

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ DE TOURS  
Département Informatique  
64 avenue Jean Portalis  
37200 Tours, France  
Tél. +33 (0)2 47 36 14 14  
[polytech.univ-tours.fr](http://polytech.univ-tours.fr)

**Projet Recherche & Développement**  
**2019-2020**

# **Apports du Machine Learning pour l'ordonnancement des applications dans les Cloud data centers**



**Tuteur académique**  
**Boukhalfa ZAHOUT**

**Étudiant**  
**Martin DE LA FUENTE (DI5)**

11 avril 2020



# Liste des intervenants

Nom	Email	Qualité
Martin DE LA FUENTE	<a href="mailto:martin.delafuente@etu.univ-tours.fr">martin.delafuente@etu.univ-tours.fr</a>	Étudiant DI5
Boukhalfa ZAHOUT	<a href="mailto:boukhalfa.zahout@univ-tours.fr">boukhalfa.zahout@univ-tours.fr</a>	Tuteur académique, Département Informatique



# Avertissement

Ce document a été rédigé par Martin De La Fuente susnommé l'auteur.

L'Ecole Polytechnique de l'Université de Tours est représentée par Boukhalfa Zahout susnommé le tuteur académique.

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

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

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

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

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

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

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



## Pour citer ce document

Martin De La Fuente, *Apports du Machine Learning pour l'ordonnancement des applications dans les Cloud data centers*, Projet Recherche & Développement, Ecole Polytechnique de l'Université de Tours, Tours, France, 2019-2020.

```
@mastersthesis{
  author={De La Fuente, Martin},
  title={Apports du Machine Learning pour l'ordonnancement des applications dans les
    Cloud data centers},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université de Tours},
  address={Tours, France},
  year={2019-2020}
}
```

# Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	vi
<b>1 Introduction</b>	<b>1</b>
1 Contexte de la réalisation.....	1
1.1 Description du problème.....	1
1.2 Objectifs .....	3
1.3 Hypothèses.....	3
1.4 Bases méthodologiques.....	4
<b>2 Description générale</b>	<b>5</b>
1 Environnement du projet .....	5
2 Caractéristiques des utilisateurs .....	5
3 Fonctionnalités du système .....	5
4 Structure générale du système .....	6
<b>3 Etat de l'art</b>	<b>7</b>
1 Machine Learning : définition.....	7
2 Techniques de Machine Learning .....	8
2.1 Régression linéaire.....	8
2.2 Régression logistique .....	9

2.3	Régression lasso .....	10
3	Architectures de Deep Learning : réseaux de neurones .....	10
3.1	Single Layer Perceptron (SLP) .....	10
3.2	Recurrent Neural Network (RNN) .....	11
3.3	Long Short-Term Memory (LSTM) .....	11
3.4	Sequence-to-sequence (seq2seq) .....	12
<b>4</b>	<b>Analyse et conception</b> .....	<b>13</b>
1	Méthodes de résolution exactes .....	13
1.1	Modèle linéaire en nombres entiers 1 (PLNE1) .....	13
1.1.1	Fonction objectif .....	13
1.1.2	Contraintes .....	13
1.1.3	Variables.....	14
1.1.4	Performances .....	14
1.2	Modèle linéaire en nombres entiers 2 (PLNE2) .....	14
1.2.1	Fonction objectif .....	14
1.2.2	Contraintes .....	15
1.2.3	Variables.....	15
1.2.4	Performances .....	15
1.3	Méthode de décomposition (branch and price) .....	15
2	Apport du Machine Learning.....	16
2.1	Architecture du système.....	16
2.1.1	Phase de prédiction .....	16
2.1.2	Phase d'apprentissage.....	17
2.2	Analyse des techniques retenues.....	19
2.2.1	Régression linéaire.....	19
2.2.2	Régression logistique.....	19
2.2.3	Perceptron .....	19
2.2.4	Sequence-to-sequence .....	20
3	Conception du système .....	20
3.1	Structure du programme PLNE .....	20
3.2	Structure du programme seq2seq .....	21
3.2.1	Phase d'apprentissage.....	21
3.2.2	Phase de prédiction .....	21
<b>5</b>	<b>Mise en oeuvre</b> .....	<b>23</b>
1	Compréhension d'un modèle seq2seq pour la traduction.....	23
1.1	Dataset utilisé .....	23
1.2	Préparation du dataset .....	24
1.3	Entraînement du modèle .....	25

1.4	Traduction.....	26
2	Implémentation du modèle seq2seq pour la prédiction d'une solution .....	26
2.1	Structure du fichier dataset .....	26
2.2	Préparation du dataset .....	27
2.2.1	Codage ASCII .....	28
2.2.2	Codage hexadécimal.....	29
2.2.3	Codage personnalisé.....	29
2.3	Entraînement du modèle .....	30
2.4	Prédiction.....	31
2.5	Résultats obtenus .....	32
3	Implémentation d'un autre modèle : seq2point .....	32
3.1	Résultats obtenus .....	32
<b>6</b>	<b>Bilan et conclusion</b> .....	<b>34</b>
1	Travail réalisé.....	34
2	Travail restant/amélioration .....	34
3	Bilan sur la planification du S9.....	35
4	Bilan sur la planification du S10.....	35
5	Bilan sur la qualité .....	36
6	Bilan auto-critique sur la gestion de projet .....	36
7	Conclusion générale .....	36
	<b>Annexes</b> .....	<b>38</b>
<b>A</b>	<b>Gestion de projet</b> .....	<b>39</b>
1	Découpage en tâches .....	39
1.1	Tâche 1 : gestion et versioning du projet.....	40
1.2	Tâche 2 : compréhension du sujet .....	40
1.3	Tâche 3 : prototypage PLNE1 .....	40
1.4	Tâche 4 : prototypage PLNE2.....	41
1.5	Tâche 5 : rédaction du cahier de spécifications.....	41
1.6	Tâche 6 : rédaction de l'état de l'art .....	41
1.7	Tâche 7 : faisabilité des méthodes de Machine Learning .....	41
1.8	Tâche 8 : préparation de la soutenance mi-parcours .....	41
1.9	Tâche 9 : rédaction du rapport final mi-parcours.....	41
1.10	Tâche 10 : création du fichier d'entrée du programme de Machine Learning	41
1.11	Tâche 11 : implémentation du programme de Machine Learning : seq2seq.	42
1.12	Tâche 12 : tests fonctionnels/unitaires et documentation du code.....	42
1.13	Tâche 13 : Documentation utilisateur/développeur/installation .....	42

1.14	Tâche 14 : préparation de la soutenance finale.....	42
1.15	Tâche 15 : Rédaction du rapport final.....	42
2	Plannings du S9 .....	42
2.1	Planning prévisionnel du S9 .....	42
2.2	Planning réel du S9.....	43
3	Plannings du S10 .....	43
3.1	Planning prévisionnel du S10 .....	43
3.2	Planning réel du S10.....	43
<b>B</b>	<b>Description des interfaces externes du logiciel</b>	<b>44</b>
1	Interfaces matériel/logiciel .....	44
2	Interfaces homme/machine .....	44
3	Interfaces logiciel/logiciel .....	44
<b>C</b>	<b>Spécifications fonctionnelles</b>	<b>45</b>
1	Définition de la fonction 1 : importInstance .....	45
2	Définition de la fonction 2 : exportResultatsCSV .....	45
3	Définition de la fonction 3 : apprentissage .....	45
4	Définition de la fonction 4 : prediction .....	46
5	Définition de la fonction 5 : exportSolutionInitiale .....	46
6	Définition de la fonction 6 : exportSolutionOptimale.....	46
<b>D</b>	<b>Spécifications non fonctionnelles</b>	<b>47</b>
1	Contraintes de développement et conception .....	47
2	Contraintes de fonctionnement et d'exploitation .....	47
2.1	Performances.....	47
2.2	Capacités.....	48
2.3	Modes de fonctionnement .....	48
2.4	Contrôlabilité.....	48
2.5	Sécurité .....	48
2.6	Intégrité.....	48
<b>E</b>	<b>Documentation d'installation</b>	<b>49</b>
<b>F</b>	<b>Documentation utilisateur</b>	<b>51</b>
1	Structure d'un fichier d'instance .....	51
2	Programme PLNE.....	52
3	Fichier résultats (dataset) .....	53
4	Programme sequence-to-sequence (seq2seq).....	53



<b>G</b>	<b>Documentation développeur</b>	<b>55</b>
1	Présentation générale .....	55
1.1	Fonctionnalités du système .....	55
1.2	Architecture globale du système .....	56
1.2.1	Phase de prédiction .....	56
1.2.2	Phase d'apprentissage.....	56
2	Conception.....	57
2.1	Arborescence du projet.....	57
2.2	Structure du script PLNE .....	58
2.3	Structure du programme seq2seq .....	59
2.3.1	Phase d'apprentissage.....	59
2.3.2	Phase de prédiction .....	60
<b>H</b>	<b>Cahier de tests</b>	<b>62</b>
<b>I</b>	<b>Glossaire</b>	<b>66</b>
<b>J</b>	<b>Webographie</b>	<b>67</b>
<b>K</b>	<b>Sources</b>	<b>68</b>

# Table des figures

## 1 Introduction

1	Données du problème .....	2
2	Ordonnancement des jobs sur les machines .....	3

## 2 Description générale

1	Diagramme de cas d'utilisation .....	6
2	Structure générale du système .....	6

## 3 Etat de l'art

1	Concepts de l'intelligence Artificielle.....	8
2	Exemple de régression linéaire .....	9
3	Représentation d'une régression logistique.....	10
4	Structure d'un perceptron.....	10
5	Architectures RNN .....	11
6	Modèle sequence-to-sequence.....	12

## 4 Analyse et conception

1	Méthode par décomposition.....	16
2	Phase de prédiction .....	17
3	Exécution du programme PLNE1 .....	18
4	Phase d'apprentissage.....	18
5	Diagramme de composants du PLNE .....	20
6	Diagramme de composants du seq2seq en phase d'apprentissage .....	21
7	Diagramme de composants du seq2seq en phase de prédiction .....	22

**5 Mise en oeuvre**

1	Dataset utilisé .....	24
2	Dictionnaires créés pour une phrase et sa traduction .....	24
3	Table ASCII .....	28
4	Affichage en console lors de l'entraînement du modèle partie 1 .....	31
5	Affichage en console lors de l'entraînement du modèle partie 2 .....	31

**A Gestion de projet**

1	Capture d'écran du Trello du projet.....	40
2	Diagramme de GANTT prévisionnel S9 .....	42
3	Diagramme de GANTT réel S9 .....	43
4	Diagramme de GANTT prévisionnel S10 .....	43
5	Diagramme de GANTT réel S10 .....	43

**E Documentation d'installation**

1	Configuration de PyCharm utilisée pour le projet .....	49
---	--	----

**F Documentation utilisateur**

1	Structure d'un fichier d'instance .....	51
2	Affichage en console lors de l'exécution du programme PLNE .....	52
3	Structure du fichier dataset (1ère partie).....	53
4	Structure du fichier dataset (2ème partie).....	53
5	Affichage d'une prédiction en console suite à l'exécution du programme seq2seq .	54

**G Documentation développeur**

1	Diagramme de cas d'utilisation .....	55
2	Architecture générale du système lors de la phase de prédiction .....	56
3	Architecture générale du système lors de la phase d'apprentissage .....	57
4	Arborescence du projet.....	58
5	Diagramme de composants du programme PLNE .....	59
6	Diagramme de composants du programme seq2seq en phase d'apprentissage .....	60
7	Diagramme de composants du programme seq2seq en phase de prédiction .....	61

# 1

## Introduction

Ce document constitue les spécifications du projet de recherche et développement (PRD) intitulé « Apports du Machine Learning pour l'ordonnancement des applications dans les Cloud data centers géographiquement distribués ». Il a pour but de rappeler le contexte et les objectifs du projet, puis de présenter les principaux éléments d'analyse et de définir un périmètre à respecter concernant l'architecture du système ainsi que les fonctionnalités à développer.

Ce projet est réalisé par Martin De La Fuente, étudiant en 5ème année d'école d'ingénieur à Polytech Tours qui représente la MOE. L'encadrant est Boukhalfa Zahout, qui représente la MOA.

## 1 Contexte de la réalisation

Aujourd'hui, le modèle du cloud que nous abordons est géographiquement distribué dans des data centers. C'est pourquoi nous utilisons le terme « Cloud data centers géographiquement distribués ». Ces Data centers sont composés de machines (clusters de machines), et permettent de stocker des données ou encore de réaliser des opérations qui nécessitent de grandes puissances de calcul. Nous pouvons distinguer d'un côté le fournisseur du cloud distribué et de l'autre les clients. Ces clients sont des utilisateurs qui souhaitent soumettre leurs applications à exécuter au sein de ces data centers.

Ces data centers possèdent différentes ressources (CPU, RAM, stockage, ...). Ces ressources peuvent être réutilisables. De plus ces ressources sont limitées car elles possèdent une capacité totale. Enfin, ces ressources peuvent être partagées entre les différents utilisateurs.

La forte demande en puissance de calcul par les utilisateurs a nécessité de trouver des méthodes d'ordonnancement des applications (jobs) sur les machines du cloud pour satisfaire le maximum de demandes tout en respectant les capacités maximales des machines ainsi que les contraintes spécifiées par les utilisateurs comme par exemple la date de début/fin d'exécution des jobs ou encore leurs besoins en ressources. Ces méthodes d'optimisation ont pour but d'aider le fournisseur à garantir la meilleure qualité de service.

### 1.1 Description du problème

Nous nous intéressons dans ce projet à un problème d'ordonnancement de type mono-critère et multi-machines. C'est-à-dire que l'on cherche à optimiser un seul critère (une seule fonction

objectif), et que nous travaillons sur  $m$  machines parallèles.

De plus, nous avons des utilisateurs qui envoient leurs applications ( $n$  jobs) à exécuter. Un job peut être ordonnancé sur une machine, ou bien rejeté. Il prend alors la valeur 0 ou 1. Les jobs sont indépendants entre eux.

Chaque job est caractérisé par :

- Sa date de début fixée :  $s_j$
- Sa date de fin fixée :  $f_j$
- Sa consommation (c'est-à-dire son besoin en ressource) :  $r_{ij}$
- Son poids qui détermine sa priorité :  $w_{ij}$

Remarque : Les poids sur les jobs peuvent être attribués de manière arbitraire, ou bien en fonction des clients.

Les caractéristiques des jobs sont des données qui sont envoyées par les utilisateurs, ce n'est donc pas à nous de les déterminer.

L'objectif de ce problème d'ordonnancement est de maximiser la valeur totale des jobs pondérés. C'est ce que l'on appelle la fonction objectif.

### Exemple

Voici un exemple simple pour illustrer le problème. Nous allons essayer d'ordonnancer 8 jobs ( $n = 8$ ) à exécuter sur 2 machines ( $m = 2$ ), et nous travaillerons avec 3 ressources qui sont : CPU, RAM, stockage.

Nous retrouvons donc dans le tableau suivant toutes les données fournies qui caractérisent les jobs (identifiant, date de début, date de fin, besoin en ressource CPU, RAM, stockage, poids) :

Job	Début	Fin	Poids	CPU	RAM	Stock
1	0	6	8	25	50	25
2	1	4	5	12	60	30
3	3	8	3	50	30	70
4	3	6	1	50	70	25
5	4	8	6	12	25	20
6	5	9	7	25	50	30
7	0	4	4	50	30	40
8	1	5	2	25	80	60

Figure 1 – Données du problème

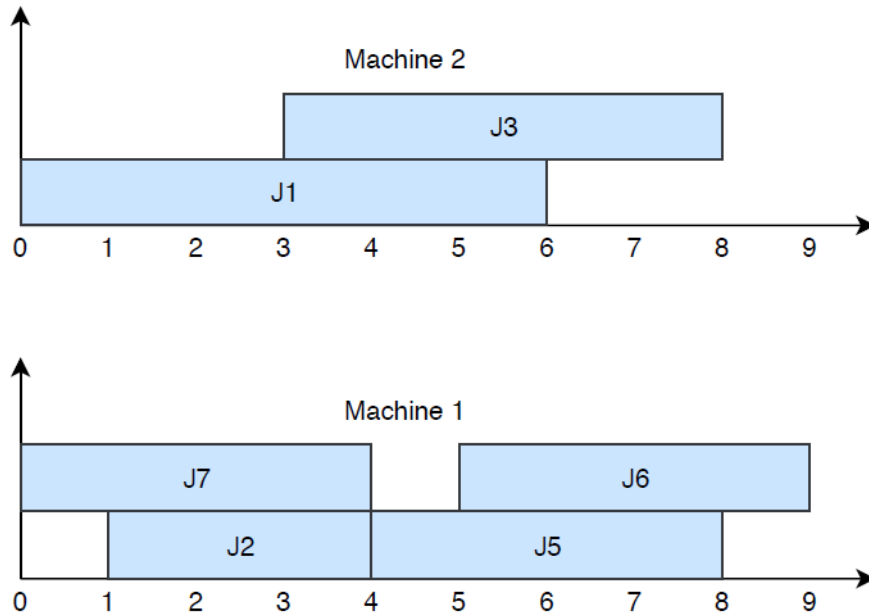
Enfin, la capacité totale disponible pour chaque ressource sur les machines est de 100.

La résolution de cet exemple nous permet d'obtenir le résultat suivant :

On peut constater que sur la machine 1, les jobs 2, 5, 6 et 7 ont été ordonnancés. Sur la machine 2, les jobs 1 et 3 ont été ordonnancés.

Les jobs 4 et 8 ont été rejetés.

La valeur de la solution optimale est de  $Z = 33$ . Cette valeur correspond simplement à la somme des poids des jobs qui ont été ordonnancés sur les 2 machines.



**Figure 2** – Ordonnancement des jobs sur les machines

Ce problème a été prouvé comme NP-difficile (Zahout et al, MISTA2017). C'est-à-dire qu'il ne peut pas être résolu en temps polynomial.

## 1.2 Objectifs

Les objectifs de ce projet sont multiples. Dans un premier temps, il est nécessaire de comprendre le contexte du problème à traiter en réalisant des prototypes sur des méthodes de résolution exactes (modèles PLNE). Puis l'implémentation d'un ou plusieurs algorithmes d'apprentissage est attendue. Ce ou ces algorithmes serviront à retourner des solutions initiales pour les méthodes exactes déjà développées (par exemple : une méthode par décomposition).

## 1.3 Hypothèses

Actuellement, la plupart des travaux sur les problèmes d'ordonnancement sont liés à l'implémentation de méthodes de résolution approchées et exactes. L'apport du Machine Learning dans ce problème d'optimisation rend ce projet totalement expérimental. Nous ne sommes pas certains d'atteindre l'objectif, à savoir d'obtenir un algorithme d'apprentissage capable de retourner des solutions initiales. Cela constitue donc un risque important dans ce projet.

Le manque de formation et d'expérience dans le domaine du Machine Learning peut entraîner un mauvais choix d'algorithme d'apprentissage et un retard dans le projet. Il faudra donc au préalable avoir sélectionné plusieurs alternatives qui puissent être mises en place rapidement. C'est pourquoi les discussions et mises en commun des recherches entre l'encadrant et l'étudiant sont primordiales pour converger vers des solutions viables et réalisables.

## 1.4 Bases méthodologiques

Une gestion de projet agile sera adoptée au cours du projet. Il y aura un échange minimum par semaine avec l'encadrant afin d'échanger sur les aspects de conception et de développement. Un diagramme de GANTT prévisionnel sera aussi établi afin de pouvoir visualiser les différents sprints mais aussi les dates de rendus de livrables, et les respecter. L'outil de gestion de projet Trello sera également utilisé pour observer les fonctionnalités « à faire, en cours, terminé ». Je rédigerai de plus un compte-rendu personnel du travail réalisé chaque semaine. Enfin, un outil de versioning Git sera utilisé pour le développement.

# 2

## Description générale

### 1 Environnement du projet

Il existe déjà une implémentation en C++ d'une méthode de résolution exacte (modèle PLNE) qui utilise le solveur CPLEX (logiciel informatique d'optimisation) pour calculer la solution optimale. Pour comprendre cette méthode exacte, une implémentation sera réalisée en Python (en utilisant une librairie CPLEX) au début du projet. Le projet ne dépend donc pas d'un environnement spécifique au niveau matériel ou logiciel.

### 2 Caractéristiques des utilisateurs

Un seul type d'utilisateur est prévu pour le système, La MOA (Boukhalfa Zahout). Il possède de solides connaissances en informatique et plus particulièrement en recherche opérationnelle sur les problèmes d'ordonnancement.

### 3 Fonctionnalités du système

Le système, développé entièrement en Python, est constitué d'un programme de résolution exacte (PLNE), et du programme de Machine Learning :

L'utilisateur a la possibilité de lancer une résolution avec une méthode exacte (PLNE) en important un ou plusieurs fichiers d'instances. Le programme calcule ensuite la solution optimale pour chaque instance, et exporte les résultats dans un fichier (chaque ligne du fichier correspondant à une instance et sa solution optimale correspondante) que l'utilisateur peut consulter.

La deuxième fonctionnalité du projet est de lancer une prédiction sur la solution d'une instance passée en paramètre dans le programme de Machine Learning (sequence-to-sequence).

La structure des fichiers d'instances utilisés est disponible dans la section 1 (Annexe F).



