École Centrale Lyon

MSO 3.8
Projet Informatique 2021-2022
Rapport

Optmisation d'une Exploitation Forestière

Élèves :
Arthur Class
Zeinab Mbaye
Enzo Pottez
Antoine Ragusa

Enseignants :
Stéphane Derrode
Alexandre Saidi

28 mars 2022

Table des matières

1 Introduction

Les éléments suivants sont tirés du document *Sujet_projet.pdf* disponible en annexe à la fin de ce rapport.

1.1 Position du problème

La société *Kiriola* cherche à développer un logiciel pour optimiser l'exploitation de ressources forestières. Pour simplifier, on considère qu'une exploitation forestière (d'une superficie typique de 200 hectares) est constituée de parcelles, toutes de la même essence – ici des sapins Douglas, dont la coupe s'effectue au bout de 50 ans – mais avec des âges différents (cf figure ci-contre) :

— 10 hectares ont 3 ans,

— 25 hectares de 30 ans

La distribution des hectares de l'exploitation en fonction de l'âge des arbres a l'allure de la figure ci-dessous. Cette distribution génère des bénéfices très irréguliers dans le temps : certaines générations d'exploitant n'ont que des dépenses, et pas de bénéfices !

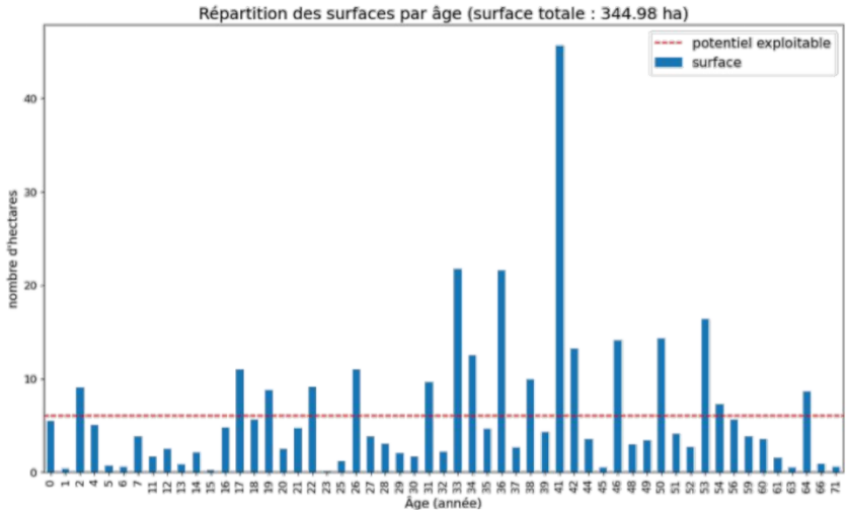


FIGURE 1 – Exemple de distribution des hectares en fonction des âges

Le problème trouve une solution en jouant sur l'âge de coupe à blanc (50 ans à plus ou moins 2 ans). Une réorganisation des parcelles de l'exploitation est proposée pour assurer un bénéfice constant (et rapide), quelle que soit l'année. A terme, la distribution des âges devrait suivre la courbe rouge en pointillés (voir fig. 1). Il s'agit donc d'un problème d'optimisation traité par des outils mathématiques/algorithmiques.

1.2 L'existant et les objectifs du projet

Suite aux collaborations avec l'ECL à travers de stages, une première version de ce logiciel permet de proposer une optimisation de la coupe pour une seule essence d'arbre. Pour le sujet actuel :

- Reprendre l'outil actuel (qui procède à une optimisation individuelle de la coupe d'arbres),l'appréhender et l'améliorer pour prendre en compte plusieurs essences d'arbre ;
- Mettre en place un outil de visualisation pour une présentation synthétique des parcelles ainsi que des prédictions (la planification future des coupes).
- Le cas échéant, faire des propositions pour mettre en place une optimisation "globale" (prise en compte de plusieurs paramètres importants).

