Étude d'algorithmes pour le problème du «Busybus».

Alejandro Caicedo

16/08/2017

Résumé

On s'intéresse à un modèle très simplifié de transport urbain. On étudie deux stratégies simple de navigation d'un bus et une plus complexe grâce à une application qui permet de connaître la destination des passagers, et ceux avant même qu'ils montent dans le bus. Finalement on s'intéresse au Machine learning afin de connaître la quantité d'information nécessaire pour résoudre ce problème.

Table des matières

Introduction

On étudie le problème du busybus [1], le concept est un modèle simplifié d'un réseau de bus. Un bus se déplace sur un réseau circulaire, comme dans la Figure 1.

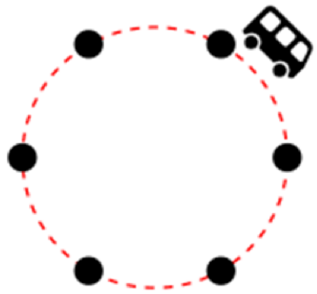

FIGURE 1 – Schéma du busybus, un bus se déplaçant dans un réseau circulaire, les points symbolisent les stations.

À chaque itération, le bus peut se déplacer d'une station à une autre adjacente (aller à gauche ou à droite) ou ne pas bouger. On suppose que le conducteur possède un programme qui lui indique où les personnes se trouvent, à chaque itération, et où elles veulent aller. Le but est de créer une stratégie de navigation, celle-ci doit indiquer à chaque itération où il doit se déplacer et qui il laisse monter (vu la capacité finie du bus) pour minimiser le nombre de personnes qui attendent dans les stations. Dans toute la suite on va utiliser le terme pps (people per station), c'est le nombre de personnes qui attendent moyenné par le nombre de stations. Dans le modèle standard, on suppose que la capacité est $C = 10$ et le nombre de stations est $N = 20$.

À chaque itération une personne arrive à une station a dirigé vers une station b (a est choisie aléatoirement uniformément puis, b est choisie de la même manière parmi les autres stations). Des personnes se trouvant là où est le bus (il commence à la station 0) peuvent monter s'ils sont autorisés (ceci est dicté par la stratégie de navigation). Le bus avance (ou pas, car il peut rester dans la même station) dans la direction établie par la stratégie de navigation. Le compteur de temps augmente, les passagers ayant atteint leur destination descendent, puis on passe à la prochaine itération.

1 Les stratégies de navigation primitives

1.1 Clock vs Greedy

D'abord on analyse les deux stratégies suivantes :

Clock [2] : Le bus se déplace toujours dans la même direction et à chaque station les personnes montent par ordre d'arrivée jusqu'au moment où le bus soit plein ou la station vide.

Greedy [2] : Le bus se dirige vers la destination du plus ancien passager, s'il y en a un, sinon il continue dans le même sens. Les personnes qui montent dans le bus sont choisies de la même manière que pour Clock.

On peut voir dans la Figure 2, l'évolution du pps au cours de $I = 10^7$ itérations.

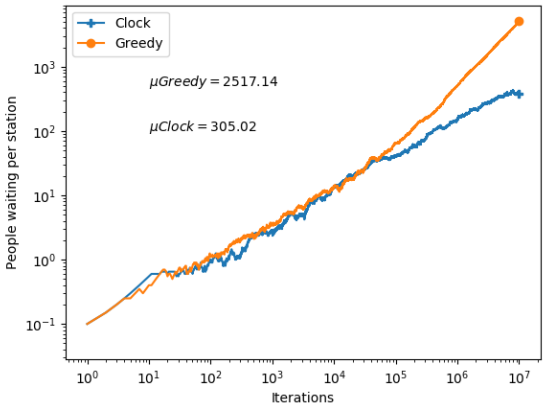

FIGURE 2 – L'évolution des pps au cours du temps pour les stratégies Clock et Greedy. [2]

On a également projeté les données sur des histogrammes, voir la Figure 3.

