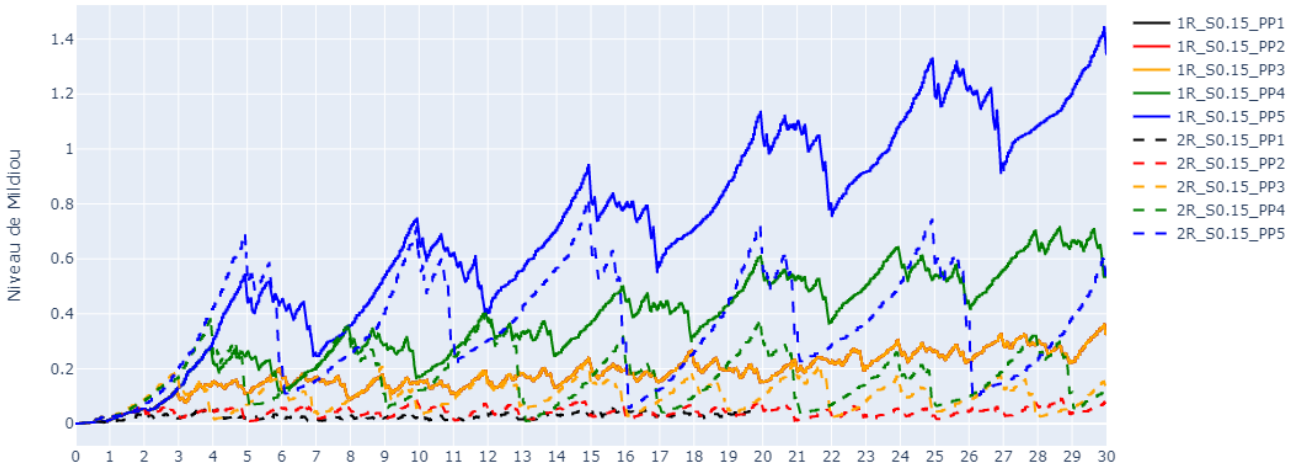

FIGURE 4.20 : Niveau moyen de la maladie dans la serre en fonction du temps (vitesse = 0,15 m/s)

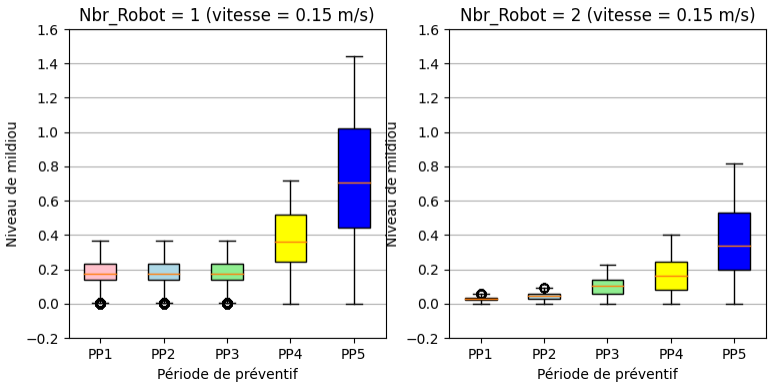

FIGURE 4.21 : Boîte à moustaches pour le traitement calendaire avec 1 et 2 robots (vitesse = 0,15 m/s)

La figure 4.22 présente le même nombre de graphiques que le cas précédent pour les simulations du traitement calendaire avec la vitesse de 0,1 m/s sur un horizon d'un mois. Nous avons adopté les mêmes notations que précédemment. Nous remarquons que les graphiques se chevauchent beaucoup plus dans les scénarios de traitement avec la vitesse de 0,1 m/s. Et pendant les cycles de traitement, le niveau moyen de la maladie diminue plus rapidement et se rapproche de 0. Parce qu'avec cette vitesse, les plantes reçoivent la dose la plus élevée, cette dose peut éradiquer la maladie à partir du niveau de contamination le plus élevé qui est de 30 dans le système UV-Robot. La particularité de ces résultats est que les courbes de PP4 et PP5 avec un robot sont meilleures que les courbes de PP4 et PP5 avec deux robots (la valeur maximale avec un robot est inférieure à la valeur maximale avec deux robots). Cela est dû au temps nécessaire pour traiter l'ensemble de la serre avec une vitesse de 0,1 m/s, car en utilisant cette vitesse, le robot traite l'ensemble de la serre après trois jours. Cependant, lorsque le robot termine le traitement du 3ème jour dans 1R_S0.1_PP4 (ou 1R_S0.1_PP5), il n'attend qu'un jour (ou deux jours) pour commencer un nouveau cycle de traitement, car les plantes traitées le premier jour ont besoin d'un traitement tous les quatre jours (ou cinq jours).

La figure 4.23 montre des boîtes à moustaches qui représentent les graphes de la figure 4.22. On remarque que la valeur maximale de la maladie avec la vitesse 0,1 m/s est de 0,8. La même remarque pour les scénarios PP1, PP2 et PP3 avec les scénarios PP1, PP2 et PP3 de la vitesse 0,15 m/s dans le cas d'un robot. Et aussi pour le cas de deux robots, il y a les traitements de PP1 et PP2 qui sont identiques, car il faut deux jours aux deux robots pour traiter toute la serre avec la vitesse 0,1 m/s.

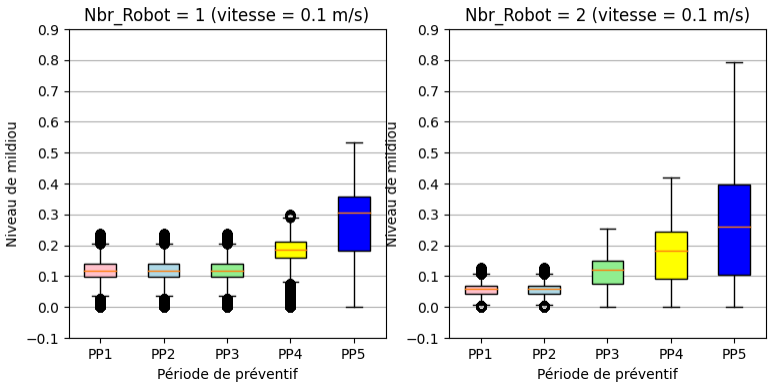

FIGURE 4.23 : Boîte à moustaches pour le traitement calendaire avec 1 et 2 robots (vitesse = 0,1 m/s)

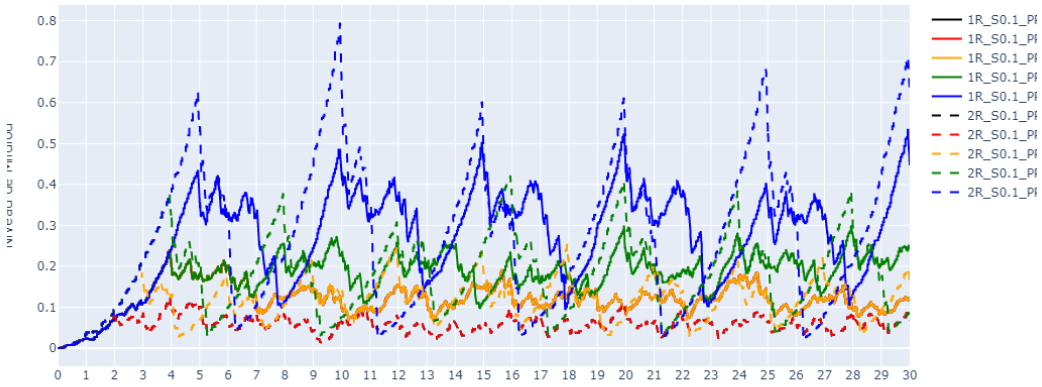

FIGURE 4.22 : Niveau moyen de la maladie dans la serre en fonction du temps (vitesse = 0,1 m/s)

Pour le traitement calendaire, nous pouvons choisir un scénario adapté à la situation de la serre. Tout d'abord le choix entre un scénario de traitement avec un ou deux robots est facile compte tenu du prix du robot qui est estimé à 30 000 euros. Donc, si nous prenons un des scénarios pour l'appliquer sur une serre, nous pouvons considérer d'autres critères, comme le type de plante, pour savoir quelle dose utiliser et si les fortes doses d'UV-C peuvent endommager les plantes. Il y a d'autres situations avec un robot pour une ou plusieurs serres, dans ce cas il est préférable de choisir un scénario qui contrôle l'évolution de la maladie avec une grande fréquence de traitement. Fondamentalement, dans le traitement calendaire, il existe plusieurs scénarios efficaces qui peuvent être appliqués dans différentes situations.

Nous pouvons remarquer sur toutes les courbes qu'un traitement ne remet pas le niveau de la maladie à 0, sauf dans certains cas où la vitesse de traitement est de 0,1 m/s. Ceci est dû au fait que lorsqu'un robot effectue un traitement calendaire avec une seule vitesse, il ne pourra pas éliminer complètement la maladie, surtout pour les niveaux où sa vitesse n'est pas suffisante. Par exemple, si le niveau de maladie d'une plante est de 30 et que le robot se déplace à une vitesse de 0,5 m/s, le traitement réduit le niveau de maladie à 20 ou 12 dans les meilleurs cas. Lorsque l'on fait la moyenne de ces phénomènes sur toutes les plantes de la serre, on se retrouve avec un résidu de maladie important

après un passage du robot. Ceci explique également le fait que lorsque le dosage du traitement est à son niveau le plus élevé (vitesse = 0,1m/s), le robot peut éliminer complètement la maladie. Il faut rappeler que ce scénario (vitesse = 0,1m/s) peut être dangereux pour les plantes s'il est prouvé que le surdosage dégrade leur état.

4.6 Conclusion

Dans ce chapitre nous avons commencé par un état de l'art sur les types de maintenance et les types de traitement en faisant une analogie entre les tâches de maintenance dans l'industrie et les tâches de traitement de maladies des plantes dans l'agriculture. Puis, nous avons expliqué les différents types de traitement qu'on a développés dans le simulateur.

Chaque traitement a un environnement spécifique qui donne une possibilité de simuler le système réel dans les serres. Cependant, le traitement conditionnel n'est pas faisable en réalité, car on ne peut pas mesurer l'évolution de maladie en temps réel. Par contre, les traitements prédictif et calendaire sont très pratique dans notre système et même dans d'autres domaines d'application tels que la désinfection COVID-19.

Les types de traitement facilitent le choix d'un scénario et son application dans un environnement réel. Les résultats du traitement conditionnel nous ont aidé à trouver le meilleur algorithme développé et ont montré certaines lacunes de l'algorithme que nous avons surmontées dans les étapes suivantes de notre travail. Les résultats de l'algorithme génétique dynamique en traitement prédictif ont montré sa performance et son respect de la contrainte liée à l'utilisation de la batterie. Les résultats du traitement calendaire donnent une idée du choix de la stratégie de traitement en fonction de la situation du système réel.

Il est possible de tester d'autres scénarios hybrides qui combinent plusieurs algorithmes d'optimisation ou d'exécuter des algorithmes d'optimisation au milieu d'un traitement calendaire. Les algorithmes d'optimisation peuvent être améliorés pour diminuer leur complexité ou changer le modèle de prédiction de l'algorithme génétique dynamique par un modèle Markovien par exemple.

Conclusion

Cette thèse représente une application réelle de la recherche opérationnelle dans des systèmes incertains et évolutifs. Le développement technologique en termes de collecte d'informations et de moyens de calcul permet d'exécuter des algorithmes complexes et de prendre des décisions précises et efficaces. Nous avons étudié un problème d'ordonnancement dynamique des tâches évolutives qui s'applique dans plusieurs domaines tels que la robotique, l'informatique, la maintenance et la production. La difficulté de ce problème réside dans le comportement incertain de l'évolution des tâches qui diminue l'efficacité du plan d'ordonnancement statique. L'étude a été menée sur une application dans le domaine agricole, où nous devons traiter un champignon sur des plantes dans des serres avec des robots autonomes. Pour être effectué efficacement, le traitement nécessite de définir un ordonnancement préalable des tâches pour le robot, sous les contraintes d'autonomie de la batterie du robot et d'évolution de la maladie qui suit un processus stochastique. Au cours de notre travail de thèse, nous avons participé à plusieurs réunions et discussions avec des professionnels et experts du secteur horticole, ce qui nous a beaucoup aider pour comprendre l'environnement du système étudié. Cependant, nous avons rencontré des difficultés concernant les données sur le comportement de la maladie, surtout dans l'horticulture. Les données utilisées sont basées sur une étude approfondie de l'état de l'art sur le comportement des maladies.

Dans le premier chapitre (cf. chapitre 1), nous avons rappelé le contexte de l'application et nous avons fixé l'objectif du besoin sur l'ordonnancement dynamique. Où, nous avons étudié un état de l'art sur le domaine agricole afin de s'adapter à l'environnement horticole. Puis une deuxième partie d'état de l'art sur le problème de l'ordonnancement dynamique et des outils d'aide à la décision. La méthode de couplage entre simulation et optimisation est une approche efficace pour les problèmes dynamiques, c'est pourquoi nous avons choisi cette approche.

Le deuxième chapitre (cf. chapitre 2) concerne la partie simulation, son état de l'art, sa modélisation et son développement. Cette partie est très importante puisqu'elle revient dans toutes les étapes de notre travail : conception, test et validation. Pour gérer le comportement incertain de la maladie, nous avons proposé une approche basée sur la modélisation markovienne pour caractériser le comportement des plantes en termes d'apparition et d'évolution de la maladie dans une serre. L'objectif est de reproduire une évolution en forme de sigmoïde du niveau global de la maladie. Les résultats des simulations sur le comportement du mildiou sont satisfaisants. L'étude du comportement est une question importante qui a enrichi les travaux sur le développement de traitements prédictifs et calendaires.

Dans le troisième chapitre (cf. chapitre 3), nous avons présenté les algorithmes d'optimisation qui sont intégrés dans l'un des agents de notre SMA. Chaque algorithme est développé pour tester différents types de traitement. Tout d'abord, nous avons mis en place un modèle mathématique, analogue au célèbre problème de bin-packing, et nous avons développé une heuristique et un algorithme génétique

pour sa résolution. Ensuite, nous avons comparé les performances de l'heuristique et de l'algorithme génétique dans un cas statique en calculant leurs GAPs par rapport à une solution exacte obtenue par un solveur. Puis, dans la deuxième partie, nous avons amélioré l'algorithme génétique en le rendant dynamique grâce à une fonction de prédiction à l'intérieur des chromosomes afin de respecter la contrainte de capacité de la batterie dans les scénarios dynamiques.

Le développement d'un simulateur assez complet, voire complexe, mais très paramétrable, nous a permis de jouer sur plusieurs paramètres pour configurer et expérimenter plusieurs scénarios possibles. L'intégration d'algorithmes d'optimisation a montré une réelle valeur ajoutée pour l'amélioration et la maîtrise du comportement de notre système. Cela n'a été possible qu'après avoir développé un simulateur efficace permettant de reproduire le plus fidèlement possible le comportement du système réel.

Dans le quatrième chapitre (cf. chapitre 4), nous avons étudié trois types de traitement dont nous avons détaillé leur fonctionnement, le plan expérimental et les résultats. Les modèles associés à ces types de traitement sont analogues à ceux de trois types de maintenance industrielle bien connus dans la littérature. Les résultats ont montré la performance des algorithmes utilisés en terme de temps de traitement et temps de calcul, et représentent un outil d'aide à la décision que les agricultures peuvent utiliser pour établir leurs stratégies de traitement.

Enfin, cette thèse propose plusieurs solutions pour optimiser et faciliter l'introduction des robots de traitement en horticulture. Le simulateur développé constitue une sorte d'outil d'aide à la décision qui permet à l'utilisateur de tester fictivement plusieurs stratégies de traitement et choisir celle qui est la mieux adaptée à son environnement. L'idée d'utiliser une fonction de prédiction du niveau de maladie (et donc sa durée de traitement) dans une méta-heuristique pour l'optimisation du traitement prédictif est simple à reproduire pour d'autres applications d'ordonnancement dynamique.

Perspectives

Les perspectives envisageables, suite aux travaux présentés dans cette thèse, concernent quatre aspects importants :

Perspectives liées aux travaux de thèse

— Une version en ligne du simulateur qui est un peu différente de la version locale. Cette version est déjà développée, il reste à la mettre sur un serveur en ligne. Elle facilite l'accès à distance par les horticulteurs et permet de tester les scénarios souhaités.
— On peut imaginer différentes architectures et dimension de serres à l'aide du simulateur. L'installation de la station de charge au milieu de la serre, ou une station aérienne mobile qui peut charger le robot n'importe où dans la serre. Il y a aussi la possibilité d'initialiser une serre avec des rangées transversales.

Perspectives liées au domaine d'application

— Cette application est également concevable pour le système des robots de nettoyage [Razali and Isa, 2021]. En effet, les robots de nettoyage autonomes doivent effectuer des trai-

tements tout en respectant la capacité de la batterie. On peut modéliser le problème de la même manière avec une approche binpacking et l'objectif de minimiser le nombre de missions de nettoyage.
— Il en va de même pour le problème du routage des véhicules électriques [Mesa et al., 2021], où la contrainte la plus importante est de respecter la capacité des batteries du véhicule. Le couplage entre la simulation et l'optimisation permet de suivre le comportement du système du véhicule électrique et de l'améliorer.

Perspectives à court terme

— L'étude de plusieurs comportements du champignon sur différents types de plantes, ceci est possible par la modification des paramètres du comportement de la maladie sur les plantes.
— Utilisation d'une fonction de prédiction dans le DGA basée sur les matrices de transition markoviennes. L'idée est d'intégrer les matrices de transition dans le chromosome de l'algorithme génétique. Cependant, le temps de calcul peut augmenter, à moins que les matrices ne soient déjà stockées dans des fichiers liés au simulateur. Nous pensons que les matrices de transition markoviennes augmentent la précision de l'algorithme génétique pour le traitement dynamique.