2 Travail prévisionnel et répartition des tâches

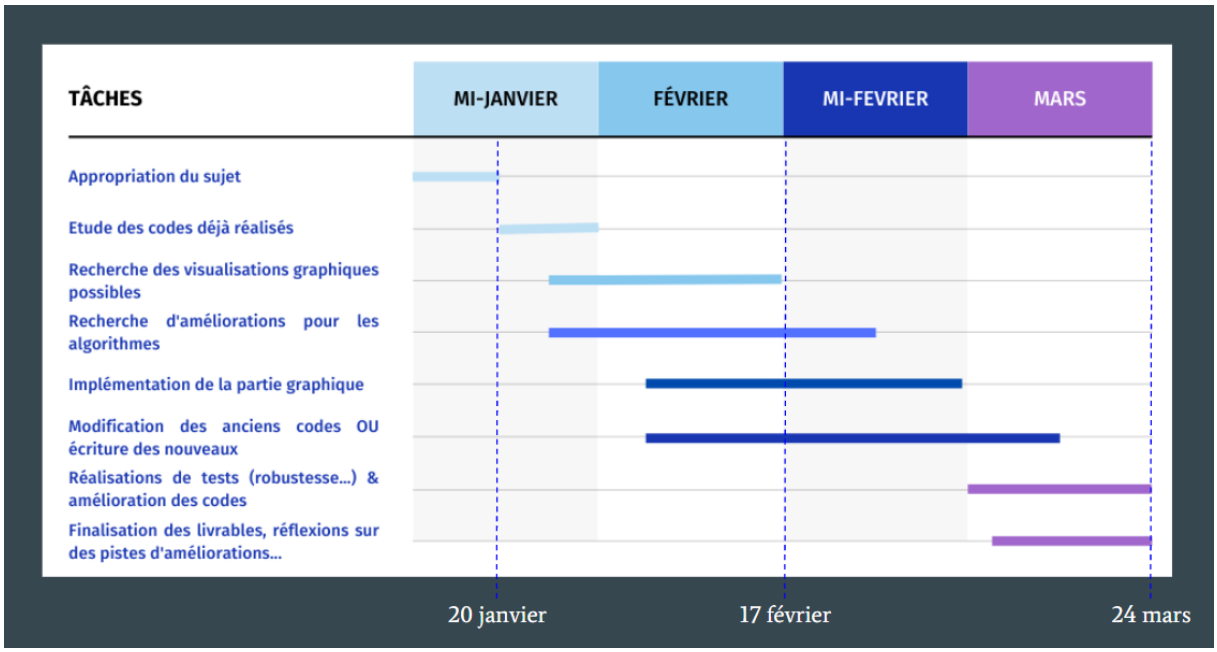


FIGURE 2 – Diagramme prévisionnel établi durant le début de notre travail

Le diagramme de Gant ci-dessus représente la prévision du travail à effectuer que nous avons établi en janvier après notre rendez-vous avec nos tuteurs Stéphane Derrode et Alexandre Saidi. Nous avons en réalité assez bien respecté ce planning : le mois de janvier a été consacré à une découverte et une appropriation du projet. Nous nous sommes ensuite réparti les tâches principales : amélioration des différents algorithmes et recherche de nouvelles solutions (pour Zeinab), clarification du code et facilitation d'utilisation de celui-ci (pour Antoine), développement d'un outil de visualisation (pour Enzo), et recherches de méthodes pour "encapsuler" le projet et faciliter la mise en place de l'environnement virtuel nécessaire à l'éxécution des algorithmes (pour Arthur).
Nous avons alors chacun travaillé sur nos parties respectives jusqu'à début mars, avant de rassembler tous nos travaux pour pouvoir tout tester et finaliser avant de penser à de nouvelles pistes d'amélioration.

3 Solutions d'optimisation

3.1 Solveur CVXOPT

Dans cette partie, nous cherchons à optimiser les algorithmes existants. Pour cela, nous nous sommes orienté vers le solveur CVXOPT. Ce dernier est un progiciel gratuit pour l'optimisation convexe basé sur le langage de programmation Python. Il peut être utilisé avec l'interpréteur Python interactif, sur la ligne de commande en exécutant des scripts Python, ou intégré dans d'autres logiciels via des modules d'extension Python.

Son objectif principal est de simplifier le développement de logiciels pour les applications d'optimisation convexe en s'appuyant sur la vaste bibliothèque standard de Python et sur les points forts de Python en tant que langage de programmation de haut niveau.

Le problème soulevé était le fait qu'au bout d'un certain nombre d'années, l'optimisation devenait impossible. En effet, on ne pouvait plus satisfaire la contrainte « Fonction objectif à minimiser > 0 » car il n'y a pas assez de surface de forêt disponible à la coupe à cause des déséquilibres dans la pyramide des âges de départ.