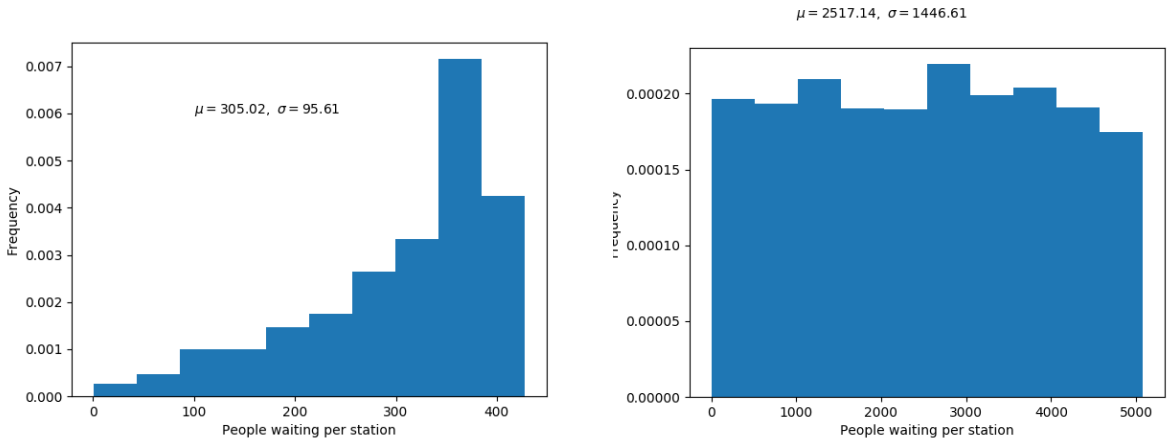

FIGURE 3 – Histogrammes de pps pour Clock (à gauche) et Greedy (à droite). [2]

On s'aperçoit que dans ce contexte ces deux stratégies ne marchent pas bien. En effet plus le temps passe plus les stations se remplissent, bien que pour la stratégie Clock c'est moins vite que pour la stratégie Greedy. Cependant, dans un contexte différent elles peuvent bien fonctionner, si par exemple le bus a une capacité différente ou s'il y a plusieurs bus, ce qui fera l'objet de la section suivante.

1.2 Clock avec une plus grande capacité

Si on augmente la capacité du bus avec la stratégie Clock, elle fonctionne assez bien. D'ailleurs, un bus avec une capacité d'au moins 38 personnes va se comporter comme si sa capacité était infinie. En effet, le plus qu'une personne peut attendre dans le bus est 19 itérations et le plus qu'elle peut attendre dans la station est également 19 itérations. Par conséquent le bus va être peut-être plein à un certain moment mais, comme les personnes descendent avant de monter il va toujours avoir de la place. On peut voir dans la Figure 4, l'évolution du pps et son histogramme.

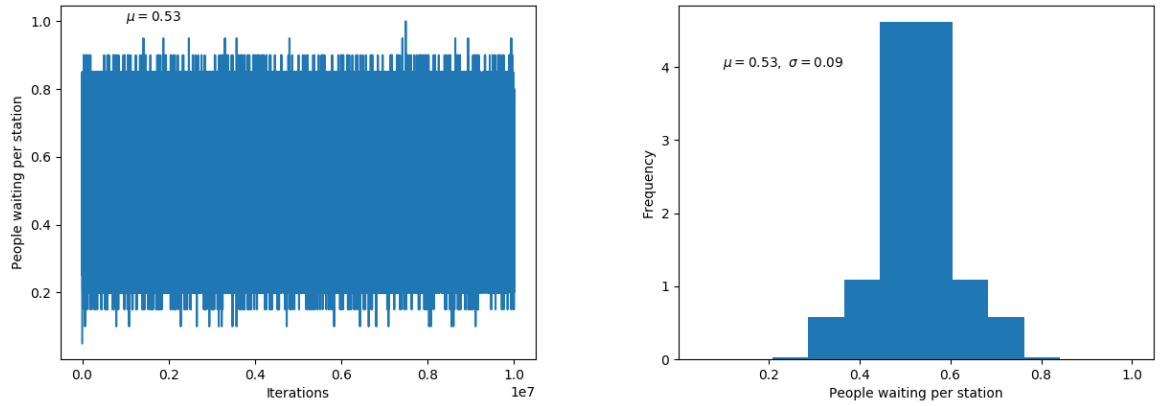

FIGURE 4 – Évolution du pps au cours du temps (à gauche) et son histogramme (à droite), pour la stratégie Clock ($C = 40$ et $N = 20$). [2]

La valeur moyenne de 0,53 est très proche de l'espérance réelle, en effet la valeur exacte (après un nombre infini d'itérations) est $\frac{1}{2} + \frac{1}{2N} = 0,525$, ce qu'on va démontrer ci-dessous.

