PROJET FIL ROUGE

Contrôle du trafic routier urbain par apprentissage par renforcement

Auteurs :
Philippe JACQUET
Thomas BINETRUY
Margaux RODRIGUES
Adrien LY

Superviseurs :
Philippe GICQUEL
Olivier ROUHAUD
Geoffroy PEETERS
Quentin MERCIER

Mastère spécialisé Big Data
Telecom Paristech

19 janvier 2020

Résumé

Mastère spécialisé Big Data
Telecom Paristech

Projet Fil Rouge

Contrôle du trafic routier urbain par apprentissage par renforcement

Philippe JACQUET, Thomas BINETRUY, Margaux RODRIGUES, Adrien LY

Superviseurs :
Philippe GICQUEL, Olivier ROUHAUD, Geoffroy PEETERS, Quentin MERCIER

Ce rapport incrémental présente dans le chapitre premier, le travail fourni en P2 pour le projet fil-rouge. Nous détaillons le projet d'optimisation de trafic routier en vue de réduire la pollution, avant d'introduire l'état de l'art en la matière. Enfin, nous présentons notre solution, à l'aide d'apprentissage par renforcement ainsi que les résultats obtenus au terme du projet.

Table des matières

Table des figures

Liste des abbréviations

RL	**R**einforcement **L**earning
DRL	**D**eep **R**einforcement **L**earning
DQL	**D**eep **Q** **L**earning

Chapitre 1

Projet Fil Rouge P1

1.1 Introduction

Alors que l'écologie devient un enjeu international d'importance, de plus en plus de recherches sont faites sur des transports verts, ou sur des moyens d'économiser de l'énergie. Dans le même temps, les avancées technologiques laissent présager une arrivée de véhicules autonomes capables de communiquer avec leur environnement, ou du moins de l'intelligence artificielle comme assistance de conduite. Dans ce contexte, plusieurs études ont été menées dans le but de réduire les émissions de polluants des véhicules par l'intermédiaire de la fluidification du trafic routier. Parmi ce domaine de recherche, on distingue notamment les travaux se concentrant sur les zones extra-urbaine (autoroutes et grands axes routiers) [citer papiers sans commenter], des zones urbaines qui sont au cœur du sujet traité dans ce rapport. La start-up CIL4SYS nous propose, dans le cadre d'un projet fil-rouge, de nous pencher sur la possibilité de réduire les émissions de polluants dans l'hypothèse de feux tricolores intelligents capables d'envoyer des consignes de vitesse aux véhicules traversants un quartier en zone urbaine dans le but de limiter les accélérations et décélérations des véhicules sources de fortes consommations en carburant.

Le projet s'articule donc autour de CIL4SYS créée en juin 2015 par Philippe GICQUEL qui a travaillé pendant 25 ans en R&D automobile chez EDAG et PSA Peugeot-Citroën. Le cœur de métier de CIL4SYS étant de proposer une solution permettant de générer des cahiers des charges et des spécifications techniques à partir diagrammes UML décrivant le fonctionnement désiré d'un système. Leur cible principale étant l'industrie automobile, un environnement de simulation a été spécifiquement développé dans le but de répondre à cette industrie. Mais aussi de SOFTEAM, grand groupe fournissant un panel de service divers, allant du consulting opérationnel et métier, à la modélisation et l'automatisation, en passant par l'innovation. Leur rôle, durant ce projet, sera de nous assister lors de la phase de machine-learning en nous apportant une expertise avancée en data science. Enfin TélécomParisTech constitue le dernier acteur de se projet par l'intermédiaire de notre groupe d'étudiants chargé d'effectuer les premiers développements et essais ainsi que de proposer les solutions et méthodes à explorer.

L'objectif principal de ce projet est donc de démontrer à l'échelle d'un quartier que par un contrôle à la fois des feux tricolores et des vitesses des véhicules (dans un contexte où les véhicules sont communicants et peuvent recevoir des instructions de vitesse), il est possible de trouver un optimum de diminution des rejets de polluants et de CO2. Les méthodes mises en place au cours du projet devront faire appel à nos connaissances en apprentissage automatique afin de proposer une première approche de résolution.

Il nous est demandé de d'utiliser nos connaissances pour proposer une solution de machine learning à ce problème.

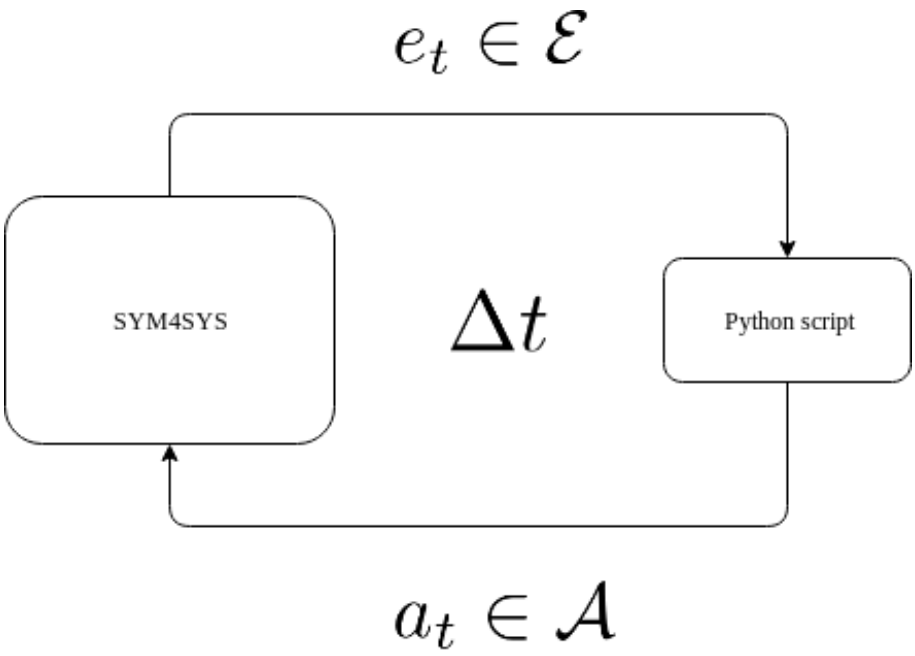


FIGURE 1.1 – Interaction de notre solution a l'outil de simulation Sim4Sys

1.2 Outils mis à disposition - CIL4Sys Softeam

1.2.1 Outil de simulation Sim4Sys

Cil4sys a mis en place un logiciel de simulation, permettant de valider leur modèles grâce à une visualisation 3D. Ce simulateur peut être exporté vers un maquette réelle pour tester les résultats d'un algorithme. Elle permet donc un potentiel passage en production de solutions résultants d'algorithmes développé sur ce logiciel.

Un des enjeux du projet sera donc d'intégrer notre solution à Sim4Sys. Pour ce faire, nous allons écrire un script Python qui prendra en entrée un état du simulateur $e \in \mathcal{E}$ et retournera une action $a \in \mathcal{A}$ à réaliser. L'état est une description du système que nous détaillons dans le reste de ce document, et l'action à prendre peut être un changement d'état des feux et/ou une recommandation de vitesse pour les voitures. La figure 1.1 schématise l'interaction de notre script avec le simulateur Sim4Sys.

1.2.2 Données à notre dispositions : TOMTOM et les taxis de Los Angeles

Grâce à Cil4Sys, nous avons accès à l'API de TOMTOM. Cela nous permet d'extraire des informations relatives à des vitesses observées réellement sur le terrain sur une zone que l'on peut définir. L'obtention de grandeurs réelles nous servira de calibrage et de validation de modèle quant au réalisme des données que nous allons manipuler.

Lors de nos recherches sur le sujet et souhaitant multiplier les informations qui pourraient nous servir par la suite, nous avons récupérer des données précises sur la circulation à Los Angeles des taxis. Ces éléments spécifiques concernant les accélérations, les vitesses, les émissions de CO2 pourraient nous servir dans le cas où nos premières solutions ne sont pas concluantes.

1.3 État de l'art : contrôle du trafic routier urbain

Notre étude de l'état de l'art porte sur les trois axes induits par le sujet du projet, à savoir la réduction des émissions de polluants, le contrôle de la vitesse des véhicules et le contrôle de la signalisation tricolore.

Le contrôle du trafic routier urbain fait l'objet de nombreuses études : la décongestion du trafic, qu'il soit urbain ou sur les grands axes représente une problématique importante tant d'un point de vue environnemental qu'en termes d'urbanisme ou de sécurité routière.

Les études les plus anciennes sur le sujet proposent une optimisation basée sur l'historique de trafic et visent à prédire le trafic en utilisant une approche de type série temporelle. Nous avons focalisé nos recherches sur le trafic urbain, et non sur des grands axes routiers, ce qui nous amène à l'étude des intersections et à la gestion de la signalisation.

1.3.1 Quantification du niveau des émissions des véhicules

Afin de répondre à la problématique de la réduction des émissions de polluants, il convient dans un premier lieu de les quantifier. Ils dépendent à priori de la vitesse et de l'accélération des véhicules, mais également de leur type, de leur âge. Selon les données utilisées dans le modèle, on peut utiliser les vitesses moyennes pour estimer les émissions ou un modèle dynamique qui prendra en compte les accélérations et arrêts par exemple.
On trouve également une modification à la fonction de coût classique qui intègre les temps de trajets et d'arrêt, qui y intègre la consommation de carburant que l'on peut considérer comme une approximation acceptable du niveau d'émissions dans un premier temps. La recherche concernant la modélisation des niveaux d'émissions a un large historique et propose des solutions diverses, par exemple modélisation par fonction linéaire du nombre d'arrêts [21] et du temps total de trajet des véhicules, ou des modèles non linéaires, réseaux de neurones [1].

Un modèle répandu est le VT-Micro [18], développé par le Virginia Tech Transportation Institute. Le modèle propose une estimation granulaire des émissions, à la fois selon des variables catégorielles et selon des variables concernant le comportement réel d'un véhicule. Les variables catégorielles concernant le véhicule (le type, le carburant utilisé, l'année de construction) dont la classification a été effectuée avec des arbres de régression, selon les quantités émises de différents polluants (CO2, NO, HC) mais aussi le mode de conduite. On distingue plusieurs modes de conduite : autoroute, route, ville, par exemple.

Enfin, le modèle utilise des variables basées sur la conduite réelle et en particulier la vitesse instantanée et l'accélération du véhicule. Le modèle de régression employé dépend de termes d'accélération et de vitesse linéaires, quadratiques et cubiques.

On note également que les simulateurs de trafic proposent une estimation des émissions des véhicules.

1.3.2 Contrôle de la vitesse des véhicules

Avec le développement des véhicules autonomes et l'amélioration des moyens de communication inter-véhicules (capteurs), des modèles de contrôle ou suggestion de vitesse des véhicules sont également développés avec l'objectif d'optimiser les intersections.[19] Des études ont également montré qu'un contrôle de la vitesse à l'abord des intersection permet de limiter les arrêts, et ainsi de diminuer les émissions de polluants [14] [15] [23].

Dans le cadre du contrôle de la vitesse des véhicules, on peut considérer soit une intersection seule en définissant une distance à partir de laquelle le véhicule entre dans la zone. A partir de ce moment la file de véhicules se situant sur une même voie se comporte en suivant le véhicule de tête, en utilisant l'hypothèse que les véhicules communiquent entre eux

afin d'éviter toute collision. Le véhicule de tête quant à lui reçoit ses instructions d'un agent planificateur qui connaît l'état de l'environnement, et envoie ses instructions aux véhicules de tête pour permettre un passage plus fluide.

Dans cette configuration, chaque véhicule qui est dernier dans la file envoie une information de vitesse au nouvel entrant, qui réglera sa vitesse sur celle du précédent. Le mécanisme permet d'éviter des collisions et seuls les véhicules de tête sont concernés par la prise de décision relative à l'intersection.

Le modèle propose également d'étudier un réseau d'intersections. Dans ce cadre le modèle d'intersection isolée est conservé mais étendu. A la sortie de l'intersection le véhicule conserve en mémoire les informations sur ceux qui le suivent. Il communique cette information à l'intersection suivante, ce qui permet d'avoir une estimation sur le nombre de véhicules qui demandent à rentrer dans l'intersection au delà de la zone délimitée précédemment. Cette information est une indication de volume auquel s'attend l'agent régulateur de l'intersection.