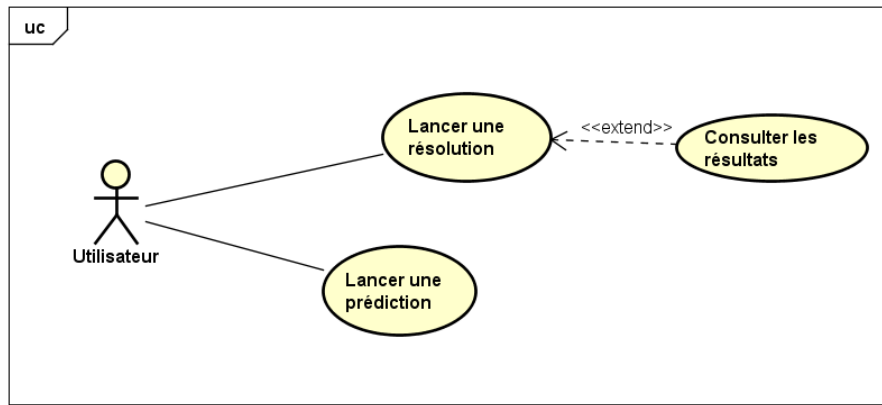


Figure 1 – Diagramme de cas d'utilisation

## 4 Structure générale du système

Le diagramme de composants suivant décrit la structure générale du système :

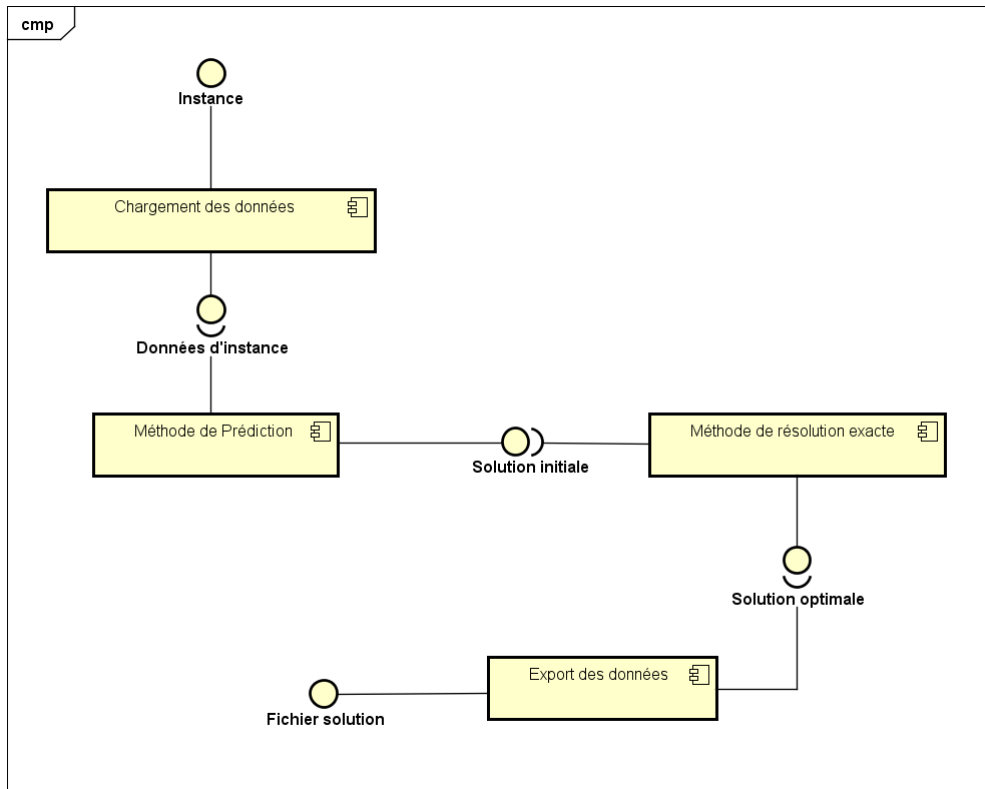


Figure 2 – Structure générale du système

4 composants interagissent entre eux :

- Un composant se charge d'importer les données d'un fichier d'instance.
- Le deuxième composant s'interface avec le premier en traitant les données importées pour y prédire une solution initiale.
- Le troisième composant est une méthode de résolution exacte et se sert de la solution initiale pour calculer la solution optimale.
- Enfin, le dernier composant s'interface avec la méthode de résolution exacte pour écrire le résultat final vers un fichier.

# 3

## Etat de l'art

Il est nécessaire de définir le concept de Machine Learning avant de réaliser un état de l'art sur les différentes techniques qui pourront être appliquées dans la phase de développement.

### 1 Machine Learning : définition

Le machine Learning (apprentissage automatique) est un domaine d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques afin de donner à un ordinateur la capacité « d'apprendre » à partir de données, et de réaliser des prédictions.

Plus précisément, le Machine Learning a pour but d'entraîner un algorithme en se basant sur des exemples afin de créer un modèle prédictif. L'objectif final est que le modèle puisse déterminer ce qui lie des données de sortie à des données d'entrée.

Une fois que l'ordinateur a appris, c'est-à-dire qu'il est entraîné, il peut résoudre des tâches ou effectuer des prédictions (classification, tendances, ...) sans avoir besoin d'être programmé au préalable. L'ordinateur va améliorer ses prédictions au cours du temps en fonction de l'expérience qu'il acquière.

Lorsqu'on souhaite créer un modèle de Machine Learning, il existe 2 phases : l'apprentissage et la prédiction.

#### Phase d'apprentissage (ou d'entraînement)

Cette phase consiste à estimer un modèle à partir de données. La phase d'apprentissage va permettre d'entraîner le modèle afin qu'il puisse être capable d'estimer. C'est dans cette phase que le modèle va essayer de trouver un lien entre les données en sortie et les données en entrée. Ces dernières constituent la base d'apprentissage du modèle.

Il existe différents types d'apprentissage, comme par exemple :

- Apprentissage supervisé : les données envoyées au modèle sont annotées de leurs sorties, c'est-à-dire qu'on leur associe des étiquettes (par exemple associer l'étiquette « avion » à une photo d'un avion). C'est donc l'utilisateur qui supervise l'apprentissage en indiquant au modèle comment se comporter pour effectuer des prédictions.

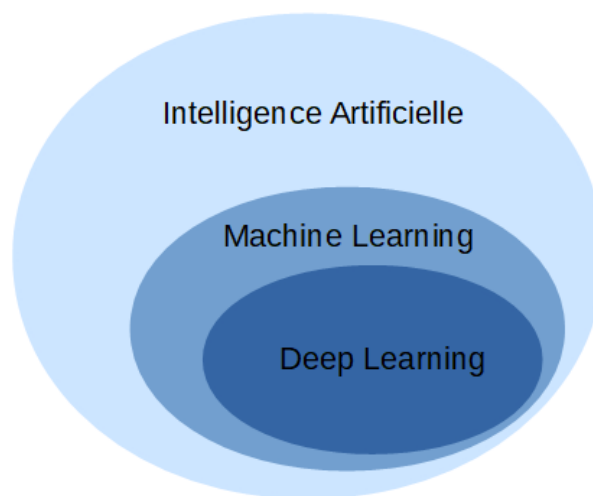
- Apprentissage non supervisé : les données envoyées au modèle ne sont pas annotées/étiquetées. Ce dernier va se débrouiller seul et essayer de regrouper les données qui présentent des caractéristiques communes entre elles.
- Apprentissage par renforcement : le modèle apprend de manière incrémentale à partir de ses expériences passées. Il va prendre des décisions par rapport à son état courant. En fonction de ses décisions, il va recevoir une récompense plus ou moins bonne. Le modèle va donc chercher à maximiser la somme des récompenses au cours du temps.

### Phase de prédiction

La phase de prédiction intervient lorsque le modèle est entraîné. En fonction des types d'apprentissage, les données d'entrée ne sont pas exactement les mêmes (exemple de la photo d'un avion : cette fois-ci, les données d'entrée seront seulement les pixels de l'image sans l'étiquette "avion").

Le machine Learning comprend de nombreuses techniques. Parmi elles, nous pouvons distinguer le Deep Learning.

Le Deep Learning (apprentissage profond) fait partie d'une famille de méthodes du Machine Learning. Il comprend des méthodes qui tentent de modéliser des données avec un haut niveau d'abstraction grâce à des architectures complexes. On peut notamment citer les réseaux de neurones artificiels qui sont inspirés des réseaux de neurones biologiques du cerveau humain.



**Figure 1** – Concepts de l'intelligence Artificielle

Nous présenterons donc dans cet état de l'art plusieurs méthodes de Machine Learning ainsi que des architectures de réseaux de neurones.

## 2 Techniques de Machine Learning

### 2.1 Régression linéaire

Le but de la régression linéaire est d'essayer de trouver une relation linéaire entre une variable expliquée et une ou plusieurs variables explicatives.

L'objectif est de faire passer au mieux une droite à travers un nuage de points qui pourrait être par exemple  $n$  observations expérimentales. Si l'on prend un exemple en deux dimensions, il nous faut les valeurs observées  $x$  et  $y$  (de  $x_1, y_1$  jusqu'à  $x_n, y_n$ ). Ensuite il faut placer toutes ces observations sur un graphique.

A travers ces points, le but est de tracer une droite de forme :

$$y = ax + b$$

Ensuite, il faut voir quelle est la distance entre les points qu'on avait mis au départ (observations) et la droite tracée (les points ajustés). Cela permet de minimiser l'erreur.

Cette technique est très utilisée en Machine Learning. Elle est de plus simple à mettre en place. Dans l'exemple ci-dessous, on a tracé une droite de régression linéaire qui montre la relation entre la taille d'un appartement et son loyer :

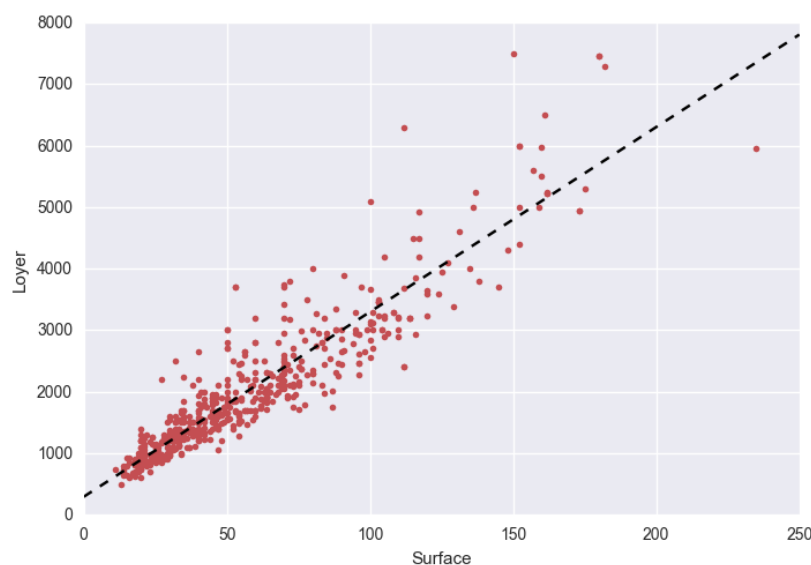


Figure 2 – Exemple de régression linéaire

Ce nuage de points correspond aux observations expérimentales (= base d'apprentissage). Ces données ont permis au programme "d'apprendre". Pour réaliser des prédictions, nous avons seulement à donner la taille d'un appartement au programme qui nous retournera une estimation du loyer. Dans cet exemple, la valeur  $y$  du loyer dépend seulement de la valeur  $x$  de la surface de l'appartement. Dans un cas plus réel, le loyer dépendrait de plusieurs variables (localisation, nombre de pièces, ...).

## 2.2 Régression logistique

L'objectif de la régression logistique est de prédire une variable aléatoire binomiale  $y$  en fonction de plusieurs variables  $x$ . C'est-à-dire prédire la valeur d'une variable représentée sous forme binaire (0 ou 1) en fonction de plusieurs paramètres en entrée. On cherche à expliquer la survenue d'un événement comme par exemple l'identification des facteurs liés à une maladie ou encore de prédire la probabilité de survie d'un passager du titanic en fonction de ses caractéristiques (âge, sexe, classe sociale, ...). Un modèle de régression logistique est représenté sous la forme suivante (c'est une fonction sigmoïde) :

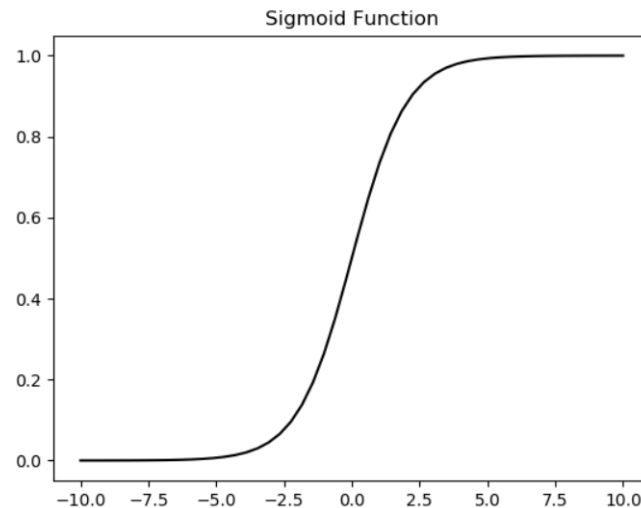


Figure 3 – Représentation d'une régression logistique

### 2.3 Régression lasso

La régression lasso est une méthode de contraction des coefficients de la régression. C'est une régression linéaire qui permet de réduire le nombre de variables du modèle en supprimant des variables qui ne sont pas nécessaires lors de la prédiction.

## 3 Architectures de Deep Learning : réseaux de neurones

### 3.1 Single Layer Perceptron (SLP)

C'est le réseau de neurones le plus simple. C'est aussi un des premiers algorithmes de Machine Learning.

Le perceptron est la modélisation d'un seul neurone qui va permettre de faire une prédiction en sortie (booléenne). Le perceptron est mono-couche (contrairement aux réseaux de neurones multi-couches). Une couche est constituée d'un ou plusieurs neurones.

Voici la structure d'un perceptron :

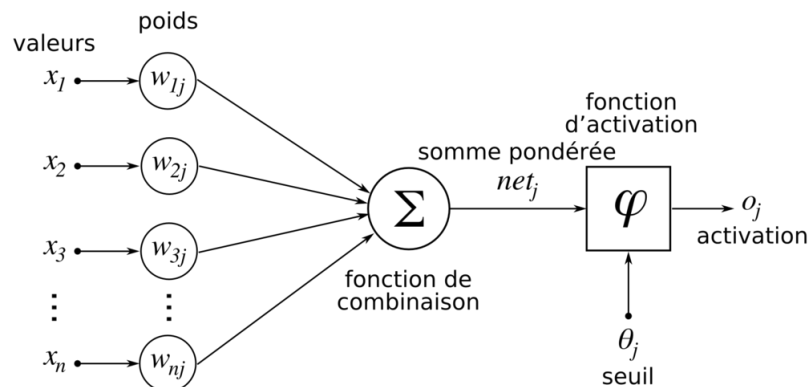


Figure 4 – Structure d'un perceptron

Le neurone va prendre des entrées (valeurs numériques indépendantes) et produire une sortie. Un poids est associé à chacune des entrées. Ces poids permettent de donner une importance

à chaque entrée. Le neurone va donc effectuer une opération qui est une somme des entrées pondérées (chaque entrée est multipliée par son poids puis on fait l'addition de toutes les entrées). La valeur obtenue est appelée la valeur de pré-activation. On applique ensuite une fonction d'activation.

Pour chaque donnée lors de la phase d'apprentissage, une fonction d'erreur sera utilisée (différence entre la sortie souhaitée et la sortie réelle) pour ajuster les poids et ainsi recadrer le modèle au niveau de ses prédictions.

### 3.2 Recurrent Neural Network (RNN)

Les RNN (en français "Réseaux de neurones récurrents") sont capables de gérer des séquences (séries temporelles), c'est-à-dire prendre des séquences en entrée et donner en sortie des séquences. Ils sont notamment utilisés dans la reconnaissance vocale et la traduction automatique.

Les réseaux de neurones récurrents offrent différentes architectures :

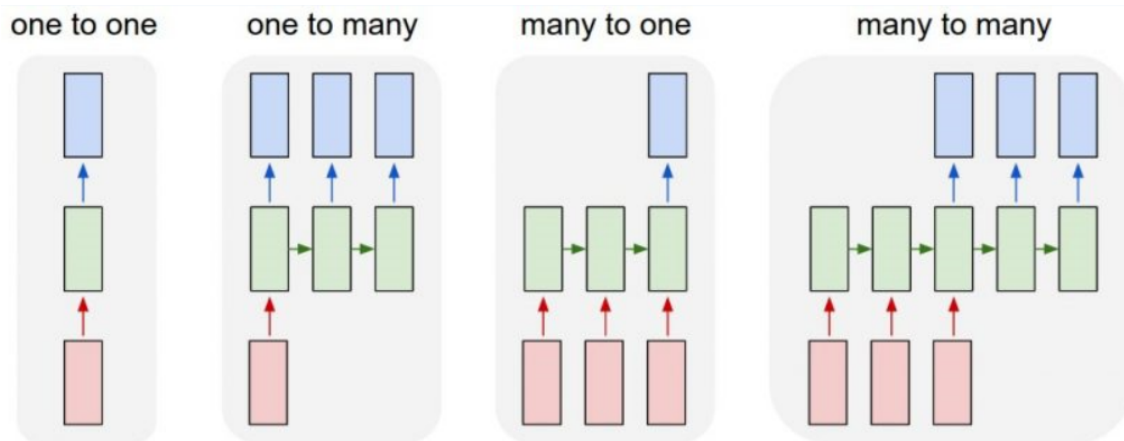


Figure 5 – Architectures RNN

1. One to one : ce réseau de neurones prend une seule entrée qui est un vecteur  $x$  et donne une sortie. Dans cette architecture la sortie prend seulement en compte le vecteur  $x$  (= architecture du perceptron).
2. One to many : ce réseau de neurones prend en entrée un vecteur  $x$  et génère en sortie une série de vecteurs. Par exemple le vecteur  $x$  pourrait être l'équivalent du mot "je", et les vecteurs de sortie seraient la suite de la phrase : "suis", "à", "la", "maison".
3. Many to one : ce réseau de neurones prend en entrée une série de vecteurs et donne une sortie. Par exemple on pourrait imaginer en entrée la phrase "Comment vas-tu?", et obtenir en sortie une valeur positive ou négative comme "est-ce que la phrase passée en entrée est une question? Oui ou non". Dans ce cas la sortie serait "oui".
4. Many to many : ce réseau de neurones prend en entrée une séquence et donne en sortie une séquence (modèle sequence-to-sequence), par exemple une traduction "Français-Anglais".

### 3.3 Long Short-Term Memory (LSTM)

C'est une architecture RNN composée d'une porte d'entrée, d'une porte d'oubli, et d'une porte de sortie. Ce type de réseaux de neurones résout le problème de « vanishing gradient » qui peut être rencontré lors de la mise en place d'un RNN classique. Ce problème ne sera pas détaillé puisque ce n'est pas le sujet de ce projet.

### 3.4 Sequence-to-sequence (seq2seq)

Le modèle sequence-to-sequence (seq2seq), qu'on appelle aussi encodeur-décodeur, fait partie des réseaux de neurones récurrent (LSTM) et est un modèle qui prédit une séquence de données à partir d'une séquence de données en entrée (architecture many-to-many). Deux réseaux de neurones sont utilisés lorsque l'on implémente un modèle séquence-to-sequence : un réseau de neurones encodeur et un réseau de neurones décodeur. L'entrée et la sortie ne sont pas forcément de la même longueur. Cette méthode est utilisée dans la traduction d'une phrase d'une langue d'origine vers une langue de destination.

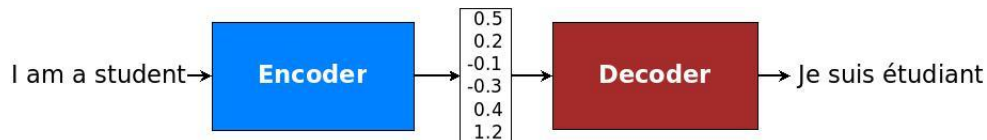


Figure 6 – Modèle *sequence-to-sequence*

Dans l'exemple ci-dessus, le premier réseau de neurones va encoder la phrase dans un vecteur de valeurs numériques. Le second réseau de neurones va se charger de décoder ce vecteur pour obtenir la phrase traduite.

# 4

## Analyse et conception

Dans ce PRD, on s'intéresse à des méthodes de résolution exactes, avec notamment des modèles linéaires en nombres entiers (PLNE pour Programmation Linéaire en Nombres Entiers). Il a fallu dans un premier temps s'approprier ces modèles. La deuxième partie de l'analyse est consacrée à l'étude de plusieurs techniques de Machine Learning afin de les utiliser avec ces méthodes de résolution exactes.

Nous allons donc expliquer deux modèles mathématiques du problème d'ordonnancement mono-critère multi-machines :

### 1 Méthodes de résolution exactes

#### 1.1 Modèle linéaire en nombres entiers 1 (PLNE1)

##### 1.1.1 Fonction objectif

$$\text{Maximiser : } \sum_{i=0}^M \sum_{j=0}^N w_j x_{ij}$$

Cette expression est la fonction objectif. C'est ce que l'on souhaite maximiser ou minimiser. Ici, on souhaite maximiser la valeur totale des jobs pondérés.

##### 1.1.2 Contraintes

$$\sum_{t=s_j}^{f_{j-1}} y_{ijt} = (f_j - s_j) * x_{ij}; \forall j \in N, \forall i \in M \quad (1)$$

$$\sum_{j=0}^N y_{ijt} * r_{jk} \leq R_k; \forall k \in K, \forall i \in M, \forall t \in [s_{min}, f_{max}] \quad (2)$$



$$\sum_{i=0}^M x_{ij} \leq 1; \forall j \in N \quad (3)$$

(1) : La première contrainte spécifie qu'un job doit être ordonnancé durant son intervalle de temps (donc  $y_{ijt} = 1$  pour tout  $t$  appartenant à  $[s_j; f_j]$  si le job est ordonnancé ( $x_{ij} = 1$ )).

(2) : La deuxième contrainte vérifie que l'ordonnancement des jobs doit respecter la capacité totale des machines.

(3) : La troisième contrainte est une contrainte d'unicité. Elle indique que chaque job doit être ordonnancé sur une seule machine.

### 1.1.3 Variables

$$x_{ij} \in \{0, 1\}; y_{ijt} \in \{0, 1\}; \forall i \in M, \forall j \in N, \forall t \in T$$

Au niveau des variables :

- $x_{ij}$  est une variable binaire qui prend la valeur 1 si le job est exécuté sur la machine  $i$ , 0 sinon.
- $y_{ijt}$  est une variable binaire qui prend la valeur 1 si le job  $j$  est exécuté sur la machine  $i$  à l'instant  $t$ , 0 sinon.