Au moment $t = i$, une personne arrive à la station a_i. La position du bus au temps $t = i$ est b_i, et $b_{i+1} = ((b_i + 1) \mod N)$. Sa capacité est supposée $C = \infty$. Soit $X_i := ((a_i - b_i) \mod N)$ le temps attendu par cette personne, c'est-à-dire que $X_i \in \{0, \dots, N-1\}$ est le nombre d'itérations que cette personne doit attendre pour que le bus arrive. C'est une copie i.i.d. de la variable aléatoire X uniformément repartie en $\{0, \dots, N-1\}$. Le nombre de personnes qui attendent à l'itération i est :

$$W_i := \sum_{j=0}^{N-2} 1_{X_{i-j} > j},$$

où $1_{(\cdot)}$ est la fonction indicatrice. On a l'espérance :

$$\mathbb{E}[W_i] = \sum_{j=0}^{N-2} \mathbb{E}[1_{X_{i-j}>j}] = \sum_{j=0}^{N-2} \mathbb{E}[1_{X>j}] = \sum_{j=1}^{N-1} \frac{j}{N} = \frac{N-1}{2}.$$

Alors,

$$\mathbb{E}[\text{pps}] = \frac{N-1}{2N} + \frac{1}{N} = \frac{N+1}{2N}.$$

On a ajouté $\frac{1}{N}$ puisque le pps prend en compte la personne qui vient d'arriver à une station.

1.3 Clock avec deux bus

À titre comparatif on a également étudié le cas où il y a 2 bus [3] qui circulent dans le système dans des sens différents (le cas de la réalité, ce qui fonctionne beaucoup mieux, surtout pour les utilisateurs car il y a toujours un bus qui va prendre la route la plus courte pour aller à leur destination). Le nombre de stations est le même ($N = 20$), on peux voir dans la Figure 5 la courbe des pps et son histogramme pour cette situation.