1.3.3 Contrôle de la signalisation au niveau des intersections

L'un des points importants à prendre en compte dans l'optimisation de trafic routier est une bonne définition de l'optimum à atteindre. En effet un optimum individuel ne permet pas d'obtenir un équilibre optimal au sens de Pareto [12]. Dans notre cas cela correspondrait au cas où un ou plusieurs véhicules minimisent leur fonction objectif (leur temps de trajet par exemple) et où le reste des véhicules ne progresse pas. Ce point nous amènera à poser des contraintes dans les fonctions objectif utilisées.

Cela se traduit par le développement de modèles multi-agents, où l'on cherche à optimiser le système au niveau de l'agent intersection. Les modèles multi-agents sont adaptés à la régulation de trafic dans la mesure où l'on ne cherche pas à optimiser une unique intersection, mais un réseau d'intersections reliées les unes aux autres. Le système multi-agent permet de définir des agents intersection, contrôlées par un agent à un niveau supérieur dont l'objectif est de coordonner les intersections entre elles (la région). Le modèle multi-agent développé par Jin et Ma, 2018 [8] décompose le système comme suit :

- Un agent de trajectoire : cela représente une combinaison possible origine-destination au sein d'une intersection à laquelle est associé un feu tricolore. Tous les agents trajectoire d'une même intersection possèdent donc le même environnement et sont représentés par une matrice de compatibilité basée sur la configuration de l'intersection (pour éviter les collisions). Ces agents possèdent deux états : actif (si la trajectoire peut être effectuée) ou inactif (le feu correspondant est rouge).
- Un agent intersection qui donne des instructions aux agents de trajectoire qui lui sont subordonnés, afin d'assurer une stratégie optimale collectivement.
- Les agents régions sont les plus hauts du système. Ils regroupent des intersections et donnent des instructions aux intersections avec une vision plus large, permettant de générer des scénarios de ligne de feux vert (*green wave scenario*). Les agents de même type sont considérés dans ce modèle comme homogènes entre eux, c'est-à-dire qu'ils possèdent les mêmes fonctions de coût ou récompense, le même environnement et les mêmes paramètres d'apprentissage.

L'approche utilisée dans cet article est l'algorithme d'apprentissage par renforcement SARSA (*State Action Reward State Action*) qui à chaque étape actualise la base de connaissances des agents trajectoire (le plus bas niveau) et qui définit le contrôle optimal au niveau de l'intersection.

L'apprentissage par renforcement pour les problèmes d'optimisation de trafic ont fait l'objet d'autres recherches, notamment utilisant le Q-learning au lieu de SARSA [4]. La différence entre les deux algorithmes se situant dans la définition de la récompense : SARSA

prend en compte la politique suivie par l'agent quand le Q-learning considère l'action suivante comme maximisant la récompense. Ces deux algorithmes proches présentent chacun des avantages et inconvénients : quand le Q-learning est plus rapide et simple à mettre en oeuvre, le SARSA permet une approche plus conservatrice, notamment si on souhaite l'utiliser dans un environnement réel et non de simulation.

L'utilisation d'un apprentissage par renforcement nous permettrait une flexibilité sur la définition de la fonction de récompense des agents, et ainsi supplémenter le modèle original d'une fonction de contrôle sur la vitesse des véhicules, permis par le contrôle situé au niveau de l'intersection mais également d'utiliser une mesure des émissions de polluants dans le modèle.

1.4 Problème d'optimisation

Plus classiquement, il est possible de donner une formulation d'un point de vue optimisation du problème. En effet, l'objectif principal demandé par CIL4SYS est de trouver des lois de commande des feux et des véhicules de manière à réduire au maximum les émissions de CO2. Au cours de ce projets les émissions de CO2 seront modélisées uniquement par l'intermédiaire des accélérations. Une première approche peut donc être de minimiser :

$$\min_{\theta(t)} \mathbb{E}\left[\sum_{j=1}^{m} \int_{t=0}^{t_j^{sortie}} |a_j(t, \theta(t), \mathcal{C}_j(\xi))| dt\right], \tag{1.1}$$

où $a_j(t)$ représente les accélérations subies par le véhicule j tout au long de sa trajectoire $\mathcal{C}_j$ affectée aléatoirement par la variable aléatoire ξ et $\theta(t)$ représente la loi de commande imposée aux véhicules.

On constate d'ors et déjà que la solution à un tel problème d'optimisation est évidente puisqu'il suffit d'imposer une loi de commande de telle sorte que les véhicules restent à l'arrêts pour minimiser les emissions. Le problème initial d'optimisation se doit donc d'être complété par l'intermédiaire de contraintes afin répondre à un certain réalisme. Un première contrainte à envisager est le temps de trajet pour la réalisation du parcours $\mathcal{C}$ du véhicule. En utilisant les données TOMTOM, il sera donc possible de déterminer au sein du quartier qui sera choisi par la suite quel est le temps de trajet moyen réalisé par un véhicule pour une trajectoire imposée $\bar{t}(\mathcal{C})$. Une contrainte seuil sur le temps de trajet pourra alors être imposé avec pour référence le temps de trajet calculé. Le problème de minimisation se reformule alors de la manière suivante

$$\begin{cases} \min_{\theta(t)} \mathbb{E}\left[\sum_{j=1}^{m} \int_{t=0}^{t_j^{sortie}} |a_j(t, \theta(t), \mathcal{C}(\xi))| dt\right] \\ \text{s.t. } t_j^{sortie} \leq \alpha \cdot \bar{t}(\mathcal{C}) \end{cases} \tag{1.2}$$

La formalisation du problème d'optimisation reste à compléter et on sera amené à ajouter d'autres contraintes au cours du projet au fur et à mesure de sa prise en main. La fonction de coût à minimiser sera elle aussi amenée à être changée si par exemple le nombre de voitures qui traversent le quartier devient aléatoire lui aussi etc...

1.5 Apprentissage par renforcement

Comme dit précédemment, une des méthodes utilisée dans la littérature actuelle pour effectuer de la fluidification de trafic est l'apprentissage par renforcement. C'est la méthode d'apprentissage sur laquelle nous avons décidée de nous concentrer car novatrice dans le

domaine de l'optimisation de trafic routier. Dans cette section, nous présentons les grandes ligne de cette méthode d'apprentissage.

L'apprentissage par renforcement est une méthode statistique de prise de décision dans un environnement donné. L'environnement est modélisé par un état, et l'acteur peut réaliser une action qui affectera l'état. L'algorithme est guidé lors de la phase d'entraînement par une fonction de récompense. Après avoir pris une action a à un état e, l'observation du nouvel état permet le calcul d'une récompense, $r \in \mathbb{R}$.

Notons $\mathcal{E}$ l'ensemble des états possibles de notre environnement, $\mathcal{A}$ l'ensemble des actions possibles, et r_t la récompense obtenue au pas de temps t. En modélisant la récompense cumulative comme un processus de Markov, elle ne dépend que de l'action prise et de l'état du système aux temps $t \geq t_0$ et est définie de la manière suivante :

$$R_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_{t+1} \tag{1.3}$$

Où le coefficient $\gamma \in [0, 1]$ permet de donner plus de poids aux récompenses proches de t_0 dans le temps et à la somme de converger.

Le principe de l'apprentissage par renforcement est alors de chercher une fonction $Q^* : \mathcal{E} \times \mathcal{A} \longrightarrow \mathbb{R}$ qui estime le retour cumulatif R_{t_0} pour une action $a \in \mathcal{A}$ réalisée à un état $e \in \mathcal{E}$.

La fonction $\pi(e, a)$ retourne la probabilité de réaliser une action a à l'état e. Nous avons donc $\sum_{a \in \mathcal{A}} \pi(e, a) = 1$. Définissons $Q^\pi(e, a)$, la fonction qui prédit l'espérance de la récompense cumulative sous π sachant e et a :

$$Q^\pi(e, a) = \mathbb{E}_\pi[R_t | e_t = e, a_t = a] \tag{1.4}$$

Définissions $\mathcal{P}^a_{ee'}$, la probabilité de passer d'un état e à un état e' sachant un état e et une action a donnés :

$$\mathcal{P}^a_{ee'} = \mathbb{P}(e_{t+1} = e' | e_t = e, a_t = a)$$

Définissions aussi $\mathcal{R}^a_{ee'}$, l'espérance de la récompense r_{t+1} sachant un état e, et une action a à l'instant t ainsi qu'un état e' à l'instant $t + 1$:

$$\mathcal{R}^a_{ee'} = \mathbb{E}(r_{t+1} | e_t = e, e_{t+1} = e', a_t = a)$$

Il est alors possible de d'exprimer Q^π de la manière suivante :

$$
\begin{aligned}
Q^{\pi}(e,a) &= \mathbb{E}_{\pi}[R_t|e_t = e, a_t = a] && (1.5)\\
&= \mathbb{E}_{\pi}[\sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t | e_{t_0} = e, a_{t_0} = a] && (1.6)\\
&= \mathbb{E}_{\pi}[r_{t+1} + \gamma \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_{t_0+1} | e_{t_0} = e, a_{t_0} = a] && (1.7)\\
&= \sum_{e'} \mathcal{P}^{a}_{ee'} \left[\mathcal{R}^{a}_{ee'} + \gamma \mathbb{E}_{\pi} \left(\sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_{t_0+1} | e_{t+1} = e' \right) \right] && (1.8)\\
&= \sum_{e'} \mathcal{P}^{a}_{ee'} \left[\mathcal{R}^{a}_{ee'} + \gamma \sum_{a'} \mathbb{E}_{\pi} \left(\sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_{t_0+1} | e_{t+1} = e', a_{t+1} = a' \right) \right] && (1.9)\\
&= \sum_{e'} \mathcal{P}^{a}_{ee'} \left[\mathcal{R}^{a}_{ee'} + \gamma \sum_{a'} \pi(e', a') Q^{\pi}(e', a') \right] && (1.10)
\end{aligned}
$$

Si la politique d'action à choisir dans un état donné consiste à maximiser la récompense cumulative, alors :

$$\pi^*(e) = \underset{a}{\text{argmax}} Q^*(e,a)$$

Cependant, nous ne connaissons pas la fonction Q^*, nous utilisons donc un modèle statistique pour l'approcher. En prenant alors :

$$\pi(e) = \underset{a}{\text{argmax}} Q^{\pi}(e,a)$$

L'équation de $Q^{\pi}(e,a)$ sous cette politique se simplifie alors :

$$Q^{\pi}(e,a) = r + \gamma Q^{\pi}(e', \pi(e'))$$

Exploration vs exploitation

Au début de l'apprentissage, la fonction approchée par notre modèle ne sera pas de bonne qualité, et donc notre politique de prendre l'action avec la plus grande valeur de Q^{π}_{θ} peut potentiellement ne pas converger. Pour encourager l'algorithme à explorer son espace d'action, on introduit un paramètre $\epsilon \in [0,1]$. On modifie alors notre politique d'action de manière à faire une action aléatoire dans ϵ% des cas :

$$\pi(e) = \begin{cases} \text{random}(\mathcal{A}), & \text{si random}([0,1]) < \epsilon. \\ \underset{a}{\text{argmax}} Q^{\pi}(e,a), & \text{sinon.} \end{cases}$$

Il est aussi possible de faire varier ϵ dans le temps, avec par exemple des valeurs plus grosses en début d'entraînement pour ensuite diminuer. Le Listing 1 propose une implémentation possible de stratégie ϵ – greedy en Python.

Deep Q-Network

Q^{π} étant inconnue, l'idée consiste à l'approcher avec un réseau de neurones dense multicouches, ce qui revient alors a choisir un paramètre θ de manière a minimiser δ, l'erreur de

```
def act(self, state):
    if np.random.rand() < self.epsilon:
        return random.randrange(self.action_size)
    else:
        probas = self.model.predict(state)
        return np.argmax(probas)
```

LISTING 1 – Stratégie ϵ – greedy en Python.

différence temporelle de notre modèle approché $Q_\theta^\pi(e,a)$:

$$\theta \in \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\|\delta\|) \tag{1.11}$$

$$\delta = Q_\theta^\pi(e,a) - (r + \gamma \underset{a'}{\operatorname{argmax}} Q_\theta^\pi(e',a')) \tag{1.12}$$

Ou $\mathcal{L}$ est une fonction de perte. En pratique, l'optimisation est réalisée par "batchs" de transitions B a l'aide d'une descente de gradient stochastique~[5].