Nous cherchons donc à résoudre le problème d'optimisation sous contraintes en minimisant l'expression ci-après :

$$\sum_{i=50}^{54} y_i x_i - Moyenne$$

Avec :
- y_i : la surface d'arbres d'âge i
- x_i : la proportion de coupe d'arbre d'âge i (entre 0 et 1)
- $Moyenne$: valeur égale au potentiel cab (coupe à blanc)

Et avec les contraintes suivantes :
- la fonction objectif à minimiser est positive
- $x_{54} = 1$ afin d'être sûr de ne pas avoir de bois de plus de 54 ans (car dans notre cas $age\_cab = 52\,ans$ et $Delta = 2$

Nous avons ci-dessus les résultats du modèle 3, en utilisant le solveur scipy.optimize pour l'un et le solveur cvxopt pour l'autre :

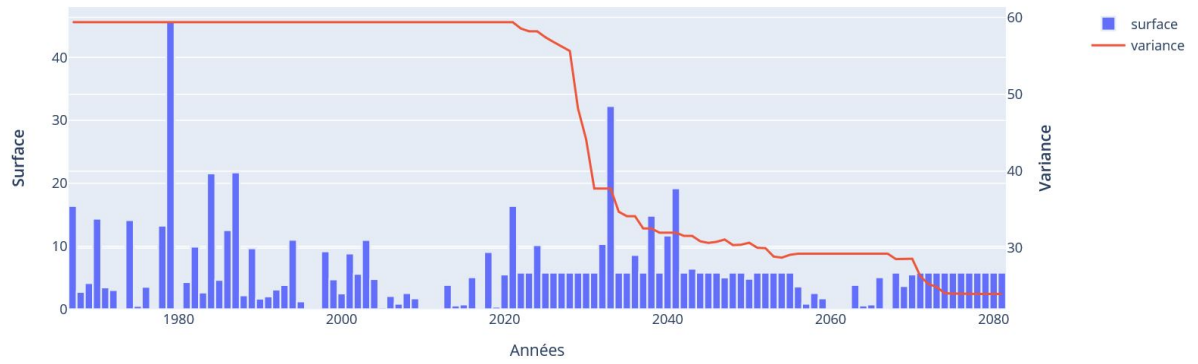


FIGURE 3 – Resultat avec le solveur de base

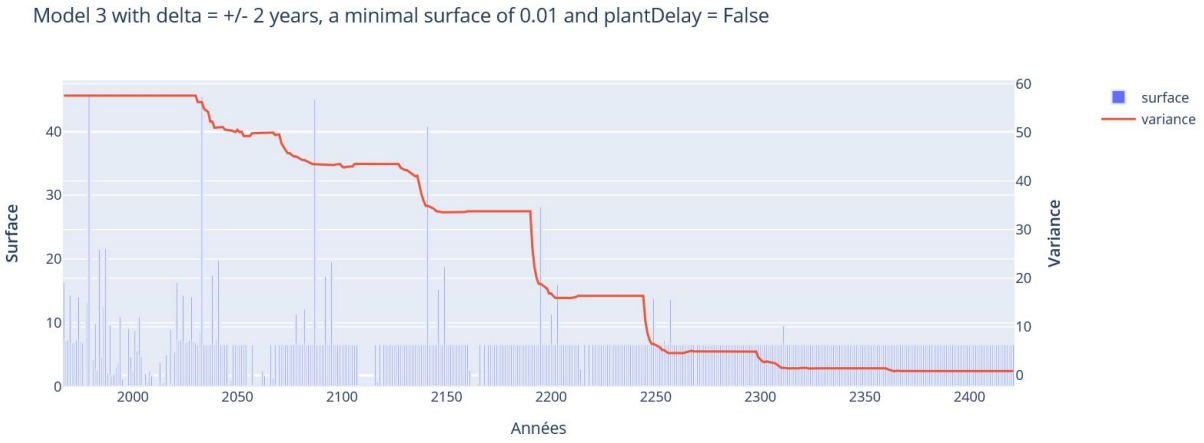


FIGURE 4 – Resultat avec le solveur CVXOPT

En comparant ces 2 graphes et en se limitant à l'année 2080, on constate que la variance du solveur baisse plus rapidement avec scipy.optimize. Par exemple, en 2070, la variance a une valeur à peu près égale à 28.61 en considérant le solveur scipy.optimize et à 45 avec le solveur CVXOPT. Ces résultats nous ont confortés dans l'idée que le solveur CVXOPT n'est pas forcément adapté sur le court-terme.