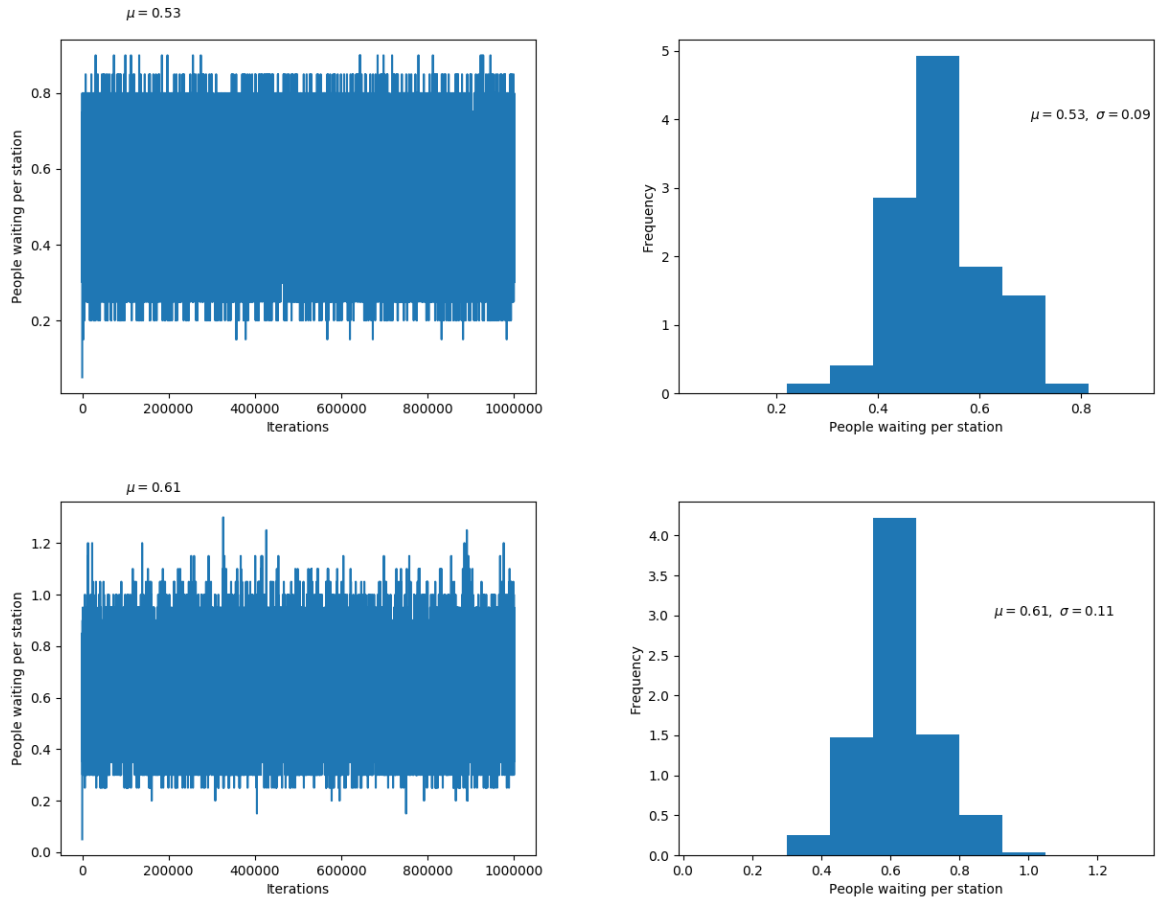

FIGURE 5 – Évolution du pps au cours du temps (à gauche) et son histogramme (à droite), pour la stratégie Clock avec 2 bus, avec une capacité $C = 10$ chacun (première ligne) et $C = 5$ chacun (deuxième ligne). [3]

2 Stratégie de Rui

On veut cependant une stratégie qui fonctionne bien mais sans devoir augmenter la capacité ni le nombre de bus, c'est le cas de la proposition de Rui Viana [4]. L'algorithme consiste en ceci :

À chaque itération l'algorithme regarde différents chemins possibles et calcule celui qui a le meilleur "score". Le bus alors va dans la direction du chemin gagnant. De même, à chaque station il décide quels passagers vont monter dans le bus en favorisant ceux qui resteront moins de temps dans le bus (le chemin du bus étant déjà fixé).

Plus concrètement :

1. Choix du chemin du bus :
   L'algorithme évalue 10 chemins possibles, chacun de 25 itérations :
   (a) Un dans le sens actuel du bus (sans changer) et un dans le sens contraire (2 chemins possibles).
   (b) Des chemins qui font continuer le bus dans la même direction pendant $j = 1, ..., 7$ itérations suivantes puis les $25 - j$ itérations restantes dans le sens contraire (7 chemins possibles).
   (c) Un dernier chemin dans lequel le bus revient à la station qu'il vient de visiter, avant de continuer dans le sens actuel (1 chemin possible).
2. Choix des passagers :
   Pour chaque chemin possible l'algorithme décide qui monte dans le bus à chaque station. En favorisant les personnes qui vont rester peu de temps dans le bus, c'est-à-dire qui vont à des stations pas très lointaines du chemin qui va suivre le bus. S'il a la capacité suffisante il va également prendre les personnes qui vont plus loin. Une fois ce choix est fait l'algorithme vérifie s'il est possible de prendre les personnes plus tôt dans sa trajectoire, pour minimiser le temps d'attente dans les stations.
3. Fonction de score :
   Les personnes qui sont délivrées à l'itération j (la différence entre l'itération où il descend du bus et l'itération actuelle) ont un score $f(i)$. La fonction f a une décroissance exponentielle (favorisant les personnes qui resterons le moins de temps dans le bus).

La Figure 6 montre l'évolution et son histogramme du pps, pour cette stratégie.

On s'aperçoit que cette stratégie est bien meilleure que Clock et Greedy, avec une moyenne de pps en dessous de 1. Cependant cette stratégie étant plus compliquée en termes de complexité, elle prend 5 fois plus de temps à computer que Clock et Greedy, à condition que les stations soient peu remplies (ceci est toujours le cas dans les conditions standards). Car sinon, l'algorithme de Rui va examiner plus de cas et va avoir une complexité encore pire.

À part du pps, il est intéressant de connaître la distribution des temps d'attente (dans la station, dans le bus et le temps total du voyage). Cela peut être observé dans les Figures 7 et 8.