En effet, après chaque action prise, le calcul de la récompense obtenue est réalisé et ces transitions sont stockées dans une mémoire $\mathcal{M}$ composée de quadruplets $(a,e,e',r) \in \mathcal{A} \times \mathcal{E}^2 \times \mathbb{R}$. A chaque action, nous mettons en mémoire le quaduplet obtenu et réalisons une descente de gradient sur un batch $B \in \mathcal{B}$.

Les paramètres de notre modélisation sont donc nombreux :

— θ, les paramètres de notre modèle statistique
— γ, le poids données aux récompenses plus tôt dans le temps
— $\mathcal{L}$, la fonction de perte
— La fonction de calcul de récompense, $\psi : \mathcal{E} \longrightarrow \mathbb{R}$

Le réseau de neurones d'un DQN prend donc en entrée les états accumulés dans la mémoire $\mathcal{M}$ au cours de l'exploration. C'est une fonction à $|\mathcal{A}|$ neurones en sortie où $\mathcal{A}$ est l'espace des actions. L'activation de cette dernière couche est **linéaire**, en effet, l'espérance de la récompense pour une action et un état donnés prend une valeur sur $\mathbb{R}$. Enfin, la fonction à minimiser est celle de la différence temporelle (équation 1.12). Une implémentation utilisant la librairie *Keras* est proposée dans le Listing 2.

```
def build_DQN(self):
    model = Sequential([
        Dense(24, input_shape=(self.state_size,)),
        Activation('relu'),
        Dense(24),
        Activation('relu'),
        Dense(self.action_size),
        Activation('linear'),
    ])
    model.compile(loss=self.temporal_difference, optimizer=Adam(lr=self.learning_rate))

    return model
```

LISTING 2 – Implémentation d'un réseau DQN dense bi-couche avec *Keras*.

Le réseau est entraîné à chaque pas de temps sur un batch *aléatoire*, pioché dans la mémoire $\mathcal{M}$. En effet, cette mémoire contient toutes les paires d'états à une action d'intervalle

explorées, il est donc possible d'estimer l'équation 1.4 tout les pas de temps sur ces paires avec le modèle. Le Listing 3 propose une implémentation en Python d'un algorithme d'entraînement du DQN.

```
def replay(self, batch_size):
    minibatch = random.sample(self.memory, batch_size)

    for state, action, reward, next_state, done in minibatch:
        q_values = self.model.predict(state)
        if done:
            q_values[0][action] = reward
        else:
            q_values[0][action] = (reward + self.gamma *
                                   np.amax(self.q_values_model.predict(next_state)[0]))
        self.model.fit(state, q_values, epochs=1, verbose=0)

    if self.epsilon > self.epsilon_min:
        self.epsilon *= self.epsilon_decay
```

LISTING 3 – Entraînement du DQN en Python.

1.6 Présentation des outils utilisés

L'apprentissage par renforcement reposant sur l'interaction avec un environnement, l'entraînement s'appuie souvent sur un simulateur. C'est le cas de notre modèle, ce qui implique donc l'interaction de plusieurs unités logiques du code entre elles. Afin de simplifier au maximum ces interactions tout en rendant le passage à l'échelle de l'entraînement possible, notre modèle repose sur un certain nombre d'outils qu'il est nécessaire d'introduire à ce stade du document, c'est ce que nous faisons à présent.

1.6.1 Sumo & Traci

Sumo est un simulateur de trafic microscopique multi-plateforme, libre, développé et maintenu par le ministère des transports allemand et possédant une API Python nommée *Traci* de haute qualité. Ce simulateur permet entre autre de requêter et modifier l'état d'une simulation à chaque pas de temps via Traci. C'est ce qui permet de prendre des actions et de calculer les récompenses lors de l'entraînement de notre agent.

Le module Traci de Sumo permet entre autre de requêter les émissions de chaque voiture, l'état de chaque feu, les collisions éventuelles, ainsi que la redirection, suppression et instanciation de véhicules à chaque pas de temps.

Dans Sumo, les routes sont représentées par des *arêtes* (*edges* en anglais), chaque arête pouvant avoir une ou plusieurs *voies* (*lanes* en anglais). Chaque arête et chaque voie est identifiée de manière unique par une chaîne de caractères, ce qui permet d'interagir facilement avec les véhicules de la simulation.

Les *feux* sont placés à certaines intersections sur la carte. Mais dans Sumo, le nombre de feux peut être supérieur au nombre d'intersections. L'état des feux à une intersection est défini par une chaîne de caractères où chaque caractère représente l'état d'un des feux de l'intersection. Par exemple, l'état des feux d'une intersection gérée par trois feux pourrait être `rGG` ce qui voudrait dire que le premier feu est *rouge* (red) et les deux suivants *verts*

(green). Le premier caractère de la chaîne représente l'état du feu le plus au nord de l'intersection gérée, le suivant le feu directement après en allant dans le sens anti-horaire, et ainsi de suite jusqu'à la fin de la chaîne. Enfin, au même titre que les arêtes, Sumo identifie chaque jeux de feux gérant une intersection par une chaîne de caractère unique permettant de requêter le feu via Traci.

1.6.2 RLlib

RLlib est une suite d'outils pour Python facilitant la mise en place d'une infrastructure d'apprentissage par renforcement. D'un part il fournit les algorithmes classiques du domaine, dont le DQN, *à la Scikit-learn*, et d'autre part il permet l'entraînement distribué d'agent. En effet, le passage à l'échelle de notre modèle implique un entraînement distribué, et donc la capacité à instancier plusieurs processus Sumo et interagir avec, ce que RLlib permet de réaliser.

1.6.3 Flow

L'intégration de Sumo est RLlib étant complexe à mettre en œuvre, un laboratoire de UC Berkeley travaille sur une abstraction open source de RLlib et Sumo de manière à faciliter l'application de l'apprentissage par renforcement au problème de fluidification de trafic routier. C'est donc à l'aide cette librairie Python appelée Flow que nous avons implémenté notre solution. Dans l'optique de permettre au lecteur de bien comprendre les prochaines sections du présent document, nous introduisons les concepts fondamentaux de Flow ainsi que son jargon.

Un **scénario** est un objet qui définit la simulation en elle même, c'est à dire le définition des arêtes et des voies, le nombre de voitures et les rails qu'elles suivent, ainsi que la position des feux. Flow fournit la classe abstraite `BaseScenario` à sous-classer lors de l'implémentation d'un scénario de calcul. Cette classe propose notamment des méthodes pour importer des maillages routiers de différents formats, dont celui de *Open Street Map*.

Un **environnement** est un objet qui sous-classe `BaseEnv` et qui définit l'espace des états, $\mathcal{E}$, celui des actions $\mathcal{A}$, les transitions d'états par une action, ainsi que la récompense via les méthodes `observation_space`, `action_space`, `_apply_rl_action` et `compute_reward`. De cette manière, il est facile de définir plusieurs modélisations du système pour un scénario donné.

Une **expérience** définit les paramètres de l'algorithme utilisé pour maximiser l'espérance de la récompense (DQN, PPO, etc...) ainsi que d'autres paramètres de simulation tels que le nombre de processus Sumo à instancier, et le nombre d'itérations à réaliser durant l'entraînement.

Enfin, un **kernel** est un objet qui fait abstraction de l'API du simulateur utilisé pour l'entraînement, cela permet d'utiliser l'abstraction de Flow pour interagir avec Sumo plutôt que *Traci* et donc de découpler notre code du simulateur. Il serait donc tout à fait possible d'implémenter un kernel Flow pour interagir avec le simulateur de CIL4SYS, *Sim4sys*, directement et donc de supprimer la dépendance à Sumo.

1.6.4 Architectures logicielles

Pour satisfaire les besoins métier tout en gardant l'état de l'art en matière d'infrastructure d'entraînement, nous avons choisi d'entraîner l'agent sur *Sumo* via *Flow* d'une part, et de l'interfacer avec le simulateur de production de CIL4SYS, *Sim4sys* d'autre part. L'analyse des données TomTom sur Issy-les-Moulineaux a permis de déterminer une zone d'étude.

(A) Feux 30677963 à l'état GrG en partant du haut et en allant dans le sens anti-horaire.

(B) File 1 de l'arête 3529628581.

FIGURE 1.2 – Nomenclature utilisée par Sumo.

L'import du terrain d'analyse dans Sumo s'effectue grâce à l'outil d'export d'*Open Street Map* et la fonctionnalité d'import de carte fournie par *Flow*.

Une fois l'agent entraîné, il sera « servi » par un serveur *Flask* communiquant en Websocket via le module Python `asyncio`. Les repères utilisés pour la génération des états étant différents entre *Sumo* et *Sym4sys*, une transformation (qui reste à finaliser à ce jour) doit donc être réalisée lors de l'interaction entre l'agent et le simulateur de production.

Les figures 1.3 et 1.4 schématisent ces architectures.

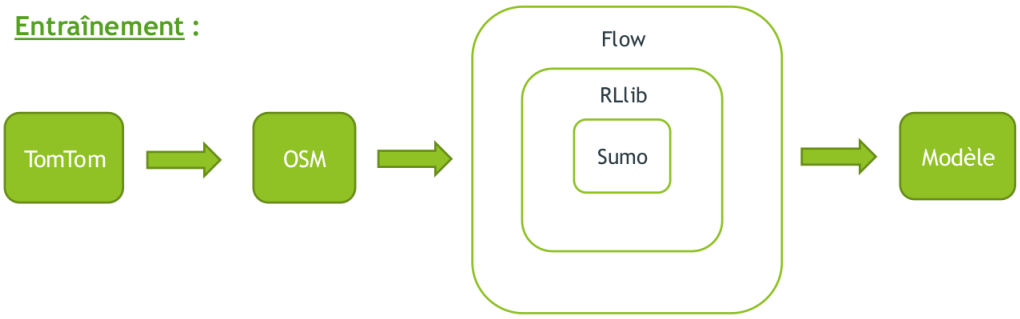


FIGURE 1.3 – Architecture d'entraînement

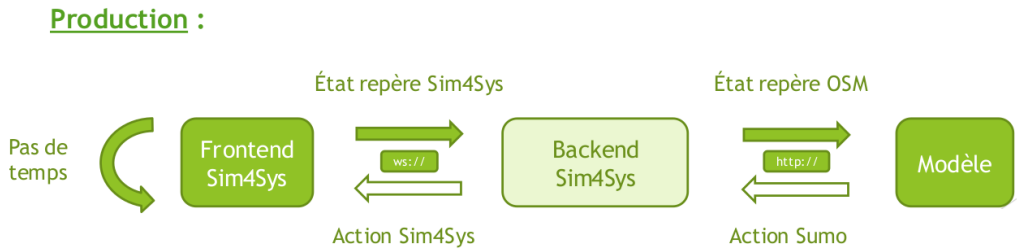


FIGURE 1.4 – Architecture de production

1.7 Quartier d'étude

Le choix du quartier d'étude doit répondre à plusieurs contraintes : la présence de feux tricolores, un trafic qui ne soit pas trop dense comme l'est celui de Paris intra-muros, mais

qui soit suffisamment congestionné pour que notre solution puisse apporter une amélioration. En considération de la haute dimension de la formulation mathématique du problème, nous avons choisi d'étudier un ensemble de 4 intersections à Issy les Moulineaux, contrôlées par trois feux tricolores.

En effet, nous avons observé des congestions dans cette zone en période de pointe à l'aide de l'API TomTom et voudrions étudier si une solution par renforcement peut réduire ces bouchons.

1.7.1 Méthodologie de choix du quartier

La figure 1.5 présente le quartier choisi pour notre modélisation. Le choix du quartier pour la mise en application de notre solution nous a été suggéré par CIL4Sys, et nous avons réalisé l'étude des données et le choix final. L'objectif convenu était de choisir une zone composée de quelques intersections, en zone de trafic dense mais non totalement congestionné.

La choix s'est alors porté sur un quartier d'Issy-les-Moulineaux, ville choisie pour sa densité de trafic et sa centralité, et également car on envisage des possibilités de fluidification. Ce quartier précis a également été choisi pour la présence de feux tricolores aux intersections. La position des feux tricolores a été corroborée par une visualisation du quartier sur OpenStreetMaps.