### 1.1.4 Performances

Pour ce modèle PLNE, on a  $NM + KMT + N$  contraintes ainsi que  $MN + MNT$  variables de décision. On a lancé ce programme pour une instance 7-3-200 (7 machines, 3 ressources et 200 jobs). On a obtenu le résultat suivant : 648.015 secondes. C'est-à-dire que le PLNE a calculé la solution optimale en 648.015 secondes (environ 10 minutes).

On a ensuite cherché à voir s'il était possible d'améliorer la performance en termes de temps de calcul de ce modèle. Pour cela nous avons proposé un nouveau modèle PLNE en réduisant le nombre de contraintes et de variables.

## 1.2 Modèle linéaire en nombres entiers 2 (PLNE2)

### 1.2.1 Fonction objectif

$$\text{Maximiser : } \sum_{i=0}^M \sum_{j=0}^N w_j x_{ij}$$

La fonction objectif est la même que le modèle PLNE précédent.

## 1.2.2 Contraintes

$$\sum_{j:s_j \leq t < f_j}^N x_{ij} * r_{jk} \leq R_k; \forall i \in M, \forall k \in K, \forall t \in [s_{min}, f_{max}] \quad (1)$$

$$\sum_{i=0}^M x_{ij} \leq 1; \forall j \in N \quad (2)$$

(1) : Cette première contrainte est l'équivalent de la fusion des deux premières contraintes du modèle PLNE précédent. Elle permet d'observer à chaque instant  $t$  les jobs qui se chevauchent.

(2) : Cette deuxième contrainte est la même que la troisième contrainte du modèle PLNE précédent.

## 1.2.3 Variables

$$x_{ij} \in \{0, 1\}; \forall i \in M, \forall j \in N$$

La variable  $x_{ij}$  est la même que le modèle précédent.

## 1.2.4 Performances

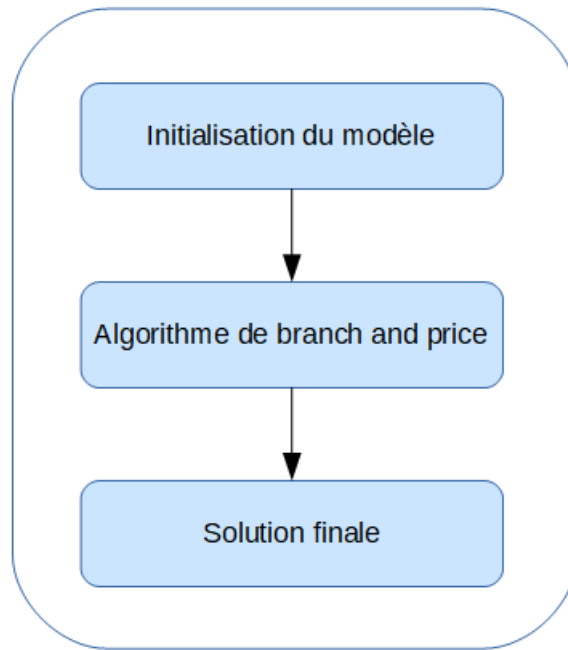
Pour ce nouveau modèle, on a  $N + KMT$  contraintes et  $MN$  variables de décision. On a décidé de lancer le modèle avec la même instance (7-3-200 pour 7 machines, 3 ressources et 200 jobs) et le même ordinateur que sur le premier PLNE pour comparer les temps de calcul. On a obtenu pour ce second PLNE le résultat suivant : 656.859 secondes. On constate donc un temps de calcul légèrement plus élevé par rapport au premier modèle PLNE. Le fait de réduire le nombre de contraintes et de variables n'améliore donc pas forcément la performance en temps de calcul au niveau de la résolution du problème.

Après réflexion, nous avons constaté que le deuxième modèle n'est pas pertinent. En effet la deuxième contrainte de ce modèle néglige la longueur des intervalles fixes des jobs. Le premier modèle, au contraire, prend en compte ces longueurs au niveau de sa première contrainte.

Pour des petites instances (de l'ordre de 60 jobs environ) les modèles PLNE sont capables de calculer la solution optimale de manière instantanée. Cependant il existe des limites à ces méthodes. Par exemple, si l'on exécute un PLNE sur une instance 10-3-200 (10 machines, 3 ressources et 200 jobs), on constate qu'il ne converge pas vers la solution optimale après une heure de calcul.

## 1.3 Méthode de décomposition (branch and price)

Une autre méthode a été mise en place. C'est une approche par décomposition avec un algorithme de branch and price. Cette méthode comprend 3 grandes phases :



**Figure 1** – *Méthode par décomposition*

Il y a tout d’abord une phase d’initialisation du modèle. Une fois ce dernier initialisé, un algorithme de branch and price est appliqué pour calculer la solution optimale.

Remarque : Cette méthode de décomposition est un exemple. Il existe d’autres méthodes comme par exemple les algorithmes NSGA-2 ou encore les algorithmes génétiques pour lesquels nous avons aussi une phase d’initialisation du modèle avec une solution de départ.

En exécutant cette méthode sur une instance 10-3-200 (10 machines, 3 ressources, 200 jobs), on constate cette fois-ci qu’elle converge vers la solution optimale après seulement 55 secondes (comparé aux 648 secondes du premier modèle PLNE).

Trouver une bonne initialisation du modèle est importante pour la convergence de ce modèle vers la solution optimale. C’est donc dans cette phase d’initialisation que les techniques de Machine Learning vont être mises en place afin de trouver une bonne initialisation du modèle.

## 2 Apport du Machine Learning

### 2.1 Architecture du système

#### 2.1.1 Phase de prédiction

Voici l’architecture globale du système :

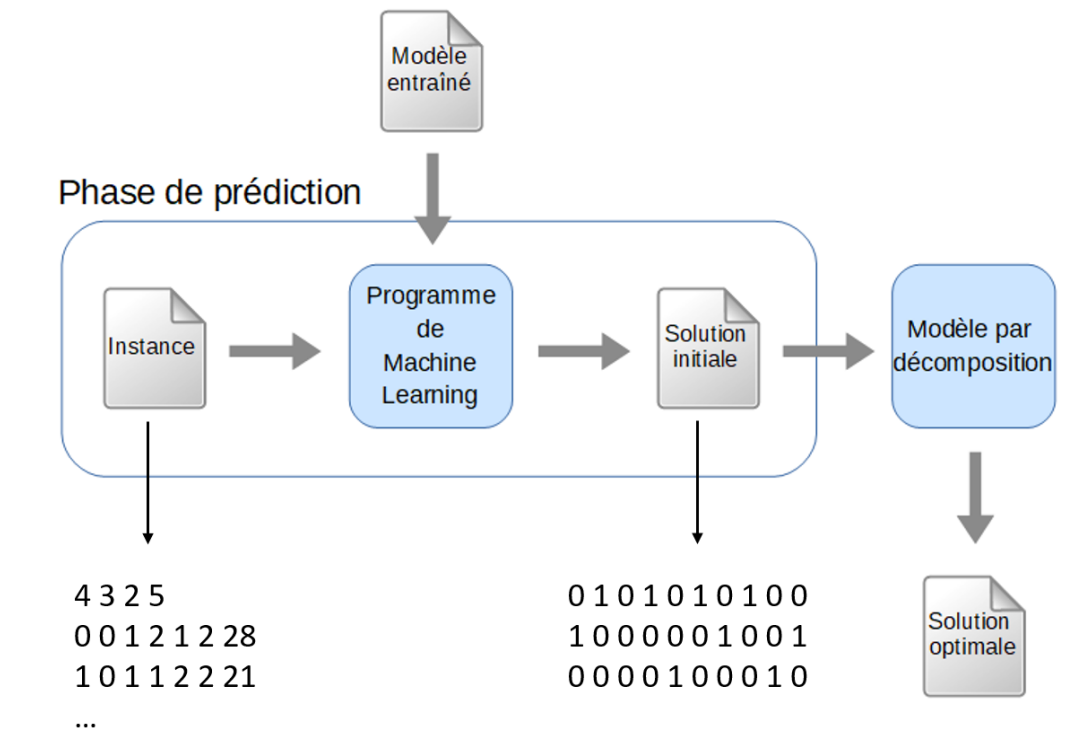


Figure 2 – Phase de prédiction

Le programme de Machine Learning doit avoir été entraîné au préalable (la phase d'apprentissage sera détaillée ensuite).

Le cheminement est le suivant :

1. Un fichier d'instance de jobs est passé en entrée au programme de Machine Learning.
2. Le programme de Machine Learning va ensuite prédire en sortie des vecteurs qui représentent une solution. C'est-à-dire les jobs ordonnancés sur les différentes machines (une ligne représente une machine  $i$  et une colonne représente un job  $j$  : si le job  $j$  est exécuté sur la machine  $i$ , alors la valeur est égale à 1, 0 sinon).
3. Cette solution, qui représente une solution initiale, sera ensuite passée en entrée au modèle par décomposition.
4. Pour finir, le modèle par décomposition va calculer la solution optimale du problème.

On espère que le programme de Machine Learning prédise une « bonne » solution initiale afin que le modèle par décomposition converge plus rapidement vers la solution optimale.

### 2.1.2 Phase d'apprentissage

Initialement, pour résoudre un problème d'ordonnancement avec une méthode exacte, nous avons l'architecture suivante :

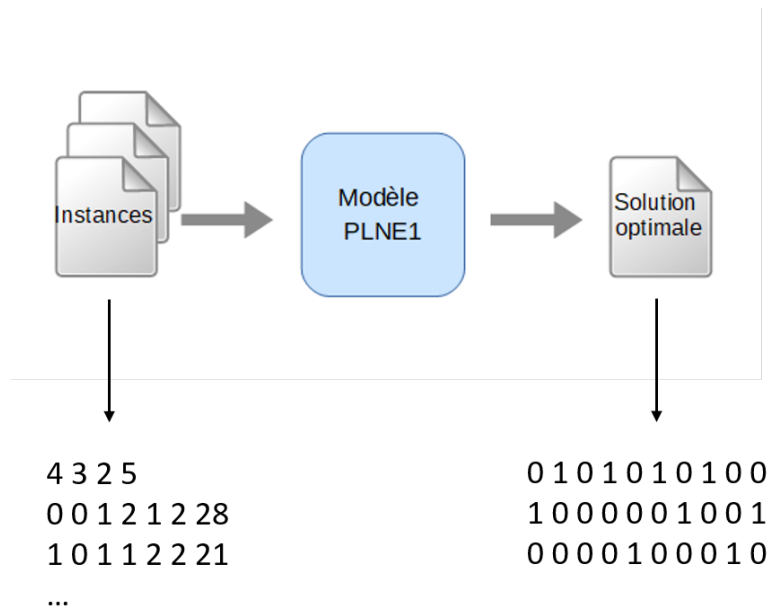


Figure 3 – Exécution du programme PLNE1

On passe un fichier d'instance en entrée d'un modèle PLNE (PLNE1 que l'on a détaillé plus tôt par exemple) qui nous calcule en sortie la solution optimale.

Ce que nous allons chercher à faire dans la phase d'apprentissage, c'est de réfléchir à une structuration d'un fichier de sortie de ce modèle PLNE, qui contiendrait les informations sur une instance ainsi que la solution optimale correspondante.

On aura donc l'architecture suivante pour la phase d'apprentissage :

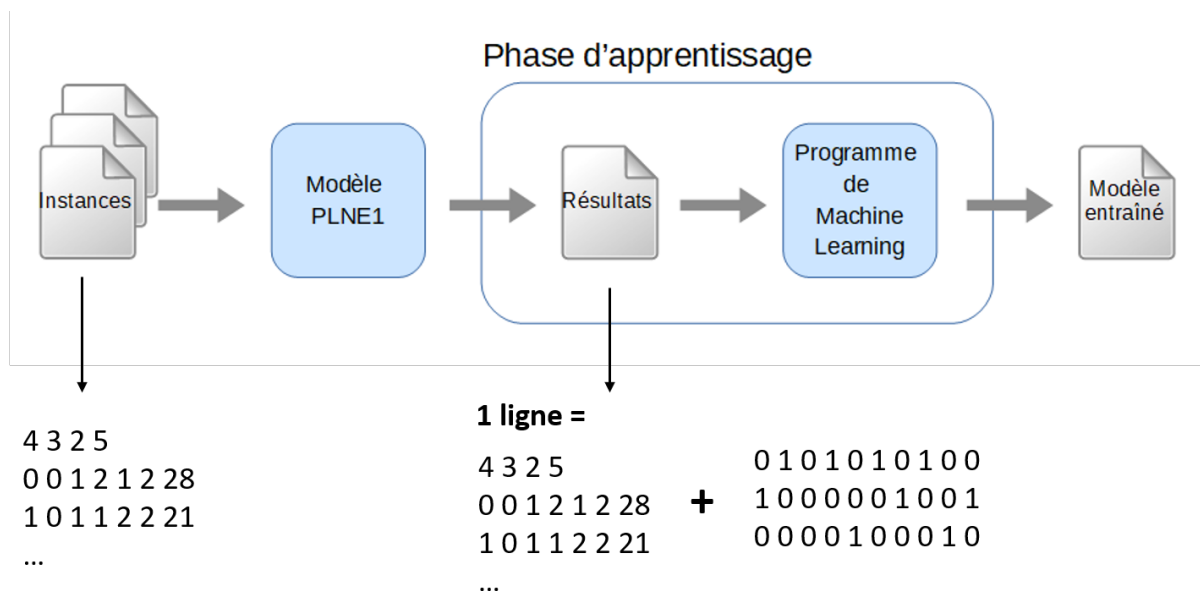


Figure 4 – Phase d'apprentissage

Le cheminement est le suivant :

1. Nous allons passer en entrée au modèle PLNE plusieurs instances.
2. Ensuite ce PLNE va produire un nouveau fichier « Résultats ». Chaque ligne correspondra à une instance et sera composée des informations sur l'instance ainsi que de la solution

optimale correspondante.

3. Ce fichier sera passé en entrée au programme de Machine Learning. Ainsi, nous pourrons indiquer au programme pour chaque ligne du fichier : « pour cette instance-là, tu devras obtenir cette solution ». Nous obtiendrons donc ensuite un modèle entraîné qui pourra être utilisé pour réaliser des prédictions sur des instances de jobs.

## 2.2 Analyse des techniques retenues

A la suite de la rédaction de l'état de l'art, il a été nécessaire d'étudier les différentes techniques de Machine Learning afin de vérifier si elles peuvent être adaptées au problème d'ordonnement.

### 2.2.1 Régression linéaire

Tout d'abord, une régression linéaire ne peut pas être mise en place dans notre cas. La principale cause est le nombre de paramètres que nous devons passer en entrée au programme de machine Learning. En effet notre fichier d'entrée sera composé de toutes les informations sur une instance et de la solution optimale. Il est impossible de représenter cette régression linéaire avec autant de variables. Nous serions alors obligé de réduire le nombre de paramètres du problème pour obtenir une représentation en 2 ou 3 dimensions. Mais cela implique de perdre des informations qui sont essentielles puisque toutes les données sur l'instance sont importantes et nécessaires pour prédire une solution.

La réduction du nombre de paramètres impliquerait donc de fausser complètement le résultat en ne prenant pas toutes les données en compte. Le modèle pourrait alors prédire des solutions impossibles.

De plus, les données d'entrée du modèle de régression linéaire doivent être uniquement des valeurs numériques. Cela impliquerait dans notre cas d'associer un identifiant entier pour chaque ligne du fichier d'entrée (donc pour chaque instance) afin de remplacer le nom de chaque instance. C'est un problème puisque, étant donné que les identifiants seront différents (par exemple première ligne = 1, deuxième ligne = 2, ...), alors le modèle de régression linéaire établira un lien entre le résultat de la prédiction et la variable identifiant qui devrait normalement n'avoir aucun impact sur la prédiction d'une solution.

### 2.2.2 Régression logistique

La régression logistique ne peut pas être mise en place. Etant donné qu'elle a pour but de prédire une probabilité sur une variable binaire, elle n'est pas adaptée à notre problème pour lequel on cherche à prédire une solution, donc une série de valeurs binaires qui représentent si les jobs ont été ordonnancés ou non sur les différentes machines.

### 2.2.3 Perceptron

Le perceptron est un classifieur linéaire, c'est-à-dire qu'il cherche à regrouper les données observées qui ont des propriétés similaires dans des classes. De plus la sortie de ce modèle est booléenne. Il ne correspond donc pas à nos critères et ne peut pas être appliqué à notre problème.

### 2.2.4 Sequence-to-sequence

A priori, le modèle sequence-to-sequence semblerait correspondre le plus à ce que l'on souhaite mettre en place. En effet la particularité de ce modèle est qu'il peut recevoir en entrée puis prédire en sortie plusieurs séquences (vecteurs). Dans l'architecture de notre système, nous pouvons visualiser nos données d'entrée et de sortie du modèle de Machine Learning comme plusieurs séquences : par exemple en entrée des vecteurs qui regrouperaient les informations sur une instance et en sortie des vecteurs représentant la solution prédite.

## 3 Conception du système

Une analyse a été faite de manière plus approfondie à la suite du diagramme de composants de la section 2 (Chapitre 2), afin d'avoir une idée plus précise sur les composants du programme PLNE qui calcule les solutions optimales des instances, et du programme seq2seq utilisé pour la prédiction de solution.

### 3.1 Structure du programme PLNE

Le diagramme de composants suivant décrit les principales fonctions et leurs interactions dans le programme PLNE :

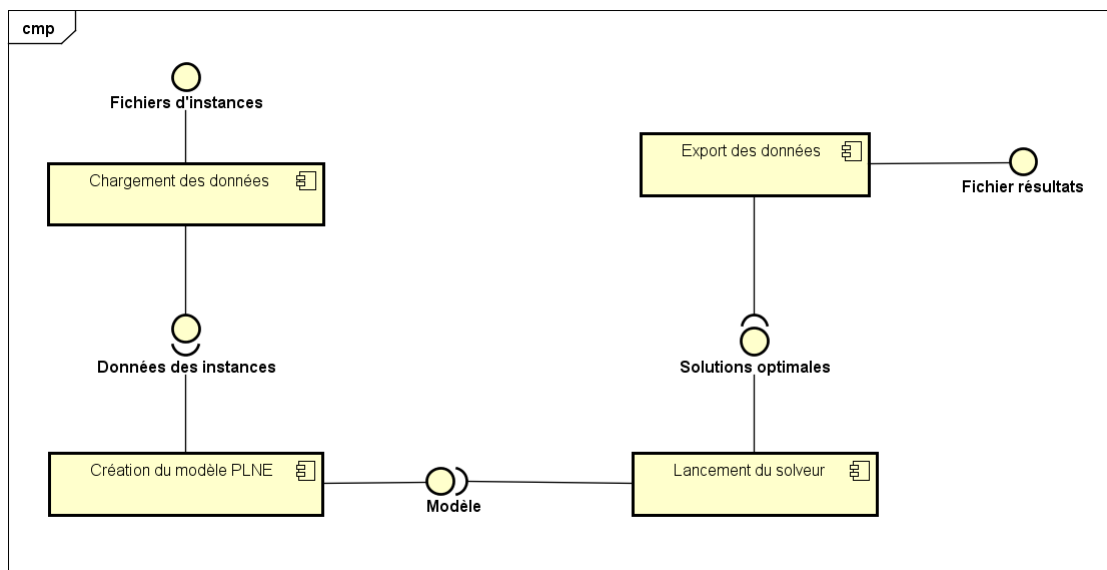


Figure 5 – Diagramme de composants du PLNE

4 composants interagissent entre eux :

- Un composant se charge d'importer les données des fichiers d'instances.
- Le deuxième composant crée le modèle PLNE en fonction des données importées.
- Le composant « Lancement du solveur » permet de lancer la résolution du modèle et de calculer les solutions optimales des instances.
- Le dernier composant permet d'exporter les données (instances + solutions calculées) dans un fichier résultats.

## 3.2 Structure du programme seq2seq

### 3.2.1 Phase d'apprentissage

Le diagramme de composants suivant décrit les principales fonctions et leurs interactions dans le programme sequence-to-sequence en phase d'apprentissage :

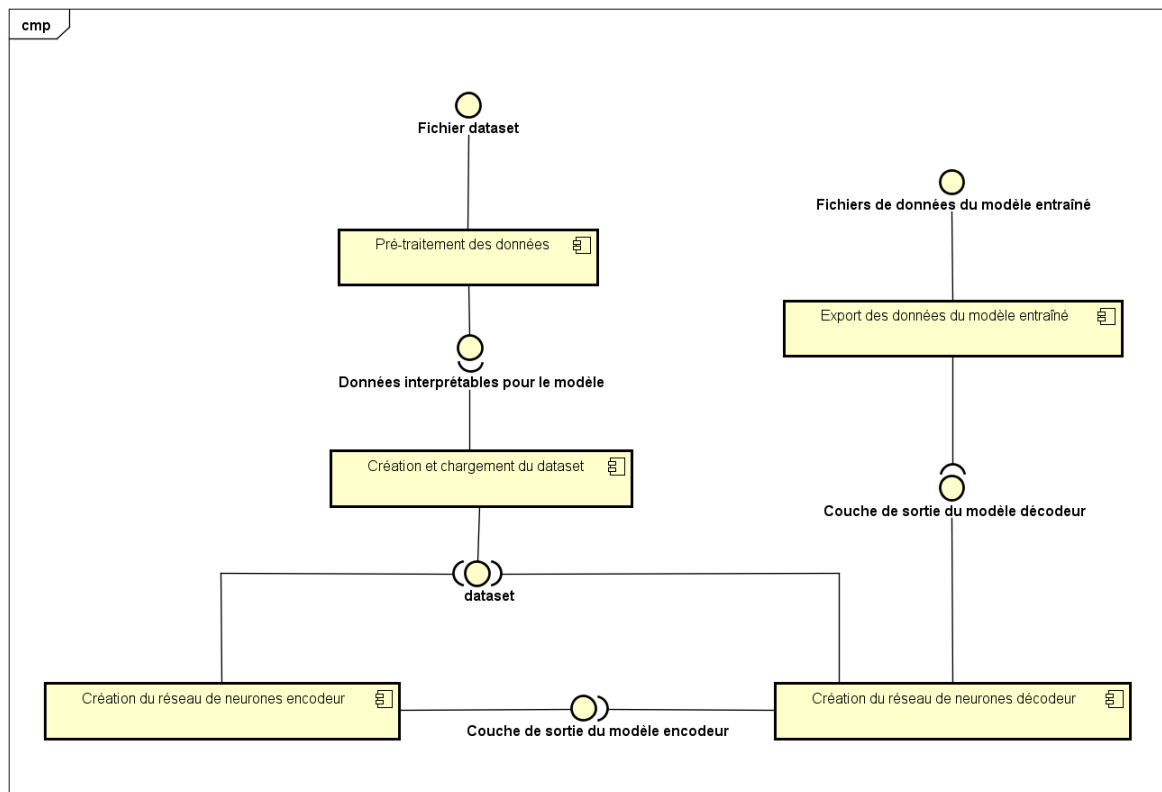


Figure 6 – Diagramme de composants du seq2seq en phase d'apprentissage

5 composants interagissent entre eux :

- Le composant « Pré-traitement des données » permet d'importer un fichier dataset et de le préparer afin qu'il puisse être mis en forme et utilisable pour le modèle.
- Le composant « Création et chargement du dataset » crée et charge le dataset.
- Les 2 composants "Encodeur" et "Décodeur" représentent les réseaux de neurones encodeur et décodeur du modèle sequence-to-sequence, et s'interfacent sur le dataset pour être créés. Lors de l'entraînement, l'encodeur est créé en premier, puis fourni le modèle dans sa couche de sortie, qui est ensuite utilisé par le décodeur. Ce dernier fournit à son tour le modèle dans sa couche de sortie.
- Le composant « Export des données du modèle entraîné » se charge de sauvegarder le modèle dans des fichiers en sortie.