Cependant, sur le long terme (année 2400 ou plus), la répartition est quasi-uniforme en utilisant le solveur CVXOPT.

3.2 Génération de nouveaux sets de données

Certains modèles peuvent être particulièrement adaptés sur un set de données, mais ne pas fonctionner sur d'autres. Cela est comparable au phénomène de "sur-apprentissage" qui a pour effet de fausser les résultats lorsqu'on soumet les modèles à de nouveaux sets de données.
L'objectif sera de générer de nouveaux sets de données et d'effectuer des tests afin de voir le comportement des modèles avec ces nouvelles données.

Pour ce faire, nous avons rédigé un script python (generatedata.py) qui nous permet de générer des données et de les stocker dans un fichier json.

Les données générées ne sont pas celles contenues dans les données fournies par le propriétaire mais celles pris en entrée par les modèles à savoir l'année et la surface à couper correspondante.

La complexité de cette partie repose sur le fait que les modèles ne prennent pas tout le temps les mêmes données.

Après cette étape, vient la phase de test qui n'a malheureusement pas abouti à des résultats concluants que l'on pourrait vous présenter dans ce rapport.

4 Corrections et simplifications d'utilisation

4.1 Objectifs

Dans l'objectif de rendre la simulation utilisable professionnellement par l'exploitation forestière, nous avons voulu la clarifier et en faciliter l'utilisation. Le premier obstacle à cette utilisation professionnelle était la difficulté de lancement. En effet, pour chaque simulation il fallait lancer un environnement virtuel via plusieurs commandes, puis écrire une série de commandes pour la simulation. Selon la version du système d'exploitation de la machine de l'utilisateur, les commandes ne sont pas forcément les mêmes pour lancer l'environnement virtuel. Nous avons chacun rencontré différentes difficultés pour le lancer car suivre les instructions données ne fonctionnait pas forcément. Dans l'objectif de pouvoir lancer les simulations directement à partir de l'outil de visualisation, nous avons créée un script permettant d'automatiser toute ces étapes. Par ailleurs certaines simulations comportaient des erreurs, qui en empêchaient le lancement.

4.2 Réalisation

Nous avons donc dans un premier temps corrigé l'ensemble des simulations. Les erreurs provenaient la plupart du temps de confusions entre les variables d'entrée des différents composants, sans doute dues au nombre important de paramètres. Nous avons ensuite écrit un script pouvant être appelé depuis la visualisation avec les différents paramètres de la simulation, qui permet de d'afficher la commande complète pour lancer la simulation. On peut ensuite en afficher le résultat directement depuis le dashboard.

5 Développement d'un outil de visualisation

5.1 Intérêt de l'outil

Plusieurs choses qui nous ont posé problème en découvrant le projet :
— L'installation de l'environnement virtuel Python ;
— L'exécution des modèles (pour tester de nouveaux ; paramètres par exemple)
— La visualisation des résultats déjà existants.

Pour l'installation de l'environnement virtuel et l'exécution des modèles, un script a été rédigé pour faciliter leurs lancements (voir partie précédente). Nous avons donc souhaité l'implémenter dans l'outil de visualisation (qui fait donc un peu plus que de la visualisation).