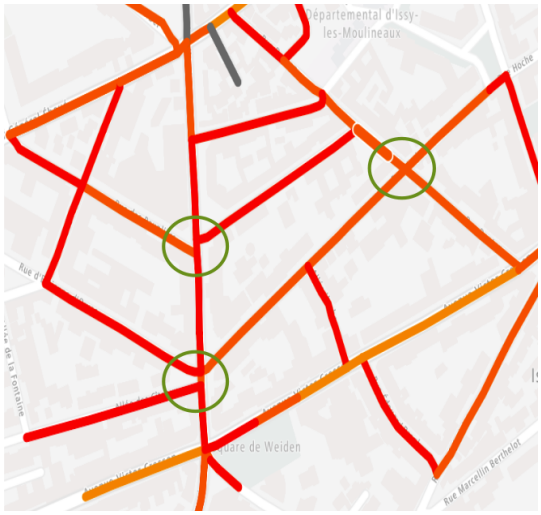


FIGURE 1.5 – Les données de trafic TomTom sur le quartier sélectionné. Les intersections contrôlées par des feux sont encerclées en vert.

1.7.2 Valeurs observées et axes d'amélioration

L'utilisation du portail développeur de TomTom, dont le détail est donné en annexe, a permis d'extraire des valeurs clés concernant le trafic.

Premièrement, après avoir défini le quartier, nous avons défini les plages de temps sur lesquelles concentrer notre analyse. Les données portent sur un historique d'un mois (février 2019). Les tranches horaires ont été divisées en deux pour tenir compte de la réalité de la circulation : une session dite d'heure creuse, de 14h à 16h, et une session dite d'heure pleine de 8h à 10h. Ces deux sections ont été mesurées chaque jour ouvré du mois de février. Les week-ends ont été exclus, car la dynamique du trafic urbain y est différente, et particulièrement à Issy-les-Moulineaux, une ville qui accueille de nombreuses entreprises.

Les données extraites à partir de cette sélection ont été analysées séparément selon qu'il s'agisse de l'heure pleine ou creuse.

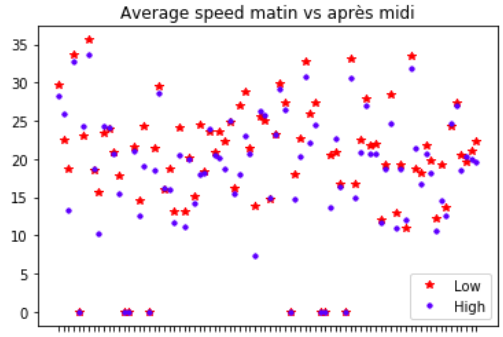


FIGURE 1.6 – Vitesses moyennes en heure creuse et pleine.

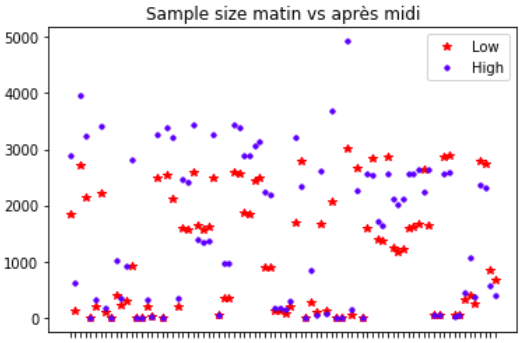


FIGURE 1.7 – Nombre de véhicules en heure creuse et pleine.

Une première analyse a été réalisée sur l'ensemble des données extraites. On a observé les différences en vitesse moyenne, en taille d'échantillon et en écart type de vitesse sur l'intégralité des segments récupérés (figures 1.6, 1.7).

On se rend compte que le nombre de véhicules est, comme attendu, plus élevé sur l'ensemble de la zone en heure pleine, et que la vitesse y est globalement plus faible.

1.7.3 Lien entre Tomtom et Sumo

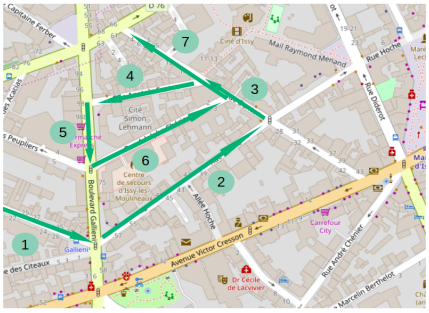
La modélisation de trafic dans Sumo requiert de définir d'une part un environnement, dans notre cas l'architecture des routes, et d'autre part un ensemble de trajectoires possibles pour les véhicules, que l'on appellera rail. Nous avons défini dans Sumo cinq rails possibles pour les véhicules.

La figure 1.8 donne les cinq rails définis. Après identification des segments concernés par ces trajectoires, nous avons pu extraire quelques statistiques descriptives concernant la circulation sur ces routes que nous utilisons comme valeur comparative lors des simulations.

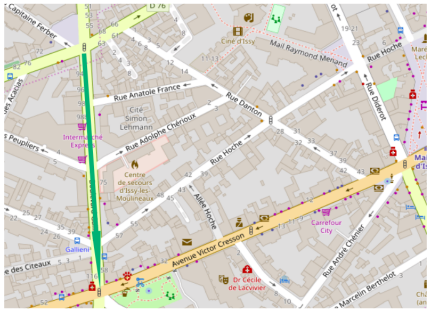
Les tables 1.1 et 1.2 montrent des pistes d'amélioration, notamment en termes de vitesse moyenne sur les rails choisis.

Rail	meanSpeed	travelTime(sum of averages) (s)	meanSampleSize	meanStdDevSpeed	meanStdTravelTime	distance (m)
rail1 (loop)	20.985	284.80	654	8.385	25.27	1025.95
rail2 (sud)	18.690	109.72	2415	9.903	12.63	297.15
rail3 (E O)	19.238	168.08	266	8.408	40.62	553.69
rail4 (O N)	21.127	190.57	3059	10.428	12.56	637.33
rail5 (E N)	17.819	177.18	2946	10.1625	15.82	429.79

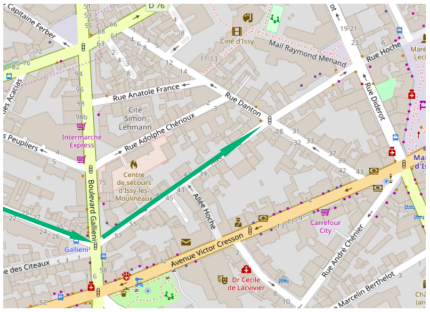
TABLE 1.1 – Résultats en heure pleine.



(A) Trajectoire 1.



(B) Trajectoire 2.



(C) Trajectoire 3.

(D) Trajectoire 4.

(E) Trajectoire 5.

FIGURE 1.8 – Rails définis pour la simulation.

Rail	meanSpeed	travelTime(sum of averages) (s)	meanSampleSize	meanStdDevSpeed	meanStdTravelTime
rail1 (loop)	21.214	264.98	437	8.373	23.49
rail2 (Sud)	19.491	109.10	2785	10.050	12.25
rail3 (E O)	19.350	161.76	170	9.118	26.53
rail4 (O N)	24.970	137.82	2178	10.400	9.07
rail5 (E N)	22.890	138.81	2228	10.435	15.08

TABLE 1.2 – Résultats en heure creuse.

1.8 Modélisations

Le DQN nous permettant de résoudre la formulation mathématique de notre problème en fonction d'un ensemble d'états, d'actions possibles sur ces états et d'une récompense entre ces transitions, il faut maintenant les définir.

1.8.1 États

Dans une logique d'apprentissage, la question devient de trouver une représentation du système à un temps t qui encode toutes les informations nécessaires pour permettre de prendre *la meilleure* action possible selon une certaine mesure qui reste à définir. Sans nécessairement formuler formellement à ce stade cette mesure, nous savons que le but de l'action prise est de fluidifier le trafic. Ainsi, nous aimerions donc que l'état encode des informations relatives à chaque véhicule simulé ainsi qu'aux feux tricolores.

Notre modèle d'état est donc une matrice composée de deux parties, l'une décrivant l'état des voitures, e^v, et l'autre des feux, e^f, à un pas de temps t donné :

$$e = [e^v, e^f], \forall e \in \mathcal{E} \tag{1.13}$$

Représentation des voitures observées, e^v

En pratique l'algorithme de gestion des feux n'aura pas forcément la capacité de connaître l'état de tout les véhicules. Nous proposons donc un modèle où l'état d'apprentissage inclut seulement un sous ensemble de véhicules - envoyés par la mairie pour sonder le trafic par exemple, à l'aube des voitures électriques autonomes, cela semble tout à fait possible. Par ailleurs, la dimensionalité du problème d'étude s'en retrouve réduite.

Formellement, pour β voitures observées à chaque pas de temps t, l'état e_i^v du i ème véhicule observé est décrit de la manière suivante :

$$e_i^v = (x_i, y_i, \theta_i, v_i, a_i, \kappa_i, t_i^{v_0}) \subset \mathbb{R}^7 \text{ pour } i = 1, \ldots, \beta \tag{1.14}$$

Où $x_i, y_i, \theta_i, v_i, a_i, \kappa_i, t_i^{v_0}$ représentent l'ordonnée, l'abscisse, l'orientation, la vitesse absolue, l'accélération, le taux d'émission de CO_2, et le temps passé à l'arrêt pour le i ème véhicule observé. En concaténant les i états, nous obtenons :

$$e^v = [e_1^v, \ldots, e_\beta^v] \subset \mathbb{R}^{7\times\beta} \tag{1.15}$$

Représentation des feux, e^f

Contrairement à l'état des voitures, nous considérons que l'algorithme de gestion des feux a connaissance de l'état de l'ensemble des feux à synchroniser à chaque pas de temps. Bien que le quartier simulé n'ait que trois intersections contrôlées par des feux tricolores, il peut y avoir plus d'un feu par croisement. En effet, chaque file peut avoir un ou plusieurs feux en considérant les feux de type *tourner à droite* ou *U-turn*. En l'occurrence, le quartier

d'analyse comporte 27 feux. De plus, nous allons encoder le nombre de pas de temps que chacune des δ intersections reste dans le même état. Nous aurons un compteur par intersection qui sera réinitialisé à chaque changement d'état.

Par souci de simplicité, nous **restreignons** l'état de chaque feu à *rouge* ou *vert* et **ignorons** le *orange*, cela consiste donc en une représentation binaire. Nous avons donc :

$$e^f \in \{0,1\}^\gamma \times \mathbb{N}^\delta \tag{1.16}$$

Représentation finale, $e \in \mathcal{E}$

Comme évoqué précédemment, l'état $e = e^v + e^f$ de l'environnement à chaque instant t pour les β voitures observées et γ feux contrôlant δ intersections est un élément de $\mathcal{E} \subset \mathbb{R}^{7\times\beta} \times \{0,1\}^\gamma \times \mathbb{N}^\delta$.

1.8.2 Actions

Dans notre modèle, l'agent peut contrôler l'état de chacun des γ feux, chaque action $a \in \mathcal{A}$ est donc un vecteur de γ booléens puisque seul les états *rouge* et *vert* sont considérés dans notre analyse. Il y a donc 2^γ actions possibles sur l'état des feux à chaque pas de temps. Le quartier comportant 27 feux, cela signifie qu'il y a un peu plus de 130 millions d'actions possibles sur le système à chaque pas de temps.

Il est donc nécessaire de choisir un sous ensemble d'actions, $\mathcal{A}^* \subset \mathcal{A}$ de manière à réduire la dimensionalité de notre problème. Par ailleurs, certaines actions théoriquement possibles ne le sont pas en pratique, nous pensons notamment aux deux actions qui passent tous les feux au même état - ce qui au mieux immobiliserait tout le trafic et au pire serait catastrophique.

La cardinalité de $\mathcal{A}^*$ ainsi que ses éléments est arbitraire bien que le but soit de trouver un « petit » ensemble d'actions qui « aient du sens » d'un point de vue de la gestion du trafic. Pour nous aider à la tâche, nous avons essayer de borner $\mathcal{A}^*$ de la manière suivante : pour trois intersections contrôlées par des feux, il faudrait que chaque intersection ait au moins deux états. Alors, si n intersections sont contrôlées par $\gamma \geq n$ feux, le nombre minimal d'actions pour que le modèle ait un sens est de $2^n \leq \gamma^n$. Dans notre cas, pour $n = 3$ et $\gamma = 27$; nous avons donc $|\mathcal{A}^*|$ bornées entre 2^3 et $|\mathcal{A}| - 2 = 2^{27} - 2$ (le -2 provenant des actions qui changent tous les feux au même état).