### 3.2.2 Phase de prédiction

Le diagramme de composants suivant décrit les principales fonctions et leurs interactions dans le programme sequence-to-sequence en phase de prédiction :



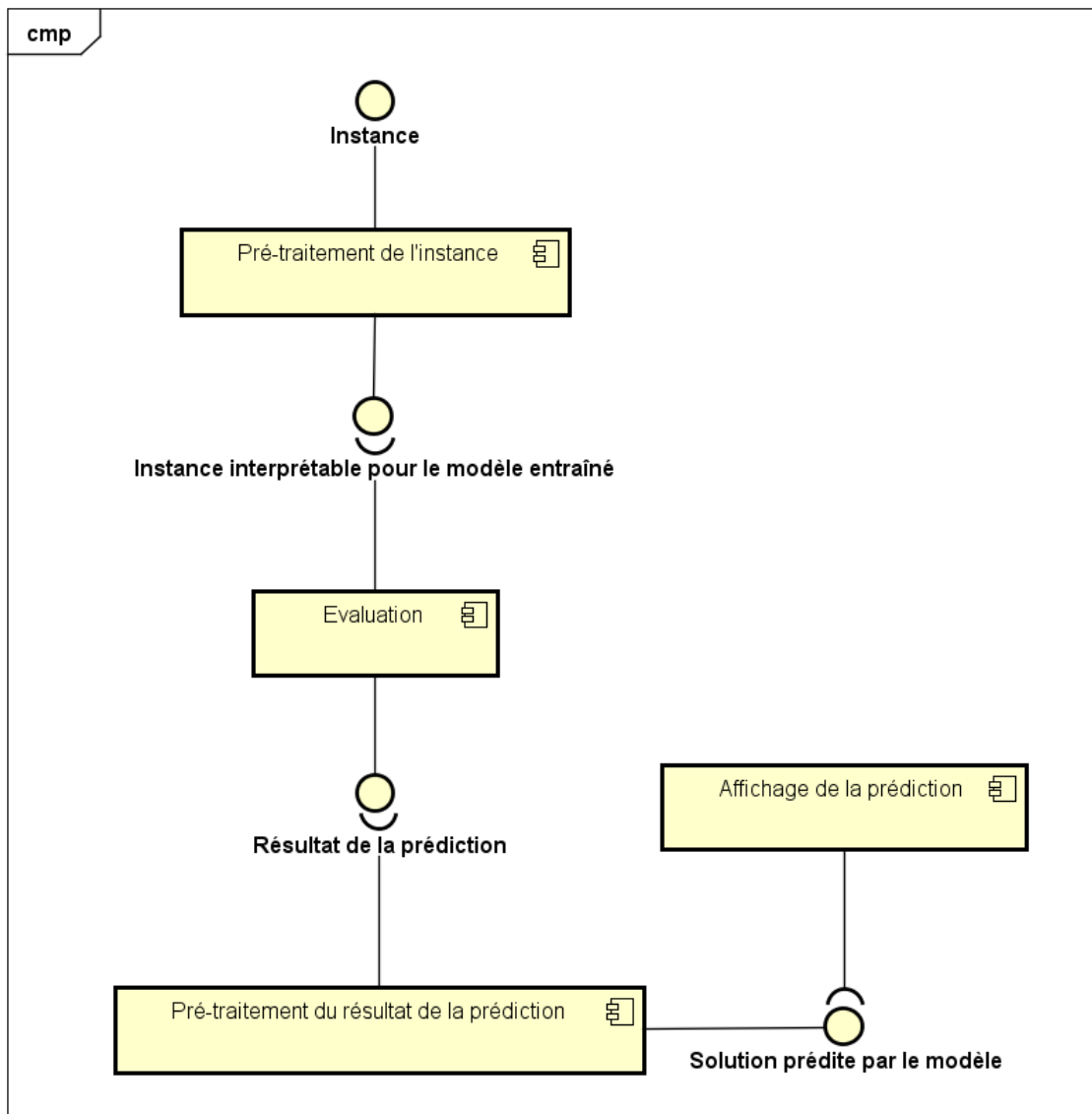


Figure 7 – Diagramme de composants du seq2seq en phase de prédiction

4 composants interagissent entre eux :

- Le composant « Pré-traitement de l'instance » permet de transformer l'instance pour qu'elle puisse être interprétable par le modèle.
- Le composant « Evaluation » s'interface à cette instance pour l'évaluer et réaliser une prédiction grâce au modèle entraîné.
- Le composant « Pré-traitement du résultat de la prédiction » se charge de convertir le résultat de la prédiction en une solution.
- Le dernier composant affiche la solution prédite.

# 5

## Mise en oeuvre

### 1 Compréhension d'un modèle seq2seq pour la traduction

Ayant très peu de notions dans le domaine du Deep Learning, il a été convenu avec mon encadrant pour gagner du temps de ne pas se focaliser sur toute l'architecture technique du modèle sequence-to-sequence. En effet, ces types de réseaux de neurones (qui font partie des modèles LSTM) ont une architecture complexe et sont donc difficiles à mettre en place. C'est pourquoi il a été choisi de sélectionner un exemple afin d'en comprendre les concepts généraux et de s'en servir de base afin d'adapter le problème au modèle.

Un très bon exemple de modèle sequence-to-sequence est disponible dans la documentation officielle de TensorFlow (accessible sur le site dans "Tutoriel > Advanced > Text > Neural machine translation with attention" ou sur le lien suivant : [https://www.tensorflow.org/tutorials/text/nmt\\_with\\_attention](https://www.tensorflow.org/tutorials/text/nmt_with_attention)).

Dans ce tutoriel, il s'agit d'entraîner un modèle pour la traduction de la langue espagnole vers la langue anglaise. Une fois le modèle entraîné, il est capable de prédire une phrase espagnole en sa traduction anglaise correspondante.

Le programme est implémenté en Python et utilise le framework TensorFlow qui se base sur la librairie Keras (librairie Python de Machine Learning).

#### 1.1 Dataset utilisé

Le dataset utilisé dans le tutoriel (disponible sur le site suivant : <http://www.manythings.org/anki/>) est un fichier contenant des phrases anglaises et leur traduction espagnole.

Ce fichier est composé pour chaque ligne d'une phrase anglaise puis de sa traduction espagnole. L'espace sert de séparateur entre la phrase d'origine et sa traduction.

En voici un exemple avec 8 phrases :

I admire your courage.	Admiro vuestro valor.
I already have a plan.	Ya tengo un plan.
I also wanted to know.	Yo también quería saber.
I always get up early.	Siempre me levanto pronto.
I always keep my word.	Siempre mantengo mi palabra.
I am 30 years old now.	Ahora tengo treinta años.
I am as rich as he is.	Soy tan rico como él.
I am as strong as you.	Soy tan fuerte como tú.

Figure 1 – Dataset utilisé

## 1.2 Préparation du dataset

L'étape de préparation du dataset est primordiale pour l'apprentissage du modèle. Elle va servir à réaliser un pré-traitement sur les phrases afin de les formater et les rendre utilisables par le modèle. Dans ce tutoriel, le pré-traitement consiste à :

- Enlever les caractères spéciaux pour des raisons de simplicité.
- Ajouter 2 chaînes de caractères supplémentaires à chaque phrase : "<start>" et "<end>". Ces tokens permettent d'indiquer au modèle le début et la fin de chaque phrase lors de la phase d'entraînement.
- Ajouter du padding pour chaque phrase du dataset afin qu'elle aient toutes la même taille.
- Créer 2 dictionnaires : le premier consiste à lier chaque mot présent dans le dataset à un identifiant, et le deuxième l'inverse.

Voici un exemple dans le dictionnaire créé pour 1 phrase espagnole et pour sa traduction anglaise :

```
Input Language; index to word mapping
1 ----> <start>
19 ----> mi
174 ----> padre
16 ----> esta
185 ----> ocupado
3 ----> .
2 ----> <end>

Target Language; index to word mapping
1 ----> <start>
21 ----> my
226 ----> father
8 ----> is
148 ----> busy
3 ----> .
2 ----> <end>
```

Figure 2 – Dictionnaires créés pour une phrase et sa traduction

Remarque : lors de la création d'un dictionnaire index → mot, un tenseur (tensor) est créé. Dans ce cas le tenseur est une matrice de chiffres de taille "*nombres de phrases\*taille de la phrase la plus longue*". Cette matrice contient en fait tous les index de chaque mot.

Si l'on prend l'exemple de la figure ci-dessus, le vecteur [1 19 174 16 185 3 2] est une ligne de la matrice créée lors de la création du dictionnaire index → mot pour les phrases espagnoles.

Une fois cette étape de pré-traitement réalisée, toutes les phrases sont stockées dans 2 listes : la 1ère contient toutes les phrases espagnoles, et la 2ème toutes les phrases anglaises.

### 1.3 Entraînement du modèle

Le fichier dataset possède plus de 100 000 lignes. Dans ce tutoriel, il a été choisi de ne garder que 30 000 lignes pour entraîner le modèle plus rapidement (plus la base d'apprentissage est petite, moins la qualité de la traduction est bonne).

Tout d'abord, il faut séparer les données d'entraînement des données de validation (test). Le dataset est donc coupé en 2 pour garder 80% des données pour l'entraînement, puis 20% pour la phase de validation du modèle.

Remarque : La phase de validation permet d'évaluer la précision (*accuracy*) d'un modèle, c'est-à-dire la qualité de son apprentissage. Les 20% restants du dataset permettent donc d'utiliser des données n'ayant pas été apprises lors de l'entraînement.

Dans ce tutoriel, il n'existe pas de phase de validation. On se sert seulement des données d'entraînement (24 000 phrases) pour faire apprendre le modèle, puis on passe directement à l'étape de prédiction.

#### Paramètres

Différents paramètres peuvent être ajustés pour l'entraînement du modèle en fonction de la taille du dataset, ou du nombre de valeurs à prédire par exemple :

- Taille du lot (*batch size*) : elle permet de définir le nombre de données qui seront propagées sur le réseau de neurones. Par exemple si la taille du lot est fixée à 64, les 64 premiers échantillons (du 1er au 64) de l'ensemble des données d'entraînement seront extraits pour former le réseau. Puis, les 64 échantillons suivants seront prélevés (65 à 128) pour entraîner à nouveau le réseau, etc jusqu'à arriver à la taille des données d'apprentissage (ici 24 000).
- Nombre de couches cachées (*embedding dim*) : ce paramètre permet de définir le nombre de couches cachées au sein du réseau de neurones (pour rappel un réseau de neurones est organisé en 3 parties : la couche d'entrée (*input layer*) d'où provient le signal d'entrée, les couches cachées (*hidden layers*), et la couche de sortie (*output layer*) qui représente la prédiction).
- Nombre de neurones présents dans chaque couche (*units*)
- Nombre d'apprentissages sur le dataset complet (*epochs*)

Dans le tutoriel, les valeurs suivantes sont proposées :

- Taille du lot = 64
- Nombre de couches cachées = 256
- Nombre de neurones dans chaque couche = 1024
- Nombre d'apprentissages sur le dataset complet = 10

#### Réseaux de neurones Encodeur et Décodeur

2 réseaux de neurones sont utilisés dans le modèle sequence-to-sequence. Ils sont représentés par 2 classes python dans le tutoriel : une classe Encodeur, et une classe Décodeur.

#### Fonction de perte

La fonction de perte (ou fonction d'erreur) est similaire à une fonction objectif en recherche opérationnelle. On cherche à minimiser l'erreur. Elle permet de déterminer la performance du modèle. Si la valeur de la fonction de perte diminue lors de son apprentissage, cela signifie que le modèle est bien en train d'apprendre.

En résumé, la phase d'apprentissage consiste à créer les réseaux de neurones encodeur et décodeur, puis à réaliser des apprentissages sur le dataset complet par lots.

Après avoir exécuté le code du tutoriel, un répertoire contenant les données du modèle entraîné est créé. Ce dernier est prêt pour réaliser des prédictions.

## 1.4 Traduction

La traduction est l'étape finale qui consiste à passer une phrase espagnole en entrée au modèle entraîné. Il prédit alors la traduction anglaise (de plus ou moins bonne qualité en fonction de l'erreur obtenue lors de la dernière itération d'apprentissage).

## 2 Implémentation du modèle seq2seq pour la prédiction d'une solution

Le code du tutoriel TensorFlow utilisé pour entraîner un modèle sequence-to-sequence a été repris afin d'implémenter notre propre modèle sequence-to-sequence. La première étape a été de construire le fichier dataset contenant les instances et leur solution. On s'est d'abord focalisé sur de petites instances (2 machines, 3 ressources, 10 jobs), et un dataset avec 20 000 instances et solutions.

### 2.1 Structure du fichier dataset

Le programme PLNE est celui qui permet de construire le dataset, utilisé pour notre modèle sequence-to-sequence.

Jusqu'à présent, le programme PLNE implémenté permettait seulement de calculer la solution optimale d'une instance passée en paramètre. Il a donc fallu modifier le code afin de l'adapter au diagramme de composants de la section 5 (Chapitre 4).

Après modification, le programme prend désormais en entrée un dossier contenant des instances, puis calcule les solutions optimales de chaque instance.

Une fonction d'export a été ajoutée pour permettre d'écrire dans un fichier les résultats (constitués des instances et de leur solution optimale correspondante). C'est dans cette fonction qu'il a fallu réfléchir à la structuration de ce fichier.

Tous les paramètres dans un fichier d'instance sont nécessaires. Ils doivent donc être tous conservés lors de la création du fichier dataset. Ils doivent de plus être structurés : on doit pouvoir comprendre le format des instances et des solutions afin de retrouver toutes les informations nécessaires. Chaque valeur doit pouvoir être identifiable.

Afin de se rapprocher au mieux du modèle sequence-to-sequence de traduction espagnol-anglais, il a été décidé de partir du principe que chaque instance correspond à une phrase, et sa solution optimale peut être assimilée à sa traduction.

La structuration globale retenue pour le dataset est donc de la forme "*instance | solution*", avec "|" qui sert de séparateur entre l'instance et la solution (au lieu de l'espace utilisé dans le dataset du tutoriel TensorFlow).

La structuration détaillée retenue pour chaque ligne du fichier est de la forme :

```
([IdInstance] : [[Job1][Job2]...]) | (SolOptimale : [[PlacementJobsM1][PlacementJobsM2]...])
```

- Toute l'instance est entre parenthèses. Les paramètres de l'instance (nb jobs, nb machines, ...) sont entre crochets et séparés des informations sur les jobs par ":". Enfin, les informations sur chaque job sont regroupées entre elles puis séparées des autres jobs par les crochets. Nous pouvons donc voir les jobs et leurs informations comme une matrice avec pour premier indice l'identifiant du job et pour second indice une information sur un job (date de début par exemple).
- Toute la solution est entre parenthèses. La valeur de la solution optimale est séparée du détail de la solution par ":". Enfin, sur le même principe des jobs regroupés dans une matrice, nous retrouvons l'ordonnancement des jobs sur les machines qui sont séparés par des crochets pour chaque machine.

Un exemple de dataset avec 3 instances est disponible dans la section 3 (Annexe F).

Nous pouvons constater que cette structuration repose sur le même principe que le dataset qui contient les phrases anglaises et espagnoles. L'étape primordiale suivante est la préparation du dataset pour qu'il soit compréhensible par le modèle encodeur-décodeur.

## 2.2 Préparation du dataset

Les instances et les solutions (considérées comme des phrases) sont des séquences d'entiers. Chacune des valeurs est pour le moment identifiable dans le dataset.

Par exemple :

- Pour l'instance "([ 1 2 3 2 10] :[[33 35 2 1 2 3][ 6 15 1 2 2 6]...])", la valeur 33 correspond à la date de début du job 1, et la valeur 15 correspond à la date de fin du job 2.
- Pour la solution "(31 :[[0 0 0 0 1 0 0 1 0 0][0 1 0 0 0 0 0 1 0]])", la valeur 31 correspond à la solution optimale, puis sur le premier vecteur après les ":" (qui correspond à l'ordonnancement des jobs sur la machine 1), on peut voir que 2 jobs ont été ordonnancés : les jobs 5 et 8.

(Le début de l'instance précédente ainsi que sa solution optimale vont servir d'exemple de base pour la suite des explications)

Le premier test effectué après avoir créé notre programme a été de réaliser un entraînement et des prédictions sur des instances non structurées. En supprimant toute la structure créée dans le dataset, on obtient seulement des séquences d'entiers séparés par des espaces. En reprenant l'exemple, la séquence était donc réduite à :

1 2 3 2 10 33 35 2 1 2 3 6 15 1 2 2 6 ... | 31 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0

Avec chaque entier correspondant à un "mot" (pour faire l'analogie avec le tutoriel de la partie précédente).

Cependant cela pose problème. En effet on perd ici toute la structure de notre instance. De plus lors de la création du dictionnaire index  $\rightarrow$  mot, le modèle va représenter les mêmes entiers par le même index. La valeur 2 de l'exemple au dessus est présente 6 fois dans l'instance. Le premier 2 correspond au nombre de machines dans l'instance, et le troisième 2 correspond à la consommation de la ressource 2 du job 1. Ce ne sont pas du tout les mêmes types d'informations et pourtant ils seront considérés par le modèle comme le même "mot". Cela implique donc un mauvais apprentissage du modèle.

Il est alors nécessaire de conserver la structure du fichier.

Le principe qui a été décidé est le suivant :

Si l'on repart du tutoriel de traduction, chaque mot d'une phrase est indexé par une valeur. En suivant le même principe, il faudrait donc créer des mots pour chaque vecteur :

- Pour une instance, cela signifierait un mot pour décrire l'instance (vecteur avant les ":"), puis un mot pour chaque job (tous les autres vecteurs compris dans le grand vecteur après les ":").
- Pour la solution, on appliquerait la même convention : un mot pour décrire la valeur de la solution optimale, et un mot pour chaque vecteur représentant un ordonnancement.

Une fois les mots créés et les phrases "nettoyées" (suppression des ":", des crochets et parenthèses restants), la séquence précédente se traduirait en une phrase de la forme :

*motInfosInstance motJob1 motJob2 ... | motSolOptimale motOrdonnancementM1 motOrdonnancementM2*

il suffit ensuite d'ajouter les tokens "<start>" et "<end>" à chaque phrase pour retrouver le fonctionnement du tutoriel seq2seq de TensorFlow.

Malheureusement, nous ne pouvons pas conserver les instances et solutions telles quelles. En effet, toujours en reprenant l'exemple au dessus, le fait de regrouper le job 1 en un mot donnerait "33352123".

Encore une fois nous perdons des informations. Comment savoir si la date de début est "33", "3" ou "333"?

Un autre job pourrait avoir le même mot, et pourtant pas les mêmes valeurs (par exemple : "333 5 2 1 2 3"). Cela fausse l'apprentissage du modèle qui considérera 2 mots identiques qui seront finalement 2 jobs avec des paramètres différents.

Il a donc été nécessaire de trouver un moyen de transformer les instances et solutions (un codage permettant de représenter les séquences d'entiers sous une autre forme). La première idée venant à l'esprit est un codage en lettres pour fonctionner exactement de la même manière que le tutoriel seq2seq de traduction.