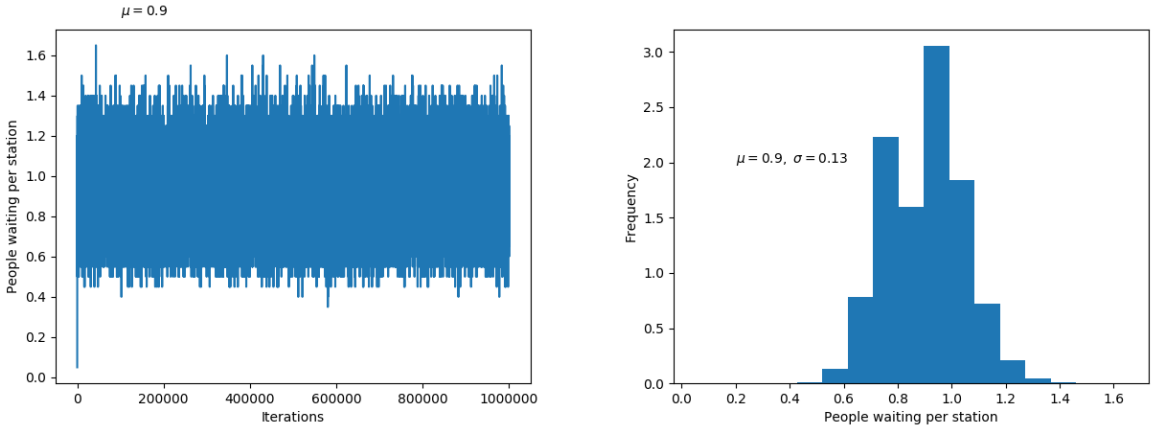

FIGURE 6 – Évolution du pps au cours du temps à gauche et son histogramme à droite, pour la stratégie de Rui. [4]

On s'aperçoit que la stratégie de Rui donne pour la plupart des personnes des temps de transport assez courts. On voit dans les histogrammes tronqués en abscisse (Figure 8), que les plus grandes fréquences sont obtenues pour les plus petites valeurs. Cependant il est intéressant de remarquer que même si on a des très faibles fréquences il existe des cas extrêmes de personnes qui ont attendu beaucoup de temps, comme le montrent les histogrammes pas tronqués, voir la Figure 7.