Pour commencer par le cas le plus simple, nous avons donc recensé deux états de feux par intersection en se demandant lesquelles permettraient de faire passer toutes les voitures à un moment ou à un autre. Les états sélectionnés sont exposés dans la structure de données du Listing 4. Cette structure de données est un dictionnaire ayant pour clefs les chaîne de caractères identifiant les feux contrôlés et pour valeurs une liste de chaîne de caractères décrivant l'état des feux selon le format défini par Sumo. Son type est donc `OrderedDict< String, List<String> >`.

Le fait que le dictionnaire soit ordonné permet de calculer le produit cartésien de ses clefs, qui représente l'ensemble des actions possibles, dans le même ordre à chaque pas de temps et donc de pouvoir associer les actions de l'agent à un nouvel état du système aisément, voir l'algorithme présenté sur le Listing 5.

Nous retenons donc huit états de feux possibles et une action pour chacun d'entre eux. Nous obtenons donc finalement $|\mathcal{A}^*| = 8$.

La figure 1.9 permet de visualiser les différents états que chaque intersection peut prendre.

```
action_spec = OrderedDict({
    # The main traffic light, in sumo traffic light state strings
    # dictate the state of each traffic light and are ordered counter
    # clockwise.
    "30677963": [
        "GGGGrrrGGGG",  # allow all traffic on the main way w/ U-turns
        "rrrrGGGrrrr",  # allow only traffic from edge 4794817
    ],
    "30763263": [
        "GGGGGGGGGG",  # allow all traffic on main axis
        "rrrrrrrrrr",  # block all traffic on main axis to unclog elsewhere
    ],
    "30677810": [  # the smallest of all controlled traffic lights
        "GGrr",
        "rrGG",
    ],
})
```

LISTING 4 – Structure de données encodant les états de feux possibles du système.

```
def map_action_to_tl_states(self, rl_actions):
        """Maps an rl_action list to new traffic light states based on
        `action_spec` or keeps current traffic light states as they are.

        Parameters
        ----------
        rl_actions: int - action number

        Returns: List<String>
        -------
            List of strings of length `self.action_spec.keys()` containing
            the new state configuration for each intersection.
    """
    all_actions = list(itertools.product(*list(self.action_spec.values())))
    return all_actions[rl_actions]
```

LISTING 5 – Fonction d'association d'une action prise à nouvel état des feux : $T : \mathcal{A}^* \longrightarrow \mathcal{E}^f$

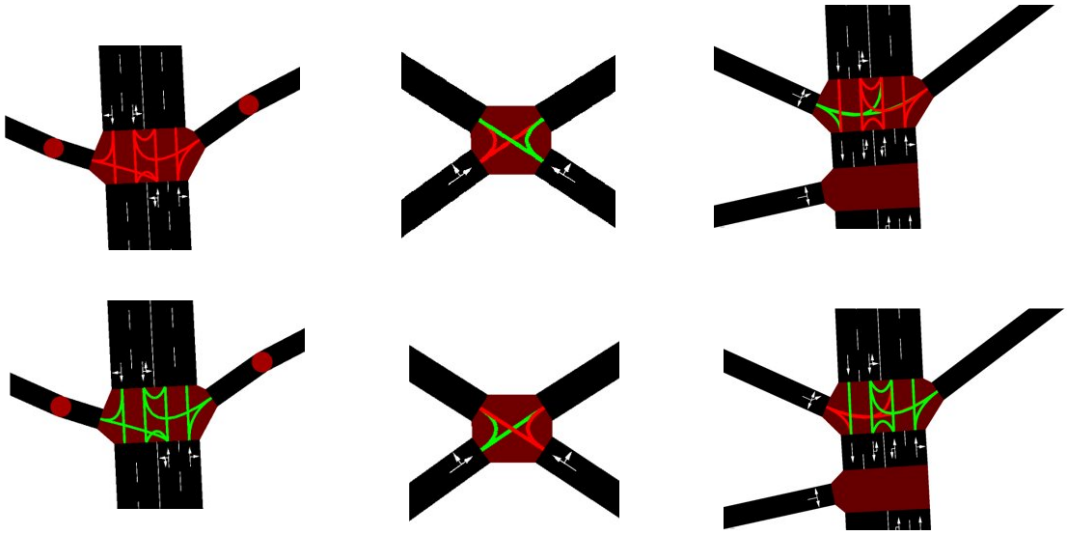


FIGURE 1.9 – Visualisation des états possibles de chaque intersection dans notre modèle.

Contraintes dures sur les actions

Pour empêcher l'agent de changer les feux trop souvent, deux solutions étaient envisageables :

1. Ajouter une neuvième action identité, et guider notre agent à l'aide d'une récompense pénalisant grandement les changements de feux,
2. Ajouter une contrainte algorithmique empêchant les changements trop fréquents des états des feux.

Nous avons finalement opté pour la seconde solution. En effet, la première ne peut pas garantir que les feux ne changeront pas plus fréquemment qu'une contrainte donnée.

La simulation contraint donc les changements de feux de la manière suivante :

— Une borne inférieure T_i spécifiant le nombre minimal de pas de temps qu'une intersection doit maintenir. Si l'agent requête un changement d'état à l'intersection α mais que cette intersection n'a pas encore maintenu son état plus de T_i pas de temps, alors la simulation ne changera pas l'état des feux à l'intersection α.

— Une borne supérieure T_s spécifiant le nombre de pas de temps maximal qu'une intersection peut maintenir à un état de feux donné. C'est à dire que si l'agent ne requête pas de changement d'état à l'intersection α pendant plus de T_s pas de temps, alors la simulation changera automatiquement l'état de cette intersection.

1.8.3 Récompenses

Les états et actions du modèle étant maintenant définis, nous pouvons concevoir les fonctions de récompenses, l'idée étant guider l'agent durant l'apprentissage. Cependant, contrairement à une approche par optimisation sous contraintes, choisir une action de manière à maximiser l'espérance de la récompense ne garantit pas une solution contrainte. Par exemple, si la fonction de récompense pénalise les changements de feux trop fréquents mais récompense autre chose, alors l'agent évitera les changements trop fréquents, *mais ne les exclura pas de la solution*. L'agent pourra par exemple choisir de changer l'état des feux à faibles intervalles occasionnellement si cela maximise l'espérance de la récompense à cet état donné. Une solution apprise par renforcement se comporte donc différemment qu'une

solution à un problème d'optimisation sous contrainte « équivalent ». Nous verrons comment pallier à ce problème ultérieurement et nous focalisons maintenant sur la conception de fonctions de récompenses pertinentes.

Nous voudrions concevoir une fonction de récompense qui encourage :
— le débit des flux
— les voitures en mouvement
Et pénalise :
— les accélérations
— les émissions de CO_2
— les changements de feux
— les voitures arrêtées pendant trop longtemps
Une fonction de récompense simple est par exemple :

$$\psi = \frac{\bar{v}}{\bar{\kappa}}$$

Où $\bar{v}$ et $\bar{\kappa}$ représentent la vitesse moyenne et le taux d'émission moyen des β véhicules observés par l'agent. Le but de cette fonction est de favoriser le débit tout en minimisant les émissions. Cependant, cette fonction est naïve et inadaptée à notre problème puisque la solution serait de laisser l'artère avec le débit de véhicules le plus important et de bloquer tous les autres. Il faudrait donc pouvoir intégrer d'autres variables à la fonction de récompense. L'image de cette fonction étant un sous ensemble de $\mathbb{R}$, ces différentes composantes ne peuvent être qu'additionnées entre elles. Cependant la fonction suivante n'a pas de sens :

$$\psi = C_1\bar{v} - C_2\bar{\kappa} - C_3\bar{t}^{v_0}$$

Où les C_i sont des constantes et $t^{\bar{v}_0}$ la moyenne d'une mesure de temps passé à l'arrêt par les voitures. En effet, le dimensionnement des constantes est très délicat puisque leurs unités sont incompatibles entre elles et ne peuvent être additionnées.

Une façon plus simple de composer la fonction de récompense et de définir un certain nombre de contraintes sur l'état des voitures et de compter les véhicules sous ou sur telle ou telle contrainte. Par exemple, une fonction qui récompense les voitures allant à une vitesse supérieure à v_{min} sera définie de la manière suivante :

$$\psi = \sum_{i=1}^{\beta} \mathbb{I}\{v_i \geq v_{min}\}$$

Cette manière de construire la récompense permet de composer ses termes. La composante due à l'état des voitures de la récompense de notre modèle, ψ^v est la suivante :

$$\psi = \sum_{i=1}^{\beta} \left[C_1 \cdot \mathbb{I}\{v_i \geq v_{min}\} + C_2 \cdot \mathbb{I}\{\kappa_i \leq \kappa_{max}\} - C_3 \cdot \mathbb{I}\{t_i^{v_0} \geq \tau\} - C_4 \cdot \mathbb{I}\{\ddot{x}_i \geq A_{max}\}\right] \tag{1.17}$$

Cette fonction de récompense dépend donc de l'état du système au temps t mais aussi à quatres constantes, « hyperparamètres » en un sens, à accorder à la main pour le moment. En effet, valider une solution revient à la visualiser puis à la valider qualitativement à ce stade. Nous n'avons pas encore réalisé de « grid search » sur ces paramètres, ces constantes doivent donc être ajustées *manuellement*. La figure 1.10 permet de visualiser le calcul de la fonction de coût.

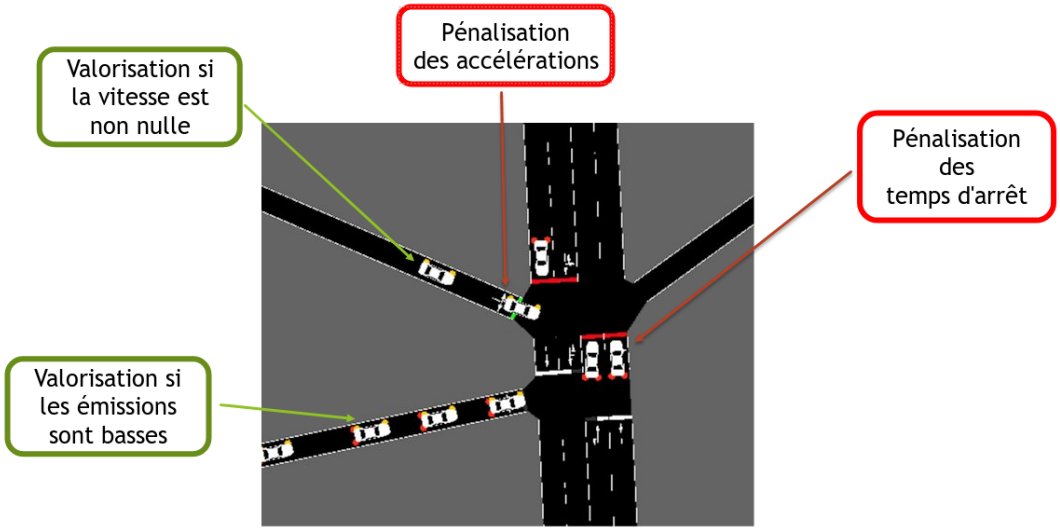


FIGURE 1.10 – Visualisation du calcul de la fonction de coût.

1.8.4 Le réseau de neurones

Dans l'algorithme du DQN, la fonction Q^{π} (Equation 1.4) est approchée par un réseau de neurones en minimisant la différence temporelle (Equation 1.12). Les temps de simulation en apprentissage par renforcement étant très longs, nous avons opté pour un modèle très simple : un réseau de neurones mono-couche de 128 neurones. En effet, la rétro-propagation du gradient s'effectue plus rapidement sur un réseau simple puisqu'il comporte moins de paramètres. La figure 1.11 permet de visualiser ce réseau ainsi que la dimensionalité de ses espaces d'entrée et de sortie.

1.9 Performances de l'agent

En termes d'apprentissage, comme nous pouvons l'observer sur la Figure 1.12, la différence de notre modèle oscille bien proche de 0 (*c.f.* Equation 1.11). De même, nous remarquons que la récompense augmente bien démontrant que le modèle apprend bien.

De manière plus qualitative, la Figure 1.13 permet de visualiser la performance de l'agent en début d'entraînement ainsi qu'en fin d'entraînement.

1.10 Conclusion

La problématique posée d'optimisation du trafic urbain pour limiter les émissions de polluants nous a amenés dans un premier temps à réaliser un état de l'art sur les méthodes mises en œuvre dans le domaine de l'optimisation de trafic urbain, et sur la réduction des émissions de polluants. Cette étude nous a amenés à considérer un modèle d'apprentissage par renforcement pour répondre à nos besoins, ce type de modèle ayant fait ses preuves dans la littérature.