## 2.2.1 Codage ASCII

ASCII (*American Standard Code for Information Interchange*) est une norme informatique de codage de caractères. C'est la norme la plus populaire. On peut y représenter notamment les textes en anglais. Voici un exemple de table ASCII :

Dec	Hx	Oct	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	MUL (null)	32	20	040	#32;	Space	64	40	100	#64;	@	96	60	140	#96;	`
1	1	001	SOH (start of heading)	33	21	041	#33;	!	65	41	101	#65;	A	97	61	141	#97;	a
2	2	002	STX (start of text)	34	22	042	#34;	"	66	42	102	#66;	B	98	62	142	#98;	b
3	3	003	ETX (end of text)	35	23	043	#35;	#	67	43	103	#67;	C	99	63	143	#99;	c
4	4	004	EOT (end of transmission)	36	24	044	#36;	\$	68	44	104	#68;	D	100	64	144	#100;	d
5	5	005	ENQ (enquiry)	37	25	045	#37;	%	69	45	105	#69;	E	101	65	145	#101;	e
6	6	006	ACK (acknowledge)	38	26	046	#38;	&	70	46	106	#70;	F	102	66	146	#102;	f
7	7	007	BEL (bell)	39	27	047	#39;	'	71	47	107	#71;	G	103	67	147	#103;	g
8	8	010	BS (backspace)	40	28	050	#40;	(	72	48	110	#72;	H	104	68	150	#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	#41;	)	73	49	111	#73;	I	105	69	151	#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	#42;	*	74	4A	112	#74;	J	106	70	152	#106;	j
11	B	013	VT (vertical tab)	43	2B	053	#43;	+	75	4B	113	#75;	K	107	71	153	#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	#44;	,	76	4C	114	#76;	L	108	72	154	#108;	l
13	D	015	CR (carriage return)	45	2D	055	#45;	-	77	4D	115	#77;	M	109	73	155	#109;	m
14	E	016	SO (shift out)	46	2E	056	#46;	.	78	4E	116	#78;	N	110	74	156	#110;	n
15	F	017	SI (shift in)	47	2F	057	#47;	/	79	4F	117	#79;	O	111	75	157	#111;	o
16	10	020	DLE (data link escape)	48	30	060	#48;	0	80	50	120	#80;	P	112	76	160	#112;	p
17	11	021	DC1 (device control 1)	49	31	061	#49;	1	81	51	121	#81;	Q	113	77	161	#113;	q
18	12	022	DC2 (device control 2)	50	32	062	#50;	2	82	52	122	#82;	R	114	78	162	#114;	r
19	13	023	DC3 (device control 3)	51	33	063	#51;	3	83	53	123	#83;	S	115	79	163	#115;	s
20	14	024	DC4 (device control 4)	52	34	064	#52;	4	84	54	124	#84;	T	116	80	164	#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	#53;	5	85	55	125	#85;	U	117	81	165	#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	#54;	6	86	56	126	#86;	V	118	82	166	#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	#55;	7	87	57	127	#87;	W	119	83	167	#119;	w
24	18	030	CAN (cancel)	56	38	070	#56;	8	88	58	130	#88;	X	120	84	168	#120;	x
25	19	031	EM (end of medium)	57	39	071	#57;	9	89	59	131	#89;	Y	121	85	169	#121;	y
26	1A	032	SUB (substitute)	58	3A	072	#58;	:	90	5A	132	#90;	Z	122	86	170	#122;	z
27	1B	033	ESC (escape)	59	3B	073	#59;	;	91	5B	133	#91;	[	123	87	171	#123;	{
28	1C	034	FS (file separator)	60	3C	074	#60;	<	92	5C	134	#92;	\	124	88	172	#124;	
29	1D	035	GS (group separator)	61	3D	075	#61;	=	93	5D	135	#93;	]	125	89	173	#125;	}
30	1E	036	RS (record separator)	62	3E	076	#62;	>	94	5E	136	#94;	^	126	90	174	#126;	~
31	1F	037	US (unit separator)	63	3F	077	#63;	?	95	5F	137	#95;	_	127	91	175	#127;	DEL

Source: [www.LookupTables.com](http://www.LookupTables.com)

Figure 3 – Table ASCII



La dernière colonne montre la valeur ASCII de chaque caractère (les 32 premiers caractères sont des caractères de contrôle, c'est-à-dire non imprimables). Les 4 premières colonnes représentent respectivement les conversions décimales, hexadécimales, octales, HTML.

Pour le moment, nos instances et solutions sont représentées par des entiers (donc elles sont dans la première colonne). Nous avons donc cherché à établir la conversion décimale → caractère ASCII.

la transformation des jobs en mots a fonctionné. Par exemple la séquence "64 98 119 48" est convertie en "@ b w 0". Pour construire le mot, il reste seulement à supprimer les espaces et on obtient "@bw0". La conversion inverse fonctionne aussi très bien pour cette chaîne puisqu'on a juste à séparer chaque caractère par des espaces et les reconvertir un par un en nombres décimaux. On résout donc le problème que l'on avait dans le premier test où l'on conservait les données brutes.

Cependant, un problème a été constaté pour plusieurs valeurs ASCII : 11, 12, 13, 28, 29, 30, 31, 32.

Si l'une de ces valeurs est présente dans la séquence d'origine, la conversion décimale → ASCII fonctionnera mais pas la conversion inverse donc on ne retrouvera pas le mot de départ.

Prenons l'exemple d'une séquence de départ contenant la valeur 10 :

D'après la table, la valeur ASCII de 10 est "nouvelle ligne". Le programme va donc représenter cette conversion par un retour à la ligne, puis continuer à convertir la suite du mot sur une nouvelle ligne.

Lors de la conversion inverse, le programme ne prendra que la première ligne puisqu'il est censé transformer une chaîne de caractère, et le caractère "\n" va indiquer la fin de la chaîne de caractère. Il manquera donc une partie de la séquence d'origine.

Il faudrait alors effectuer un contrôle et un traitement spécifique pour chaque valeur qui pose problème. C'est possible mais c'est une tâche plutôt longue et fastidieuse. Nous avons donc décidé d'abandonner ce codage.

### 2.2.2 Codage hexadécimal

Le codage hexadécimal est une alternative pour pallier le problème du codage ASCII. En effet, en observant la colonne 2 de la table ASCII, il n'y a pas de caractères de contrôle.

Cependant, pour des raisons de clarté nous avons décidé de trouver un codage permettant de convertir les séquences de chiffres en lettres uniquement pour se rapprocher au mieux du tutoriel TensorFlow et utiliser des mots composés de lettres.

### 2.2.3 Codage personnalisé

Enfin, un dernier codage a été proposé. Le principe est d'attribuer une lettre de l'alphabet à un entier. Une fois toutes les lettres utilisées, on rajoute une lettre pour que chaque entier puisse être transformé en une combinaison de lettres unique.

Le codage est de la forme :

$$1 \rightarrow a, 2 \rightarrow b, \dots, 27 \rightarrow aa, \dots$$

Pour le codage inverse :

$$a \rightarrow 1, b \rightarrow 2, \dots, aa \rightarrow 27, \dots$$



Remarque : le 0 est codé par la chaîne "zero", et inversement.

Avec ce codage, la conversion fonctionne des 2 côtés. Par exemple, la séquence "1 26 7 3 28" est convertie en "a z g c ab". Pour pouvoir toujours identifier les valeurs séparées par les espaces, on rajoute le caractère "\_" entre chaque lettre. Le mot formé est ainsi : "a\_z\_g\_c\_ab".

Une fois tous les mots de la phrase convertis, il reste à rajouter les 2 tokens "<start>" et "<end>" dans la chaîne.

Pour une phrase avec 2 jobs, on aura par exemple la séquence suivante :

```
"<start> a_z_g_c_ab ba_v_t_c_w <end>"
```

Cette séquence peut facilement être reconvertie en chiffres. Il suffira de :

- Supprimer les 2 tokens
- Traduire chaque mot séparé par un espace
- Remplacer les "\_" par des espaces au sein de chaque mot

Le mécanisme fonctionne exactement de la même manière pour une séquence d'entiers qui représente une solution.

### 2.3 Entraînement du modèle

La phase d'entraînement du modèle est similaire à celui du tutoriel TensorFlow. Une fois que le dataset a été pré-traité et que les instances et solutions ont été converties en phrases de mots, elles peuvent être passées en entrée au modèle pour qu'il puisse apprendre.

Les réseaux de neurones encodeur et décodeur, la manière dont le modèle a été entraîné ainsi que la fonction de perte du tutoriel TensorFlow ont donc été directement repris sans modification.

De plus, les mêmes paramètres ont été appliqués :

- Taille du lot = 64
- Nombre de couches cachées = 256
- Nombre de neurones dans chaque couche = 1024
- Nombre d'apprentissages sur le dataset complet = 10

Voici sur les 2 captures suivantes les différentes itérations (epochs) effectuées lors de l'apprentissage ainsi que l'évolution de la valeur de la fonction d'erreur (loss). Nous pouvons constater que l'erreur diminue et tend vers 0. Cela signifie donc que le modèle a bien appris :

(Par manque de temps, l'étape de validation du modèle n'a pas été implémentée. Nous ne pouvons donc pas évaluer la précision du modèle)

```

Epoch 1 Batch 0 Loss 5.3143
Epoch 1 Batch 100 Loss 2.9358
Epoch 1 Batch 200 Loss 2.7744
Epoch 1 Loss 3.1828
Time taken for 1 epoch 441.43631505966187 sec

Epoch 2 Batch 0 Loss 2.7740
Epoch 2 Batch 100 Loss 2.8422
Epoch 2 Batch 200 Loss 2.6984
Epoch 2 Loss 2.8001
Time taken for 1 epoch 434.3580894470215 sec

Epoch 3 Batch 0 Loss 2.7012
Epoch 3 Batch 100 Loss 2.7166
Epoch 3 Batch 200 Loss 2.6449
Epoch 3 Loss 2.6768
Time taken for 1 epoch 434.7636082172394 sec

Epoch 4 Batch 0 Loss 2.5420
Epoch 4 Batch 100 Loss 2.4164
Epoch 4 Batch 200 Loss 2.3471
Epoch 4 Loss 2.4144
Time taken for 1 epoch 437.1547648906708 sec

Epoch 5 Batch 0 Loss 2.0566
Epoch 5 Batch 100 Loss 1.9507
Epoch 5 Batch 200 Loss 1.8440
Epoch 5 Loss 1.9449
Time taken for 1 epoch 434.3743243217468 sec

```

Figure 4 – Affichage en console lors de l'entraînement du modèle partie 1

```

Epoch 6 Batch 0 Loss 1.4686
Epoch 6 Batch 100 Loss 1.1876
Epoch 6 Batch 200 Loss 1.0591
Epoch 6 Loss 1.2106
Time taken for 1 epoch 441.14523339271545 sec

Epoch 7 Batch 0 Loss 0.6318
Epoch 7 Batch 100 Loss 0.5112
Epoch 7 Batch 200 Loss 0.4040
Epoch 7 Loss 0.4794
Time taken for 1 epoch 438.8920154571533 sec

Epoch 8 Batch 0 Loss 0.1951
Epoch 8 Batch 100 Loss 0.1164
Epoch 8 Batch 200 Loss 0.2396
Epoch 8 Loss 0.1519
Time taken for 1 epoch 434.2444951534271 sec

Epoch 9 Batch 0 Loss 0.0785
Epoch 9 Batch 100 Loss 0.0478
Epoch 9 Batch 200 Loss 0.0609
Epoch 9 Loss 0.0511
Time taken for 1 epoch 436.69182419776917 sec

Epoch 10 Batch 0 Loss 0.0272
Epoch 10 Batch 100 Loss 0.0222
Epoch 10 Batch 200 Loss 0.0135
Epoch 10 Loss 0.0151
Time taken for 1 epoch 437.6304793357849 sec

```

Figure 5 – Affichage en console lors de l'entraînement du modèle partie 2

## 2.4 Prédiction

Cette phase est similaire à la phase de traduction du tutoriel TensorFlow. La seule modification a lieu au niveau de la fonction utilisée pour pré-traiter les données d'entrée que l'on souhaite traduire. Étant donné que l'on souhaite prédire la solution d'une instance, il faut donner en entrée au programme une instance en respectant le format du dataset, à savoir :

`(([Idinstance] : [[Job1][Job2]...])`

L'instance est ensuite pré-traitée comme pour la phase d'apprentissage, puis le modèle l'utilise et prédit une solution. La solution prédite est aussi de la forme "mots de lettres" et a besoin d'être reconvertie en une séquence d'entiers.

Un exemple de prédiction est disponible dans la section 5 (Annexe F).

## 2.5 Résultats obtenus

Après différents tests avec des instances du dataset d'entraînement, on a remarqué que le modèle prédit la solution optimale correspondante environ 9 fois sur 10.

En revanche, il a été constaté que si l'on passe en entrée au modèle une instance inconnue (non présente dans le dataset d'entraînement), une erreur est présente.

En effet, lors de la phase d'entraînement, le modèle enregistre chaque mot. C'est-à-dire que si l'on prédit une phrase avec un mot qu'il n'a jamais appris (donc qu'il ne connaît pas), il ne sera pas capable de l'interpréter. C'est un problème puisque dans notre format d'instance qui est "`(([Idinstance] : [[Job1][Job2]...])`", le premier vecteur à gauche des ":" contient les informations sur l'instance, dont un identifiant, ce qui fait que le premier mot de chaque instance apprise est différent.

Pour que le modèle fonctionne, il faut donc lui donner en entrée une instance qu'il a déjà appris.

## 3 Implémentation d'un autre modèle : seq2point

Un autre modèle a été mis en place : le modèle sequence-to-point (seq2point). Tout le principe est exactement similaire au modèle sequence-to-sequence développé. La seule chose qui change est la structure du dataset et la séquence prédite.

L'idée était ici de conserver dans l'instance uniquement les informations sur les jobs, et de prédire seulement la valeur de la solution optimale.

Cela a nécessité de construire un nouveau fichier dataset avec la structure suivante :

`[[Job1][Job2]...] | Solution`

Les étapes de préparation du dataset, d'entraînement et validation n'ont pas changé par rapport au premier modèle implémenté.

### 3.1 Résultats obtenus

Après différents tests avec des instances du dataset d'entraînement, on a remarqué que le modèle prédit la solution optimale correspondante environ 9 fois sur 10.

On retrouve le même problème d'erreur que dans le modèle précédent si l'on passe en entrée une instance avec un mot inconnu.

A la différence du modèle seq2seq, l'instance est seulement composée des jobs. On peut tout de même créer une nouvelle instance composée de plusieurs jobs d'autres instances. C'est ce que nous avons fait pour tester la qualité d'apprentissage du modèle seq2point et ainsi vérifier si ce dernier est capable de prédire la solution d'une instance qu'il n'a jamais vu auparavant.

Etant donné que nous ne connaissons pas la solution optimale de cette nouvelle instance, nous l'avons calculé avec le programme PLNE. Une fois cela fait, nous avons passé l'instance en entrée au modèle seq2point.

Malheureusement, pour l'instance passée en entrée, la prédiction du modèle ne correspondait pas à la valeur de la solution optimale (la valeur de la prédiction était plus élevée).

Comme pour le modèle seq2seq, le modèle seq2point n'est efficace que si on lui donne en entrée une instance qu'il a déjà appris.

# 6

## Bilan et conclusion

### 1 Travail réalisé

Concernant la phase de recherche et d'analyse, une grosse étape de documentation/ compréhension du problème d'ordonnancement et du domaine du Machine Learning ont été réalisées pour s'approprier au mieux le sujet. Les étapes de prototypages m'ont de plus permis de vraiment comprendre le contexte du problème d'ordonnancement ainsi que le modèle. L'état de l'art ainsi que l'étude de faisabilité des techniques de Machine Learning étaient nécessaires pour essayer de trouver la technique la plus adaptée au problème.

La phase de développement avait pour objectif d'implémenter un algorithme de Machine Learning, plus précisément un modèle de Deep Learning (sequence-to-sequence) qui est un réseau de neurones. Les tests réalisés en fin de développement ont permis d'en conclure que les résultats obtenus ne sont malheureusement pas ceux espérés : le modèle entraîné est capable de prédire la solution d'une instance qu'il connaît déjà, mais pas pour une instance jamais rencontrée.

Le croisement entre les domaines de Machine Learning et de recherche opérationnelle est relativement récent. Il n'existe donc pas énormément de documentation et d'expérimentation sur le sujet. Cependant nous obtenons quand même des résultats qui pourront être utilisés ou servir de base pour de futurs projets similaires.

### 2 Travail restant/amélioration

L'objectif du PRD a été atteint. Le modèle sequence-to-sequence a pu être utilisé pour prédire des solutions d'instances.

Même si l'on obtient pas les résultats espérés pour n'importe quelle instance, ces derniers nous montrent que le modèle prédit de manière instantanée la solution pour une instance pour laquelle il a déjà appris cette solution. Ces résultats pourront être utilisés lors d'un futur projet ou bien pour la reprise de ce PRD, notamment en changeant de réseau de neurones ou de technique de Machine Learning, ou encore en essayant de trouver une alternative à la structure actuelle des instances pour ainsi avoir moins de paramètres.

Quelques améliorations supplémentaires peuvent être ajoutées au projet :

- Création d'un "checker" pour le programme PLNE permettant de vérifier de manière automatique que le modèle est bien implémenté (contraintes non violées, solution trouvée = solution optimale, etc...).
- Création d'une interface graphique pour le programme PLNE.
- Création d'une interface graphique pour le programme sequence-to-sequence.
- Ajout de l'étape de validation dans le programme sequence-to-sequence afin d'évaluer la précision (*accuracy*) du modèle de Deep Learning.

### 3 Bilan sur la planification du S9

Etant donné que toutes les tâches définies sont liées, un retard sur une tâche entraîne un retard sur toutes les tâches suivantes.

Les différences entre le diagramme de GANTT prévisionnel et le diagramme de GANTT réel sont les suivantes :

- Retard de 4 jours sur le prototypage du premier modèle PLNE. Cela est notamment dû à la prise en main d'un nouveau langage de programmation (Python) et de la librairie utilisée pour implémenter le modèle (docplex).
- Retard d'un jour sur la rédaction des spécifications.
- Ajout d'une nouvelle tâche après la réalisation de l'état de l'art sur les techniques de Machine Learning. Elle a pour but l'étude sur la faisabilité des méthodes de Machine Learning sélectionnées. Cette tâche a été oubliée lors de la planification. Elle constitue une étape essentielle pour essayer d'adapter les méthodes sélectionnées au problème d'ordonnancement, et ainsi éliminer celles qui ne sont pas adéquates.
- Retard sur la rédaction du rapport final à mi-parcours (dû aux retards pris sur les tâches précédentes).

Remarque : les dates limites de remise des livrables ont été prolongées.

Les diagrammes de GANTT prévisionnel et réel du S9 sont disponibles respectivement dans les sections 2.1 (Annexe A), 2.2 (Annexe A).

### 4 Bilan sur la planification du S10

Les différences entre le diagramme de GANTT prévisionnel et le diagramme de GANTT réel sont les suivantes :

- Ajout d'une nouvelle tâche permettant de s'approprier le code d'un exemple de modèle sequence-to-sequence.
- Avance de 2 jours sur la création du format de fichier d'entrée du programme de Machine Learning.
- Retard de plusieurs jours sur l'implémentation du modèle sequence-to-sequence. Cela est dû à divers problèmes (compréhension/adaptation du code de base, problèmes de codages des instances/solutions lors du pré-traitement, le modèle a "mal appris").
- Les tests unitaires/fonctionnels ainsi que la validation des résultats ont pris moins de temps que prévu.
- Ajout d'une nouvelle tâche de rédaction de documentation utilisateur/développeur/installation.
- La rédaction du rapport final et la préparation de la soutenance finale ont pris moins de temps que prévu.

Remarque : la date limite prévisionnelle de remise des livrables a été volontairement placée au 26 mars afin d'avoir une marge d'environ 1-2 semaines en cas de retard sur le projet. La date limite finale de remise des livrables a été placée au 12 avril.

Les diagrammes de GANTT prévisionnel et réel du S10 sont disponibles respectivement dans les sections 3.1 (Annexe A), 3.2 (Annexe A).

## 5 Bilan sur la qualité

La notion de qualité dans un projet est primordiale. La reprise et l'utilisation du système dans ce PRD dépend de la démarche adoptée en termes de qualité et de mise en oeuvre.

La qualité inclut tout d'abord la qualité du code produit mais aussi des éléments de modélisation associés, de la documentation, du versionning, ou encore des tests.

Afin de pouvoir reprendre mon projet pour l'utiliser et l'améliorer, j'ai donc mis en place/rédigé les éléments suivants :

- Hébergement de mon projet sur GitHub pour versionner mon code.
- Documentation dans le code sous forme de Pydoc (disponible dans la console mais aussi sous forme de fichiers html portant le nom des programmes).
- Documentation développeur/utilisateur/d'installation du projet avec des diagrammes de modélisation, versions des logiciels utilisés, captures d'écrans des programmes.
- Document de tests regroupant les différents tests unitaires mis en place ainsi que les tests fonctionnels effectués.

## 6 Bilan auto-critique sur la gestion de projet

Concernant la gestion de projet de ce PRD, j'ai pu remarquer à quel point il est important d'utiliser différents outils de gestion de projet et de mettre en place un planning prévisionnel afin de le respecter au mieux.

Le fait de découper son projet en tâches et estimer leur importance ainsi que leur durée est nécessaire puisqu'il permet de fractionner et répartir son projet sur toute sa durée. Une organisation de telle manière permet de plus de rester toujours focalisé sur les attentes du client, et respecter au mieux les délais.

Le suivi sur l'avancement des tâches avec Trello, ainsi que mes comptes-rendus hebdomadaires m'ont permis de rester focalisé sur les tâches qui étaient en cours, et d'y relever les questions et problèmes éventuels pour en discuter avec mon encadrant par la suite.

Enfin, la mise à jour du diagramme de GANTT tout au long du projet est primordiale pour gérer les événements imprévisibles qui impactent sur la durée/les tâches du projet.

Je me suis plutôt bien organisé tout au long de mon PRD. Cependant lors de mon prochain projet, je ferai en sorte d'évaluer le temps de travail que nécessitent les tâches d'une manière plus précise, en y introduisant plusieurs paramètres afin d'avoir une estimation plus précise sur la durée de chaque tâche.

## 7 Conclusion générale

Ce projet m'a permis d'améliorer mes compétences de manière générale au niveau de la gestion d'un projet. De la phase de recherche et d'analyse à la phase de développement, en passant

par la gestion de projet, j'ai ainsi pu gagner en autonomie et en adaptabilité en prenant des initiatives et différentes décisions.

Je tiens à remercier M. Boukhalfa Zahout pour avoir supervisé mon PRD, qui m'a encadré et apporté son aide tout au long de celui-ci.