Perspectives à long terme

— Hypothèse de traitement d'une partie de la rangée, qui nécessite un autre modèle mathématique en nombre réel (par exemple : si on traite 30 % d'une rangée et qu'il reste 70 % à traiter).
— Possibilité de développer et d'intégrer d'autres algorithmes d'optimisation, puis de comparer les résultats avec les algorithmes existants.
— Optimisation sur plusieurs serres en même temps, pour permettre à l'horticulteur d'utiliser un robot dans plusieurs serres, et de garder une bonne production en même temps.
— Création d'un modèle dynamique par l'introduction de la variable temps.

Bibliographie

[Abosuliman and Almagrabi, 2021] Abosuliman, S. S. and Almagrabi, A. O. (2021). Routing and scheduling of intelligent autonomous vehicles in industrial logistics systems. *Soft Computing*, pages 1–14.

[Ackerman, 2020] Ackerman, E. (2020). Autonomous robots are helping kill coronavirus in hospitals. *IEEE Spectrum*, 11.

[Aydin and Öztemel, 2000] Aydin, M. E. and Öztemel, E. (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2-3) :169–178.

[Baglee and Jantunen, 2014] Baglee, D. and Jantunen, E. (2014). Can equipment failure modes support the use of a condition based maintenance strategy ? *Procedia CIRP*, 22 :87–91.

[Baker et al., 2002] Baker, S. F., Morton, D. P., Rosenthal, R. E., and Williams, L. M. (2002). Optimizing military airlift. *Operations Research*, 50(4) :582–602.

[Barenji et al., 2017] Barenji, A. V., Barenji, R. V., Roudi, D., and Hashemipour, M. (2017). A dynamic multi-agent-based scheduling approach for smes. *The International Journal of Advanced Manufacturing Technology*, 89(9-12) :3123–3137.

[Baykasoğlu and Ozsoydan, 2018] Baykasoğlu, A. and Ozsoydan, F. B. (2018). Dynamic scheduling of parallel heat treatment furnaces : A case study at a manufacturing system. *Journal of manufacturing systems*, 46 :152–162.

[Begić, 2017] Begić, A. (2017). Application of service robots for disinfection in medical institutions. In *International Symposium on Innovative and Interdisciplinary Applications of Advanced Technologies*, pages 1056–1065. Springer.

[Ben-Daya et al., 2009] Ben-Daya, M., Duffuaa, S. O., Raouf, A., Knezevic, J., and Ait-Kadi, D. (2009). *Handbook of maintenance management and engineering*, volume 7. Springer.

[Benders, 1962] Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1) :238–252.

[Berndt et al., 2015] Berndt, S., Jansen, K., and Klein, K.-M. (2015). Fully dynamic bin packing revisited. *Mathematical Programming*, pages 1–47.

[Bonadies et al., 2016] Bonadies, S., Lefcourt, A., and Gadsden, S. A. (2016). A survey of unmanned ground vehicles with applications to agricultural and environmental sensing. In *Autonomous Air and Ground Sensing Systems for Agricultural Optimization and Phenotyping*, volume 9866, page 98660Q. International Society for Optics and Photonics.

[Bouanan, 2016] Bouanan, Y. (2016). *Contribution à une architecture de modélisation et de simulation à événements discrets : application à la propagation d'information dans les réseaux sociaux*. PhD thesis, Université de Bordeaux.

[Byon and Ding, 2010] Byon, E. and Ding, Y. (2010). Season-dependent condition-based maintenance for a wind turbine using a partially observed markov decision process. *IEEE Transactions on Power Systems*, 25(4) :1823–1834.

[Cai et al., 2017] Cai, Z., Li, X., Ruiz, R., and Li, Q. (2017). A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds. *Future Generation Computer Systems*, 71 :57–72.

[Campi and Garatti, 2018] Campi, M. C. and Garatti, S. (2018). Wait-and-judge scenario optimization. *Mathematical Programming*, 167(1) :155–189.

[Carson and Maria, 1997] Carson, Y. and Maria, A. (1997). Simulation optimization : methods and applications. In *Proceedings of the 29th conference on Winter simulation*, pages 118–126.

[Chan et al., 2009] Chan, J. W.-T., Wong, P. W., and Yung, F. C. (2009). On dynamic bin packing : An improved lower bound and resource augmentation analysis. *Algorithmica*, 53(2) :172–206.

[Chetto et al., 1990] Chetto, H., Silly, M., and Bouchentouf, T. (1990). Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3) :181–194.

[Claude, 2007] Claude, M. (2007). Mildiou de la vigne - bilan de la campagne 2007. In *ACTUALITÉS PHYTOSANITAIRES*, pages 99–105. IFV.

[Coffman et al., 1983] Coffman, Jr, E. G., Garey, M. R., and Johnson, D. S. (1983). Dynamic bin packing. *SIAM Journal on Computing*, 12(2) :227–258.

[Cowling and Johansson, 2002] Cowling, P. and Johansson, M. (2002). Using real time information for effective dynamic scheduling. *European journal of operational research*, 139(2) :230–244.

[Crino et al., 2004] Crino, J., Moore, J., Barnes, J., and Nanry, W. (2004). Solving the theater distribution vehicle routing and scheduling problem using group theoretic tabu search. *Mathematical and computer modelling*, 39(6-8) :599–616.

[Cubero et al., 2020] Cubero, S., Marco-Noales, E., Aleixos, N., Barbé, S., and Blasco, J. (2020). Robhortic : A field robot to detect pests and diseases in horticultural crops by proximal sensing. *Agriculture*, 10(7) :276.

[Dahane et al., 2015] Dahane, M., Sahnoun, M., Bettayeb, B., Baudry, D., and Boudhar, H. (2015). Impact of spare parts remanufacturing on the operation and maintenance performance of offshore wind turbines : a multi-agent approach. *Journal of Intelligent Manufacturing*, pages 1–19.

[Dahl and Nygaard, 1966] Dahl, O.-J. and Nygaard, K. (1966). Simula : an algol-based simulation language. *Communications of the ACM*, 9(9) :671–678.

[Dantzig, 1990] Dantzig, G. B. (1990). Origins of the simplex method in : Nash sg, editor. a history of scientific computing.

[De-An et al., 2011] De-An, Z., Jidong, L., Wei, J., Ying, Z., and Yu, C. (2011). Design and control of an apple harvesting robot. *Biosystems engineering*, 110(2) :112–122.

[Desaulniers et al., 2006] Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2006). *Column generation*, volume 5. Springer Science & Business Media.

[Dhillon, 2002] Dhillon, B. S. (2002). *Engineering maintenance : a modern approach*. cRc press.

[Drogoul, 1993] Drogoul, A. (1993). *De la simulation multi-agents à la résolution collective de problèmes*. PhD thesis, Thesis at University of Paris IV.

[Epstein and Levy, 2010] Epstein, L. and Levy, M. (2010). Dynamic multi-dimensional bin packing. *Journal of Discrete Algorithms*, 8(4) :356–372.

[Farzanegan and Vahidipour, 2009] Farzanegan, A. and Vahidipour, S. (2009). Optimization of comminution circuit simulations based on genetic algorithms search method. *Minerals Engineering*, 22(7) :719–726.

[Federici, 2006] Federici, D. (2006). *Habilitation à Diriger des Recherches Discipline : Informatique Simulation Concurrente de Systèmes à Evénements Discret : Concepts et Applications*. PhD thesis, Université Pascal Paoli.

[Feng et al., 2012] Feng, Q., Wang, X., Zheng, W., Qiu, Q., and Jiang, K. (2012). New strawberry harvesting robot for elevated-trough culture. *International Journal of Agricultural and Biological Engineering*, 5(2) :1–8.

[Ferber, 1999] Ferber, J. (1999). *Multi-agent systems : an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading.

[Fu, 2002] Fu, M. C. (2002). Optimization for simulation : Theory vs. practice. *INFORMS Journal on Computing*, 14(3) :192–215.

[Garredu, 2013] Garredu, S. (2013). *Approche de méta-modélisation et transformations de modèles dans le contexte de la modélisation et simulation à évènements discrets : application au formalisme DEVS*. PhD thesis, Université Pascal Paoli.

[Gonzalez and Sahni, 1976] Gonzalez, T. and Sahni, S. (1976). Open shop scheduling to minimize finish time. *Journal of the ACM (JACM)*, 23(4) :665–679.

[Gonzalez-de Soto et al., 2016] Gonzalez-de Soto, M., Emmi, L., Perez-Ruiz, M., Aguera, J., and Gonzalez-de Santos, P. (2016). Autonomous systems for precise spraying–evaluation of a robotised patch sprayer. *biosystems engineering*, 146 :165–182.

[Gu et al., 2021] Gu, S., Wang, Z., Chen, W., and Wang, J. (2021). Early identification of aspergillus spp. contamination in milled rice by e-nose combined with chemometrics. *Journal of the Science of Food and Agriculture*.

[Guettari et al., 2020] Guettari, M., Gharbi, I., and Hamza, S. (2020). UVC disinfection robot. *Environmental Science and Pollution Research*, pages 1–6.

[Guo et al., 2018] Guo, S., Liu, J., Yang, Y., Xiao, B., and Li, Z. (2018). Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing. *IEEE Transactions on Mobile Computing*, 18(2) :319–333.

[Hameed et al., 2009] Hameed, Z., Hong, Y., Cho, Y., Ahn, S., and Song, C. (2009). Condition monitoring and fault detection of wind turbines and related algorithms : A review. *Renewable and Sustainable energy reviews*, 13(1) :1–39.

[Hassas, 2003] Hassas, S. (2003). Systèmes complexes à base de multi-agents situés. *Mémoire HDR, University Claude Bernard Lyon*.

[Hermenier et al., 2008] Hermenier, F., Lorca, X., Cambazard, H., Menaud, J.-M., and Jussien, N. (2008). Reconfiguration dynamique du placement dans les grilles de calculs dirigée par des objectifs. In *6ième Conférence Francophone sur les Systèmes d'Exploitation (CFSE'06)*.

[Hevesh, 1967] Hevesh, A. H. (1967). Maintainability of phased array radar systems. *IEEE Transactions on Reliability*, 16(1) :61–66.

[Holmberg et al., 1992] Holmberg, K., Folkeson, A., Bergman, B., Ostvik, R., and Aage, C. (1992). *Operational reliability and systematic maintenance*. CRC Press.

[House et al., 2009] House, L., Street, T., and Spa, L. (2009). *How to Contact the Xpress Team*. Fair Isaac Corporation.

[Huff et al., 2021] Huff, D., Dai, S., and Hanrahan, P. (2021). Clockwork : Resource-efficient static scheduling for multi-rate image processing applications on fpgas. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 186–194. IEEE.

[Hwang and Sistler, 1985] Hwang, H. and Sistler, F. (1985). The implementation of a robotic manipulator on a pepper transplanting machine. In *Proc. CAD/CAM, Robotics Automat. Int. Conf.*, pages 553–556.

[Ivanov et al., 2016] Ivanov, D., Dolgui, A., Sokolov, B., Werner, F., and Ivanova, M. (2016). A dynamic model and an algorithm for short-term supply chain scheduling in the smart factory industry 4.0. *International Journal of Production Research*, 54(2) :386–402.

[Janani et al., 2016] Janani, A., Alboul, L., and Penders, J. (2016). Multi robot cooperative area coverage, case study : spraying. In *Conference Towards Autonomous Robotic Systems*, pages 165–176. Springer.

[Janisiewicz et al., 2016] Janisiewicz, W. J., Takeda, F., Nichols, B., Glenn, D. M., Jurick II, W. M., and Camp, M. J. (2016). Use of low-dose uv-c irradiation to control powdery mildew caused by podosphaera aphanis on strawberry plants. *Canadian journal of plant pathology*, 38(4) :430–439.

[Jin et al., 2014] Jin, X., Liu, H. H., Gandhi, R., Kandula, S., Mahajan, R., Zhang, M., Rexford, J., and Wattenhofer, R. (2014). Dynamic scheduling of network updates. In *ACM SIGCOMM Computer Communication Review*, volume 44(4), pages 539–550. ACM.

[Johnson, 1973] Johnson, D. S. (1973). *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology.

[Johnson, 1989] Johnson, E. L. (1989). Modeling and strong linear programs for mixed integer programming. In *Algorithms and model formulations in mathematical programming*, pages 1–43. Springer.

[Jordan and Hunter, 1972] Jordan, V. and Hunter, T. (1972). The effects of glass cloche and coloured polyethylene tunnels on microclimate, growth, yield and disease severity of strawberry plants. *Journal of horticultural science*, 47(3) :419–426.

[Jun et al., 2021] Jun, J., Kim, J., Seol, J., Kim, J., and Son, H. I. (2021). Towards an efficient tomato harvesting robot : 3d perception, manipulation, and end-effector. *IEEE Access*, 9 :17631–17640.

[Karimi et al., 2021] Karimi, H., Bahmani, R., and Jadid, S. (2021). Stochastic multi-objective optimization to design optimal transactive pricing for dynamic demand response programs : A bi-level fuzzy approach. *International Journal of Electrical Power & Energy Systems*, 125 :106487.

[Kemeny and Snell, 1976] Kemeny, J. G. and Snell, J. L. (1976). *Markov chains*, volume 6. Springer-Verlag, New York.

[Klement, 2014] Klement, N. (2014). *Planification et affectation de ressources dans les réseaux de soin : analogie avec le problème du bin packing, proposition de méthodes approchées*. PhD thesis, Université Blaise Pascal-Clermont-Ferrand II.

[Kouiss et al., 1997] Kouiss, K., Pierreval, H., and Mebarki, N. (1997). Using multi-agent architecture in fms for dynamic scheduling. *Journal of intelligent Manufacturing*, 8(1) :41–47.

[Le et al., 2020] Le, T. D., Ponnambalam, V. R., Gjevestad, J. G., and From, P. J. (2020). A low-cost and efficient autonomous row-following robot for food production in polytunnels. *Journal of Field Robotics*, 37(2) :309–321.

[Leinberger et al., 1999] Leinberger, W., Karypis, G., and Kumar, V. (1999). Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *Proceedings of the 1999 International Conference on Parallel Processing*, pages 404–412. IEEE.

[Li et al., 2017] Li, J., Wang, P., and Geng, C. (2017). The disease assessment of cucumber downy mildew based on image processing. In *Computer Network, Electronic and Automation (ICCNEA), 2017 International Conference on*, pages 480–485. IEEE.

[Li et al., 2015a] Li, Y., Chen, M., Dai, W., and Qiu, M. (2015a). Energy optimization with dynamic task scheduling mobile cloud computing. *IEEE Systems Journal*, 11(1) :96–105.

[Li et al., 2015b] Li, Y., Tang, X., and Cai, W. (2015b). Dynamic bin packing for on-demand cloud resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 27(1) :157–170.

[Lim et al., 2009] Lim, M. K., Zhang, Z., and Goh, W. (2009). An iterative agent bidding mechanism for responsive manufacturing. *Engineering Applications of Artificial Intelligence*, 22(7) :1068–1079.

[Magableh, 2006] Magableh, M. A. (2006). *A theoretical and empirical analysis of the Wagner hypothesis of public expenditure growth.* PhD thesis, University of Western Sydney.

[Mazar et al., 2020a] Mazar, M., Bettayeb, B., Klement, N., Louis, A., and Sahnoun, M. (2020a). Optimisation dynamique des traitement de maladies agricoles. Technical report, ROADEF.

[Mazar et al., 2020b] Mazar, M., Bettayeb, B., Klement, N., Sahnoun, M., and Louis, A. (2020b). Dynamic scheduling of robotic mildew treatment by uv-c in horticulture. In *10th SOHOMA European Workshop on Service Oriented, Holonic and Multi-Agent Manufacturing Systems for Industry of the Future.*

[Mazar et al., 2021] Mazar, M., Bettayeb, B., Klement, N., Sahnoun, M., and Louis, A. (2021). Multi-agent simulation to predict global behavior of population based on elementary local markovian model. In *2021 1st International Conference On Cyber Management And Engineering (CyMaEn)*, pages 1–5. IEEE.

[Mazar et al., 2019] Mazar, M., M'Hammed Sahnoun, B. B., Benatia, M. A., and Louis, A. (2019). Optimisation de la planification des tâches de traitement robotisé de maladies en horticulture.

[Mazar et al., 2018] Mazar, M., Sahnoun, M., Bettayeb, B., and Klement, N. (2018). Optimization of robotized tasks for the uv-c treatment of diseases in horticulture. In *The 2018 International Conference of the African Federation of Operational Research Societies (AFROS 2018).*

[Mazar et al., 2020c] Mazar, M., Sahnoun, M., Bettayeb, B., Klement, N., and Louis, A. (2020c). Simulation and optimization of robotic tasks for uv treatment of diseases in horticulture. *Operational Research*, pages 1–27.

[Mei et al., 2005] Mei, Y., Lu, Y.-H., Hu, Y. C., and Lee, C. G. (2005). A case study of mobile robot's energy consumption and conservation techniques. In *ICAR'05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, pages 492–497. IEEE.

[Mesa et al., 2021] Mesa, F., Granada, J. G., and Velez, G. C. (2021). Electric vehicle routing problem optimal solution. In *Journal of Physics : Conference Series*, volume 1981, page 012006. IOP Publishing.

[Oberti et al., 2016] Oberti, R., Marchi, M., Tirelli, P., Calcante, A., Iriti, M., Tona, E., Hočevar, M., Baur, J., Pfaff, J., Schütz, C., et al. (2016). Selective spraying of grapevines for disease control using a modular agricultural robot. *Biosystems Engineering*, 146 :203–215.