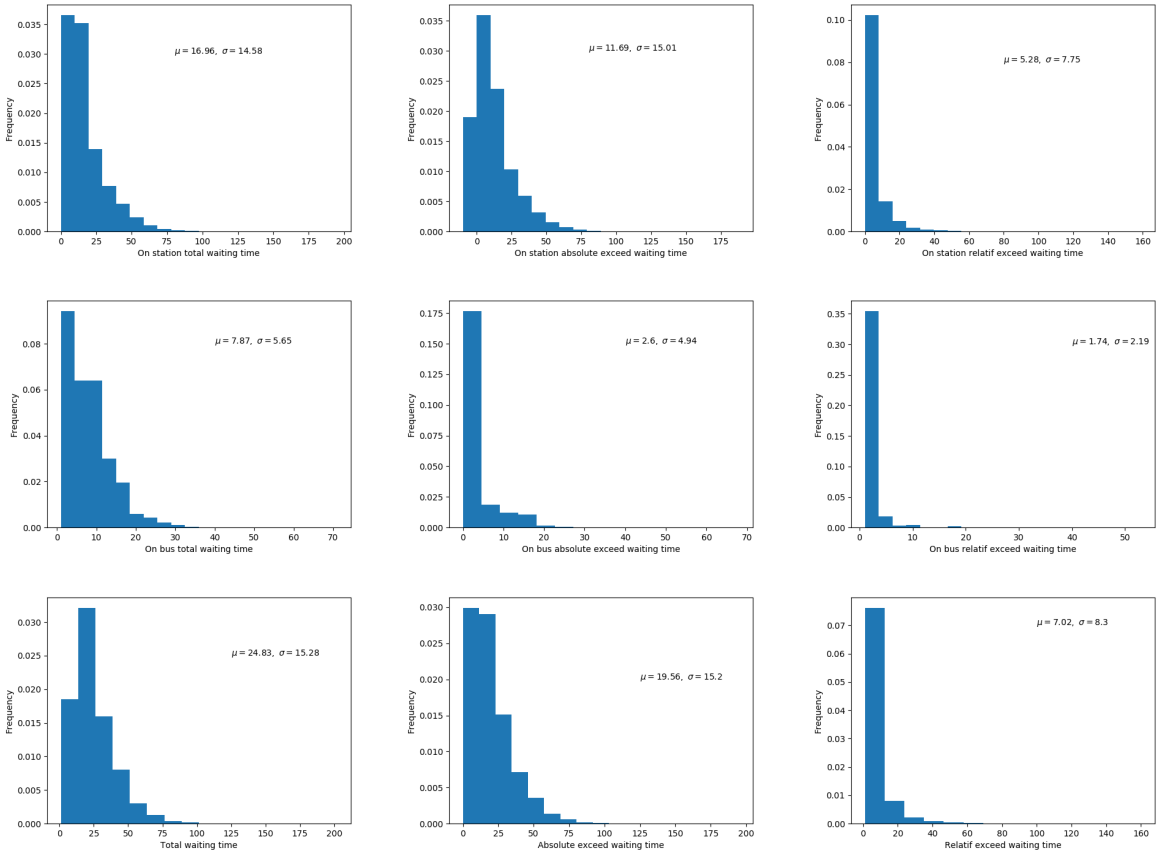

FIGURE 7 – Différents histogrammes des temps d'attente pour la stratégie de Rui. La première ligne est le temps d'attente dans la station, la deuxième dans le bus et la troisième le temps total. La première colonne désigne le nombre d'itérations totales attendues par personne, la deuxième l'excès absolu attendu (par rapport à la distance à parcourir) et la troisième l'excès relatif. [4]

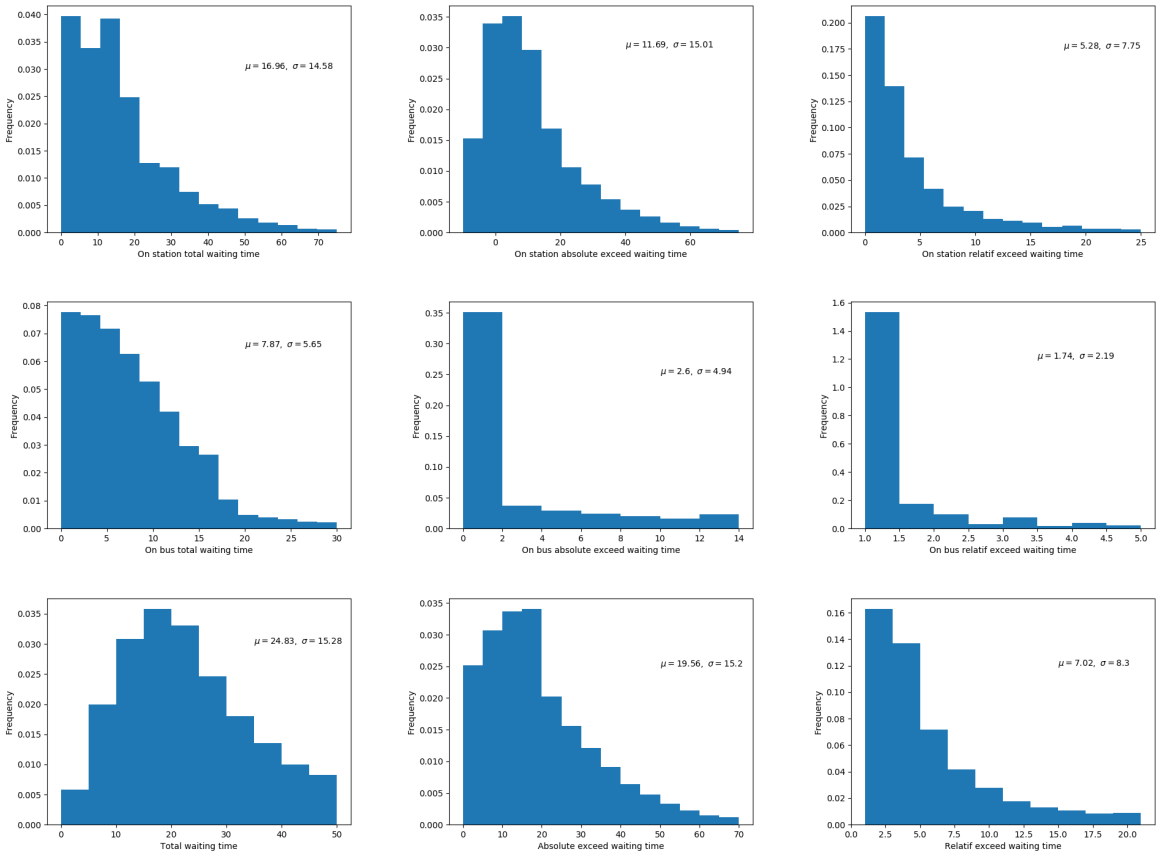

FIGURE 8 – Même histogrammes que dans la Figure 7 mais tronqués. [4]