## Annexes

# A

## Gestion de projet

Une gestion de projet pseudo-agile a été mise en place. Des rendez-vous hebdomadaires sont fixés avec l'encadrant afin de faire un point sur l'avancement du projet et sur les problèmes rencontrés. De plus des dates ont été fixées au début du projet pour la remise des livrables demandés par l'encadrant :

- Prototypes modèles PLNEs en python
- Etat de l'art des méthodes de Machine Learning
- Cahier des spécifications
- Programme de Machine Learning
- Documentation
- Rapport final et diaporama

Différents outils de gestion de projet ont été utilisés avec notamment :

- Microsoft Project pour la planification du projet avec un diagramme de GANTT
- Trello pour représenter et suivre l'avancement des tâches du projet (à faire/en cours/terminé)
- Git pour le versioning du projet

### 1 Découpage en tâches

Voici l'outil de gestion de projet Trello utilisé au cours du projet. C'est ici que j'ai pu suivre et contrôler l'avancement de mes tâches/sous-tâches :

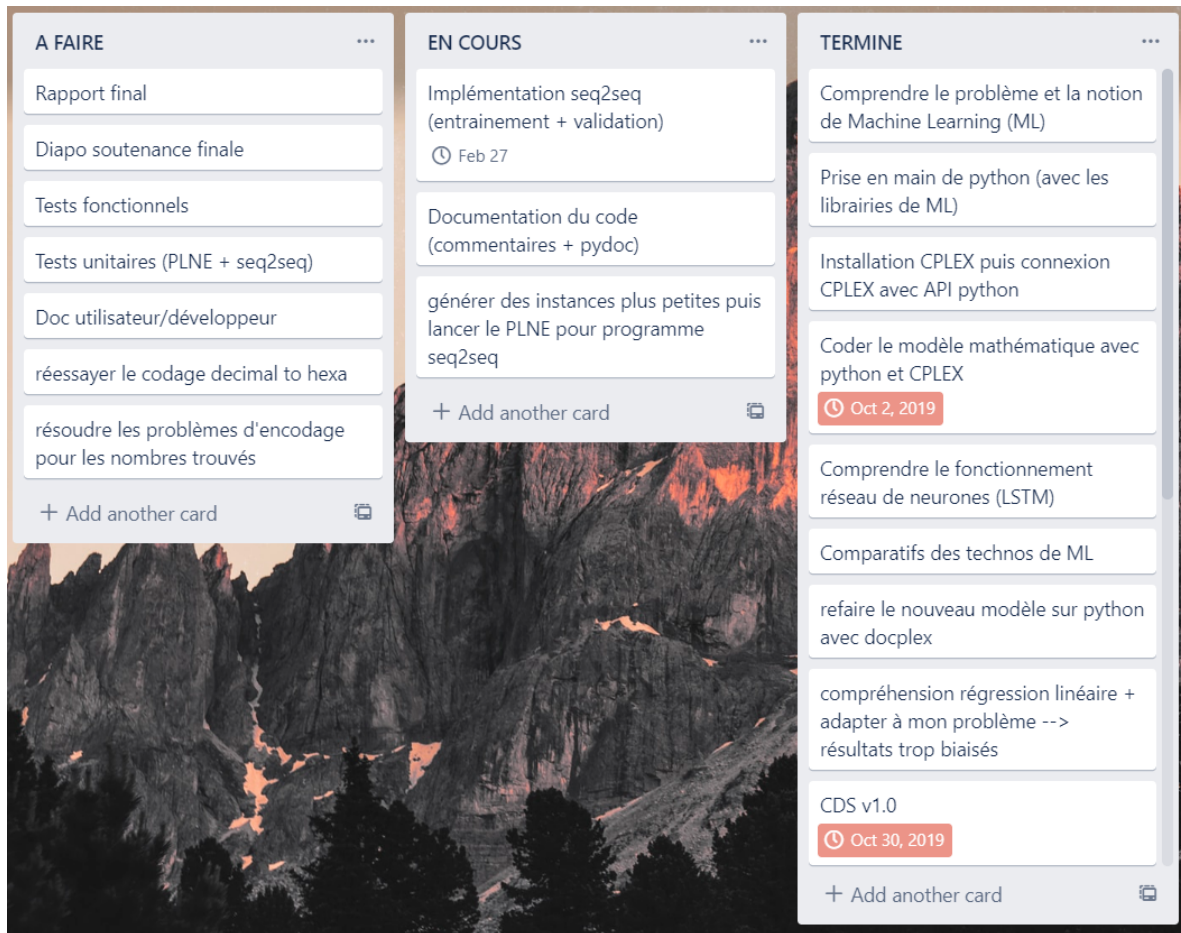


Figure 1 – Capture d'écran du Trello du projet

### 1.1 Tâche 1 : gestion et versioning du projet

- Description de la tâche : cette tâche a pour but de mettre en places les outils de gestion de projet (Git, trello, ms project, google drive).
- Durée : 1 heure.

### 1.2 Tâche 2 : compréhension du sujet

- Description de la tâche : cette tâche a pour but de s'approprier le sujet (contexte, objectifs). Cela inclut de la documentation et de la lecture sur des problèmes d'ordonnancement et du Machine Learning.
- Durée : 15 jours.

### 1.3 Tâche 3 : prototypage PLNE1

- Description de la tâche : cette tâche a pour but d'implémenter le prototypage d'un premier modèle PLNE.
- Durée : 9 jours.

**1.4 Tâche 4 : prototypage PLNE2**

- Description de la tâche : cette tâche a pour but d'implémenter le prototypage d'un deuxième modèle PLNE.
- Durée : 4 jours.

**1.5 Tâche 5 : rédaction du cahier de spécifications**

- Description de la tâche : cette tâche a pour but de rédiger le cahier de spécifications. Ce dernier sera repris pour être amélioré jusqu'à être validé par l'encadrant.
- Durée : 10 jours.

**1.6 Tâche 6 : rédaction de l'état de l'art**

- Description de la tâche : cette tâche a pour but de réaliser l'état de l'art sur les différentes techniques de Machine Learning.
- Durée : 5 jours et +.

**1.7 Tâche 7 : faisabilité des méthodes de Machine Learning**

- Description de la tâche : cette tâche a pour but de réaliser une étude sur la faisabilité des différentes méthodes de Machine Learning retenues dans la phase d'état de l'art.
- Durée : 3 jours.

**1.8 Tâche 8 : préparation de la soutenance mi-parcours**

- Description de la tâche : cette tâche a pour but de préparer la soutenance mi-parcours.
- Durée : 4 jours.

**1.9 Tâche 9 : rédaction du rapport final mi-parcours**

- Description de la tâche : cette tâche a pour but de rédiger le rapport final à mi-parcours du projet.
- Durée : 8 jours.

**1.10 Tâche 10 : création du fichier d'entrée du programme de Machine Learning**

- Description de la tâche : cette tâche a pour but de réfléchir à une structuration du fichier (et de créer ce fichier) qui sera passée en entrée du programme de Machine Learning dans la phase d'apprentissage.
- Durée : 9 jours.

**1.11 Tâche 11 : implémentation du programme de Machine Learning : seq2seq**

- Description de la tâche : cette tâche a pour but d'implémenter le programme de Machine Learning. Cela inclut la partie "Apprentissage" du modèle et la partie "prédiction".
- Durée : 36 jours.

**1.12 Tâche 12 : tests fonctionnels/unitaires et documentation du code**

- Description de la tâche : cette tâche a pour but de réaliser les tests fonctionnels/unitaires sur les programme de Machine Learning en lançant des prédictions, et sur le PLNE. Elle consiste en plus à documenter les fichiers Python sous forme de Pydoc.
- Durée : 1 jour.

**1.13 Tâche 13 : Documentation utilisateur/développeur/installation**

- Description de la tâche : cette tâche a pour but de réaliser les documentations développeur, utilisateur, et d'installation utiles pour la reprise du projet.
- Durée : 3 jours.

**1.14 Tâche 14 : préparation de la soutenance finale**

- Description de la tâche : cette tâche a pour but de préparer la soutenance finale.
- Durée : 5 jours.

**1.15 Tâche 15 : Rédaction du rapport final**

- Description de la tâche : cette tâche a pour but de rédiger le rapport final du projet.
- Durée : 5 jours.

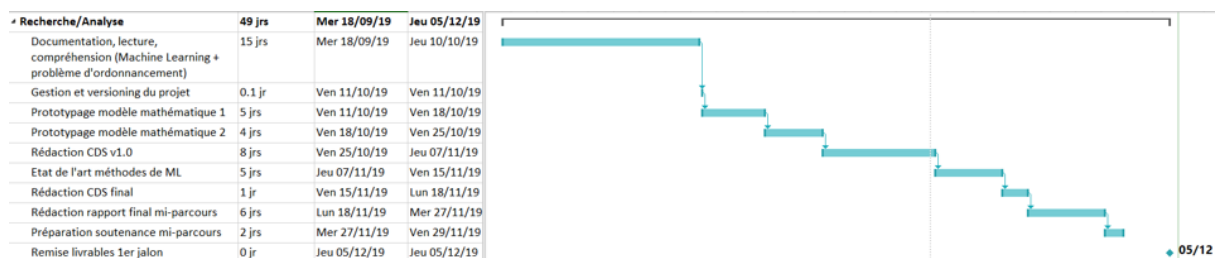
**2 Plannings du S9****2.1 Planning prévisionnel du S9**

Figure 2 – Diagramme de GANTT prévisionnel S9

## 2.2 Planning réel du S9

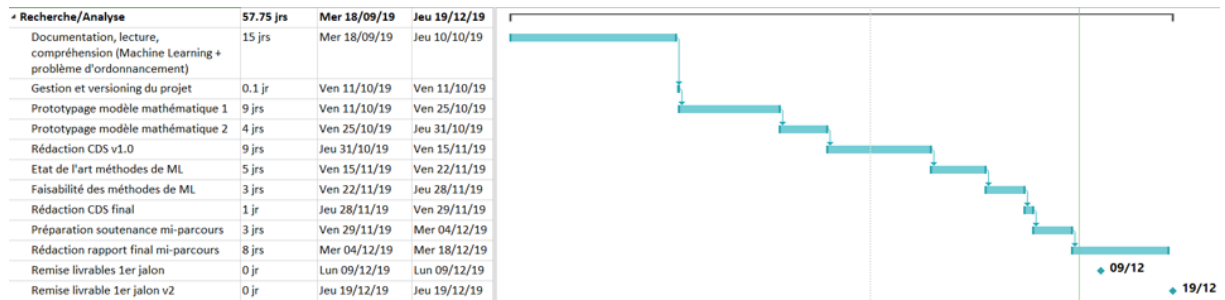


Figure 3 – Diagramme de GANTT réel S9

## 3 Plannings du S10

### 3.1 Planning prévisionnel du S10



Figure 4 – Diagramme de GANTT prévisionnel S10

### 3.2 Planning réel du S10

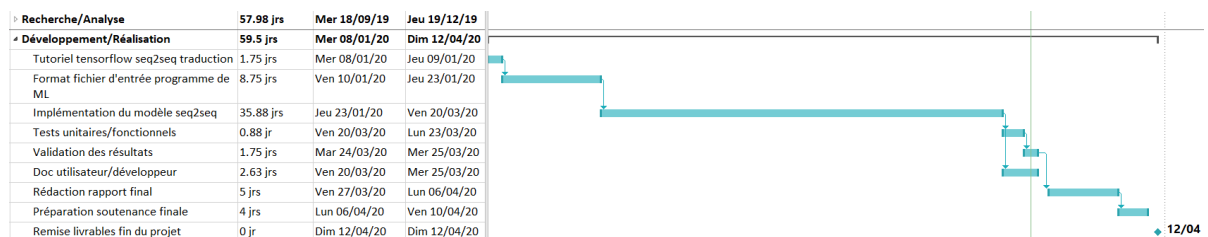


Figure 5 – Diagramme de GANTT réel S10

# B

## Description des interfaces externes du logiciel

### 1 Interfaces matériel/logiciel

Aucune interface matériel/logiciel n'est prévue. Cependant si l'utilisateur souhaite entraîner le modèle de Machine Learning avec de nombreuses instances ou bien lancer l'exécution d'une méthode exacte sur une grande instance, il devra se munir d'un ordinateur doté d'une bonne puissance de calcul afin d'obtenir un résultat avec un temps de calcul acceptable.

### 2 Interfaces homme/machine

Aucune interface homme/machine n'est souhaitée pour le moment. L'interface sera donc en mode console avec un menu au lancement qui proposera différents choix.

### 3 Interfaces logiciel/logiciel

Aucune interaction n'est nécessaire avec d'autres logiciels.

# C

## Spécifications fonctionnelles

### 1 Définition de la fonction 1 : importInstance

1. Identification de la fonction 1 :  
Cette fonction permet d'importer un fichier d'instance passé en paramètre. Sa priorité est primordiale.
2. Description de la fonction 1 :  
Cette fonction prend en entrée un fichier d'instance et retourne un tableau contenant toutes les données du fichier. Chaque ligne du fichier correspond à une case du tableau.

### 2 Définition de la fonction 2 : exportResultatsCSV

1. Identification de la fonction 2 :  
Cette fonction permet d'exporter le résultat obtenu par une méthode de résolution exacte dans un fichier csv. Ce résultat est constitué de la solution optimale ainsi que des informations sur l'instance pour laquelle le calcul a été lancé. Sa priorité est primordiale.
2. Description de la fonction 2 :  
Cette fonction prend en entrée les informations sur une instance et sa solution optimale correspondante, et stocke les informations dans un fichier CSV. Ce fichier correspond au fichier d'entrée du programme de Machine Learning en phase d'apprentissage.

### 3 Définition de la fonction 3 : apprentissage

1. Identification de la fonction 3 :  
Cette fonction permet d'entraîner le modèle de Machine Learning avec des données. Sa priorité est primordiale.
2. Description de la fonction 3 :  
Cette fonction prend en entrée un fichier csv qui contient pour chaque ligne des informations sur une instance avec la solution optimale correspondante. Ce fichier constitue donc la base d'apprentissage du modèle de Machine Learning. Il est créé à la suite de l'exécution d'une méthode de résolution exacte sur plusieurs instances. La précondition est donc que



le programme de résolution exacte doit avoir été exécuté au moins une fois pour créer le fichier d'entrée du modèle de Machine Learning. Le traitement principal dans cette fonction est d'indiquer au programme de Machine Learning que « pour tel instance passée en entrée, il doit prédire en sortie la solution optimale correspondante ».

#### 4 Définition de la fonction 4 : prediction

1. Identification de la fonction 4 :  
Cette fonction permet de lancer une prédiction en exécutant le modèle de Machine Learning entraîné. Sa priorité est primordiale.
2. Description de la fonction 4 :  
Cette fonction prend en entrée un fichier d'instance puis donne en sortie une solution (une matrice  $ixj$  : chaque ligne représente les jobs  $j$  ordonnancés sur la machine  $i$  (0 ou 1)). Le traitement dans cette fonction est que le modèle entraîné va prédire une solution en se basant sur les données qui lui ont servi de s'entraîner pour la phase d'apprentissage.

#### 5 Définition de la fonction 5 : exportSolutionInitiale

1. Identification de la fonction 5 :  
Cette fonction permet d'exporter le résultat obtenu par le modèle de Machine Learning lors d'une prédiction. Sa priorité est primordiale.
2. Description de la fonction 5 :  
Cette fonction prend en entrée une matrice  $ixj$  (chaque ligne représente les jobs  $j$  sur les machines  $i$ ) qui correspond à une solution initiale, et l'écrit dans un fichier.

#### 6 Définition de la fonction 6 : exportSolutionOptimale

1. Identification de la fonction 6 :  
Cette fonction permet d'exporter le résultat de la solution optimale (calculée par une méthode de résolution exacte) dans un fichier. Sa priorité est primordiale.
2. Description de la fonction 6 :  
Cette fonction prend en entrée la solution optimale et l'écrit dans un fichier.

# D

# Spécifications non fonctionnelles

## 1 Contraintes de développement et conception

Aucun langage de programmation n'était imposé. Il a été convenu avec la MOA d'utiliser le langage Python (v3.6.7) qui est très utilisé dans le domaine du Machine Learning. Le projet est réalisé sur Windows 10. L'environnement de développement intégré utilisé pour programmer en Python est PyCharm 2019.2.3 (Community Edition). Il n'existe pas de contrainte liée au système d'exploitation, tant que le langage Python est installé. Un dépôt git sera mis en place afin de gérer les versions de code (GitHub).

La structure des fichiers d'instance est imposée. Ces fichiers ont des extensions « .data », et respectent un format précis avec toutes les informations nécessaires pour lancer une résolution.

Le logiciel CPLEX (v12.8) a dû être installé puis une API Python a été mise en place afin de pouvoir programmer le modèle mathématique en Python et lancer une résolution par la méthode exacte afin de comprendre le problème à traiter (phase de prototypage).

Enfin, plus la machine utilisée pour exécuter les programmes de Machine Learning et de résolution exacte possèdera une bonne puissance de calcul, plus les temps d'exécution des programmes seront réduits.

La bibliothèque Python de Machine Learning Keras sera utilisée en se basant sur le framework TensorFlow (version 2).

## 2 Contraintes de fonctionnement et d'exploitation

### 2.1 Performances

Le programme de Machine Learning est censé prédire une solution initiale « correcte ». En passant cette solution à un programme de résolution exacte déjà développé (comme par exemple une méthode par décomposition), nous souhaitons améliorer les performances en termes de temps de calcul. En effet si une bonne solution de départ est passée en entrée au modèle de résolution exacte (c'est-à-dire qu'on aura trouvé une bonne phase d'initialisation pour ce modèle), alors ce modèle convergera plus rapidement vers la solution optimale. C'est pourquoi

nous souhaitons dans l'idéal obtenir un temps de réponse instantané lors du calcul de la solution optimale (par une méthode de décomposition par exemple).

Ce temps de calcul dépendra de la qualité du modèle de Machine Learning implémenté (trop/pas assez entraîné) mais aussi de la qualité de la solution retournée par celui-ci.

## 2.2 Capacités

Aucune limite n'est fixée au niveau de la taille des instances. Il serait tout de même intéressant de pouvoir réaliser des prédictions sur de grandes instances puisque ce sont elles qui posent le plus de problèmes actuellement étant donné les temps de calcul très longs qui sont nécessaires pour trouver la solution optimale.

## 2.3 Modes de fonctionnement

Le seul mode de fonctionnement est de lancer le programme Python en ligne de commande.

## 2.4 Contrôlabilité

Des messages de log seront affichés dans le programme Python pour avoir différentes informations sur les instances utilisées, les solutions optimales, les sorties du programme de Machine Learning, les temps de calcul.

## 2.5 Sécurité

Le code du projet sera utilisé uniquement au sein de Polytech par des enseignants chercheurs/doctorants. Aucune sécurité n'est donc à prévoir.

## 2.6 Intégrité

Il existe des situations non protégées au niveau de la phase d'apprentissage. En effet, si on coupe le modèle pendant qu'il « apprend », alors il sera faussé puisqu'il ne possèdera pas toutes les informations. Il faut donc que la phase d'apprentissage soit réalisée du début à la fin sans arrêt manuel pour ne pas avoir de perte d'information. Si c'est le cas, il faut supprimer le modèle et relancer sa phase d'apprentissage. Au niveau de la prédiction, il n'y a aucun problème d'intégrité puisqu'une déconnexion imprévue du programme impliquera seulement que le modèle ne prédira rien. Il suffira donc de relancer le modèle pour obtenir une prédiction.

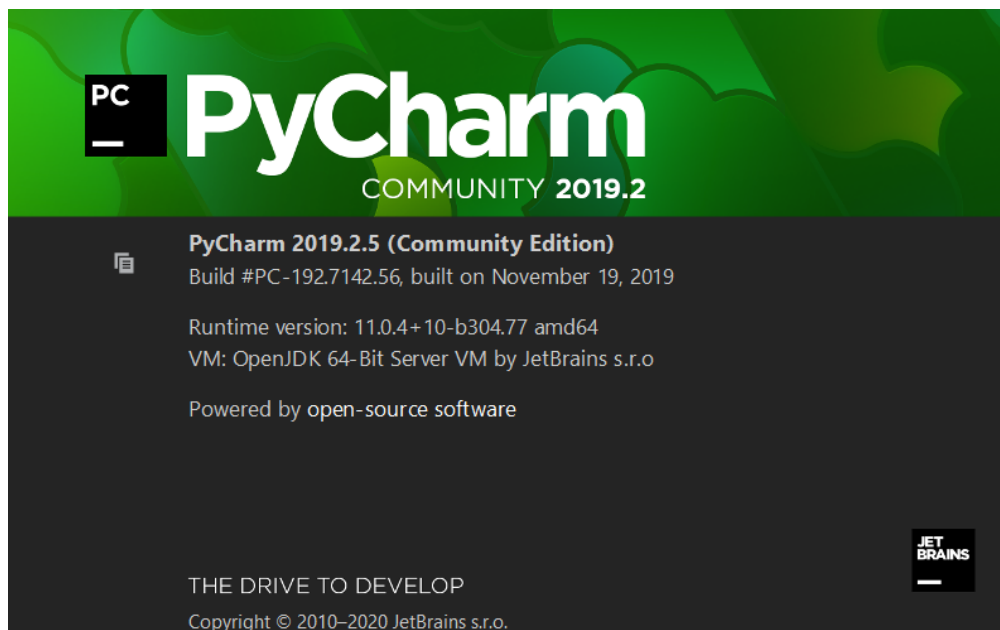
# E

## Documentation d'installation

Voici les étapes à suivre pour installer l'environnement de développement afin de mettre en place le projet sur un ordinateur en local et utiliser les différentes applications (PLNE, programme de Machine Learning) :

1. Installer Python v3.6.7 (<https://www.python.org/downloads/>).
2. (Optionnel) Installer un environnement de développement intégré comme PyCharm (<https://www.jetbrains.com/pycharm/>).