[O'Driscoll Jr and Rizzo, 2002] O'Driscoll Jr, G. P. and Rizzo, M. J. (2002). *The Economics of Time and Ignorance : With a New Introduction.* Routledge.

[Ören et al., 2014] Ören, T., Yilmaz, L., and Ghasem-Aghaee, N. (2014). A systematic view of agent-supported simulation past, present, and promising future. In *Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), 2014 International Conference on*, pages 497–506. IEEE.

[Ouelhadj and Petrovic, 2009] Ouelhadj, D. and Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling*, 12(4) :417–431.

[Padberg and Rinaldi, 1991] Padberg, M. and Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1) :60–100.

[Peries, 1962] Peries, O. (1962). Studies on strawberry mildew, caused by sphaerotheca macularis (wallr. ex fries) jaczewski. *Annals of Applied Biology*, 50(2) :211–224.

[Powell, 2005] Powell, W. B. (2005). The optimizing-simulator : Merging simulation and optimization using approximate dynamic programming. In *Proceedings of the 37th conference on Winter simulation*, pages 96–109. Winter Simulation Conference.

[Powell, 2008] Powell, W. B. (2008). Approximate dynamic programming : lessons from the field. In *Winter Simulation Conference 2008*, pages 205–214. IEEE.

[Qin et al., 2018] Qin, W., Zhang, J., and Song, D. (2018). An improved ant colony algorithm for dynamic hybrid flow shop scheduling with uncertain processing time. *Journal of Intelligent Manufacturing*, 29(4) :891–904.

[Rao et al., 2009] Rao, S. C., Dove, G., Cascino, G. D., and Petersen, R. C. (2009). Recurrent seizures in patients with dementia : frequency, seizure types, and treatment outcome. *Epilepsy & behavior*, 14(1) :118–120.

[Rawlings et al., 2019] Rawlings, B. C., Avadiappan, V., Lafortune, S., Maravelias, C. T., and Wassick, J. M. (2019). Incorporating automation logic in online chemical production scheduling. *Computers & Chemical Engineering*, 128 :201–215.

[Razali and Isa, 2021] Razali, N. S. M. and Isa, K. (2021). Simulation of adaptive control system for multiagent autonomous underwater cleaning robots. *Evolution in Electrical and Electronic Engineering*, 2(2) :709–718.

[Ren and Tang, 2016] Ren, R. and Tang, X. (2016). Clairvoyant dynamic bin packing for job scheduling with minimum server usage time. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 227–237. ACM.

[Ribrant, 2006] Ribrant, J. (2006). Reliability performance and maintenance-a survey of failures in wind power systems. *KTH school of Electrical Engineering*, pages 59–72.

[Ryer, 1999] Ryer, D. M. (1999). Implementation of the metaheuristic tabu search in route selection for mobility analysis support system. Technical report, DTIC Document.

[Sahni and Vidyarthi, 2015] Sahni, J. and Vidyarthi, D. P. (2015). A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. *IEEE Transactions on Cloud Computing*, 6(1) :2–18.

[Sahnoun et al., 2015] Sahnoun, M., Baudry, D., Mustafee, N., Louis, A., Smart, P. A., Godsiff, P., and Mazari, B. (2015). Modelling and simulation of operation and maintenance strategy for offshore wind farms based on multi-agent system. *Journal of Intelligent Manufacturing*, pages 1–17.

[Sahnoun et al., 2019] Sahnoun, M., Baudry, D., Mustafee, N., Louis, A., Smart, P. A., Godsiff, P., and Mazari, B. (2019). Modelling and simulation of operation and maintenance strategy for offshore wind farms based on multi-agent system. *Journal of Intelligent Manufacturing*, 30(8) :2981–2997.

[Sahnoun et al., 2016] Sahnoun, M., Bettayeb, B., Bassetto, S.-J., and Tollenaere, M. (2016). Simulation-based optimization of sampling plans to reduce inspections while mastering the risk exposure in semiconductor manufacturing. *Journal of Intelligent Manufacturing*, 27(6) :1335–1349.

[Sakai et al., 2008] Sakai, S., Iida, M., Osuka, K., and Umeda, M. (2008). Design and control of a heavy material handling manipulator for agricultural robots. *Autonomous Robots*, 25(3) :189–204.

[Sarri et al., 2017] Sarri, D., Martelloni, L., and Vieri, M. (2017). Development of a prototype of telemetry system for monitoring the spraying operation in vineyards. *Computers and Electronics in Agriculture*, 142 :248–259.

[Selim and Gurel, 2007] Selim, M. and Gurel, A. S. (2007). Machining conditions-based preventive maintenance. *International journal of production research*, 45(8) :1725–1743.

[Shannon and Johannes, 1976] Shannon, R. and Johannes, J. D. (1976). Systems simulation : the art and science. *IEEE transactions on systems, man, and cybernetics*, 10 :723–724.

[Shcherbakov et al., 2020] Shcherbakov, M. V., Glotov, A. V., and Cheremisinov, S. V. (2020). Proactive and predictive maintenance of cyber-physical systems. In *Cyber-Physical Systems : Advances in Design & Modelling*, pages 263–278. Springer.

[Shi et al., 2021] Shi, L., Guo, G., and Song, X. (2021). Multi-agent based dynamic scheduling optimisation of the sustainable hybrid flow shop in a ubiquitous environment. *International Journal of Production Research*, 59(2) :576–597.

[Sistler, 1987] Sistler, F. (1987). Robotics and intelligent machines in agriculture. *IEEE Journal on Robotics and Automation*, 3(1) :3–6.

[Southall et al., 2002] Southall, B., Hague, T., Marchant, J. A., and Buxton, B. F. (2002). An autonomous crop treatment robot : Part i. a kalman filter model for localization and crop/weed classification. *The international journal of robotics research*, 21(1) :61–74.

[Suresh and Chaudhuri, 1993] Suresh, V. and Chaudhuri, D. (1993). Dynamic scheduling—a survey of research. *International journal of production economics*, 32(1) :53–63.

[Talbot, 2014] Talbot, D. (2014). A nimble-wheeled farm robot goes to work in minnesota.

[Tang et al., 2016] Tang, D., Dai, M., Salido, M. A., and Giret, A. (2016). Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization. *Computers in Industry*, 81 :82–95.

[Thakor and Shah, 2011] Thakor, D. and Shah, A. (2011). D_edf : An efficient scheduling algorithm for real-time multiprocessor system. In *2011 World Congress on Information and Communication Technologies*, pages 1044–1049. IEEE.

[Thiessen et al., 2016] Thiessen, L., Keune, J., Neill, T., Turechek, W., Grove, G., and Mahaffee, W. (2016). Development of a grower-conducted inoculum detection assay for management of grape powdery mildew. *Plant Pathology*, 65(2) :238–249.

[Tranier, 2007] Tranier, J. (2007). *Vers une vision intégrale des systèmes multi-agents.* PhD thesis, Université Montpellier II, Montpellier, Thèse de doctorat.

[Trentesaux, 2009] Trentesaux, D. (2009). Distributed control of production systems. *Engineering Applications of Artificial Intelligence*, 22(7) :971–978.

[Troitzsch, 2009] Troitzsch, K. G. (2009). Perspectives and challenges of agent-based simulation as a tool for economics and other social sciences. In *AAMAS (1)*, pages 35–42. Citeseer.

[Umar et al., 2015] Umar, U. A., Ariffin, M., Ismail, N., and Tang, S. (2015). Hybrid multiobjective genetic algorithms for integrated dynamic scheduling and routing of jobs and automated-guided vehicle (agv) in flexible manufacturing systems (fms) environment. *The International Journal of Advanced Manufacturing Technology*, 81(9-12) :2123–2141.

[Van Henten et al., 2002] Van Henten, E. J., Hemming, J., Van Tuijl, B., Kornet, J., Meuleman, J., Bontsema, J., and Van Os, E. (2002). An autonomous robot for harvesting cucumbers in greenhouses. *Autonomous Robots*, 13(3) :241–258.

[Van Rossum and Drake Jr, 1995] Van Rossum, G. and Drake Jr, F. L. (1995). *Python tutorial*, volume 620. Centrum voor Wiskunde en Informatica Amsterdam.

[Vicino, 2015] Vicino, D. A. (2015). *Improved time representation in discrete-event simulation.* PhD thesis, Université Nice Sophia Antipolis.

[Wainer, 2009] Wainer, G. A. (2009). *Discrete-event modeling and simulation : a practitioner's approach.* CRC press.

[Wang et al., 2017] Wang, D.-J., Liu, F., and Jin, Y. (2017). A multi-objective evolutionary algorithm guided by directed search for dynamic scheduling. *Computers & Operations Research*, 79 :279–290.

[Wang, 2012] Wang, W. (2012). An overview of the recent advances in delay-time-based maintenance modelling. *Reliability Engineering & System Safety*, 106 :165–178.

[White, 2012] White, S. D. (2012). *Implications of new sustainable greenhouse systems for pests, diseases and biological control : A modelling approach using Oidium neolycopersici and Tetranychus urticae.* PhD thesis, University of Warwick.

[Wilensky and Evanston, 1999] Wilensky, U. and Evanston, I. (1999). Netlogo : Center for connected learning and computer-based modeling. *Northwestern University, Evanston, IL*, pages 49–52.

[Wspanialy and Moussa, 2016] Wspanialy, P. and Moussa, M. (2016). Early powdery mildew detection system for application in greenhouse automation. *Computers and Electronics in Agriculture*, 127 :487–494.

[Wu et al., 2021] Wu, J., Xiang, B., Bai, J., Li, W., Liu, Y., Xiang, H., and Qu, L. (2021). Analysis of types and treatment methods of cervical massive hemorrhage. *Zhonghua yi xue za zhi*, 101(29) :2283–2287.

[Wu et al., 2003] Wu, T., Powell, W. B., and Whisman, A. (2003). The optimizing simulator : An intelligent analysis tool for the military airlift problem. *Unpublished Report. Department of Operations Research and Financial Engineering, Princeton University, Princeton NJ.*

[Xiang and Lee, 2008] Xiang, W. and Lee, H. P. (2008). Ant colony intelligence in multi-agent dynamic manufacturing scheduling. *Engineering Applications of Artificial Intelligence*, 21(1) :73–85.

[Zankoul et al., 2015] Zankoul, E., Khoury, H., and Awwad, R. (2015). Evaluation of agent-based and discrete-event simulation for modeling construction earthmoving operations. In *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, volume 32, page 1. IAARC Publications.

[Zeigler et al., 2000] Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of modeling and simulation : integrating discrete event and continuous complex dynamic systems*. Academic press.

[Zhai et al., 2020] Zhai, Z., Martínez, J. F., Beltran, V., and Martínez, N. L. (2020). Decision support systems for agriculture 4.0 : Survey and challenges. *Computers and Electronics in Agriculture*, 170 :105256.

[Zhang et al., 2002] Zhang, N., Wang, M., and Wang, N. (2002). Precision agriculture-a worldwide overview. *Computers and electronics in agriculture*, 36(2-3) :113–132.

[Zhang et al., 2018] Zhang, S., Ding, F., Peng, H., Huang, Y., and Lu, J. (2018). Molecular cloning of a cc–nbs–lrr gene from vitis quinquangularis and its expression pattern in response to downy mildew pathogen infection. *Molecular Genetics and Genomics*, 293(1) :61–68.

Annexe A

.1 Outils de calculs

Dans cette annexe nous allons parler brièvement des outils qu'on a utilisés pour le développement du simulateur et des algorithmes d'optimisation.

.1.1 Netlogo

NetLogo est un environnement de modélisation et de simulation programmable pour les systèmes multi-agents. [Wilensky and Evanston, 1999] ont conçu et rédigé Netlogo à l'université Northwestern située à Evanston en banlieue nord de Chicago, dans l'État de l'Illinois. Il est utilisé dans l'enseignement par des étudiants, d'enseignants et de chercheurs et dans le domaine professionnel aussi. Le logiciel est gratuit et open source, il a été écrit avec les langage Scala et Java. Il existe des versions pour tous les systèmes d'exploitation.

L'utilisation de Netlogo est facile pour le développement d'un environnement avec les systèmes multi-agents. L'autre avantage est que le logiciel donne une liberté totale de manipuler les agents, puisque le développement se fait par des lignes de code. La difficulté vient quand l'environnement devient complexe et quand on veut intégrer des algorithmes d'optimisation, car il faut savoir ou les intégrer et les paramétrer.

Le premier simulateur que nous avons développé était basique, on a juste commencé par l'installation des agents et maîtrisé la taille de la serre. Puis nous avons développé le robot qui se déplace dans la serre et surtout entre les rangées. Les premiers déplacements était sans traitement, jusqu'à la partie d'intégration d'algorithmes d'optimisation. Cette dernière est une tâche un peu difficile, car il faut savoir où intégrer les algorithmes et le moment de les déclencher. En effet, le simulateur tourne sans arrêt et si pour chaque pas de simulation on lance un des algorithmes d'optimisation, le temps de calcul monte en exponentiel.

Netlogo contient aussi des extensions qui permettent d'utiliser d'autres codes sur le simulateur. Parmi ses extensions on trouve CSV, GIS, Matrices, R, Arduino, Bitmap et Python. Nous avons utilisé CSV pour la sauvegarde des données, Matrices pour les algorithmes d'optimisation et Python pour réduire quelques codes et augmenter le temps de calcul.

.1.2 Python

Python est un langage de programmation puissant, facile à lire et à comprendre. Il démontre la plupart de ces fonctionnalités communes à de nombreux autres langages et est utile pour les applications du monde réel. Il s'agit également d'un logiciel libre, doté d'une implémentation standard et d'une communauté importante ce qui augmente sa puissance. La communauté développe pas mal d'algorithmes de tous genre et crée beaucoup de bibliothèques. Il est facile à manipuler pour un développeur qui maîtrise déjà un autre langage [Van Rossum and Drake Jr, 1995].

Nous avons utilisé du code Python dans deux cas, le premier pour l'affichage des résultats et le deuxième à l'intérieur du simulateur puisque Netlogo accepte du code Python. Nous avons utilisé les bibliothèques suivantes :

- Numpy : permet de manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.
- Pandas : permet la manipulation et l'analyse des données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles.
- Matplotlib : est destinée à tracer et visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy.
- Plotly.graph_objects : crée des graphes représentées par des structures de données arborescentes qui sont automatiquement sérialisées en JSON pour être rendues par la bibliothèque JavaScript Plotly.js.
- Csv : permet de lire et d'écrire des fichiers csv très facilement.

.1.3 FICO XPRES

FICO Xpress est un solveur qui a un puissant outil d'optimisation mathématique bien adapté à un large éventail de problèmes d'optimisation. Il dispose d'interfaces via la ligne de commande «optimiseur» de Console Xpress, via l'application d'interface graphique IVE et via une bibliothèque accessible depuis toutes les principales plates-formes de programmation. Il combine une fonctionnalité d'accès aux données flexible et des algorithmes d'optimisation, qui permettent à l'utilisateur de gérer les problèmes de plus en plus complexes qui se posent dans l'industrie et le milieu universitaire. Il traite les problèmes de la programmation linéaire (LP), la programmation linéaire mixte en nombres entiers (MILP), la programmation quadratique convexe (QP), la programmation quadratique contrainte quadratique convexe (QCQP) et La programmation non linéaire (NLP). Plusieurs méthodes de recherche opérationnelle y sont intégrées : méthode du simplex, méthode de quasi-Newton (Newton Barrier Method), Branch and Bound, QCQP Methods et des méthodes de résolution des programmations non-linéaires convexes (Méthodes objectives non linéaires convexes). Les interfaces de programmation disponibles de la bibliothèque incluent des interfaces pour C/C++, .NET, Java et Visual Basic pour Applications (VBA) [House et al., 2009].

Le solveur nous a aidé pour résoudre notre problème pour le traitement conditionnel. Nous avons introduit notre modèle mathématique dans FICO XPRES pour obtenir une solution optimale qui est le nombre minimum des missions pour faire un traitement total de la serre. La solution obtenue est introduite manuellement dans le simulateur. En effet, nous avons mis dans le simulateur le choix d'utiliser une méthode exacte. Si le simulateur tourne avec la méthode exacte, il tourne jusqu'au déclenchement d'un traitement conditionnel. En ce moment, le simulateur se met en pause en générant

une matrice d'état de la serre. Cette matrice constitue les paramètres d'entrée du solveur.

Annexe B

.1 Codage des agents

Dans cette annexe il y a l'explication du fonctionnement de chaque agent développé dans le simulateur. En détaillant ainsi ses variables, son initialisation, et sa mise à jour. Un diagramme représente le comportement global de chaque agent sous forme de logigramme. Cette partie est la suite de la représentation des algorithmes de simulation utilisés dans le chapitre 2.

.1.1 Greenhouse

La serre représente tout l'environnement de simulateur. C'est là où tous les autres agents sont installés. C'est un agent simple, il est caractérisé par la taille de l'environnement de la serre et la couleur de ses `patches` qui sont initialisé au début.

.1.2 Plants

L'agent `Plants` est une classe d'agents qui contient tous les agents `Plant`. Si on veut modifier une variable sur les plantes on fait appel à l'agent `Plants`, et si on fait une modification sur une plante précise on fait appel à l'agent `Plant` (numéro de la plante). On a besoin de programmer le comportement des agents `Plant` pour simuler le comportement réel des plantes.