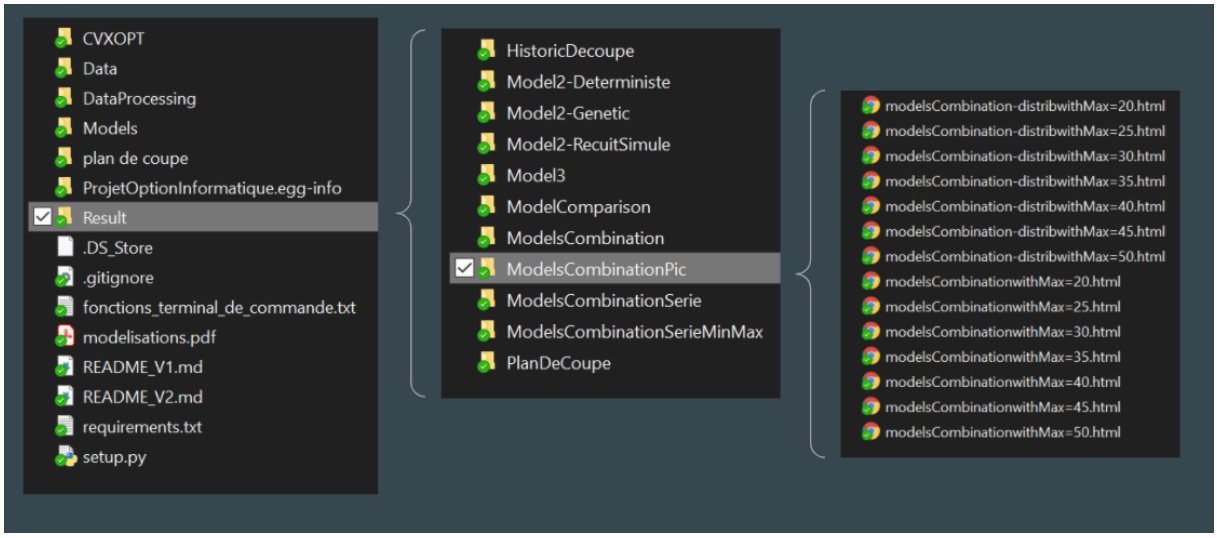


FIGURE 5 – Aperçu de l'arborescence pour accéder aux résultats des modèles

La visualisation des résultats était assez compliquée car les fichiers sont stockés sous différents formats (texte, json ou html) en fonction des modèles. Il n'existait donc pas un format de sauvegarde "universel". De plus, il était nécessaire d'explorer les dossiers et sous-dossiers du projet pour trouver les résultats qui nous intéressent (voir figure 5). Nous souhaitions donc regrouper les résultats dans un même outil de visualisation.

5.2 Choix technologiques

Nous avons choisi de réaliser un **dashboard** comme outil de visualisation. Il y a plusieurs raisons qui nous ont motivés à réaliser un dashboard :
— Possibilité de répondre aux besoins explicités (affichage simple des résultats, lancement de nouveaux tests...) ;
— Les graphiques réalisés lors des projets précédents utilisent la librairie Python Plotly, qui est conçue pour être facilement dans un dashboard.

Le dashboard a été réalisé avec la librairie Python Dash. Cette librairie permet de réaliser des dashboards sous Python et elle est conçue pour être parfaitement compatible

avec Plotly (librairie pour générer les graphiques). Cette solution est celle qui nous a semblé la plus naturelle et la plus facile à implémenter.

D'un point de vue plus technique, Dash et Plotly permettent de générer des fichiers en html / javascript / css en ne codant qu'une partie en Python. L'exécution se fait via un navigateur web car l'application fait tourner un serveur local. On obtient alors des graphiques dynamiques et une qualité affichage très bonne. Nous avons tenté de structurer le code du dashboard de façon claire, de sorte que son contenu soit compréhensible a posteriori (pour facililiter le travail d'un éventuel futur groupe de projet...) :

- Le dossier *assets* contient tous les logos / images utilisés dans le dashboard ;
- Le dossier *components* contient les différentes fonctions utilisés dans le dashboard ;
- Le dossier *data* contient les résultats des simulations sous format json (nous avons d'ailleurs dû modifié les codes des différents modèles pour s'assurer que le format et l'emplacement de sauvegardes étaient corrects) ;
- Le dossier *pages* contient les codes qui structurent les pages du dashboard ;
- Dans le reste du dossier, on trouve un fichier requirements.txt qui contient les librairies à installer, le fichier callback.py qui assure l'interactivité du dashboard, le fichier index qui sert à lancer le serveur...

Dash permet aussi d'avoir un outil de debug de l'application (disponibilité du serveur, erreurs, callbacks qui ne fonctionnent pas...).

5.3 Fonctionnalités du dashboard

La dashboard propose plusieurs fonctionnalités, réparties en 3 pages :

- Une page d'accueil qui présente le projet, le dashboard et ses fonctionnalités
- Une page *dashboard* (figure 6) qui permet de visualiser les résultats disponibles dans le dossier *data*. On dispose d'un menu déroulant pour sélectionner le modèle, et deux items pour sélectionner l'affichage souhaité (soit la surface d'arbres à couper et la variance, soit la répartition des surfaces coupés par âge de coupe).
- Une page *calcul* (figure 7) qui permet de sélectionner le modèle et ses paramètres et de générer le script à entrer dans un terminal de commande. Il est possible de suivre l'avancement du calcul dans le terminal (une des raisons pour laquelle nous avons choisi cette solution). Les résultats sont ensuite stockés dans le dossier *data*.