La configuration utilisée pour le projet est la suivante :



**Figure 1** – Configuration de PyCharm utilisée pour le projet

3. Installer une version payante de Cplex (v12.8 minimum). La version Cplex studio 12.8.0 a été utilisée pour le projet (<https://www.ibm.com/fr-fr/analytics/cplex-optimizer>). Pour installer Cplex, suivre les instructions d'installation de Cplex sur le site officiel d'IBM : [https://www.ibm.com/support/knowledgecenter/SSSA5P\\_12.8.0/ilog.odms.studio.help/Optimization\\_Studio/topics/COS\\_installing.html](https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_installing.html)
4. Une fois que Cplex est installé, mettre en place l'api python de Cplex afin de pouvoir

utiliser Cplex dans un programme python.

Pour mettre en place l'api python de Cplex, suivre les instructions sur le site officiel d'IBM :

[https://www.ibm.com/support/knowledgecenter/SSSA5P\\_12.7.0/ilog.odms.cplex.help/Cplex/GettingStarted/topics/set\\_up/Python\\_setup.html](https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.cplex.help/Cplex/GettingStarted/topics/set_up/Python_setup.html)

5. Le gestionnaire de paquets "pip" est inclut par défaut lors de l'installation de Python. Il permet d'installer et de gérer des paquets écrits en Python. La dernière étape est donc d'installer les librairies qui ont besoin d'être utilisées dans les scripts Python du projet avec la commande "pip install nom\_du\_paquet".

Remarques concernant la librairie TensorFlow :

- Si vous utiliser le programme de Machine Learning sur un ordinateur avec une carte graphique nvidia (Cudnn), il faut installer la version 2.0 de TensorFlow avec la commande "pip install tensorflow==2.0.0" (sinon il y aura une erreur avec Cudnn)
- Si vous souhaitez utiliser la carte graphique de la machine pour faire les calculs plus rapidement, il faut installer TensorFlow v2.0.0 gpu avec la commande "pip install tensorflow-gpu==2.0.0" (<https://www.tensorflow.org/install/gpu>)

## 1 Structure d'un fichier d'instance

Les fichiers d'instances contiennent des informations sur les jobs. Ils sont utilisés par les modèles de résolution exactes (PLNE) pour calculer la solution optimale.

Ils ont une extension ".data" et leur nom est défini comme ceci :

*instance\_NbMachines\_NbRessources\_CapaciteMaxRessources\_NbJobs*  
(ou bien : *instance\_IdInstance\_NbMachines\_NbRessources\_CapaciteMaxRessources\_NbJobs*)

Par exemple, pour une instance avec 4 machines disponibles, 3 ressources avec pour capacité maximale la valeur 2 et 100 jobs, le nom de l'instance sera :

*instance\_4\_3\_2\_100.data*  
(ou bien : *instance\_0\_4\_3\_2\_100.data*)

Voici le contenu de ce fichier d'instance :

```
4 3 2 100
0 0 1 2 1 2 28
1 0 1 1 2 2 21
2 0 5 2 1 2 542
3 0 11 2 2 2 1053
4 0 12 1 1 1 265
5 1 2 1 1 1 14
6 1 3 2 1 1 237
7 1 6 1 1 2 208
8 1 6 1 1 2 153
```

Figure 1 – Structure d'un fichier d'instance

Ligne 1 : nombre de machines, nombre de ressources, capacité maximale des ressources, nombre de jobs.

Ligne 2 et + : identifiant du job, date de début, date de fin, ressource 1, ressource 2, ressource 3, poids du job.

## 2 Programme PLNE

Ce script écrit en python permet, pour une ou plusieurs instances, de calculer la solution optimale, et d'exporter les résultats dans un fichier. Il ne possède pas d'interface graphique. Il s'exécute donc en ligne de commande.

Au lancement du script, 2 informations sont à renseigner :

- Saisir le chemin d'accès à un dossier contenant des instances (qui respectent le format de la partie précédente)
- Saisir l'emplacement et le nom de fichier dans lequel on souhaite exporter les résultats

Une fois que cela est fait, le PLNE calcule les solutions optimales de toutes les instances fournies, et exporte les résultats dans un fichier. Chaque ligne du fichier est composée d'une instance, et de sa solution optimale correspondante (nous y reviendrons dans la partie suivante).

Des informations sur la résolution de chaque instance sont affichées dans le terminal (nom de l'instance, valeur de la solution optimale, temps de calcul) au cours de l'exécution du programme.

Voici un exemple d'affichage de la console pour la résolution des 3 premières instances d'un dossier :

```
instance_0_2_3_2_10.data
* model probleme solved with objective = 31
status = integer optimal solution
time   = 0.031 s.
problem = MILP
gap     = 0%

iteration 1--- 0.5311625003814697 seconds ---

instance_10000_2_3_2_10.data
* model probleme solved with objective = 20
status = integer optimal solution
time   = 0.047 s.
problem = MILP
gap     = 0%

iteration 2--- 0.32811617851257324 seconds ---

instance_10001_2_3_2_10.data
* model probleme solved with objective = 35
status = integer optimal solution
time   = 0.031 s.
problem = MILP
gap     = 0%

iteration 3--- 0.2812645435333252 seconds ---
```

Figure 2 – Affichage en console lors de l'exécution du programme PLNE

### 3 Fichier résultats (dataset)

Comme dit précédemment, ce fichier de résultats est défini globalement comme ceci :

instance → solution optimale

Lors de l'exécution du programme PLNE, on a simplement recopié pour chaque instance ses informations (dates début, fin des jobs etc...), puis sa solution optimale. Ce qui donne donc un format comme ceci :

`(([Idinstance] : [[Job1][Job2]...]))(SolOptimale : [[Placement]JobsM1][Placement]JobsM2]...))`

Remarque : le "|" sert de séparateur entre l'instance et la solution.

Voici sur les 2 images suivantes une capture du fichier pour 3 instances (3 lignes) :

```
(( 1 2 3 2 10):[[33 35 2 1 2 3][ 6 15 1 2 2 6][27 57 2 2 2 4][ 6 45 1 2 1 9][10 36 2 2 1 7][33 65 2 1 1 3][38 53 2 2 2 3]
([ 2 2 3 2 10]:[[21 40 1 2 2 1][34 69 1 2 2 9][16 56 2 1 2 1][13 20 2 1 1 1][27 64 2 2 1 9][35 51 1 2 2 9][34 44 1 2 1 3]
([ 3 2 3 2 10]:[[19 35 2 2 1 10][16 33 2 1 2 4][32 52 1 1 2 6][ 0 10 1 2 1 4][25 46 2 1 2 3][11 22 2 2 1 5][27 39 1 1 2 2]
```

Figure 3 – Structure du fichier dataset (1ère partie)

```
[38 77 1 1 1 10][28 56 2 2 2 8][34 38 2 2 2 4]] | (31:[0 0 0 0 1 0 0 1 0 0][0 1 0 0 0 0 0 0 1 0])
[31 70 2 1 2 4][21 48 1 2 2 6][ 8 21 1 1 2 1]] | (20:[0 0 0 1 1 0 0 0 0 0][0 1 0 0 0 0 0 0 0 1])
[22 28 1 2 2 10][28 47 2 2 1 1][30 59 1 1 2 1]] | (35:[1 0 0 1 0 0 0 0 0 0][0 0 1 0 0 1 0 1 0 0])
```

Figure 4 – Structure du fichier dataset (2ème partie)

La structure de ce fichier peut paraître lourde et fastidieuse. Elle est cependant nécessaire puisque ce fichier de résultats est utilisé dans le programme sequence-to-sequence (qui est un réseau de neurones de type LSTM) lors de la phase d'apprentissage du modèle. On peut donc aussi parler de fichier dataset. Il va permettre d'indiquer au réseau de neurones que "pour telle instance, on obtient cette solution".

### 4 Programme sequence-to-sequence (seq2seq)

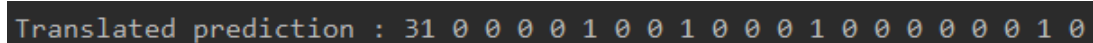
Le programme sequence-to-sequence s'exécute lui aussi en ligne de commande. Il peut être exécuté en "2 parties" :

- Phase d'apprentissage : vous venez d'exécuter le programme PLNE avec les instances de votre choix afin de construire votre fichier résultats. Vous souhaitez désormais faire apprendre le modèle pour qu'il puisse, une fois entraîné, prédire une solution à partir d'une instance passée en entrée.
- Phase de prédiction : quelqu'un a déjà réalisé la phase d'apprentissage à votre place. Le modèle est donc déjà entraîné : vous n'avez qu'à lancer le programme, et saisir une instance dans la console pour observer sa prédiction (Remarque : l'instance saisie doit respecter le format "([IdInstance] : [[Job1][Job2]... ])").

Le programme sequence-to-sequence n'a pas été conçu pour pouvoir être utilisé en phase d'apprentissage pour le moment. Vous avez donc seulement la possibilité de lancer une prédiction et d'observer les résultats.

Voici un exemple de prédiction correcte du modèle pour une instance à 2 machines et 10 jobs :



A screenshot of a terminal window with a dark background. The text 'Translated prediction : 31 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0' is displayed in a light-colored monospace font. The sequence consists of 20 elements: a decimal value '31' followed by 19 binary digits (0s and 1s).

```
Translated prediction : 31 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0
```

**Figure 5** – *Affichage d'une prédiction en console suite à l'exécution du programme seq2seq*

## 1 Présentation générale

### 1.1 Fonctionnalités du système

Le diagramme de cas d'utilisation suivant présente les fonctionnalités du système :

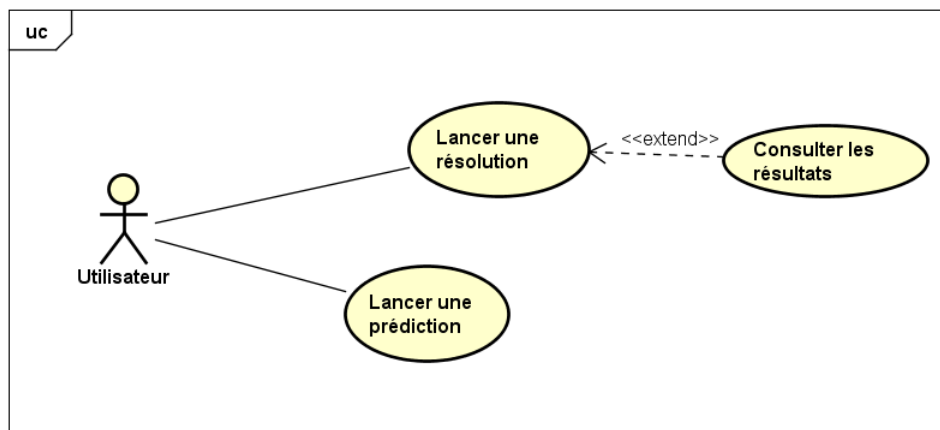


Figure 1 – Diagramme de cas d'utilisation

L'utilisateur a la possibilité de lancer une résolution avec une méthode exacte (PLNE) en important un ou plusieurs fichiers d'instances. Le programme calcule ensuite la solution optimale pour chaque instance, et exporte les résultats dans un fichier (chaque ligne du fichier correspondant à une instance et sa solution optimale correspondante) que l'utilisateur peut consulter.

La deuxième fonctionnalité du projet est de lancer une prédiction sur la solution d'une instance passée en paramètre dans le programme de Machine Learning (sequence-to-sequence).

## 1.2 Architecture globale du système

### 1.2.1 Phase de prédiction

Le schéma suivant représente l'architecture générale du système lors de la phase de prédiction :

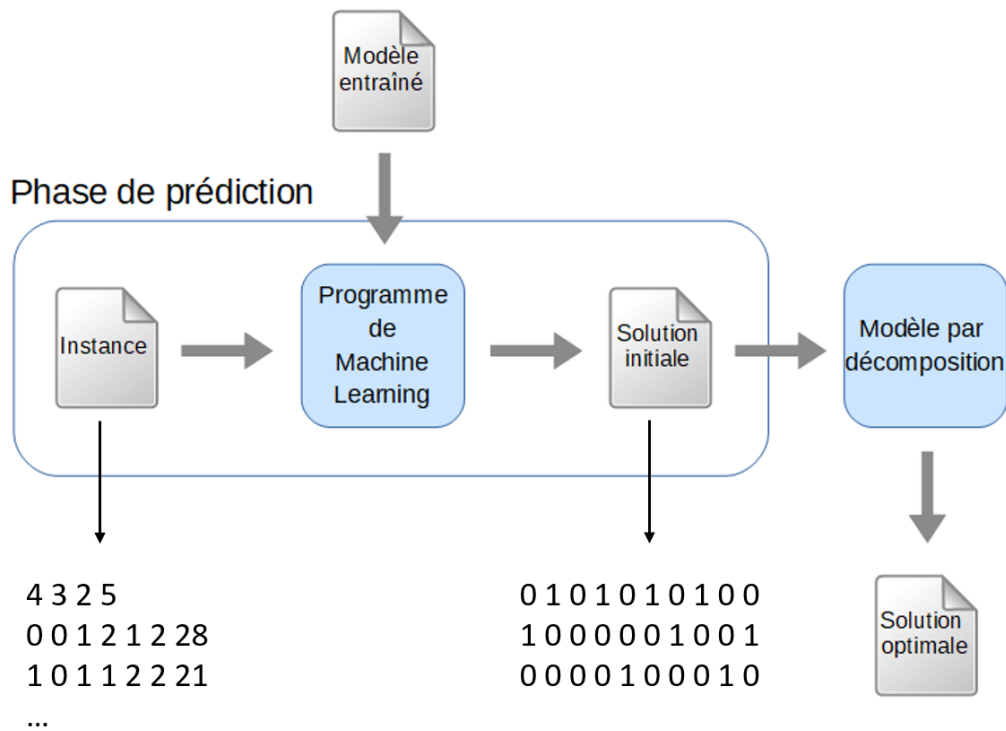


Figure 2 – Architecture générale du système lors de la phase de prédiction

Le programme de Machine Learning doit avoir été entraîné au préalable (la phase d'apprentissage est détaillée dans la partie suivante).

Le cheminement est le suivant :

1. Un fichier d'instance de jobs est passé en entrée au programme de Machine Learning.
2. Le programme de Machine Learning va ensuite prédire en sortie des vecteurs qui représentent une solution. C'est-à-dire les jobs ordonnancés sur les différentes machines (une ligne représente une machine  $i$  et une colonne représente un job  $j$  : si le job  $j$  est exécuté sur la machine  $i$ , alors la valeur est égale à 1, 0 sinon).

Remarque : si la solution prédite ne correspond pas déjà à la solution optimale, on peut la passer en entrée à un modèle par décomposition qui calculera la solution optimale.

### 1.2.2 Phase d'apprentissage

Voici l'architecture suivante pour la phase d'apprentissage :

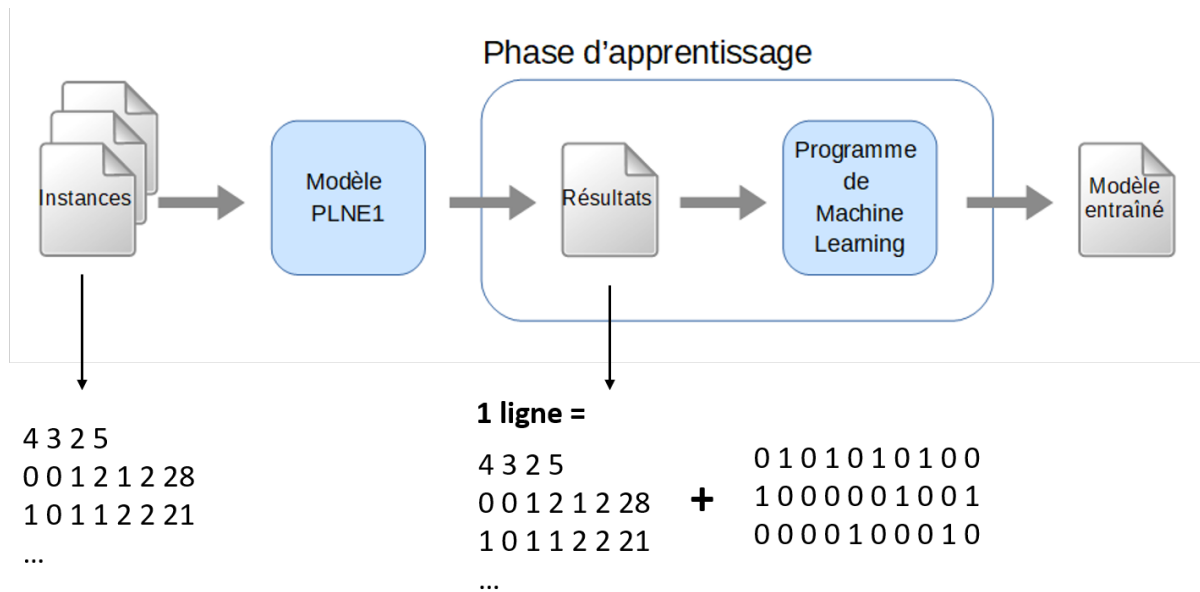


Figure 3 – Architecture générale du système lors de la phase d'apprentissage

Le cheminement est le suivant :

1. On passe en entrée au modèle PLNE plusieurs instances.
2. Le PLNE calcule toutes les solutions optimales et produit un nouveau fichier de résultats qui contient pour chaque ligne les informations sur l'instance et sa solution optimale trouvée.
3. Ce fichier est ensuite passé en entrée au programme de Machine Learning pour "apprendre".

## 2 Conception

### 2.1 Arborescence du projet

L'arborescence du projet est la suivante :

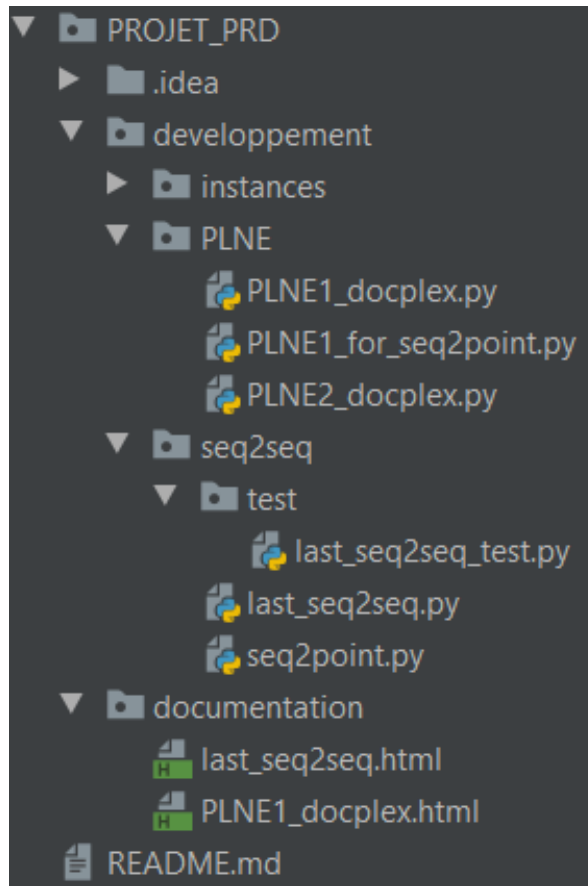


Figure 4 – Arborescence du projet

Le package "documentation" comprend les pydoc (générées au format html) des scripts python de PLNE et de sequence-to-sequence.

3 packages sont présents dans le dossier "développement" :

- instances : ce package contient des fichiers d'instances classiques utilisés à des fins de tests
- PLNE : ce package contient le script PLNE utilisé pour la résolution d'instances
- seq2seq : ce package contient le modèle sequence-to-sequence et le modèle sequence-to-point (permettant de réaliser des prédictions), et un package test qui comprend un fichier de tests unitaires sur le modèle seq2seq

## 2.2 Structure du script PLNE

Le diagramme de composants suivant décrit les principales fonctions et leurs interactions dans le programme PLNE :

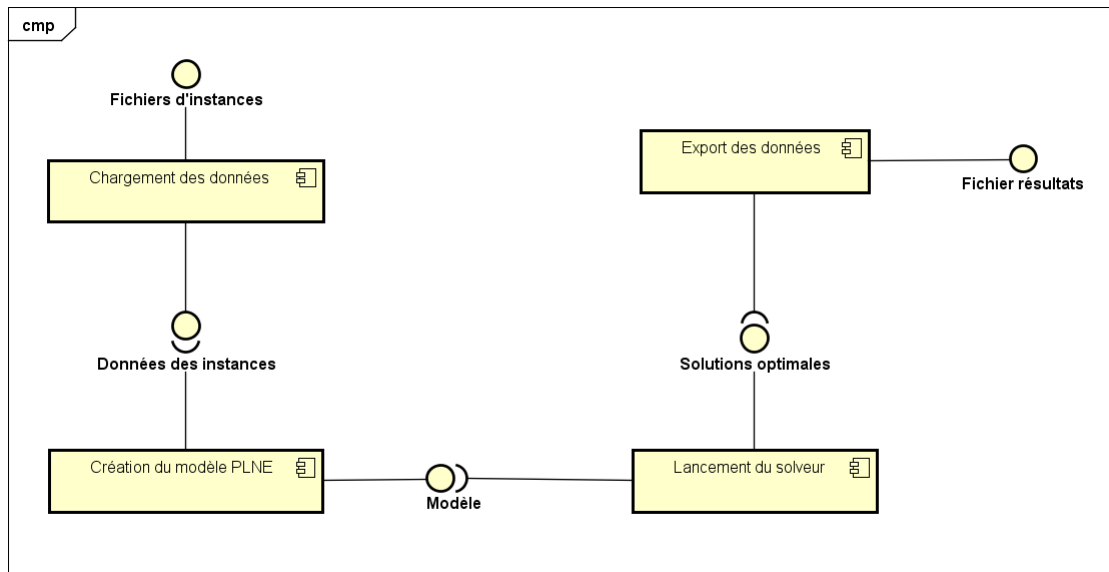


Figure 5 – Diagramme de composants du programme PLNE

4 composants interagissent entre eux :

- Un composant se charge d’importer les données des fichiers d’instances.
- Le deuxième composant crée le modèle PLNE en fonction des données importées.
- Le composant « Lancement du solveur » permet de lancer la résolution du modèle et de calculer les solutions optimales des instances.
- Le dernier composant permet d’exporter les données (instances + solutions calculées) dans un fichier résultats.