Variables des plantes :

Les variables de la classe d'agents `Plantes` sont présentes dans la figure 24. La variable `level-mildiew` indique le niveau de maladie de chaque plante, elle augmente en fonction du comportement du mildiou, et diminue avec le traitement. Au début du projet, les partenaires ont mis en hypothèse que chaque quatre plantes suivent le même comportement, c'est pour ça nous avons ajouté la variable `tray` qui indique chaque plateau des plantes. Les deux variables `treat-left` et `treat-righ` sont utilisées pour savoir si la plante est traitée des deux côtés. La variable `last-treatment` est utilisée pour connaître le dernier jour de traitement, et `last-infect` est une variable qui s'incrémente toutes les heures, pour connaître la dernière heure de traitement. En effet, la séparation de ces deux variables permet d'accélérer l'accès aux données pour différents besoins lors de l'exécution du simulateur. Le comportement du mildiou est différent d'une plante à l'autre, car chacune possède sa propre matrice markovienne `Markovian-matrix`.

FIGURE 24 : Les variables locales de chaque agent `plant`

Initialisation des plantes :

La fonction d'initialisation de la classe d'agents `Plants`, qui est appelée par la fonction `Setup`, est montré dans la figure 25. Au début, le programme crée des plantes grâce à la syntaxe de NetLogo `create-ordered-(classe d'agent) (nombre d'agents à créer)`. Le nombre de plantes est définie par la variable globale `Plants-number` qui se trouve dans l'interface de simulation. Après la création des plantes, le programme initialise les variables de chaque plante : `shape` pour l'apparence sur l'interface, `size` pour la taille, `treat-left` et `treat-right` initialisées à `false` au début de simulation et `last-infect` initialisée à 0 au début de simulation. Enfin, le programme fait appel à la fonction `line-plants` qui aligne les plantes dans la serre. Le logigramme du programme `line-plants` est représenté sur la figure 26. Le programme initialise d'abord les variables (x,y) qui sont les coordonnées de la première plante. La première chose faite dans la boucle est d'attribuer les coordonnées (x,y) et `tray` qui identifie un ensemble de quatre plantes appelé "plateau". Ensuite, si l'utilisateur a choisi de démarrer une simulation avec un pourcentage de plantes malade, le programme distribue aléatoirement la maladie par les plateaux. Vient ensuite la création de la matrice Markovienne de chaque plante. À la fin, si on atteint la limite d'une rangée, on commence une nouvelle rangée avec de nouvelles coordonnées (x,y). Lorsque la boucle `For` se termine, le programme appelle la fonction `state-plants`.

Fonction mise à jour des plantes :

La fonction de mise à jour des plantes est appelée par le bouton `Go` après le lancement de la simulation. Cette fonction appelle les deux autres fonctions `state-plants` et `infection-plants`. Le programme de la fonction `state-plants` est détaillé dans le logigramme de la figure 27. Se programme se lance à chaque fois que les niveaux de maladies sur les plantes changent. Il y a une boucle `For-all` qui permet de parcourir toutes les plantes afin de définir une couleur pour chaque

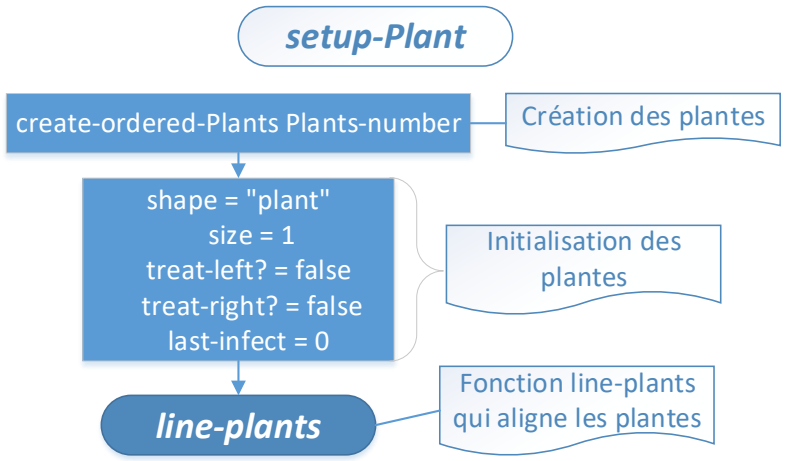

FIGURE 25 : Logigramme du code d'initialisation des plantes

plante par rapport à son niveau de maladie. Cette fonction permet aux utilisateurs de voir l'état des plantes dans l'interface de simulation. La deuxième fonction de la mise à jour est `infection-plants` qui gère l'évolution de mildiou sur les plantes. La figure 28 montre le logigramme de cette fonction. Comme dans toutes les fonctions de l'agent `Plants`, il y a une boucle, pour parcourir toutes les plantes, qui commence par incrémenter la variable `last-infect`. Puis, on vérifie si le niveau de maladie de la plante est inférieur à 30 (le niveau maximal). Si oui, le programme génère une variable aléatoire (R) qui sera comparé par la suite avec la probabilité d'évolution de la maladie d'une plante i (P_i). Si ($R \leq P_i$), le programme augmente le niveau de maladie. Sinon, le niveau de maladie de la plante n'évolue pas. Enfin, la fonction `infect-plants` fait appel à la fonction `state-plants`.

.1.3 Robot

Le robot a plusieurs fonctions dans le simulateur qui lui permet de faire des traitements, de se déplacer, de se décharger et de se charger. Le type d'agents `Robots` peut contenir plusieurs agents `Robot`, selon le nombre de robots dans la serre indiqué par l'utilisateur.

Variables des robots :

Chaque robot a besoin de plusieurs variables comme le montre la figure 29. Dans un premier temps, nous avons défini les variables les plus nécessaires comme celle d'énergie du robot, `Energy` pour suivre l'état de sa batterie, la permission de déplacement défini par `move?`, une variable booléenne, si elle vraie le robot peut bouger, sinon il est en mode arrêt, et la variable `destination` pour lui indiquer sa prochaine destination. D'autres variables sont défini tels que `Row-matrix` qui contient les coordonnées de chaque rangée. Puis nous avons les deux variables `left-lamp` et `right-lamp` pour identifier les lampes de chaque côté du robot, la variable `nose` pour identifier le E-nose et `speed` qui permet

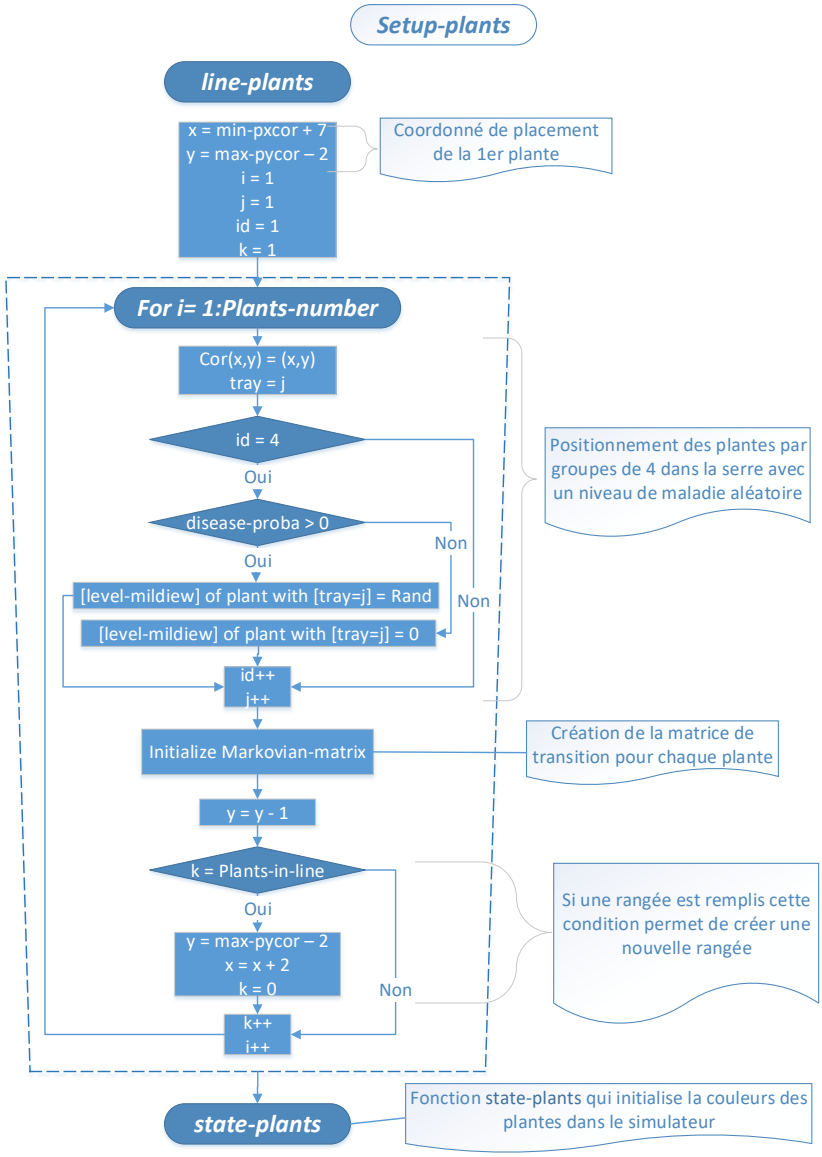

FIGURE 26 : Logigramme de la fonction `line-plants`

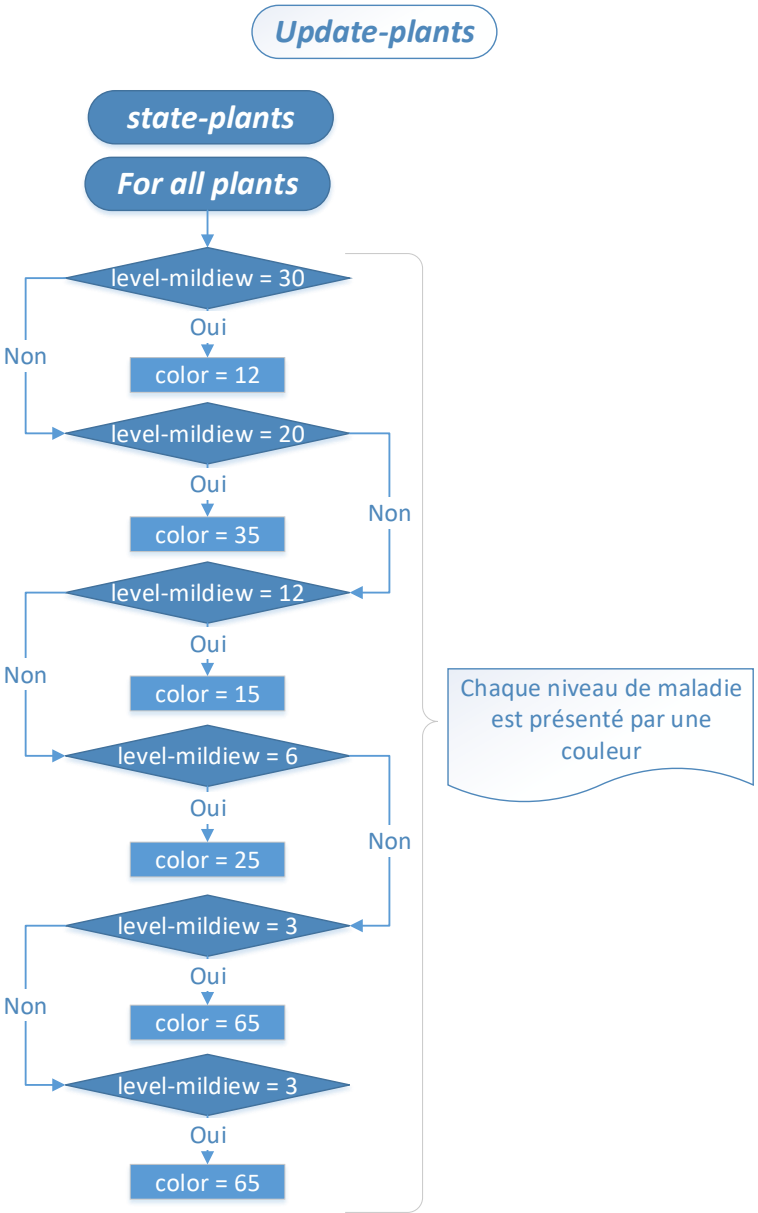

FIGURE 27 : Logigramme de la fonction `state-plants`

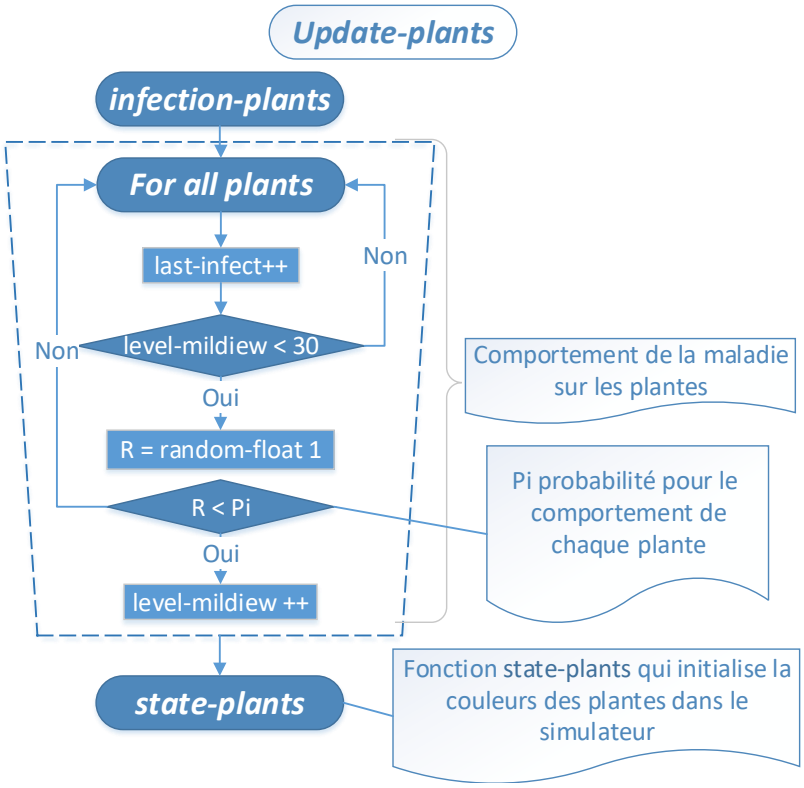

FIGURE 28 : Logigramme de la fonction `infection-plants`

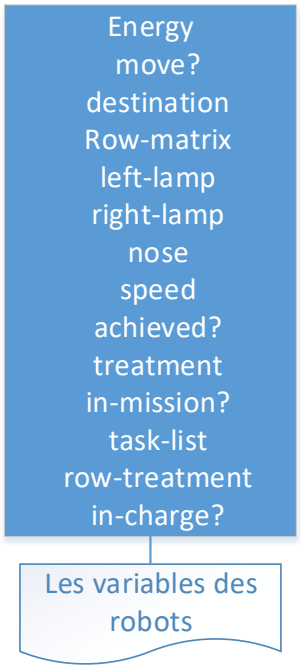

FIGURE 29 : Les variables locales de chaque agent `Robot`

d'identifier la vitesse de déplacement du robot. La variable `achieved?` est une variable booléenne qui est vraie si le robot a atteint une destination. La différence entre les deux variables booléenne `treatment` et `in-mission` est que la première est vraie juste lorsque le robot est devant une plante pendant un traitement, alors que la deuxième est vraie tant que le robot a quitté la station de charge pour faire des traitements. La variable `task-list` contient la liste des rangées qui seront traitées par le robot. La variable `row-treatment` est un peu spéciale, elle sert à gérer les déplacements du robot à l'intérieur des rangées. Cette variable est égale à 0 si le robot se déplace vers l'entrer de la rangée, est égale à 1 si le robot se trouve dans la rangée et sa destination est le font de la rangée, ou égale à 2 si le robot a terminé son traitement et il est dans le chemin de retour pour quitter la rangée. Enfin, la variable `in-charge?` est une variable booléenne qui est vraie quand le robot est dans la station de charge et qu'il n'est pas à 100% de sa charge.

Initialisation des robots :

L'initialisation des robots se fait sur deux parties, la première pour initialiser les variables de chaque robot et la deuxième pour initialiser les coordonnées des rangées dans une matrice qui permet aux robots de se repérer facilement dans la serre.