Nous avons utilisé à cet effet le simulateur de trafic Sumo, qui nous a permis de tester les algorithmes développés et leur efficacité. Nous avons mis en place un environnement de simulation en l'approchant des conditions réelles grâce aux données de trafic fournies par l'API Tomtom qui nous donne une évaluation réaliste de demande de trafic.

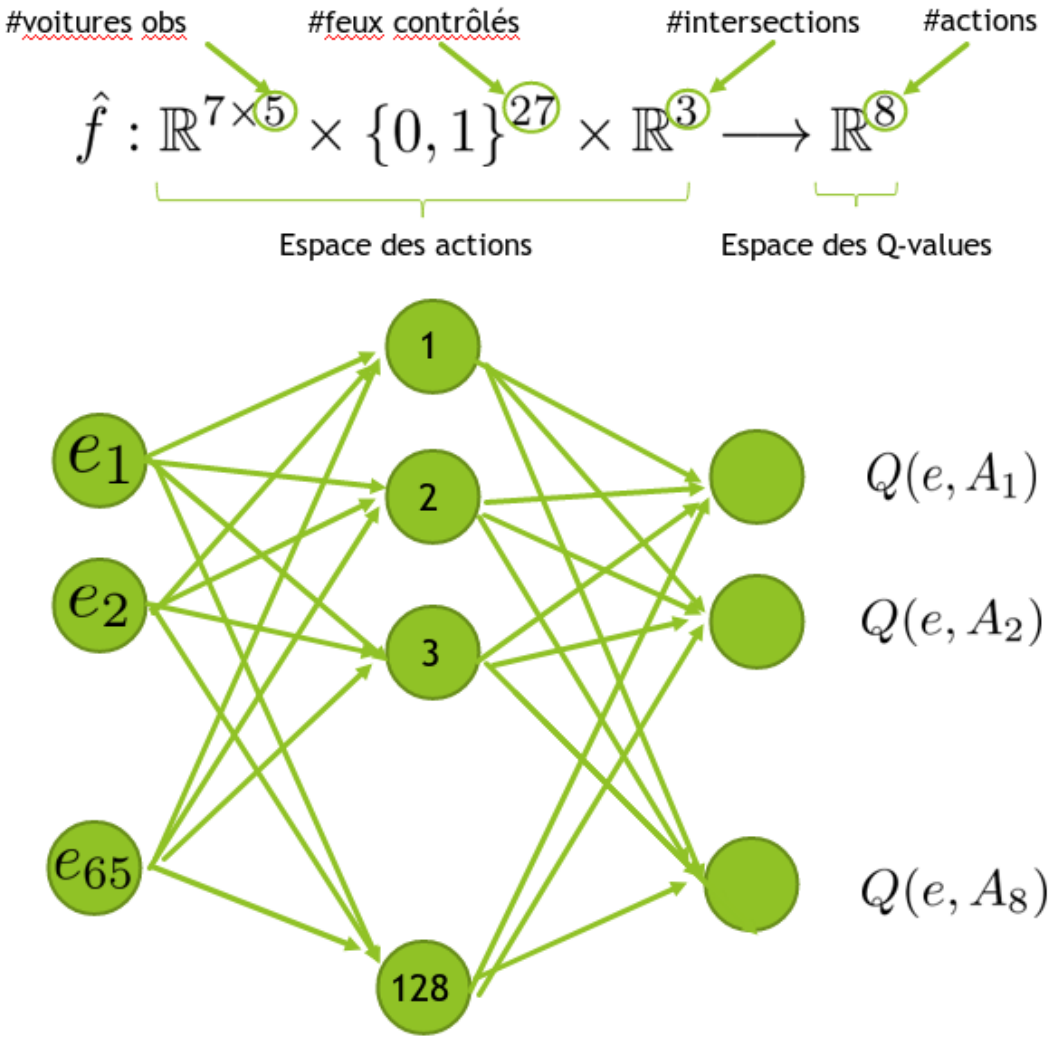
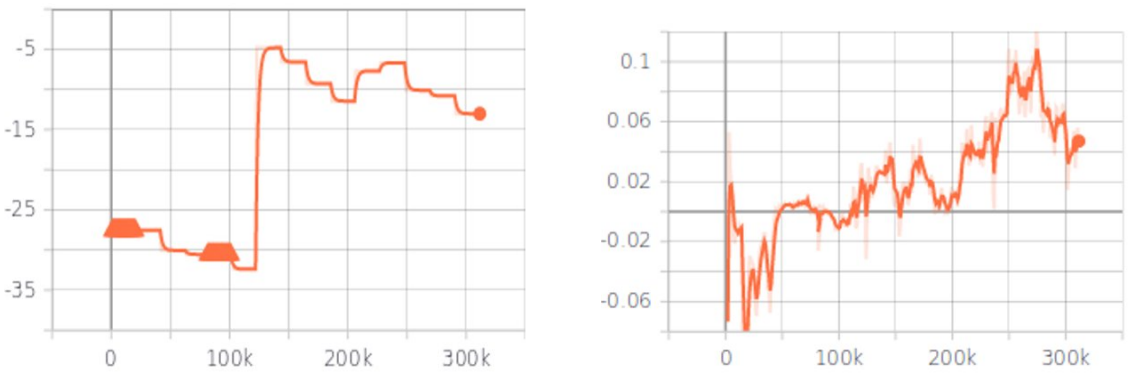
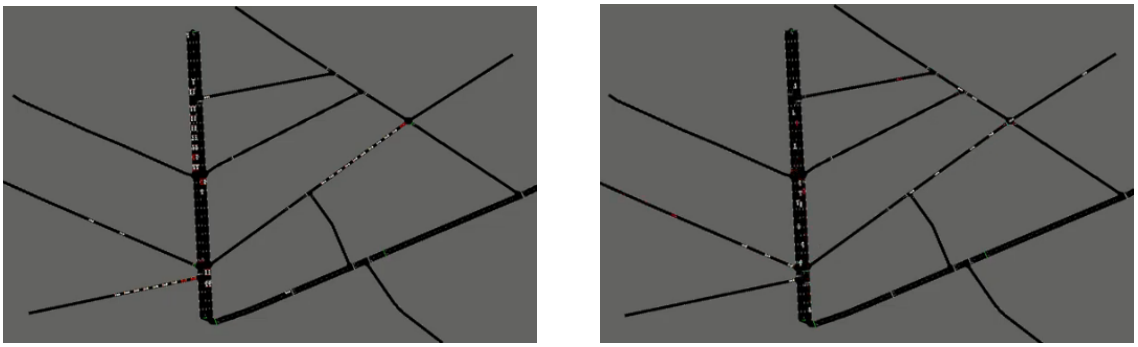


FIGURE 1.11 – Visualisation du réseau de neurones approchant la fonction Q.



(A) Évolution de la récompense moyenne (B) Évolution de la différence temporelle

FIGURE 1.12 – Entraînement de notre modèle



(A) Agent en début d'entraînement (B) Agent en fin d'entraînement

FIGURE 1.13 – Résultat qualitatif : Nous observons qu'au même pas de temps (160), la simulation de l'agent en début d'entraînement est très congestionnée alors que celle du l'agent entraîné ne l'est pas du tout

Les modèles développés n'ont pu être intégrés au simulateur Sim4Sys que partiellement, faute de temps, et dans la mesure où les commandes de voitures ne gèrent pas encore les feux. Cependant, le gros de l'interface entre notre agent et le simulateur Sim4Sys a été réalisé.

Nous aurions aimé pouvoir entraîner notre agent sur des machines plus puissantes que nos ordinateurs, en effet, cela à beaucoup ralenti le temps de calibrage des paramètres de la fonction de récompense. Cela nous aurez aussi permis de pouvoir entraîner un agent sur un espace d'action plus large.

Cependant, le résultats des simulations est satisfaisant dans la mesure où, qualitativement, l'environnement de simulation est bien plus fluide entre le début de l'apprentissage et après. Nous regrettons cependant de ne pas avoir eu le temps de développer un algorithme calculant des statistiques sur l'ensemble d'une simulation pour avoir un résultat quantitatif.

Annexe A

Sumo sous Docker

Pour simplifier le processus d'installation de Sumo, nous avons mis en place un image Docker permettant d'utiliser Sumo en quelques commandes. Nous détaillons dans cette Annexe les étapes à réaliser pour lancer le script de test `sumo.py` dans un container Docker avec le support de l'interface graphique du simulateur.

A.1 Installation

Tout d'abords nous clonnons le repository pour ensuite builder le container.

```
git clone git@github.com:tbinetruy/CIL4SYS
cd CIL4SYS/sumo-docker
sudo docker build -t sumo .
```

A.2 Utilisation

L'environnement Sumo permet de visualiser le système via une interface graphique. Il faut donc permettre au serveur graphique de l'hôte d'accepter des connexions venant de Docker. Les commandes suivantes lancent le script `sumo.py` sur Sumo depuis Docker en mode GUI.

```
# allow connection to x11 server from docker
xhost +

# run sumo.py in dockerised sumo-gui
sudo docker run -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix \
    --privileged -ti --mount src="$(pwd)",target=/host,type=bind  \
    sumo /bin/sh -c "cd /host && python sumo.py"
```

Si plutôt que d'exécuter le script l'utilisateur veux seulement ouvrir un interpréteur dans Docker (avec le support GUI), alors les commandes suivantes sont plus adaptées :

```
# allow connection to x11 server from docker
xhost +

# run sumo.py in dockerised sumo-gui
sudo docker run -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix \
    --privileged -ti --mount src="$(pwd)",target=/host,type=bind  \
    sumo /bin/bash
```

Annexe B

Simpler DQN problems

B.1 Cartpole

Dans le but de se familiariser avec les concepts liés à l'apprentissage par renforcement, nous avons étudié sa mise en oeuvre dans le cadre d'un modèle "jouet" bien connu, celui du pendule inversé. C'est un problème classique de contrôle utilisé pour développer des algorithmes d'apprentissage par renforcement dans un cadre simple.

OpenAI Gym [2] est un framework développé pour faciliter la recherche en apprentissage par renforcement. Cet outil permet de charger des environnements déjà implémentés avec lesquels il est ensuite possible d'interagir. OpenAI Gym permet aux développeurs de ne pas se préoccuper de la partie simulation physique des systèmes pour ne se concentrer que sur le développement d'algorithmes de RL.

Il est possible de charger des environnements de type jeux Atari, jeux de plateau, robots 2D ou 3D (cf. figure B.1).

Même si Gym ne propose pas d'environnement propre au trafic routier, il constitue un framework référence pour l'apprentissage par renforcement, duquel s'inspire en partie le framework Flow que nous utiliserons au cours du projet. La prise en main de Gym constitue donc une bonne entrée en matière.

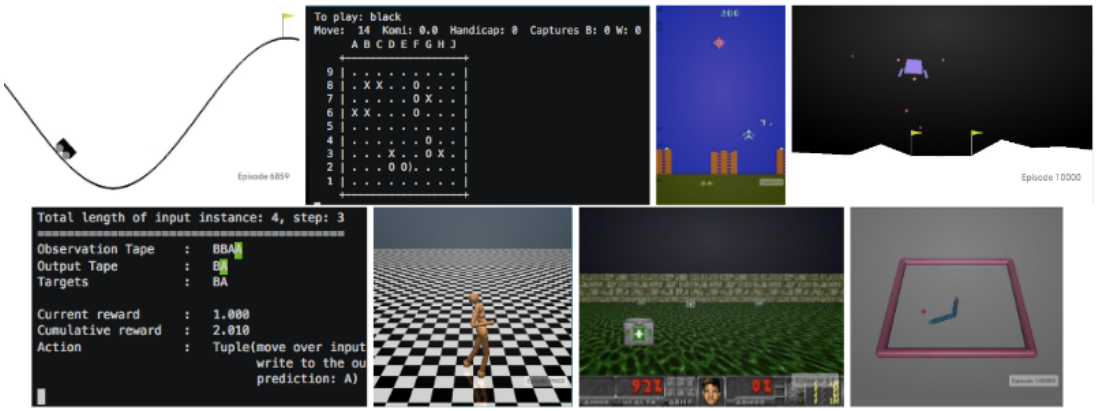


FIGURE B.1 – Exemples d'environnement OpenAI Gym

Le pendule inversé est constitué d'un mat en métal positionné sur un chariot mobile. Le chariot peut se déplacer sur la gauche et à la droite et la tâche de contrôle consiste à maintenir le mat en équilibre. Ce problème de contrôle est facilement soluble par les outils de contrôle classiques (e.g. commande optimale). Cet environnement constitue en revanche un banc d'essai intéressant pour implémenter des premiers algorithmes de RL.