3 Machine Learning

Ayant établi les statistiques sur la stratégie de Rui, on sait ce qu'on peut espérer d'une très bonne stratégie. Dans le but d'améliorer cette stratégie on s'est intéressé à une classe d'algorithmes de Machine Learning [5, 6], basé sur des réseaux neuronales [7] (on utilisera l'abréviation r.n. dans la suite). Le neurone formel est conçu comme un automate doté d'une fonction de transfert. Un réseau neuronale, comme son nom l'indique, s'inspire du fonctionnement des neurones biologiques. Les neurones du système ont la capacité d'apprendre grâce à l'expérience, elles apprennent par induction. Par confrontation à des situations ponctuelles, elles reçoivent des entrées (input). Grâce à plusieurs couches d'un certain nombre de neurones formelles, elles traitent l'information en essayant de faire des prédictions (ouput) pour résoudre un problème concret. Il y a trois types de Machine Learning (pouvant utiliser ou non les réseaux neuronales) :

1. Supervised learning (Apprentissage supervisé) :

   Ici on connaît bien ce qu'on veux avoir comme ouput, par exemple, essayer de copier une autre stratégie, comme on va le voir dans les parties suivantes. Dans ce cas l'information est étiquetée, pour n'importe qu'elle entrée il y a un unique ouput. Ainsi le r.n. va apprendre à faire les bonnes décisions pour arriver à avoir l'ouput qu'on veut.

2. Unsupervised learning (Apprentissage non supervisé) :

   Ici on ne connaît rien sur le type d'output qu'on veut, le r.n. va analyser les données pour essayer de trouver de relations entre elles. Ce n'est pas outil dans notre modèle car on connaît les lois qui le régissent. Cependant dans un cas plus réaliste où, par exemple, l'apparition des personnes dans les stations n'est pas uniforme, ce type d'apprentissage permettrait de connaître cette répartition.

3. Reinforcement learning (Apprentissage renforcé) :

   Ceci se ressemble au Supervised learning mais on ne sait pas exactement le type d'ouput qu'on veut, on ne sait pas si on peut réduire à moins de 0.7 pps après 1000 itérations par exemple. On va alors confronter nos données à une définition de récompense, dans notre cas ça pourrait être desservir une personne (le fait qu'elle monte dans le bus et puis arrive à sa destination). Ainsi l'algorithme va essayer de maximiser cette notion de récompense. Un des exemples les plus célèbres est le alphaGo [8], un programme crée par Google qui grâce au Reinforcement Learning a appris à jouer du Go au point de battre les meilleurs joueurs du monde.

On veut faire du Reinforcement learning dans le but de trouver la meilleure stratégie possible. Cependant, on doit avant comprendre la complexité nécessaire du réseau neuronale. Pour cela on va faire du Supervised learning pour essayer d'apprendre automatiquement la stratégie de Rui. Concrètement on s'est placé dans un cas plus simple ($C = 3$ et $N = 6$) et les modélisations sont faites pour $I = 1000$. On a utilisé la librairie keras [9] de python pour modéliser le r.n., dans la suite on va mentionner plusieurs termes qu'utilise cette librairie pour mésurer la qualité d'apprentissage du r.n. :

- Epoch : Quand on parle de l'époch (époque en angalis) n d'apprentissage cela veux dire que c'est la énième fois que le r.n. s'entraîne (chaque époque ayant plusieurs échantillons). À chaque époque on a un chiffre de accuracy et de loss (et sin on le veut de validation accuracy et validation loss également) qui est donné.

- Sample : Un sample est un échantillon, c'est-à-dire une décision prise par la stratégie de Rui.