La figure 30 montre un logigramme d'initialisation des variables du robot. Au début, il y a la création des robots à l'aide de la fonction NetLogo `creat-Robots Nbr_Robots`. Le nombre de robots créés est choisi par l'utilisateur via l'interface de simulation avec la variable globale `Nbr_Robots`. Par la suite on initialise les variables `in-mission?`, `Energy`, `move?`, `achieved?` et `in-charge?`

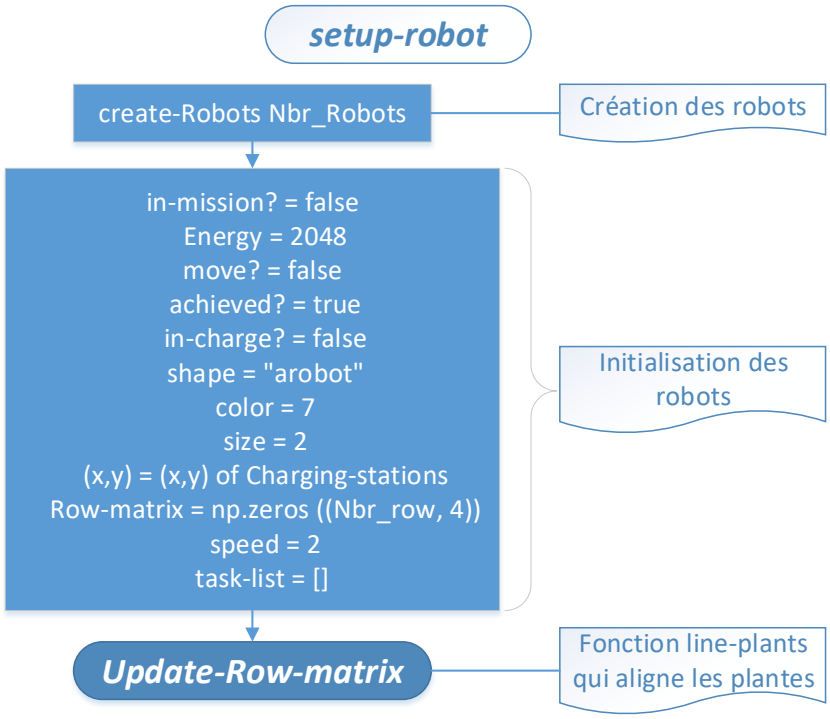

FIGURE 30 : Logigramme d'initialisation des robots

définies pour un robot sans mission se trouvant dans la station de charge avec une batterie chargée à 100%. La charge maximal des batteries du robot est de 2048 Kw. Ensuite, les variables `shape`, `color`, `size` et `(x,y)` sont utilisées pour définir sa forme, sa couleur, sa taille et les coordonnées de son emplacement initial, qui sont les mêmes de la station de charge. Dans la dernière étape, on initialise la matrice des coordonnées des rangées `Row-matrix`, la liste des tâches `task-list` et la vitesse de départ `speed` qui est égale à 2 mètres par seconde. La figure 31 montre le logigramme d'initialisation de la matrice des coordonnées des rangées. La fonction `Update-Row-matrix` crée une matrice des coordonnées pour tous les robots. Cette fonction commence par initialiser la première rangée. Puis, elle construit la matrice `Row-matrix` à partir des premiers données. Les deux boucles `For i in rang (0, Nbr_row)` et `For j in rang (0, 4)` permettent de parcourir toutes les lignes et les colonnes de la matrice et la remplir. La matrice contient `Nbr_row` lignes qui sont le nombre de rangée, et 4 colonnes qui correspondent aux coordonnées des rangées, dont les deux premières colonnes sont x et y de l'entrée de chaque rangée, et les deux dernières colonnes correspondent aux x et y du fond de chaque rangée.

Fonction mise à jour des robots :

L'agent de type `Robots` a besoin de plusieurs fonctions de mise à jour, pour assurer le bon fonctionnement des robots dans le simulateur. La figure 32 montre le logigramme du programme d'exécution des robots. Ce code explore tous les robots un par un et commence par vérifier si un robot a at-

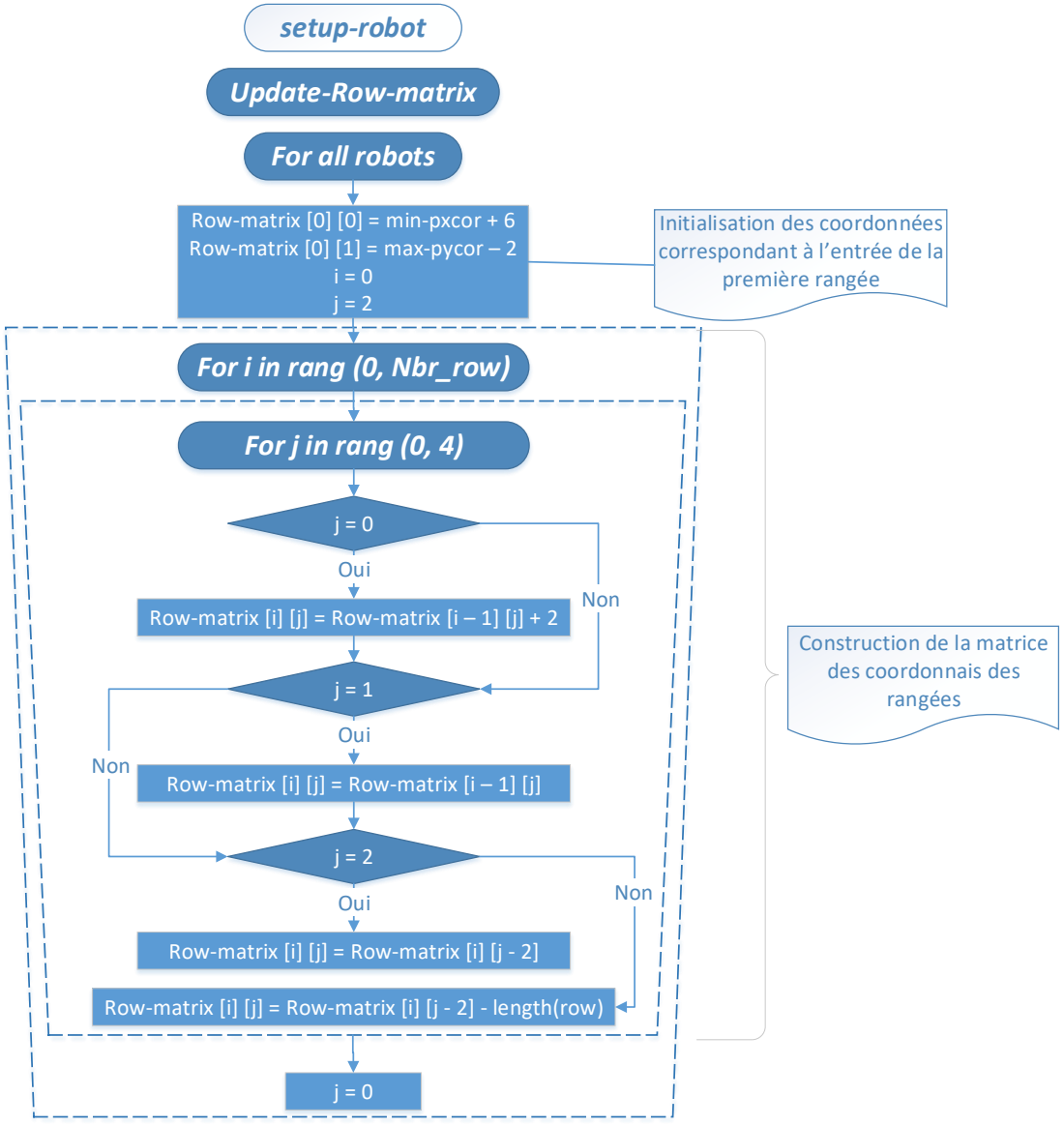

FIGURE 31 : Logigramme d'initialisation de la matrice de coordonnées des rangées

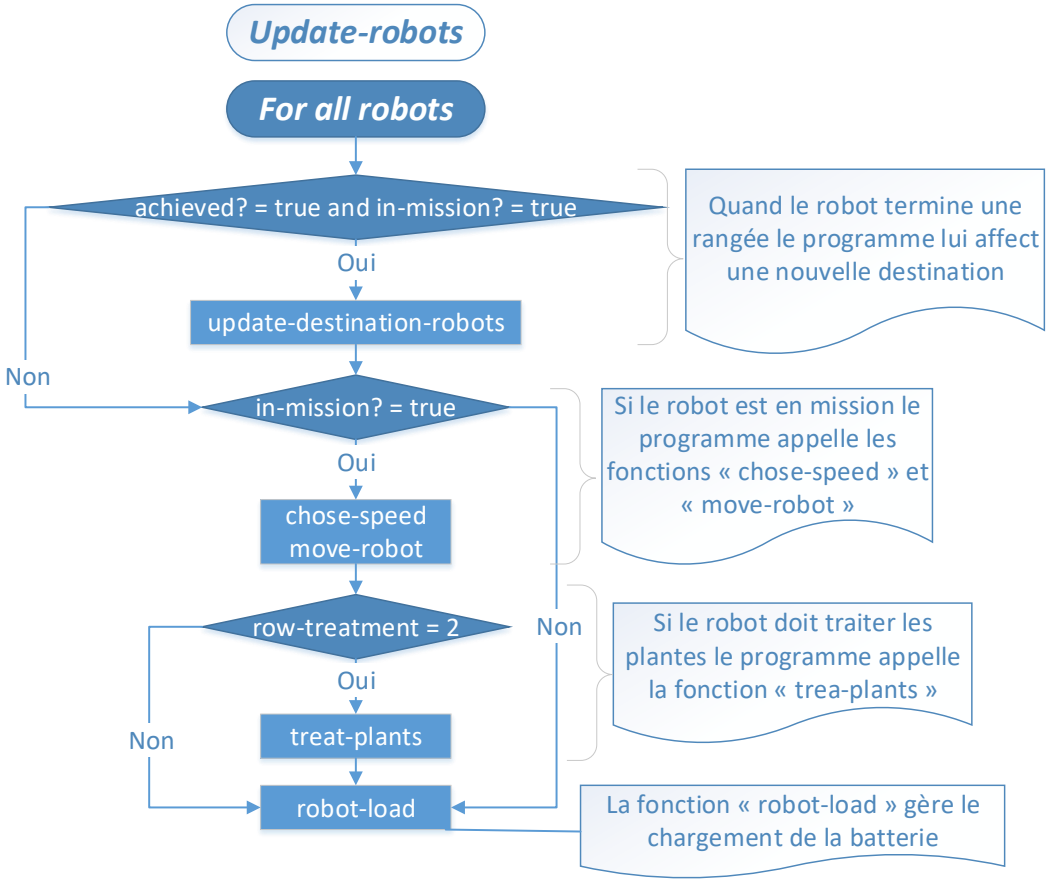

FIGURE 32 : Logigramme du fonctionnement de la mise à jour des robots

teint une destination et lui en assigne une nouvelle grâce à la fonction `update-destination-robots`. Sinon, s'il n'a pas atteint sa destination mais est en mission, le programme appelle les fonctions `chose-speed` et `move-robot` pour assurer le déplacement du robot. Ensuite, la fonction mise à jour vérifie si le robot doit traiter des plantes afin d'appeler la fonction `treat-plants`. Enfin, le programme `Update-robots` exécute la fonction `robot-load` pour gérer la charge de la batterie du robot.

Chaque robot a six vitesses dont une pour les déplacements sans traitement et cinq pour le traitement de différent niveaux de maladies. Le logigramme de la fonction qui gère les vitesses du robot se trouve dans la figure 33. Au début le programme, la fonction `chose-speed` vérifie si le type de traitement est préventif pour fixer une vitesse constante pour tous les traitements. Puis, il initialise la variable `nearest-plant`. Cette variable contient le niveau de maladie le plus petit d'une des plantes qui se trouve à côté du robot. Ensuite, si le robot se déplace entre les rangées sans traiter, le programme lui affecte la vitesse la plus rapide qui est de 2 mètres par seconde. Sinon, il choisit une des vitesses de traitement qui correspond à chaque niveau de maladie.

La figure 34 montre le logigramme de la fonction `move-robot` qui s'occupe des déplacements

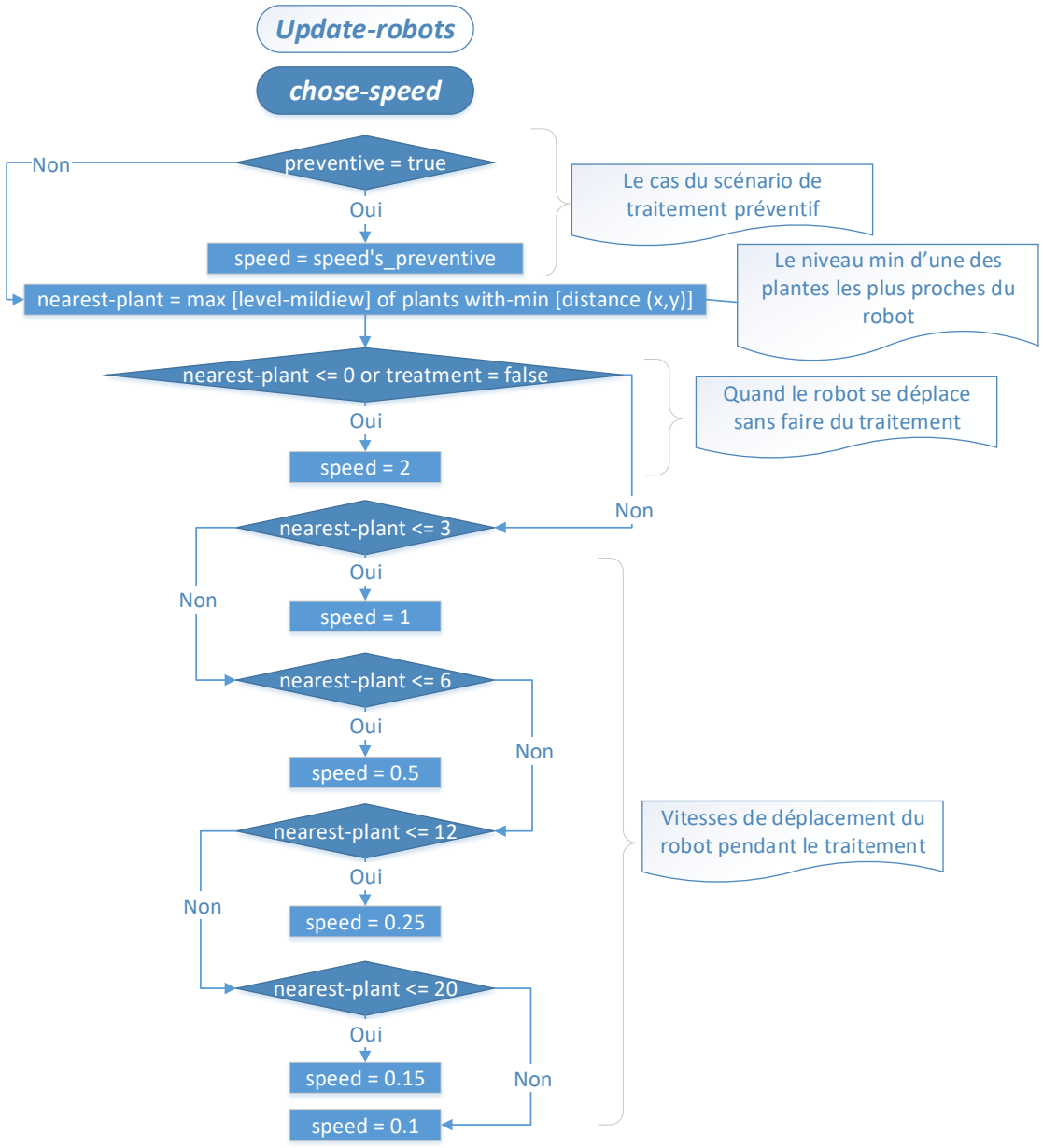

FIGURE 33 : Logigramme de la fonction `chose-speed`

des robots. Le programme commence par vérifier si le robot peut se déplacer avec la condition `move? = true`. Si le robot est très proche de sa destination, il termine son chemin de suite. Dans le cas où sa destination est la station de charge, il accomplit sa mission. Sinon il reste face à sa destination et continue de bouger avec le pas de sa vitesse. La commande `fd = speed` permet de se déplacer avec un pas `fd` chaque `tick`. Pour notre cas le robot se déplacer d'un pas égal à `speed` chaque seconde. Enfin, le programme appelle les fonctions `energy-consumption` et `Update-lamps`.

Dans le simulateur, le robot a plusieurs destinations qui sont soit les rangées des plantes, soit la station de charge. Dans la figure 35, le logigramme explique la fonction `update-destination-robots` qui choisit les destinations des robots. Dans le programme, les destinations sont divisées sur quatre cas. Le premier cas est lorsque le robot traite toutes les rangées qui se trouvent dans sa liste `task-list = []`, alors il doit se retourner vers la station de charge. Le deuxième cas est quand la variable `row-treatment = 0`, alors le robot doit se déplacer vers l'entrée d'une nouvelle rangée pour la traiter. Le troisième cas est si la variable `row-treatment = 1` où le robot doit se déplacer vers le font de la rangée pour effectuer un traitement. Enfin, le quatrième cas est si la variable `row-treatment = 2` dans lequel le robot a terminé son traitement et sa prochaine destination est l'entrée de la rangée traitée. L'entrée de chaque rangée dans la serre est aussi sa sortie pour le robot, car le changement de rangée ne peut se faire que d'un côté de la serre. L'autre côté est réservé à la tuyauterie.