## 2.3 Structure du programme seq2seq

### 2.3.1 Phase d’apprentissage

Le diagramme de composants suivant décrit les principales fonctions et leurs interactions dans le programme sequence-to-sequence en phase d’apprentissage :

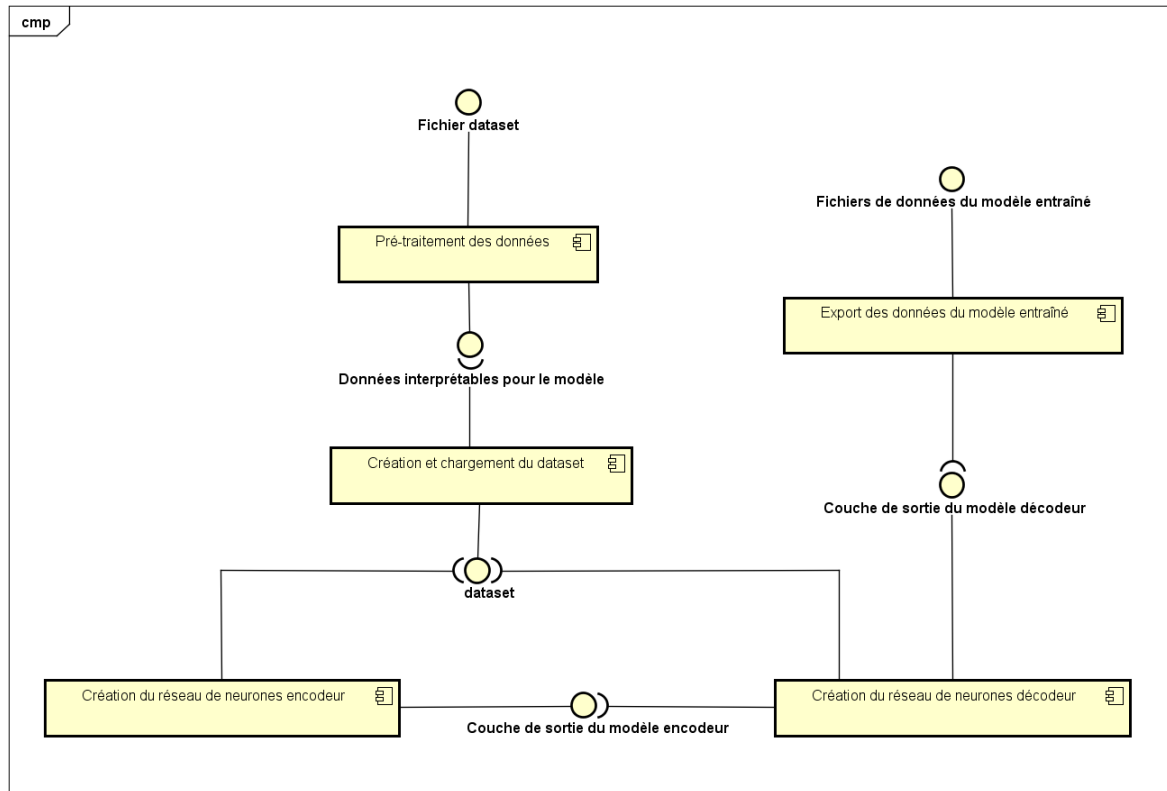


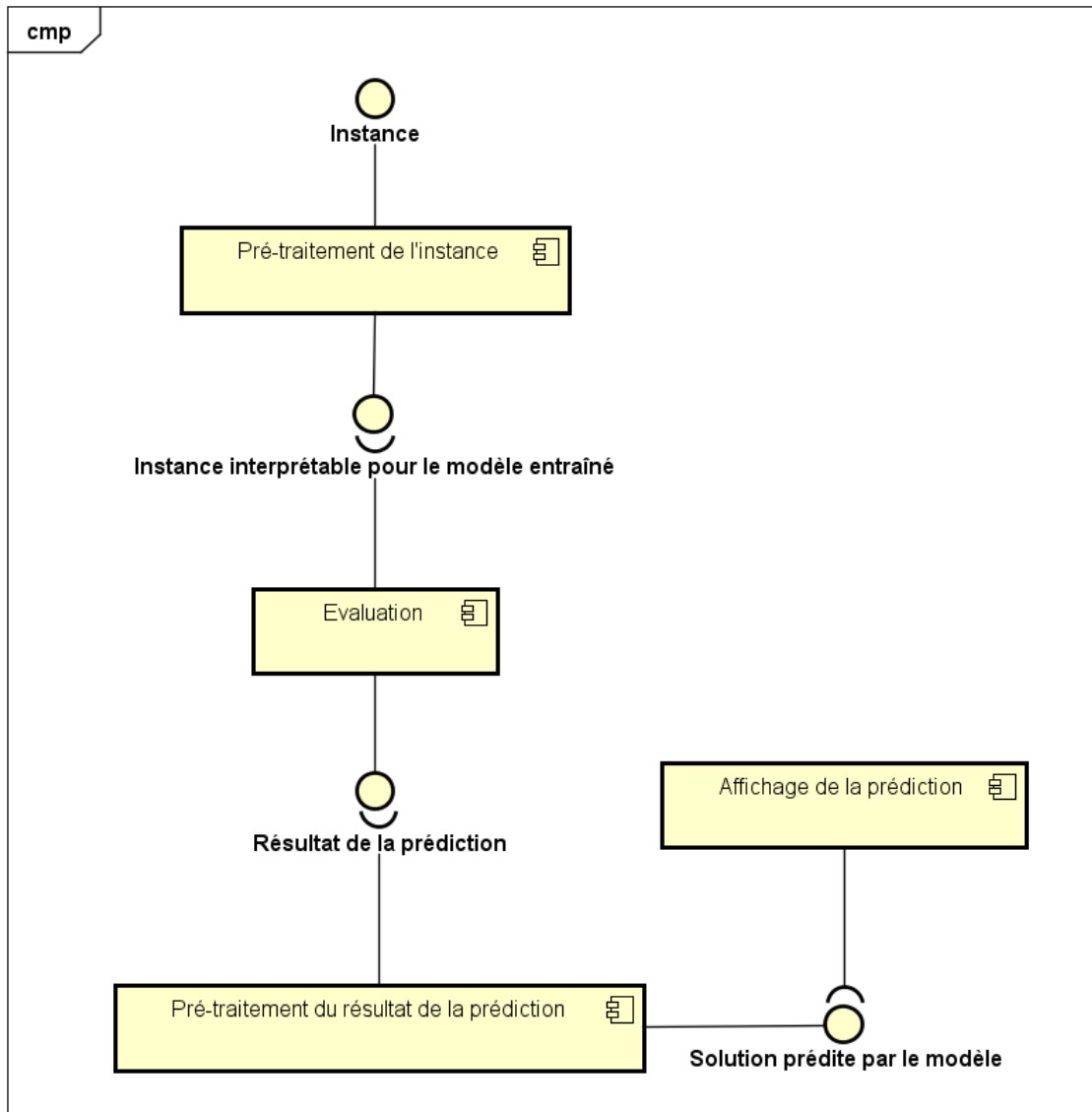
Figure 6 – Diagramme de composants du programme seq2seq en phase d'apprentissage

5 composants interagissent entre eux :

- Le composant « Pré-traitement des données » permet d'importer un fichier dataset et de le préparer afin qu'il puisse être mis en forme et utilisable pour le modèle.
- Le composant « Création et chargement du dataset » crée et charge le dataset.
- Les 2 composants "Encodeur" et "Décodeur" représentent les réseaux de neurones encodeur et décodeur du modèle sequence-to-sequence, et s'interfacent sur le dataset pour être créés. Lors de l'entraînement, l'encodeur est créé en premier, puis fourni le modèle dans sa couche de sortie, qui est ensuite utilisé par le décodeur. Ce dernier fournit à son tour le modèle dans sa couche de sortie.
- Le composant « Export des données du modèle entraîné » se charge de sauvegarder le modèle dans des fichiers en sortie.

### 2.3.2 Phase de prédiction

Le diagramme de composants suivant décrit les principales fonctions et leurs interactions dans le programme sequence-to-sequence en phase de prédiction :



**Figure 7** – Diagramme de composants du programme seq2seq en phase de prédiction

4 composants interagissent entre eux :

- Le composant « Pré-traitement de l'instance » permet de transformer l'instance pour qu'elle puisse être interprétable par le modèle.
- Le composant « Evaluation » s'interface à cette instance pour l'évaluer et réaliser une prédiction grâce au modèle entraîné.
- Le composant « Pré-traitement du résultat de la prédiction » se charge de convertir le résultat de la prédiction en une solution.
- Le dernier composant affiche la solution prédite.



H

Cahier de tests

Nom du fichier : last\_seq2seq.py

Objectif : vérifier le bon fonctionnement du programme

Pré-requis : le modèle doit avoir été entraîné au préalable

#### Tests unitaires

N°	Action	Attendu	Résultat
1	Exécution de la fonction qui convertie un chiffre/nombre en une ou plusieurs lettres pour la valeur 0	La fonction doit retourner la valeur 'ZERO'	OK
2	Exécution de la fonction qui convertie un chiffre/nombre en une ou plusieurs lettres pour la valeur 1	La fonction doit retourner la valeur 'A'	OK
3	Exécution de la fonction qui convertie un chiffre/nombre en une ou plusieurs lettres pour la valeur 27	La fonction doit retourner la valeur 'AA'	OK
4	Exécution de la fonction qui convertie une ou plusieurs lettres en un chiffre/nombre pour la chaîne 'ZERO'	La fonction doit retourner la valeur 0	OK
5	Exécution de la fonction qui convertie une ou plusieurs lettres en un chiffre/nombre pour la chaîne 'A'	La fonction doit retourner la valeur 1	OK
6	Exécution de la fonction qui convertie une ou plusieurs lettres en un chiffre/nombre pour la chaîne 'AA'	La fonction doit retourner la valeur 27	OK
7	Exécution de la fonction qui convertie une chaîne de valeurs numériques en une chaîne de lettres séparées par '_' pour la chaîne '1 26 7 3 28'	La fonction doit retourner la chaîne 'A_Z_G_C_AB'	OK
8	Exécution de la fonction qui convertie une instance en une séquence de lettres pour la chaîne '([ 1 2 3 2 3]:[[33 35 2 1 2 3][ 6 15 1 2 2 6][27 57 2 2 2 4]]'	La fonction doit retourner la chaîne '<start> A_B_C_B_C AG_AI_B_A_B_C F_O_A_B_B_F AA_BE_B_B_B_D <end>'	OK
9	Exécution de la fonction qui traduit une séquence de mots de lettres en une séquence de valeurs numériques pour la chaîne 'ae zero_zero_zero_zero_a zero_a_zero_zero_zero'	La fonction doit retourner la chaîne '31 0 0 0 1 0 1 0 0 0 '	OK
10			

#### Tests fonctionnels

N°	Action	Attendu	Résultat
1	Saisir une instance respectant le bon format	Affichage du résultat de la prédiction dans la console	OK

2	Saisir une instance ne respectant pas le bon format	Message d'erreur dans la console indiquant que le format de l'instance n'est pas respecté	Message absent
3	Saisir une instance avec un "mot" que le modèle n'a pas appris	Message d'erreur dans la console indiquant qu'un des mots n'a pas été appris par le modèle	Message absent
4			
5			
6			
7			
8			
9			
10			

*Nom du fichier* : **PLNE1\_docplex.py**

*Objectif* : vérifier le bon fonctionnement du programme

*Pré-requis* : aucun

*Tests unitaires* : aucun

*Tests fonctionnels*

N°	Action	Attendu	Résultat
1	Pour l'import, saisir un chemin vers un dossier existant contenant des fichiers avec l'extension ".data"	Résolution des instances et export des résultats	OK
2	Pour l'import, saisir un chemin vers un dossier ne contenant aucune instance avec l'extension ".data"	Fin du programme	OK
3	Pour l'import, saisir un chemin vers un dossier inexistant	Message d'erreur dans la console indiquant que le chemin spécifié n'existe pas	OK
4	Pour l'import, saisir un chemin vers un fichier	Message d'erreur dans la console indiquant que le nom spécifié après le chemin n'est pas un dossier	OK
5	Pour l'export des résultats, saisir un chemin correct et un fichier qui n'existe pas encore	Ecriture des résultats puis création du fichier	OK
6	Pour l'export des résultats, saisir un chemin correct et un fichier déjà existant	Ecriture des résultats à la suite du fichier	OK
7	Pour l'export des résultats, saisir un chemin incorrect	Message d'erreur dans la console indiquant que le chemin pour l'export est incorrect	Message absent
8			
9			
10			

*Autre* : fonction qui affiche toutes les contraintes du modèle à respecter, puis les variables égales à 1 à la suite de la résolution de l'instance afin de visualiser qu'elles ne sont pas violées

# I

## Glossaire

**CPLEX** : Logiciel d'optimisation.

**Machine Learning** (Apprentissage machine) : Champ d'étude de l'intelligence artificielle. Il se base sur un historique de données pour faire « apprendre » la machine pour qu'elle puisse ensuite réaliser des prédictions sur les données futures.

**PLNE** : Programmation Linéaire en Nombres Entiers. Forme de modélisation d'un problème d'optimisation avec une fonction objectif, des contraintes linéaires et des variables entières.

**Python** : Langage de programmation interprété.



## Webographie

- [1] Introduction au Machine Learning : <https://openclassrooms.com/fr/courses/4011851-initiez-vous-au-machine-learning>
- [2] Régression linéaire/logistique : <https://techdifferences.com/difference-between-linear-and-logistic-regression.html>
- [3] Perceptron : <https://fr.wikipedia.org/wiki/Perceptron>
- [4] RNN : [https://en.wikipedia.org/wiki/Recurrent\\_neural\\_network](https://en.wikipedia.org/wiki/Recurrent_neural_network)
- [5] LSTM : [https://en.wikipedia.org/wiki/Long\\_short-term\\_memory](https://en.wikipedia.org/wiki/Long_short-term_memory)
- [6] RNN-LSTM : <https://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm/>
- [7] Document de recherche seq2seq : <https://arxiv.org/pdf/1409.3215.pdf>
- [7] Fonctionnement seq2seq détaillé : <https://jalamar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>
- [8] Tutoriel seq2seq : [https://www.tensorflow.org/tutorials/text/nmt\\_with\\_attention](https://www.tensorflow.org/tutorials/text/nmt_with_attention)
- [9] Document de recherche seq2point : <https://arxiv.org/pdf/1612.09106.pdf>

# K

## Sources

Figure "Exemple de régression linéaire" : <https://openclassrooms.com/fr/courses/4011851-initiez-vous-au-machine-learning>

Figure "Structure d'un perceptron" : <https://fr.wikipedia.org/wiki/Perceptron>

Figure "Architectures RNN" : <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Figure "Modèle sequence-to-sequence" : <https://www.tensorflow.org/>

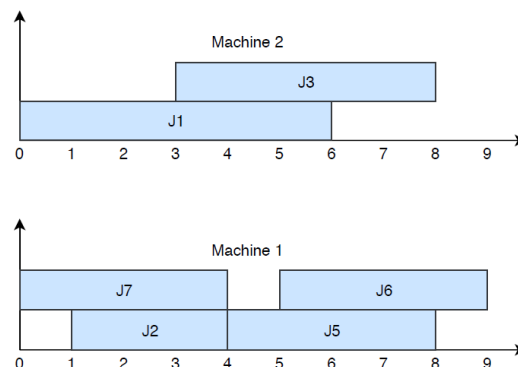
# Apports du Machine Learning pour l'ordonnancement des applications dans les Cloud data centers

Martin De La Fuente

Encadrement : Boukhalfa Zahout

## Objectifs

L'objectif de ce projet de recherche et développement est d'étudier et de mettre en place une méthode de machine learning au sein d'un problème d'ordonnancement.



Exemple d'ordonnancement de 6 jobs sur 2 machines

## Mise en œuvre

Après le développement d'un prototype PLNE, la mise en œuvre consiste à implémenter une technique de machine learning, puis d'interpréter les résultats obtenus afin d'en évaluer l'efficacité.

$$\sum_{i=0}^M \sum_{j=0}^N w_j x_{ij}$$

$$\sum_{t=s_j}^{f_{j-1}} y_{ijt} = (f_j - s_j) * x_{ij}; \forall j \in N, \forall i \in M$$

$$\sum_{j=0}^N y_{ijt} * r_{jk} \leq R_k; \forall k \in K, \forall i \in M, \forall t \in [s_{min}, f_{max}]$$

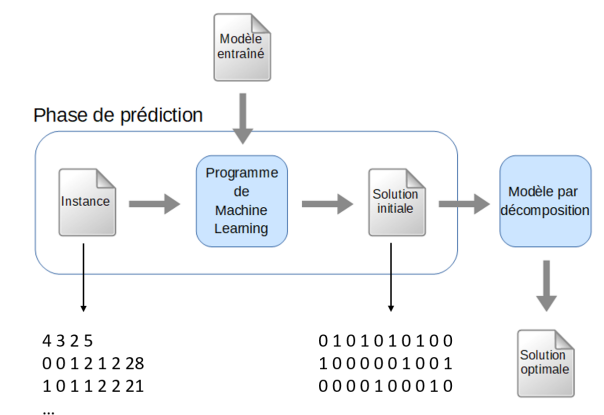
$$\sum_{i=0}^M x_{ij} \leq 1; \forall j \in N$$

$$x_{ij} \in \{0,1\}; y_{ijt} \in \{0,1\}; \forall i \in M, \forall j \in N, \forall t \in T$$

Programme Linéaire en Nombres Entiers

## Résultats attendus

Le résultat attendu est que le programme de machine learning, une fois entraîné, puisse prédire une solution initiale (qualifiée de "bonne solution initiale") qui sera ensuite donnée à une méthode exacte afin qu'elle puisse calculer la solution optimale plus rapidement.



Architecture du système lors de la prédiction d'une solution



# Apports du Machine Learning pour l'ordonnancement des applications dans les Cloud data centers

Martin De La Fuente

Encadrement : Boukhalfa Zahout

## Objectifs

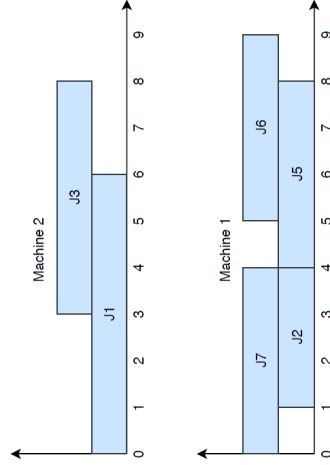
L'objectif de ce projet de recherche et développement est d'étudier et de mettre en place une méthode de machine learning au sein d'un problème d'ordonnancement.

## Mise en œuvre

Après le développement d'un prototype PLNE, la mise en œuvre consiste à implémenter une technique de machine learning, puis d'interpréter les résultats obtenus afin d'en évaluer l'efficacité.

## Résultats attendus

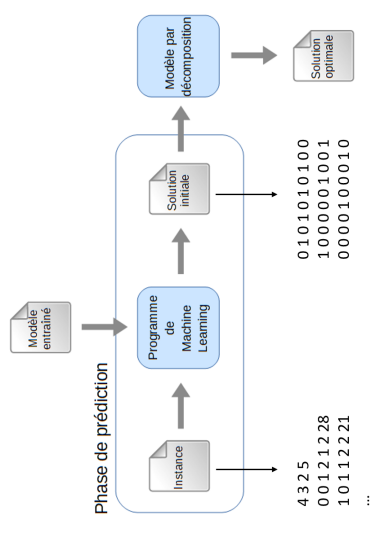
Le résultat attendu est que le programme de machine learning, une fois entraîné, puisse prédire une solution initiale (qualifiée de "bonne solution initiale") qui sera ensuite donnée à une méthode exacte afin qu'elle puisse calculer la solution optimale plus rapidement.



Exemple d'ordonnancement de 6 jobs sur 2 machines

$$\sum_{i=0}^M \sum_{j=0}^N w_{ij} x_{ij}$$
$$\sum_{t=s_j}^{t=f_j} y_{ijt} = (f_j - s_j) * x_{ij}; \forall j \in N, \forall i \in M$$
$$\sum_{j=0}^N y_{ijt} * \tau_{jk} \leq R_k; \forall k \in K, \forall i \in M, \forall t \in [s_{min}, f_{max}]$$
$$\sum_{i=0}^{M_1} x_{ij} \leq 1; \forall j \in N$$
$$x_{ij} \in \{0,1\}; y_{ijt} \in \{0,1\}; \forall i \in M, \forall j \in N, \forall t \in T$$

Programme Linéaire en Nombres Entiers



Architecture du système lors de la prédiction d'une solution

# Apports du Machine Learning pour l'ordonnancement des applications dans les Cloud data centers

## Résumé

Ce document a pour but de présenter les recherches qui ont été menées concernant l'apport du Machine Learning au sein d'un problème d'ordonnancement de type mono-critère et multi-machines (dont l'objectif est de maximiser la valeur des jobs pondérés). Dans un premier temps, il se focalise sur la présentation de deux méthodes de résolution exactes (modèles linéaires en nombres entiers) afin d'analyser leur performance en temps de calcul. Ce document détaille ensuite différentes techniques de Machine Learning puis propose une architecture mettant en place un algorithme d'apprentissage qui servira à retourner des solutions initiales pour des méthodes exactes déjà développées.

## Mots-clés

Apprentissage machine, Job, mono-critère, multi-machines, Optimisation, Ordonnancement, Prédiction, Programmation Linéaire en Nombres Entiers

## Abstract

This document shows research on adding Machine Learning in a mono-objective and multi-machine scheduling problem (whose objective is to maximize the weighted number of jobs). Firstly, this document presents two exact methods (Integer Programming models) and analyzes their performance. Then it shows several Machine Learning techniques and explains an architecture with a Machine Learning algorithm which will return initial solutions for exact methods already developed.

## Keywords

Machine Learning, Job, mono-objective, multi-machine, Optimization, Scheduling, Prediction, Integer Programming