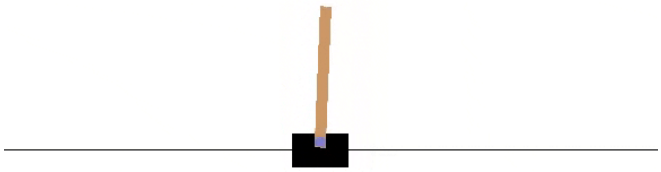


FIGURE B.2 – Environnement pendule inversé

Nous avons choisi dans un premier temps de nous concentrer sur le deep Q-learning. Dans cette approche, l'agent est un réseau de neurones qui prend en entrée l'état du système et donne en sortie l'action à entreprendre.

Dans l'exemple suivant, un agent de deep Q-learning a été codé dans la classe DQNSolver :

```
class DQNSolver:

    def __init__(self, observation_space, action_space):
        self.exploration_rate = EXPLORATION_MAX
        self.action_space = action_space
        self.memory = deque(maxlen=MEMORY_SIZE)
        self.model = Sequential()
        self.model.add(Dense(24, input_shape=(observation_space,),
                             activation="relu"))
        self.model.add(Dense(24, activation="relu"))
        self.model.add(Dense(self.action_space, activation="linear"))
        self.model.compile(loss="mse", optimizer=Adam(lr=LEARNING_RATE))

    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))

    def act(self, state):
        if np.random.rand() < self.exploration_rate:
            return random.randrange(self.action_space)
        q_values = self.model.predict(state)
        return np.argmax(q_values[0])

    def experience_replay(self):
        if len(self.memory) < BATCH_SIZE:
            return
        batch = random.sample(self.memory, BATCH_SIZE)
        for state, action, reward, state_next, terminal in batch:
            q_update = reward
            if not terminal:
                q_update = (reward + GAMMA *
                            np.amax(self.model.predict(state_next)[0]))
            q_values = self.model.predict(state)
            q_values[0][action] = q_update
            self.model.fit(state, q_values, verbose=0)
```

```
        self.exploration_rate *= EXPLORATION_DECAY
        self.exploration_rate = max(EXPLORATION_MIN,
                                    self.exploration_rate)
```

B.2 Deep Q-learning pour une intersection

L'étape suivante consiste à implémenter un algorithme de DQL à un système de trafic routier. Dans cette partie, l'environnement est simulé par l'intermédiaire de SUMO. Le système le plus simple que l'on puisse imaginer pour le contrôle du trafic routier par feux tricolores est une intersection unique. La figure B.3 montre une capture d'écran de l'interface graphique de SUMO au cours d'une simulation.

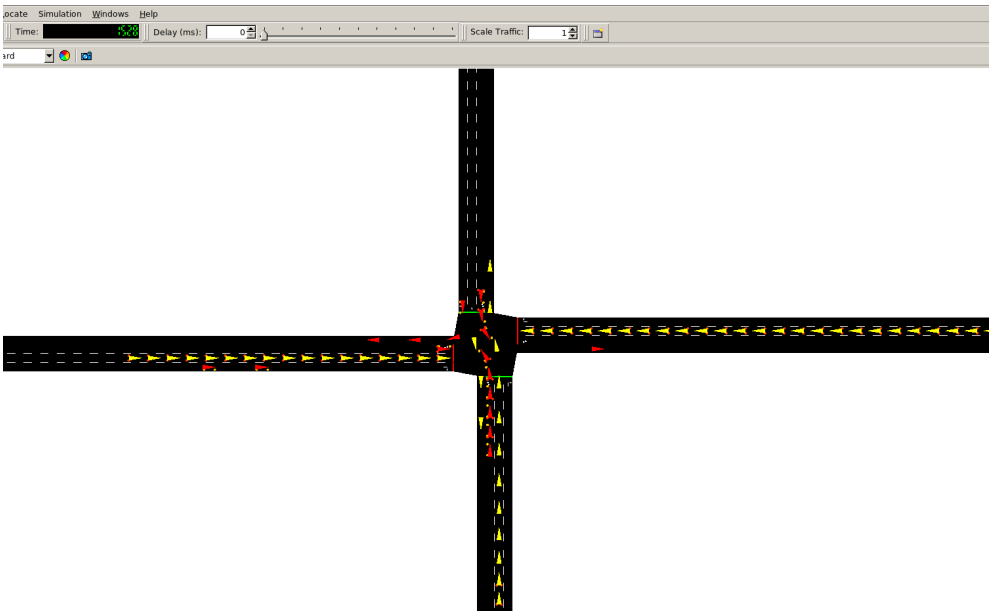


FIGURE B.3 – Intersection SUMO

La démarche est la suivante : on définit une architecture (ici une intersection), une demande de trafic et la période de temps de la simulation. Cela constitue un scénario qu'il est possible de répéter autant de fois que l'on veut. L'intervalle de temps entre chaque pas de simulation peut être configuré, ce qui permet de choisir la "vitesse" d'exécution de la simulation. Dans notre cas, une heure de simulation correspond par exemple à un peu moins d'une minute d'exécution.

Il faut ensuite définir le triplet suivant $(\mathcal{E}_t, \mathcal{A}_t, r_t)$ pour les états, les actions et les récompenses du système.

B.2.1 Etats $\mathcal{E}_t$

Une façon de définir l'état de l'intersection consiste à "découper" chaque voie en cellules, et définir une matrice de positions et de vitesses comme suit (cf. figure B.4) : pour la matrice de positions on associe à chaque cellule la valeur 1 si un véhicule est présent dans la cellule, 0 sinon ; pour la matrice de vitesses, on prend la vitesse du véhicule présent dans la cellule, normalisée par la vitesse limite.

B.2.2 Actions $\mathcal{A}_t$

L'agent a le choix entre deux actions $\mathcal{A}_t = \{0,1\}$:

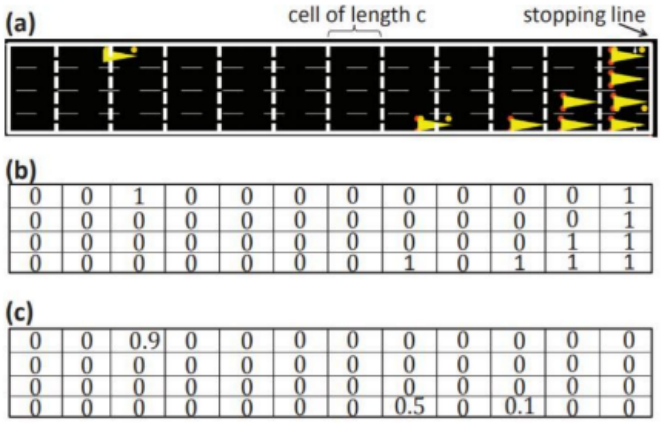


FIGURE B.4 – Etat de l'intersection

- 0 $\rightarrow$ feu vert pour la direction Est-Ouest
- 1 $\rightarrow$ feu vert pour la direction Nord-Sud

B.2.3 Récompenses r_t

La récompense est définie à partir des délais encourus par les véhicules sur les voies menant à l'intersection. Pour chaque véhicule, les temps de décélération, d'arrêt et d'accélération conduisent à un retard en comparaison avec un scénario sans intersection. On peut donc définir un retard cumulé pour l'ensemble des véhicules sur le réseau. A chaque fois qu'un véhicule traverse l'intersection, on soustrait son retard du retard cumulé. De cette façon il est possible de définir une récompense de la manière suivante :

$$r_t = C_d^{g^-} - C_d^{g^+} \tag{B.1}$$

où $C_d^{g^-}$ est le retard cumulé au début de la phase de feu vert après l'action a_t et $C_d^{g^+}$ le retard cumulé à la fin de la phase.

Avec une telle définition, on voit que l'agent de RL reçoit après chaque action un signal qui l'"informe" de la qualité de l'action entreprise. Une récompense positive traduit une diminution du retard cumulé et une récompense négative une augmentation de ce retard.

Choisir une récompense r_t revient d'une certaine façon à choisir ce que l'on veut optimiser. D'autres possibilités pourraient être la minimisation de la longueur des files d'attentes, la minimisation du niveau d'émissions, etc...

B.2.4 Architecture du DQN

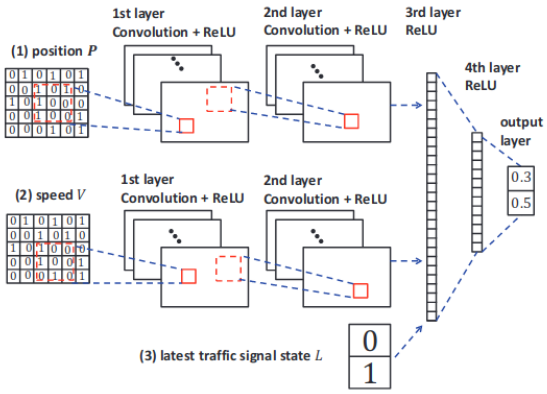


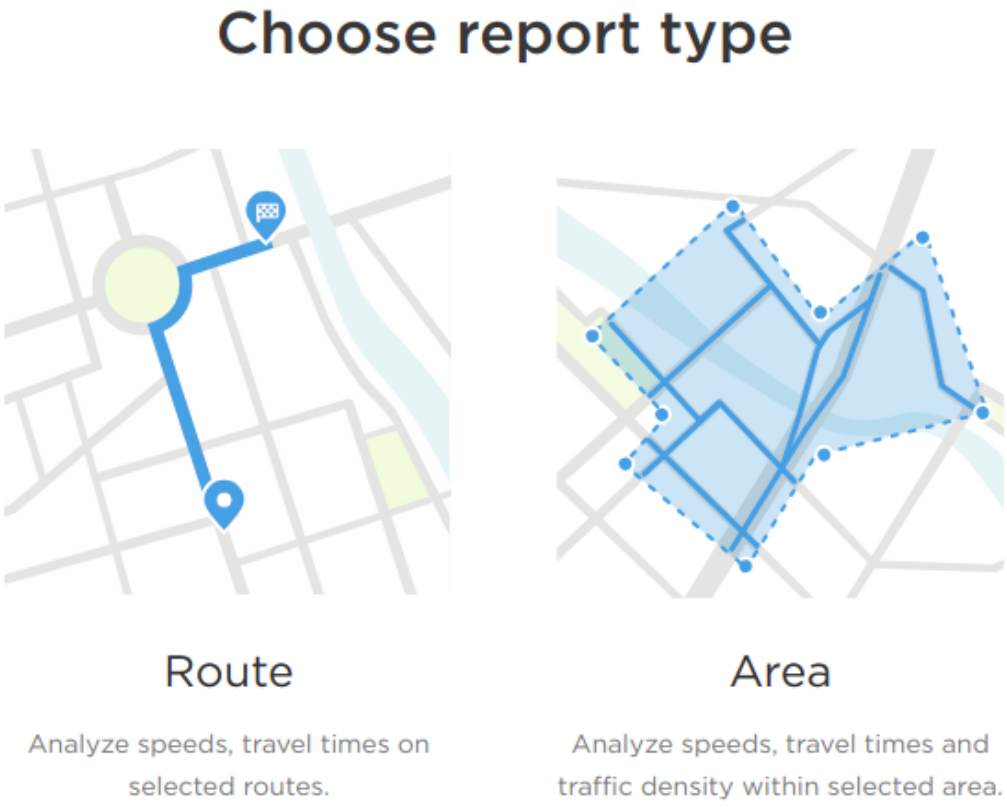
FIGURE B.5 – Architecture du DQN

Annexe C

Guide Tomtom

C.1 Utilisation du portail Tomtom Move

Deux options :



C.1.1 Routes

On peut obtenir des données similaires en version gratuite sans interface graphique (voir plus loin).

Lien vers la documentation ici.

Possibilité de choisir : départ, arrivée et éventuellement point de transit (à la Google Maps) On peut ajouter jusqu'à 20 routes.