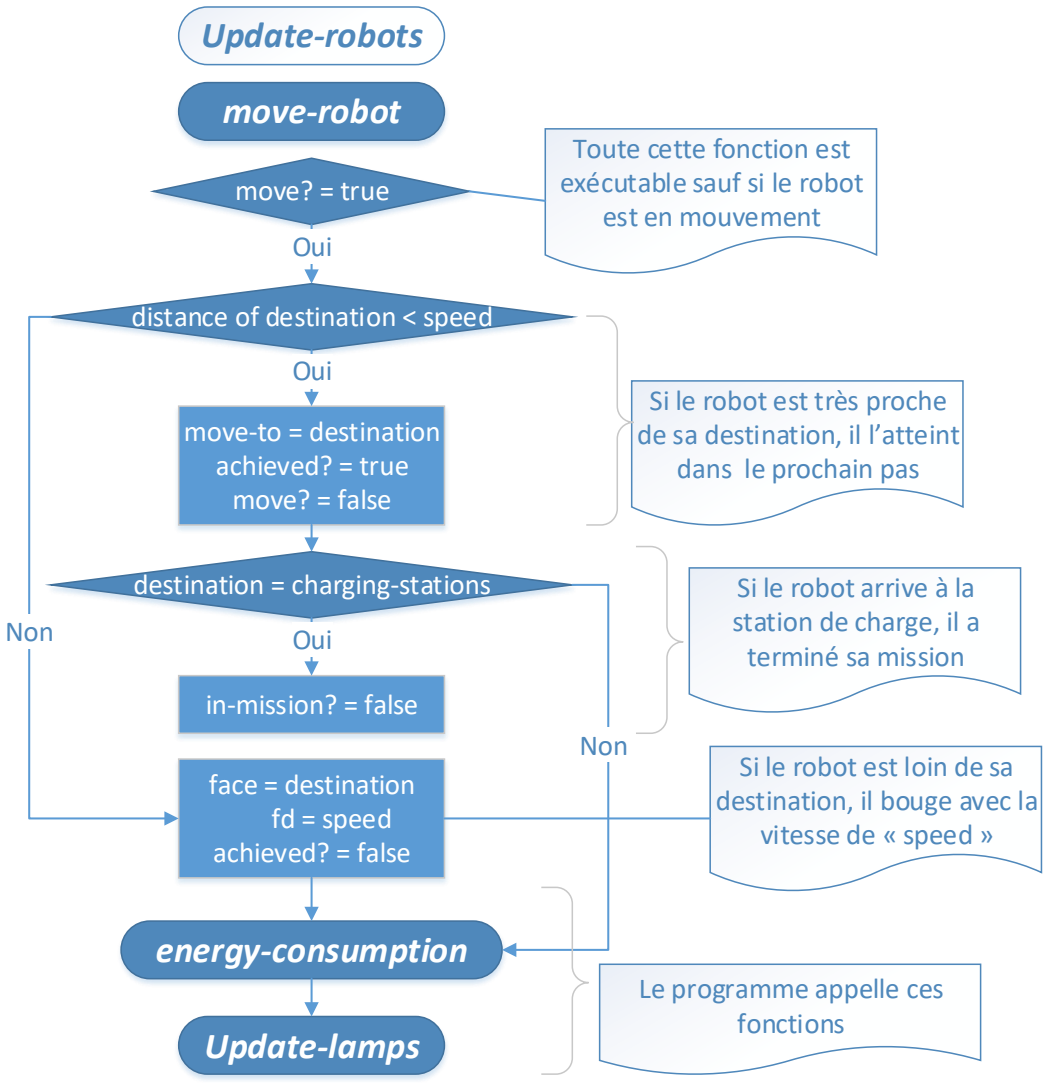

FIGURE 34 : Logigramme du code de déplacement des robots dans le simulateur

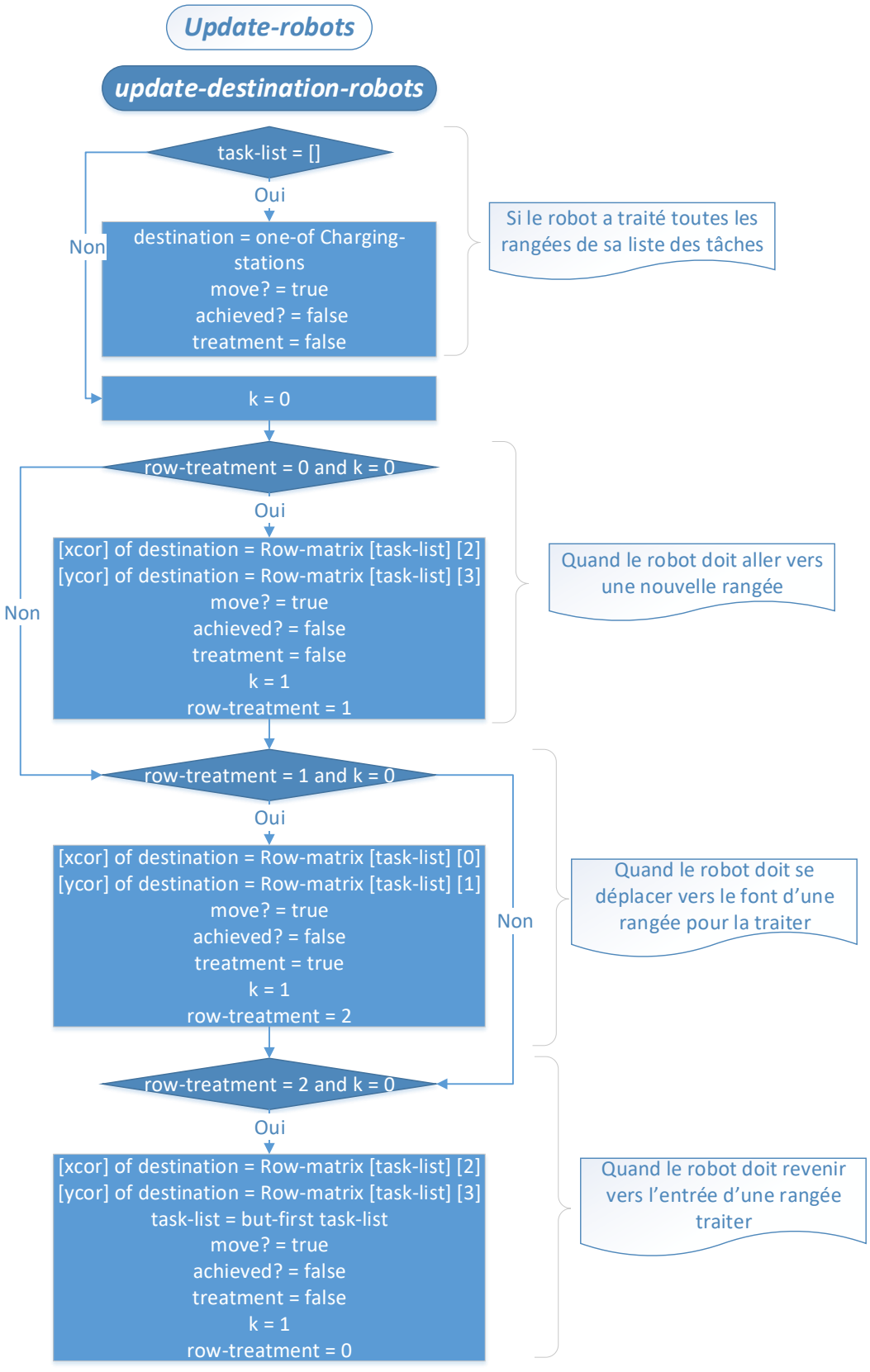

FIGURE 35 : Logigramme de la fonction `update-destination-robots`

La consommation d'énergie de la batterie du robot dépend de sa vitesse et de l'état des lampes, même si les lampes UV-C consomme plus d'énergie que les déplacements de robot. La figure 36 montre le logigramme de la fonction `energy-consumption` qui gère la consommation d'énergie.

La figure 37 montre le logigramme de la fonction `robot-load` qui charge la batterie des robots. Au début de cette fonction, le programme vérifie si les coordonnées du robot correspondent au celles de la station de charge. Puis, le programme met à jour les variables `move? = false` et `achieved? = true` parce que le robot est dans la station de charge. Ensuite le programme vérifie si la batterie est toujours en dessous de 2048 Kw, sinon il ajoute 0.23 Kw chaque seconde pour respecter la charge totale de la batterie en 2h et 30 min. Enfin, si le robot atteint sa charge maximale de 2048 Kw, il est prêt pour exécuter des missions `do-the-treatment? = true`.

Le traitement des plantes se fait grâce à la fonction `treat-plants` qui est montré dans le logigramme de la figure 38. Ce programme commence par vérifier les plantes qui se trouve dans un rayon de 2 mètres par rapport au robot, donc il peut traiter à la fois 2 plantes dans chaque côté. Ensuite, si le robot traite une plante d'un côté, sa variable de traitement deviendra vraie : `treat-left? = true` pour le côté gauche et `treat-right? = true` pour le côte droit.

.1.4 UV-Lamps

La classe d'agent `UV-Lamps` contient deux sous-classes d'agents, une pour les lampes qui sont à gauche du robot, et l'autre pour ceux du côté droit.

Variables des lampes :

Les variables associées au type d'agents `UV-Lamps` sont montré dans la figure 40. La variable binaire `on-off` permet d'allumer ("true") et éteindre ("false") la lampe UV-C si elle est `true`, sinon `false` pour l'éteindre. La variable `in-robot` permet d'identifier le robot de chaque lampe UV-C. La variable `side-lamp` permet de connaître le côté de chaque lampe : côté gauche ou droit du robot. Enfin, la variable `defective-lamp` indique l'état de la lampe : fonctionnelle ou défaillante.

Initialisation des lampes :

La figure 40 représente le processus d'initialisation de l'agent `UV-Lamps`, où l'on retrouve la création de deux lampes en même temps des deux côtés du robot (gauche, droite). Les deux codes sont similaires, les différences entre eux sont juste leurs placements. Après la création d'une lampe avec `create-UV-lamps 1` vient l'identification du robot grâce à la variable `in-robot` qui reçoit l'identifiant du robot. Ensuite nous initialisons les variables `on-off` et `defective-lamp` en `false`. La variable `Side-lamp` est initialisée tout dépend du côté de la lampe. Ensuite, on donne à la lampe les mêmes coordonnées (x,y) que le robot qui la porte. `Shape` pour l'apparence de la lampe dans l'interface de simulation. Après l'initialisation des lampes, le programme appelle l'agent `Robot` pour attribuer les identifiants de chaque lampe.