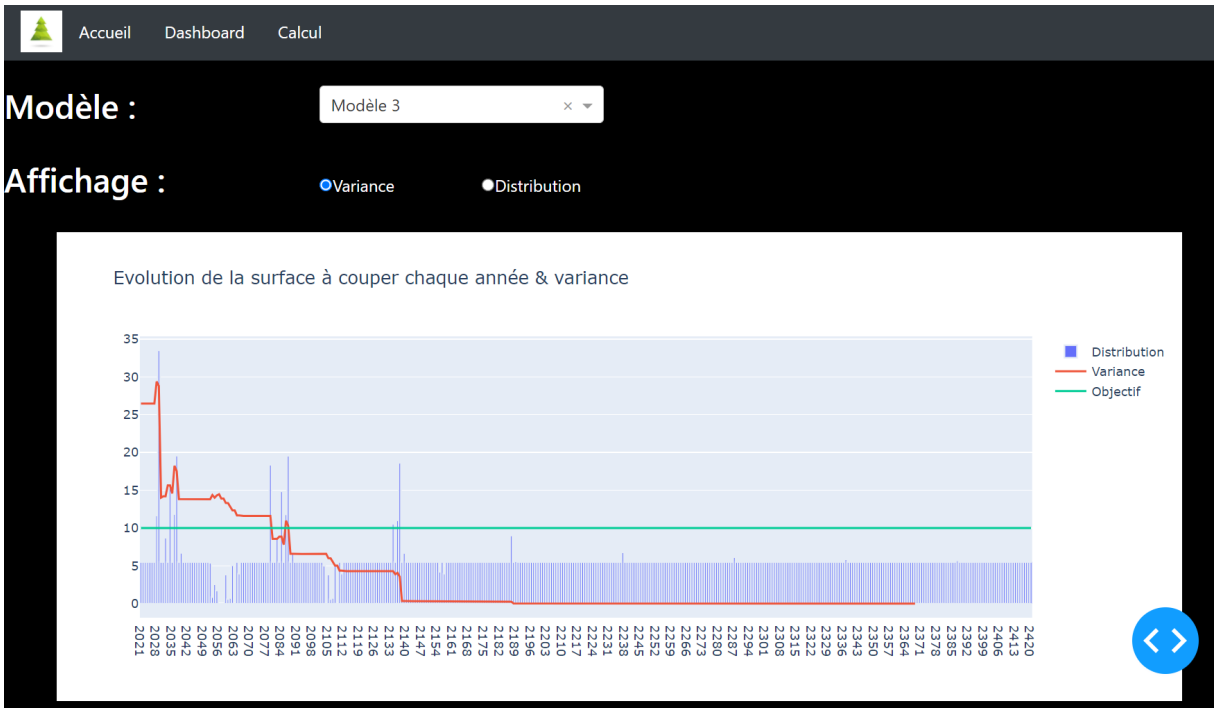


FIGURE 6 – Page "Dashboard" de l'outil de visualisation

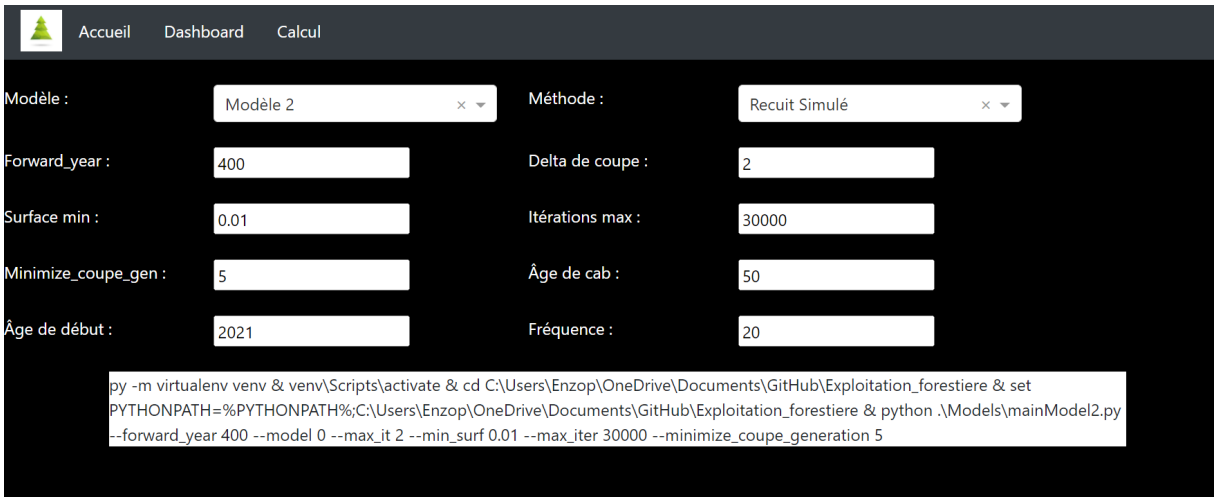


FIGURE 7 – Page "Calcul" de l'outil de visualisation

6 Tentative d'encapsulation du projet avec *Docker*

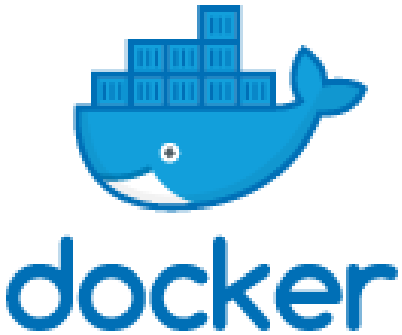


FIGURE 8 – Le logo de l'application *Docker*

Une autre piste de simplification pour l'installation et de l'exécution des fichiers du projet a été explorée : l'utilisation de l'application *Docker*. Elle fournit une solution alternative pour mettre en place un déploiement efficace d'une application, de manière adaptable, sur n'importe quel serveur. "Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur" (*Wikipédia*).
Ainsi Docker nous a paru être une bonne solution pour éviter des problèmes liés à la mise en place "manuelle" d'un environnement virtuel permettant l'exécution des différents scripts et algorithmes d'optimisation.Au lieu de mettre en place un environnement virtuel, il faut procéder au chargement d'une image *Docker*.Il s'agit d'un package qui inclut tout ce qui est nécessaire à l'exécution d'une application, à savoir : le code, l'exécution, les variables d'environnement, les bibliothèques, les fichiers de configuration. La perspective de tout mettre en place automatiquement nous a paru très intéressante car elle permettrait à n'importe qui d'utiliser nos programmes, même à des personnes peu familières avec l'informatique comme par exemple les responsables de l'exploitation forestière.

Cependant après plusieurs essais nous sommes parvenus à la conclusion que *Docker* allait résoudre certains problèmes tout en en créant d'autres. En effet la création du script permettant de centraliser les différents algorithmes et de lancer automatiquement l'environnement virtuel via la bibliothèque python *system* est une