Etape 2 : ajout de dates (jours de la semaine, étendue du temps pour l'historique)

C.1.2 Area

On sélectionne la zone à la main en dessinant un polygone sur la carte. Il est possible d'effectuer des sélections par jour de la semaine et par horaire. Les résultats seront présentés dans le fichier de résultat avec les différents éléments choisis.

C.2 Résultats

On récupère un fichier json.

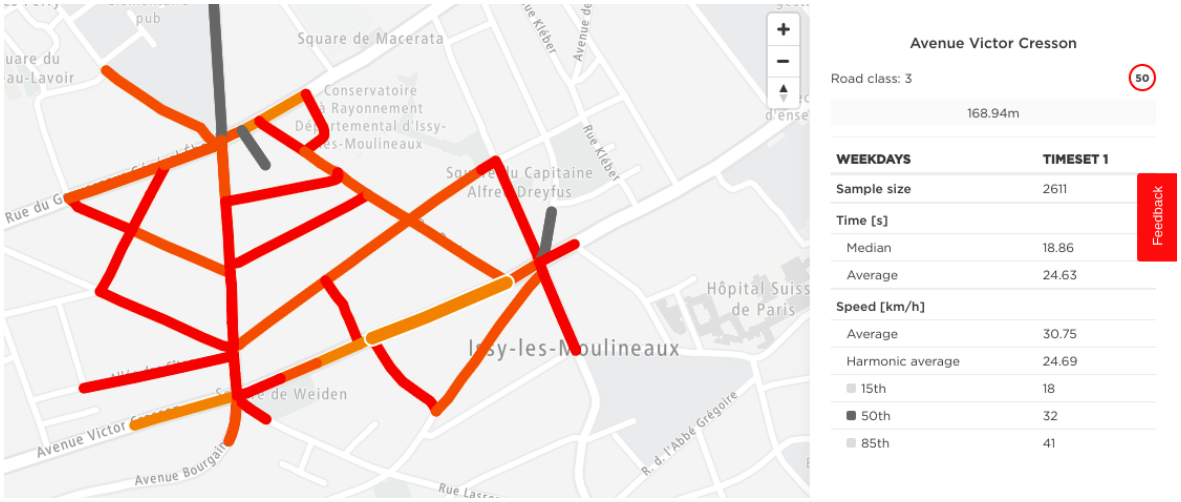
C.2.1 Structure du json pour la section area

Les premiers items sont les caractéristiques de ce qu'on a demandé.

```
"jobName": "Issy",
"creationTime": "2019-04-06T15:30:48.109Z",
"userPreference": {
    "distanceUnit": "KILOMETERS"
},
"dateRanges": [
    {
        "@id": 1,
        "name": "Fevrier2019",
        "from": "2019-02-01",
        "to": "2019-02-28",
        "exclusions": [],
        "excludedDaysOfWeek": []
    }
],
```

Dans cet exemple, on a choisi un seul univers de temps (dateRanges peut se composer de plus d'unités).

Ensuite on a un item par segment. Les segments sont visibles sur le portail TomTom.



En sélectionannt le segment, on obtient sur la partie droite de l'écran ses caractéristiques. Une des difficultés de manipulation de l'API est de trouver les correspondances entre les segments présentés sur la carte et les segments identifiés par Id dans l'API.

Une solution proposée est si l'on connaît les coordonnées GPS de la route, les utiliser car elles sont récupérées par la requête.

Exemple

Données générales sur le segment :


```
"segmentResults": [
    {
        "segmentId": -12500001162438,
        "newSegmentId": "-00004632-3000-0400-0000-00000004739d",
        "speedLimit": 30,
        "frc": 6,
        "streetName": "Rue Danton",
        "distance": 43.22,
        "shape": [
            {
                "latitude": 48.82469,
                "longitude": 2.27008
            },
            {
                "latitude": 48.82482,
                "longitude": 2.26978
            },
            {
                "latitude": 48.82492,
                "longitude": 2.2696
            }
        ],
```


On a ensuite les jeux de résultats par tranche horaire qu'on a choisie : La tranche est donnée par 'timeSet' et dateRange.


```
"segmentTimeResults": [
                    {
                        "timeSet": 2,
                        "dateRange": 1,
                        "harmonicAverageSpeed": 24.74,
                        "medianSpeed": 29.13,
                        "averageSpeed": 28.23,
                        "standardDeviationSpeed": 8.19,
                        "travelTimeStandardDeviation": 9.97,
                        "sampleSize": 2891,
                        "averageTravelTime": 17.05,
                        "medianTravelTime": 14.48,
                        "travelTimeRatio": 1.0,
                        "speedPercentiles": [
```


On a les moyennes sur les vitesses. sampleSize donne la taille de l'échantillon sur lequel sont calculés les statistiques. On utilise ces données comme référence.

C.3 Analyse de trajets

On peut aussi utiliser la version non graphique de l'API (accessible sans licence) pour obtenir les détails d'un trajet (temps, vitesse moyenne). Il faut pour cela entrer les coordonnées du point de départ et du point d'arrivée.

On obtient avec cette requête un temps de trajet à l'instant où on fait la requête :


```
"summary": {
            "lengthInMeters": 1370,
            "travelTimeInSeconds": 467,
            "trafficDelayInSeconds": 125,
            "departureTime": "2019-06-18T18:39:31+02:00",
            "arrivalTime": "2019-06-18T18:47:18+02:00"
        },
        "legs": [
            {
                "summary": {
                    "lengthInMeters": 1370,
                    "travelTimeInSeconds": 467,
                    "trafficDelayInSeconds": 125,
                    "departureTime": "2019-06-18T18:39:31+02:00",
                    "arrivalTime": "2019-06-18T18:47:18+02:00"
```


Ceci est un extrait du résultat, l'API renvoie également tous les points du segements nécessaires à son tracé.

Cela peut nous permettre de comparer les résultats de nos modèles à des données réelles, notamment pour la vitesse moyenne et l'écart-type. Comme on cherche à éviter la situation où un véhicule avance avec tous les autres véhicules arrêtés, on pourra utiliser l'écart-type comme indicateur de la non-dégradation de la situation au carrefour.

On a également utilisé la taille d'échantillon comme proxy pour définir les flux entrants de véhicules dans le simulateur.

Bibliographie

[1] Kyoungho AHN et al. « MICROSCOPIC FUEL CONSUMPTION AND EMISSION MODELING ». In : (jan. 2019).

[2] Greg BROCKMAN et al. « OpenAI Gym ». In : *arXiv* (2016). URL : http://arxiv.org/abs/1606.01540.

[3] Yuanfang CHEN et al. « DeepTFP : Mobile Time Series Data Analytics based Traffic Flow Prediction ». In : (oct. 2017).

[4] Samah EL-TANTAWY et Baher ABDULHAI. « Multi-Agent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controllers (MARLIN-ATSC) ». In : sept. 2012, p. 319–326. ISBN : 978-1-4673-3064-0. DOI : 10.1109/ITSC.2012.6338707.

[5] Juntao GAO et al. « Adaptive Traffic Signal Control : Deep Reinforcement Learning Algorithm with Experience Replay and Target Network ». In : *arXiv* (2017). URL : http://arxiv.org/abs/1705.02755.

[6] José GARCÍA-NIETO, Enrique ALBA et Ana OLIVERA. « Swarm intelligence for traffic light scheduling : Application to real urban areas ». In : *Eng. Appl. of AI* 25 (2012). DOI : 10.1016/j.engappai.2011.04.011.

[7] Wade GENDERS et Saiedeh RAZAVI. « Using a Deep Reinforcement Learning Agent for Traffic Signal Control ». In : *arXiv* (2016). URL : http://arxiv.org/abs/1611.01142.

[8] Junchen JIN et Xiaoliang MA. « Hierarchical multi-agent control of traffic lights based on collective learning ». In : *Engineering Applications of Artificial Intelligence* (2017). DOI : 10.1016/j.engappai.2017.10.013.

[9] Mohamed A. KHAMIS et Walid GOMAA. « Adaptive Multi-objective Reinforcement Learning with Hybrid Exploration for Traffic Signal Control Based on Cooperative Multi-agent Framework ». In : *Engineering Applications of Artificial Intelligence* 29 (2014). DOI : 10.1016/j.engappai.2014.01.007.

[10] Daniel KRAJZEWICZ et al. « Recent Development and Applications of SUMO - Simulation of Urban MObility ». In : *International Journal On Advances in Systems and Measurements*. International Journal On Advances in Systems and Measurements 5.3&4 (2012), p. 128–138. URL : http://elib.dlr.de/80483/.

[11] Pablo Alvarez LOPEZ et al. « Microscopic Traffic Simulation using SUMO ». In : *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL : https://ieeexplore.ieee.org/document/8569938.

[12] Marin LUJAK, Stefano GIORDANI et Sascha OSSOWSKI. « Fair route guidance : Bridging system and user optimization ». In : *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*. IEEE. 2014, p. 1415–1422.

[13] Patrick MANNION, Jim DUGGAN et Enda HOWLEY. « An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control ». In : *Autonomic Road Transport Support Systems* (2016), p. 47–66. DOI : 10.1007/978-3-319-25808-9_4.

[14] Yukimasa MATSUMOTO et Kazuya NISHIO. « Reinforcement Learning of Driver Receiving Traffic Signal Information for Passing through Signalized Intersection at Arterial Road ». In : *Transportation Research Procedia* 37 (2019). 21st EURO Working Group on Transportation Meeting, EWGT 2018, 17th – 19th September 2018, Braunschweig, Germany, p. 449 –456. ISSN : 2352-1465. DOI : https://doi.org/10.1016/j.trpro.2018.12.219. URL : http://www.sciencedirect.com/science/article/pii/S2352146518306434.

[15] Yukimasa MATSUMOTO, Tatsuya OSHIMA et Ruka IWAMOTO. « Effect of Information Provision around Signalized Intersection on Reduction of CO2 Emission from Vehicles ». In : *Procedia - Social and Behavioral Sciences* 111 (2014). Transportation : Can we do more with less resources ? – 16th Meeting of the Euro Working Group on Transportation – Porto 2013, p. 1015 –1024. ISSN : 1877-0428. DOI : https://doi.org/10.1016/j.sbspro.2014.01.136. URL : http://www.sciencedirect.com/science/article/pii/S1877042814001372.

[16] Volodymyr MNIH et al. « Human-level control through deep reinforcement learning ». In : *Nature* 518 (2015), p. 529–33. DOI : 10.1038/nature14236.

[17] Volodymyr MNIH et al. « Playing Atari with Deep Reinforcement Learning ». In : *arXiv* (2013). URL : http://arxiv.org/abs/1312.5602.

[18] Hesham RAKHA, Kyoungho AHN et Antonio TRANI. « Development of VT-Micro model for estimating hot stabilized light duty vehicle and truck emissions ». In : *Transportation Research Part D : Transport and Environment* 9.1 (2004), p. 49 –74. ISSN : 1361-9209. DOI : https://doi.org/10.1016/S1361-9209(03)00054-3. URL : http://www.sciencedirect.com/science/article/pii/S1361920903000543.

[19] Hesham RAKHA et al. « Intersection Management Using In-Vehicle Speed Advisory/Adaptation ». In : (sept. 2016).

[20] David SILVER et al. « Mastering the game of Go without human knowledge ». In : *Nature* 550 (2017), p. 354–359. DOI : 10.1038/nature24270.

[21] Aleksandar STEVANOVIC et al. « Optimizing traffic control to reduce fuel consumption and vehicular emissions : Integrated approach with VISSIM, CMEM, and VISGAOST ». In : *Transportation Research Record : Journal of the transportation research board* 2128 (2009), p. 105–113.

[22] Richard S SUTTON et Andrew G BARTO. *Reinforcement learning : An introduction*. MIT press, 2018.

[23] Tessa TIELERT et al. « The impct of traffic-light-to-vehicle communication on fuel consumption and emmisions ». In : nov. 2010. DOI : 10.1109/IOT.2010.5678454.

[24] Cathy WU et al. « Flow : Architecture and Benchmarking for Reinforcement Learning in Traffic Control ». In : *arXiv* (2017). URL : http://arxiv.org/abs/1710.05465.

[25] Dongbin ZHAO, Yujie DAI et Zhen ZHANG. « Computational Intelligence in Urban Traffic Signal Control : A Survey ». In : *IEEE Transactions on Systems, Man, and Cybernetics - TSMC* 42 (2012), p. 485–494. DOI : 10.1109/TSMCC.2011.2161577.