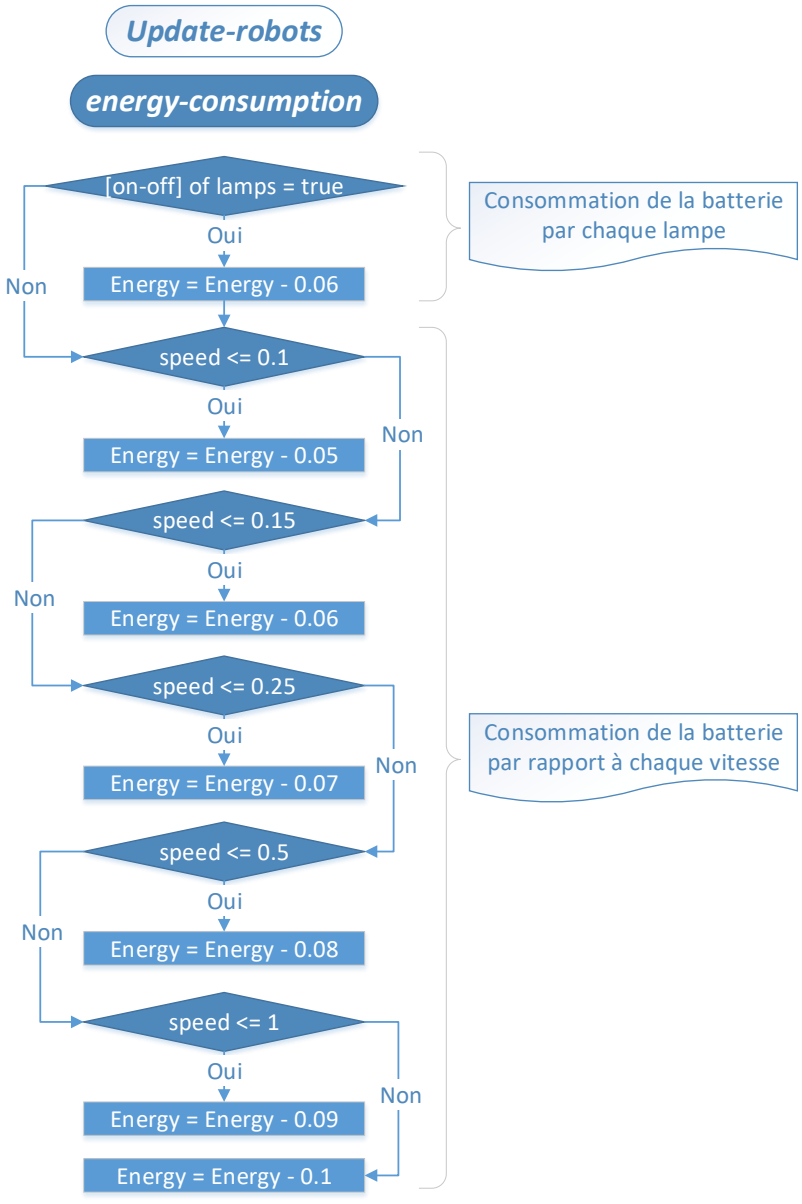

FIGURE 36 : Logigramme de la fonction `energy-consumption`

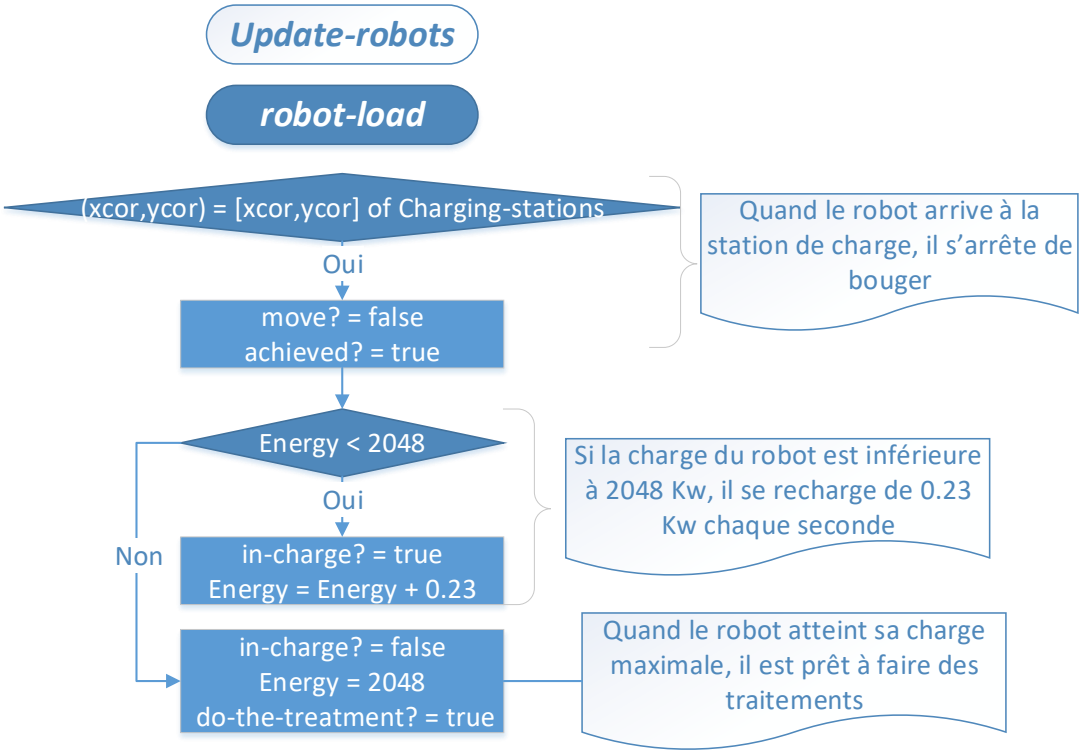

FIGURE 37 : Logigramme de la fonction `robot-load`

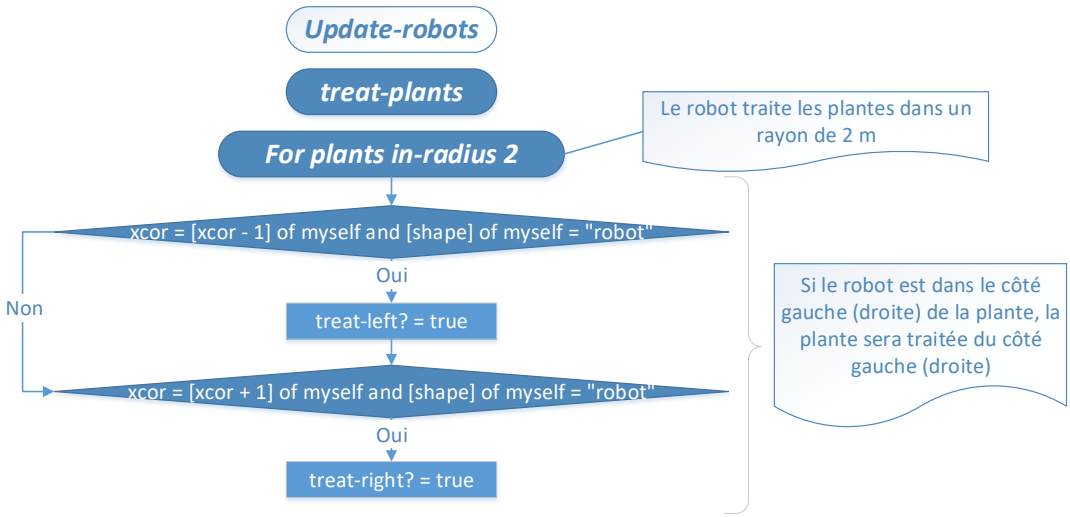

FIGURE 38 : Logigramme de la fonction `treat-plats`

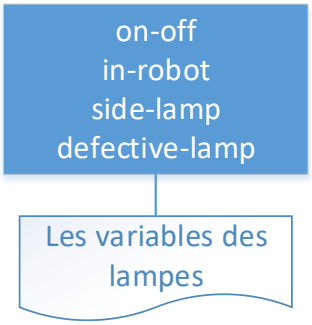

FIGURE 39 : Les variables locales de chaque agent `UV-Lamps`

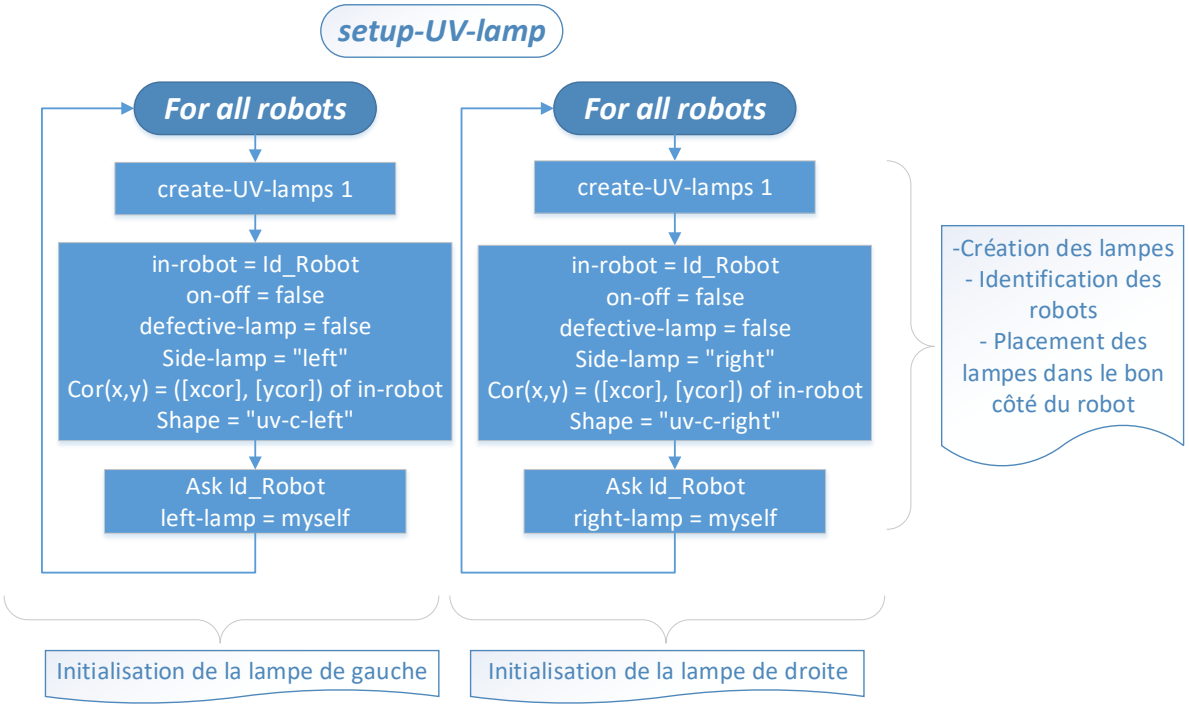

FIGURE 40 : Logigramme du code d'initialisation des lampes

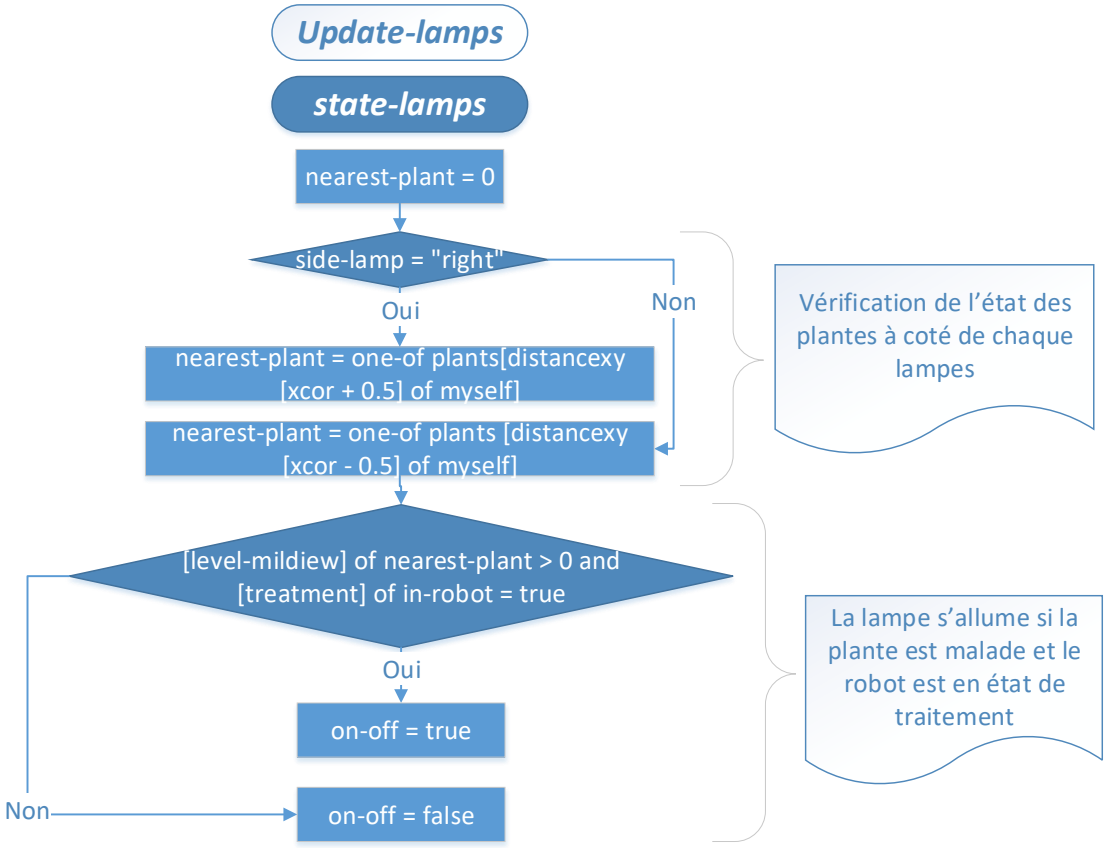

FIGURE 41 : Logigramme de la fonction `state-lamps`

Fonction mise à jour des lampes :

Dans le cas des lampes, le simulateur a besoin de simuler juste leur état (allumé ou éteint). Dans la figure 41 il y a le logigramme de la fonction `state-lamps`, qui est la seule fonction dans `Update-lamps`. Au début le programme de la fonction `state-lamps` initialise une variable `nearest-plant`. Ensuite pour la lampe de droite (gauche) il prend le nombre de plantes dans une distance de $xcor + 0,5$ ($xcor - 0,5$). Enfin il vérifie s'il y a des plantes à côté de chaque lampe pour l'allumer.

.1.5 Agriculteur

Dans cette simulation, l'agriculteur n'a pas beaucoup de tâches, il surveille l'état du robot et le niveau de sa batterie. Si le robot manque d'énergie, l'agriculteur s'en occupe pour le recharger.

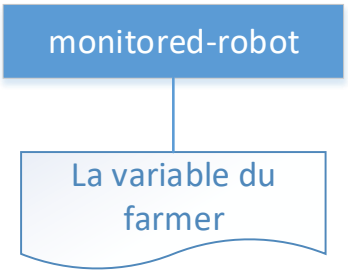

FIGURE 42 : Variable d'agriculteur

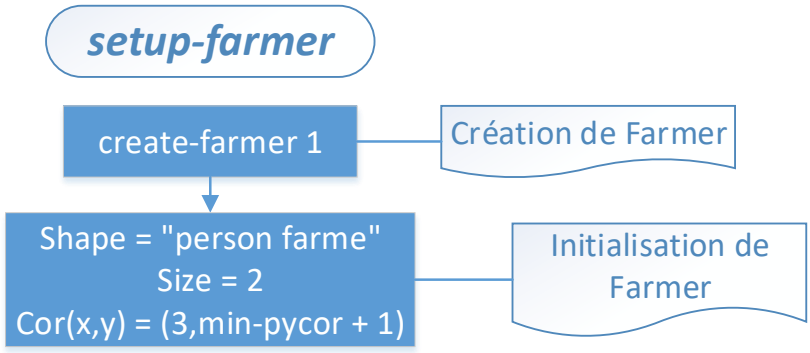

FIGURE 43 : Logigramme du code d'initialisation d'agriculteur

Variables d'agriculteur :

Dans la figure 42 il y a la seule variable de l'agriculteur ou l'agent `Farmer` qui lui permet d'identifier le robot `monitored-robot` .

Initialisation d'agriculteur :

La fonction d'initialisation `create-calendars 1` montré dans la figure 43, elle crée un agriculteur, et le place au milieu de la serre.

Fonction mise à jour d'agriculteur :

La fonction de mise à jour `Update-farmer` de l'agent `farmer` qui est montré dans le logigramme de la figure 44, elle vérifie tous les robots qui se trouve dans la serre à l'aide de la boucle `For all robot` . Puis elle prend au début l'identifiant du robot dans sa variable local `Monitored-robot` . Puis elle vérifie l'énergie du robot, si la batterie du robot est déchargée l'agriculteur prend en charge le robot pour le conduire manuellement vers la station de charge.

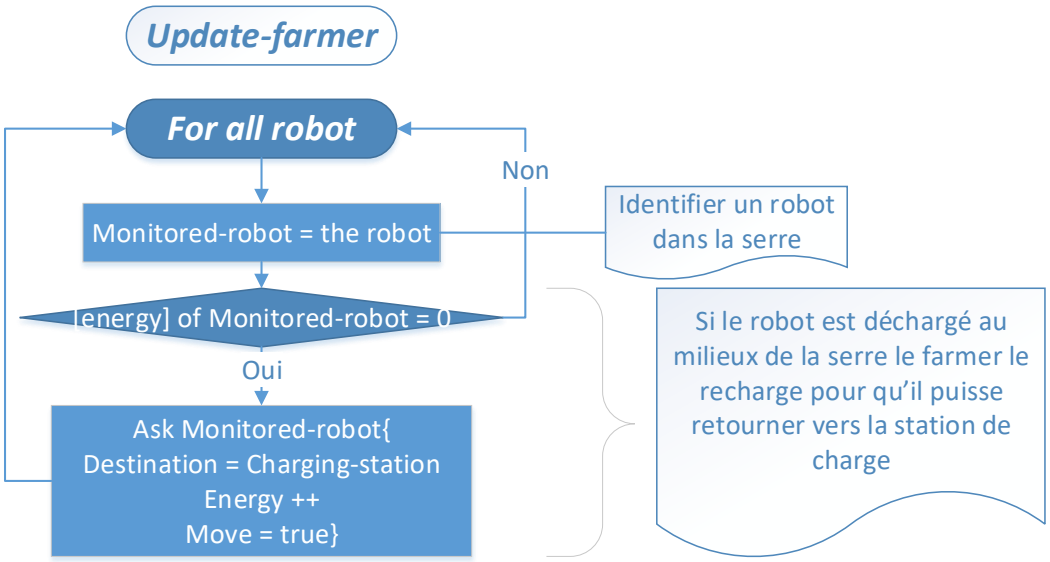

FIGURE 44 : Logigramme du code pour le fonctionnement d'agriculteur

.1.6 Station de charge

La station de charge où l'agent `Charging-station` a que le programme d'initialisation qui est montré dans la figure 45. Le programme d'initialisation de la station de charge crée une station dans la serre grâce à `create Charging stations 1`. Puis il initialise son apparence `shape = electric`, sa taille `size = 1.5` et son emplacement au milieu de la serre `(xcor,ycor ) = middle (xcor,ycor) of greenhouse`. Lorsque le robot atteint le point de la station de charge, sa fonction de charge est activée.

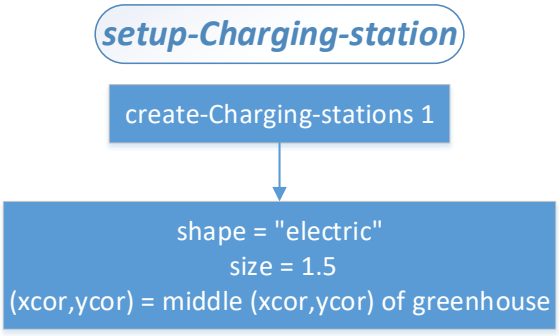

FIGURE 45 : Logigramme du code d'initialisation de la station de charge

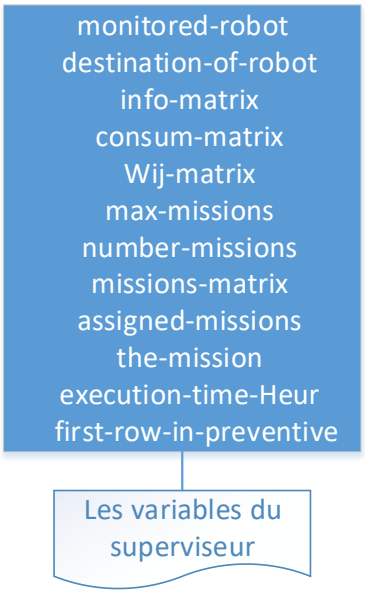

FIGURE 46 : Variables de l'agent `Monitoring`

.1.7 Superviseur

L'agent superviseur où `Monitorings` est un agent qui dirige l'ordonnancement des tâches du robot et permet de suivre l'état des plantes. La planification des tâches sera menée par le superviseur en fonction du type de traitement sélectionné par l'utilisateur. Beaucoup de calcul s'exécute dans l'agent `Monitorings`, car il intègre des algorithmes d'optimisation.

Variables du superviseur :

Les variables de l'agent `Monitorings` se trouvent dans la figure 46. La première variable est `monitored-robot`, qui prend l'identifiant du robot qui sera supervisé. On initialise quatre matrices défini comme suit :

- `info-matrix` par les informations des niveaux de maladie des plantes.
- `consum-matrix` représente l'énergie que le robot doit consommer pour traiter chaque rangée
- `Wij-matrix` représente l'énergie que le robot doit consommer pour effectuer tous les déplacements et les traitements
- `missions-matrix` est utilisée dans l'algorithme génétique qui est présenté dans le chapitre 3.

L'étape suivante consiste à définir la variable booléenne `assigned-missions` qui permet aux algorithmes d'optimisation de programmer des missions si elle est vraie. La variable `execution-time-Heur` permet de calculer le temps d'exécution de chaque algorithme. Enfin, la variable `first-row-in-preventive` est utilisée dans le programme de traitement préventif qui est présenté dans le chapitre 4.

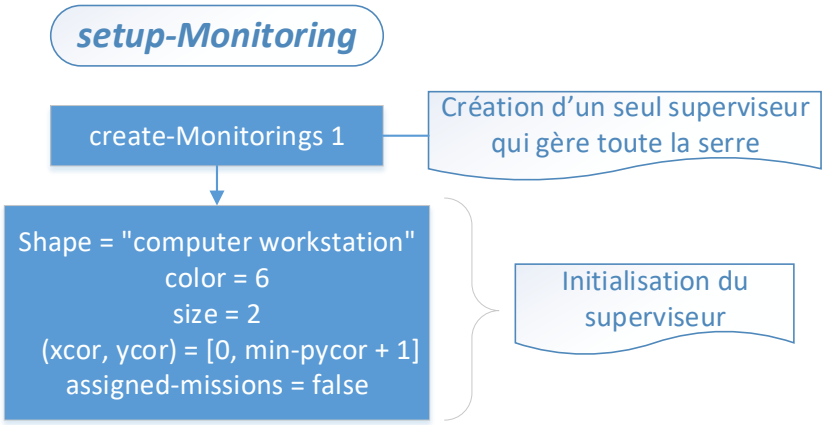

FIGURE 47 : Logigramme du code d'initialisation du superviseur

Initialisation du superviseur :

L'initialisation du superviseur se fait comme le montre le logigramme de la figure 47. Dans un premier temps le programme crée `create-Monitorings 1` un seul superviseur qui sera suffisant pour gérer toute la serre. Ensuite, le programme initialise les variables de cet agent pour choisir son apparence `Shape = computer workstation`, sa couleur `color = 6` et sa taille `size = 2`. Puis, on définit les coordonnées de l'emplacement du superviseur dans le simulateur `(xcor, ycor) = [0, min-pycor + 1]`. Enfin, la variable `assigned-missions = false` qui est fausse au début de la simulation tant qu'il n'y a pas de demande de planification des tâches.

Fonction mise à jour du superviseur :

Une partie seulement des codes développés à l'intérieur de l'agent `Monitorings` est présentée ici. L'autre partie qui contient les algorithmes de planification des tâches est présentée dans le chapitre 3. La première fonction d'exécution appelé par l'agent `Monitorings` est la fonction `robot-monitor`, qui est montré dans la figure 48. Au début de cette boucle le programme prend l'identifiant du robot `monitored-robot = The-robot`. Puis, si le robot identifié est prés de faire des traitements, le programme appelle la fonction `Update-Row-info-matrix` qui prépare la matrice d'informations d'états des plantes. Ensuite, si la première condition est vérifiée et le robot est en mission `in-mission? = true`, le programme appelle la fonction `Update-tasks-of-robot-in-preventive`. Sinon, s'il n'est pas en mission le programme planifie une mission grâce à la fonction `Update-tasks-of-robot`.