solution pour l'instant assez convenable, qui nécessite seulement d'avoir un environnement python installé sur sa machine. De toute manière l'utilisation de *Docker* implique d'installer un logiciel sur sa machine et d'exécuter plusieurs lignes de code, donc le problème de rendre le plus accessible possible nos programmes à n'importe quelle personne désireuse d'observer rapidement leurs effets n'aurait pas été résolu.

7 Conclusion

Les livrables que nous avions à rendre pour ce projet étaient assez souples. Nous avons défini, lors d'une réunion de lancement avec nos tuteurs A. Saidi et S. Derrode, des directions clés et des pistes d'amélioration. Il était ensuite nécessaire de s'approprier le projet et le code réalisé par les élèves ayant déjà travaillé sur le projet. Nous avons ensuite réfléchi aux améliorations possibles pour le projet.

Tout d'abord, la phase d'appropriation a été assez longue. Lors de celle-ci, nous avons rencontré plusieurs bugs qu'il a été nécessaire de régler. Cela venait par exemple du fait que les installations de Python sur les différentes machines étaient différentes. Nous avons donc trouvé des solutions pour permettre de faire tourner les codes de façon plus "universelle". Nous avons également dû modifier légèrement les codes Python définissant les modèles, qui ne fonctionnaient parfois pas.

L'outil de visualisation graphique (dashboard) est quant à lui fonctionnel. Il permet effectivement de lancer l'exécution des modèles et de visualiser les résultats plus facilement. Il reste cependant largement améliorable : tous les modèles ne sont pas disponibles pour la visualisation (les "pics" notamment) car les fichiers générés sont différents des autres modèles ; il est possible d'y ajouter des fonctionnalités (comparaison des graphes, meilleure gestion des résultats sauvegardés...).