Le logigramme de la fonction `Update-Row-info-matrix` est montré dans la figure 49. Son programme commence par initialiser les matrices et les variables d'indice utilisé dans cette fonction. Puis le programme récupère les données sur l'état des plantes en deux étapes. La première étape consiste à récupérer les données des niveaux de maladie sur les plantes en commençant de la première rangée vers la dernière. Cette étape prend au début l'indice de la première plante

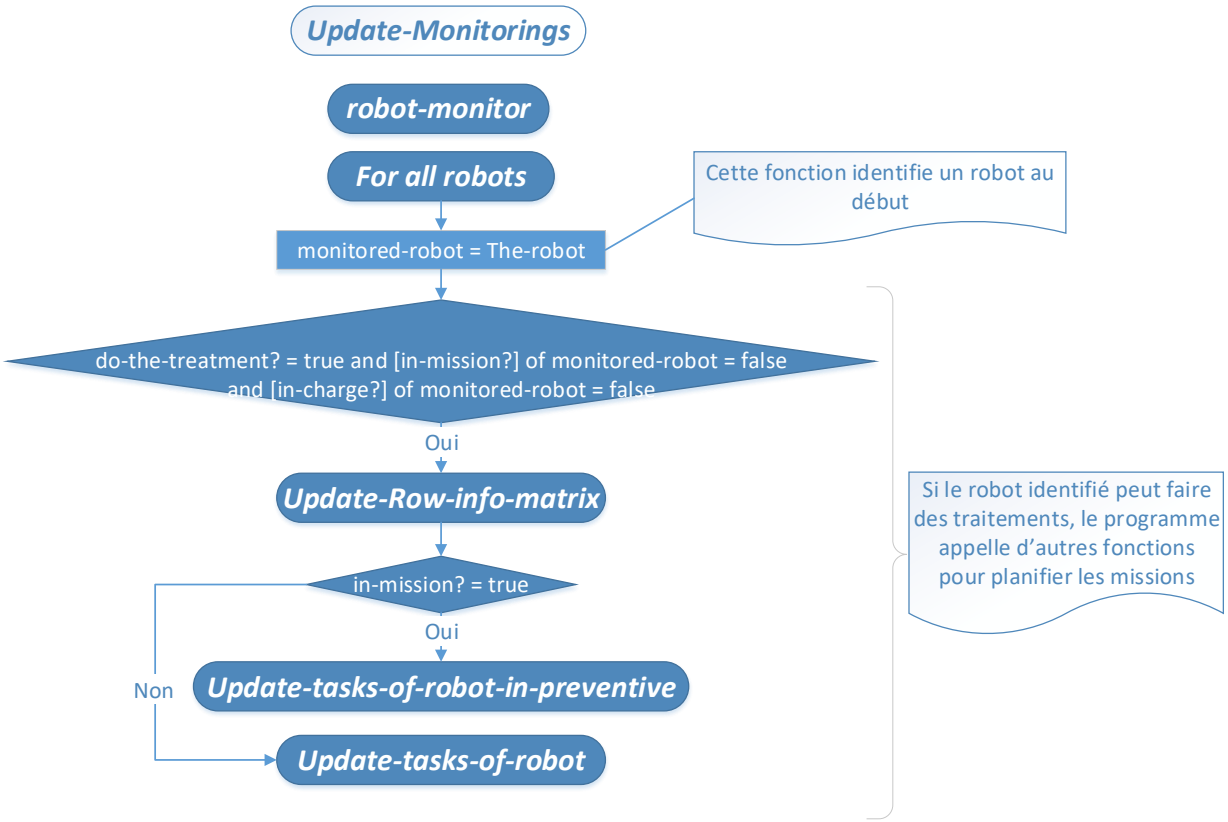

FIGURE 48 : Logigramme de la fonction `robot-monitor`

`first-plant = [who] of plants with-min [who]` , puis le programme parcours toutes les plantes en utilisant la boucle `For i in rang (0, Plants-in-line / 4)` . Cette boucle permet de collecter les informations sur les données qui sont à droite de chaque rangée. Puisque dans une rangée il y a des plantes dans les deux côtés (gauche et droit). Le programme utilise une deuxième étape pour compléter les informations sur les rangées. La deuxième étape est l'inverse de la première où le programme commence par l'indice de la dernière plante `last-plant = [who] of plants with-max [who]` . Puis il utilise la même boucle `For i in rang (0, Plants-in-line / 4)` pour parcourir les rangées dans l'autre sens (de la dernière rangée vers la première). Ensuite il ajoute à chaque fois le niveau de maladie des plantes qui se trouve sur le côté gauche de chaque rangée `info-matrix [k] [j] = info-matrix [k] [j] + [level-mildiew] of plant last-plant` . Enfin, le programme appelle la fonction `Update-Row-consumption-matrix` qui calcule les consommations d'énergie à partir de la matrice d'information `info-matrix` .

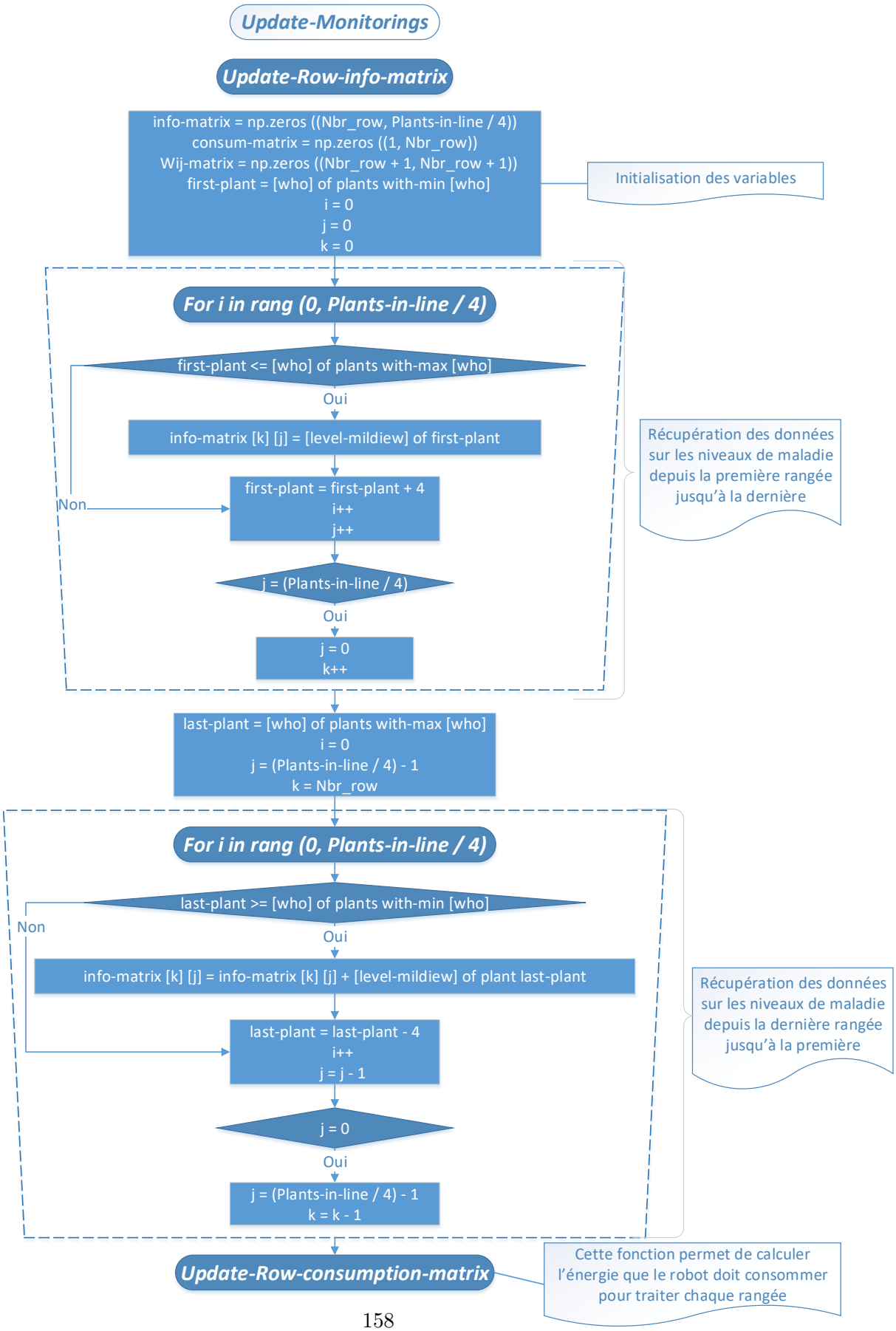

FIGURE 49 : Logigramme de collecte d'informations sur l'état des plantes

La fonction de construction de la matrice des consommations d'énergie se trouve dans le logigramme de la figure 50. Le programme commence par initialiser la variable d'incrémentation `i = 0` et la variable `Sum = 0` pour cumuler les consommations d'énergie de toutes les rangées. Puis à l'aide des boucles `For i in rang (0, Plants-in-line / 4)` et `For j in rang (0, Plants-in-line / 4)` le programme parcours toute la matrice `info-matrix`. Ensuite il cumule les données de chaque rangée avec les consommations d'énergie dans les colonnes de la matrice `consum-matrix [0] [i]`. En plus avant de basculer vers d'autres rangées la fonction ajoute les données précédentes dans la variable `Sum`. Après la construction de la matrice `consum-matrix` le programme vérifie si la somme de consommation égale à zéro `Sum = 0`. Dans ce cas il n'y a pas de maladie dans la serre, le programme ne demande pas de traitement. Sinon le programme fait appel à des algorithmes d'optimisation afin de planifier des missions pour les robots. Enfin le programme fait appel à la fonction `Update-Wij-matrix` qui construit la matrice `Wij`. Cette matrice contient toutes les consommations d'énergie de traitement et de déplacement du robot, la matrice `consum-matrix` est une diagonale de la matrice `Wij`.

La planification des tâches se fait à l'aide de plusieurs algorithmes d'optimisation qui sont présentés dans le chapitre 3. La sélection d'un des algorithmes se fait avec la fonction `Update-tasks-of-robot` qui est montré dans la figure 51. L'utilisateur choisit un algorithme sur l'interface de simulation. Puis, la fonction `Update-tasks-of-robot` appelle un des algorithmes `Genetic-Algorithm`, `Dynamic-Genetic-Algorithm` ou `Heuristic`.

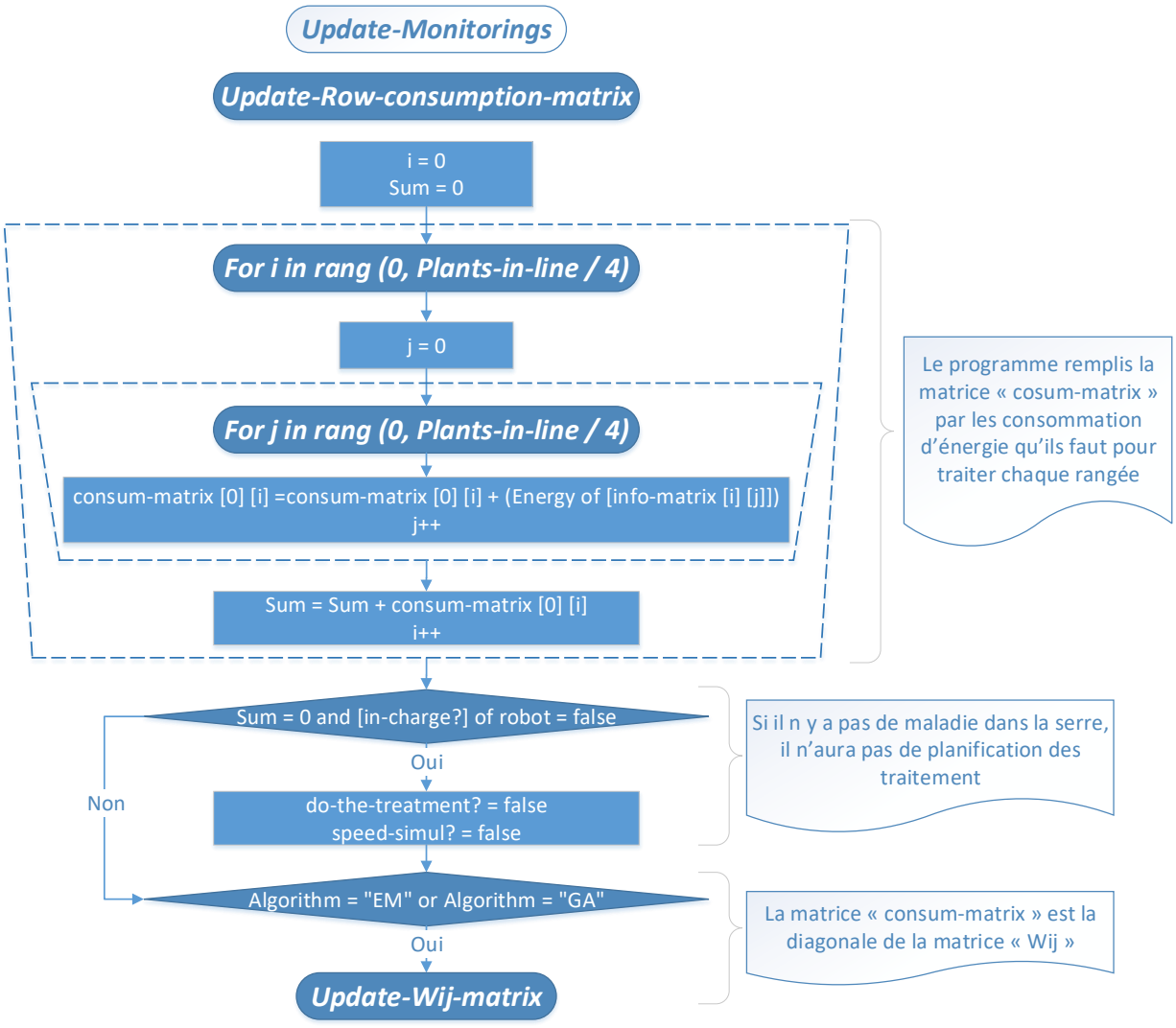

FIGURE 50 : Logigramme de calcul de l'énergie nécessaire pour traiter chaque rangée

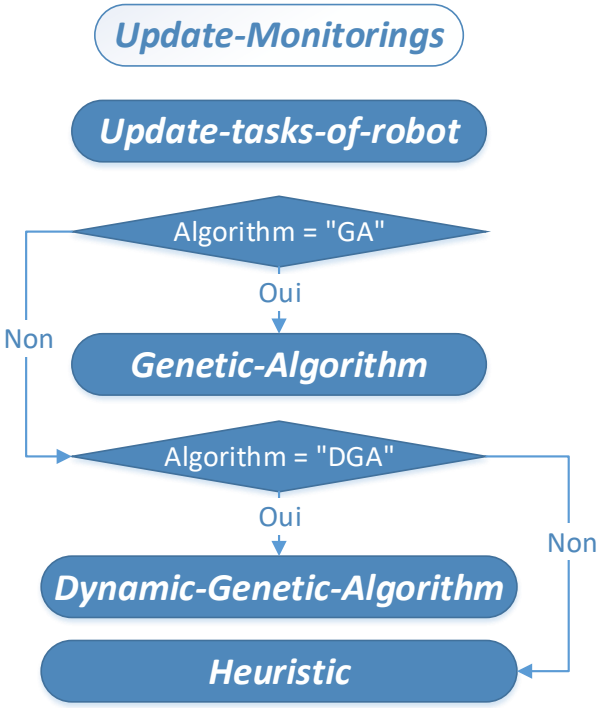

FIGURE 51 : Logigramme du choix algorithme d'ordonnancement

Annexe A

Glossaire

ACO Colonie de fourmis (ant colony optimization).

COVID-19 Coronavirus.

DGA Algorithme génétique dynamique (dynamic genetic algorithm).

EM Méthode exacte (exact method).

E-nose Nez électronique intelligent (smart electronic nose).

GA Algorithme génétique (genetic algorithm).

HA Algorithme heuristique (heuristic algorithm).

IFV Institut Français de la Vigne.

PCR Réaction en chaîne par polymérase (polymerase chain reaction).

SA Recuit simulé (simulated annealing).

SC Systèmes Complexes.

SMA Système Multi-Agents.

UV-C Ultraviolet de type C.

Merouane MAZAR

Simulation et optimisation de la gestion dynamique de tâches évolutives sur des robots mobiles autonomes

Résumé : L'ordonnancement de tâches sur des systèmes stochastiques complexes est une activité qui ne peut être optimisée avec les méthodes d'optimisation classiques. La principale difficulté réside dans le comportement stochastique et dynamique de ces systèmes, qui peut venir modifier les propriétés et/ou le nombre de tâches et/ou les ressources nécessaires à leur exécution. En effet, il est difficile d'estimer le temps nécessaire à l'exécution d'une tâche et définir un ordonnancement efficient qui prend en compte le comportement et l'évolution du système. Cette thèse propose de traiter ce problème en utilisant une approche basée sur le couplage entre simulation et optimisation dans plusieurs situations. Nous avons appliqué cette approche dans le contexte de l'Agriculture 4.0, où nous avons automatisé le traitement robotique de maladie du mildiou en horticulture. Nous avons développé un simulateur basé sur des systèmes multi-agents, où nous avons incorporé des moteurs d'optimisation basés sur des méthodes exactes et approchées. Nous avons modélisé le comportement de la maladie dans une serre en se basant sur le formalisme de chaine de Markov pour chaque agent représentant une plante. Les scénarios de simulation sont basés sur différents types de traitements préventifs (conditionnels, prédictifs et calendaires). Les résultats obtenus ont montré l'efficacité de l'approche et ont permis de proposer aux horticulteurs un outil combinant la simulation et l'optimisation pour les aider à définir et paramétrer la politique de traitement robotique appropriée.
Mots clés : Ordonnancement dynamique, Simulation, Optimisation, Agriculture 4.0, Industrie 4.0, Traitement préventif, Heuristique, Algorithme Génétique, Robot autonome.

Abstract : Task scheduling on complex stochastic systems is an activity that cannot be optimized with classical optimization methods. The main difficulty lies in the stochastic and dynamic behavior of these systems, which can modify the properties and/or the number of tasks and/or the resources required for their execution. Indeed, it is difficult to estimate the time needed to execute a task and to define an efficient scheduling that takes into account the behavior and the evolution of the system. This thesis proposes to solve this problem using an approach based on the coupling between simulation and optimization in several situations. We have applied this approach in the context of Agriculture 4.0, where we have automated the robotic treatment of mildew disease in horticulture. We developed a simulator based on multi-agent systems, where we incorporated optimization engines based on exact and approximate methods. We modeled the disease behavior in a greenhouse based on the Markov chain formalism for each agent representing a plant. The simulation scenarios are based on different types of preventive treatments (conditional, predictive and calendar). The results obtained showed the efficiency of the approach and allowed to propose to horticulturists a tool combining simulation and optimization to help them to define and parameterize the appropriate robotic treatment.
Keywords : Dynamic scheduling, Simulation, Optimization, Agriculture 4.0, Industry 4.0, Preventive treatment, Heuristics, Genetic algorithm, Autonomous robot.