Nous n'avons pas mis en place de solution permettant de prendre en compte plusieurs essences d'arbres. Les datasets utilisés étant assez complexes, il est difficile d'en générer de nouveaux. Il pourrait être intéressant de s'en procurer d'autres auprès des exploitants forestiers.

De manière générale, nous avons trouvé assez difficile de s'approprier un projet créé par d'autres étudiants. Nous avons rencontré plusieurs bugs qui ont freiné notre avancement. De plus, le fait de ne pas avoir de livrables bien définis mais plutôt des pistes d'amélioration était assez perturbant.

8 Annexes

Projet 6 : Optimisation d'une exploitation forestière

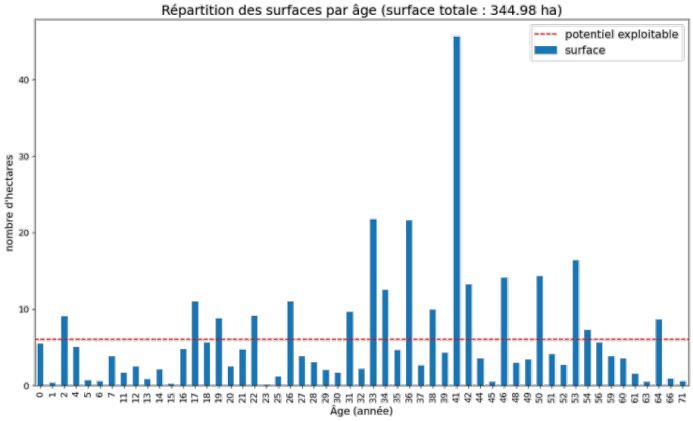
Commanditaire éventuel : Société Kiriola (Charles de Rambuteau)

Rattachement administratif
Département : MI **Équipe d'enseignement :** Informatique

Tuteur(s) : Alexandre Saidi et Stéphane Derrode

Position du problème :
La société Kiriola cherche à développer un logiciel pour optimiser l'exploitation de ressources forestières.
Pour simplifier, on considère qu'une exploitation forestière (d'une superficie typique de 200 hectares) est constituée de parcelles, toutes de la même essence --ici des Sapin Douglas, dont la coupe s'effectue au bout de 50 ans--, mais avec des âges différents (cf figure ci-contre) :
- 10 hectares ont 3 ans,
- 25 hectares de 30 ans…

La distribution des hectares de l'exploitation en fonction de l'âge des arbres a l'allure de la figure ci-dessous. Cette distribution génère des bénéfices très irréguliers dans le temps : certaines générations d'exploitant n'ont que des dépenses, et pas de bénéfices !



Le problème trouve une solution en jouant sur l'âge de coupe à blanc (50 ans à + ou – 4 ans). Une réorganisation des parcelles de l'exploitation est proposée pour assurer un bénéfice constant (et rapide), quel que soit l'année. A terme, la distribution des âges devrait suivre la courbe rouge en pointillés (voir fig. de gauche).
Il s'agit donc d'un problème d'optimisation traité par des outils Mathématiques / algorithmiques.
L'existant : suite aux collaborations avec l'ECL à travers de stages, une première version de ce logiciel permet de proposer une optimisation de la coupe pour une seule essence d'arbre. Pour le sujet actuel :
- Reprendre l'outil actuel (qui procède à une optimisation individuelle de la coupe d'arbres), l'appréhender et l'améliorer pour prendre en compte plusieurs essences d'arbre;
- Mettre en place un outil de visualisation pour une présentation synthétique des parcelles ainsi que des prédictions (la planification future des coupes).
- Le cas échéant, faire des propositions pour mettre en place une optimisation "globale" (prise en compte de plusieurs paramètres importants).

Objectifs de réalisation : Logiciel x rapport x objet
Objectifs généraux et / ou pédagogiques :
Il s'agit de concevoir et programmer une solution optimale pour résoudre le problème énoncé ci-dessus.

Nature principale du travail :
Il s'agira d'une part de modéliser le problème avec des variables pertinentes, puis de programmer la solution dans un solveur, et enfin tester cette solution à partir d'un jeu de données réelles
.
Moyens mis à disposition pour la réalisation du travail :
Pas de moyen particulier nécessaire pour la réalisation du travail.