- Accuracy and Loss : Ces deux termes mesurent comment le réseau se comporte face aux données qu'il reçoit, en général l'accuracy augmente, se rapprochant de 1 et loss diminue se rapprochant de 0. Plus précisément pour chaque époque, on confronte un ensemble $\overline{Y}$ de prédiction des input et un ensemble Y des ouput souhaités. $\overline{Y}$ est un ensemble de listes où chaque liste de prédiction décrit la probabilité que le r.n. choisisse une action. De même pour Y, sauf que dans ce cas tous les listes sont conformés de cases avec des 0 sauf dans une avec un 1 (car la stratégie de Rui est déterministe). Pour l'accuracy, on considère la moyenne de la "cathegorical prediction accuracy", c'est-à-dire le pourcentage de prédictions correctes pour cette époque. Pour la Loss, on considère l'entropie croisée entre $\overline{Y}$ et Y : $H(X,Y) := -\sum_k Y(k)\log(\overline{Y}(k))$, où k est une action concrète est $X(k)$ est la probabilité que le r.n. la choisisse (jamais 0, ni jamais 1), de même $Y(k)$ est la probabilité la stratégie de Rui la fasse (0 ou 1).

- Validation accuracy et Validation loss : En plus de juste s'entraîner à chaque époque sur tous les échantillons on à l'option de ne pas s'entraîner sur tous et laisser quelques uns pour valider. En gros, ces deux valeurs mesurent comment se comporterait le modèle dans une situation semblable (avec des donnés semblables). Cela nous permet de faire un suivi plus approfondi de l'apprentissage du r.n.

3.1 Imitation de la stratégie de Rui

Le programme consiste à mettre en fonctionnement la stratégie de Rui et d'interpoler les données. C'est-à-dire, à chaque fois que Rui prend une décision, aller à droite ou à gauche et qui faire monter dans le bus, on donne au r.n. des données. Celles-ci étant l'état du monde actuel (qui se trouve dans chaque station, dans le bus, où ils veulent aller et également si dans la dernière itération le bus est aller à gauche ou à droite, vue que le programme de Rui utilise ces données) et la décision en question, le but étant que notre r.n. reproduise la stratégie de Rui dans cette situation.

3.1.1 Apprentissage offline

Vu que le r.n. apprend en fonction de comment la stratégie de Rui agit, pour qu'il fonctionne d'une manière optimale il doit avoir observé Rui agir dans la plupart des situations et si possible plusieurs fois. En effet, dès qu'il rencontre un scénario que la stratégie de Rui n'avait pas rencontré il ne sait pas comment agir, cela nécessite beaucoup de temps d'apprentissage. Vu qu'au début on ne savait pas si notre modèle allait fonctionner on a commencé par ce qu'on va appeler l'apprentissage offline [10]. En effet dans ce cas on montre au robot comment agit la stratégie de Rui mais toujours dans le même scénario,

c'est-à-dire avec les personne qui apparaissent toujours au même moment, au même endroit et avec la même destination. Il va clairement pas agir bien dans un cas général, mais dans ce cas précis il va fonctionner assez bien et va l'apprendre à le faire assez vite, comme on peut le voir dans la Figure 9. Dans ce cas il est inutile d'introduire des échantillons pour valider vu qu'on ne veut pas qu'il apprenne à se comporter dans une situation semblable mais exactement dans celle qui lui est donnée. Plus précisément pour faire cela on a utilisé 4 couches de neurones formelles, 2 de 64 neurones et 2 de 128 neurones.

3.1.2 Apprentissage online

Contrairement à ce qu'on a appelé l'apprentissage offline, quand les données changent avec chaque entraînement c'est de l'apprentissage online [11]. Ici par contre, 10% des échantillons sont utilisés pour valider le r.n. De plus on a changé un peu le réseau neuronale, les 2 couches de 64 neurones sont maintenant de 128 neurones. On voit bien la progression du robot à travers les Epochs, en regardant la décroissance de $\mathbb{E}$[pps] dans la Figure 10. Cette courbe est produite en faisant une modélisation du système au milieu de l'apprentissage, ce n'est pas la vrai espérance du robot mais l'espérance observée dans ce parcours. On peut bien remarquer qu'elle n'est pas tout à fait décroissante mais on a parfois des fortes augmentations. En effet, si le robot rencontre une situation inconnue il peut se 'tromper' (pas agir comme la stratégie de Rui), ce qui dans la plus grande partie des cas va faire augmenter le pps et donc finir dans une situation encore pire (Rui n'a jamais des grands pps, ainsi le r.n. va bugger et faire n'importe quoi). Ce n'est donc pas une stratégie à coup sûr mais, plus il s'entraîne, moins grande sera la probabilité qu'un tel cas se produise. Pour mieux observer ceci on peut voir la Figure 11, dans cet histogramme on s'aperçoit qu'il y a des valeurs extrêmes, des cas où le robot a eu une erreur de comportement. Mais, en soit la moyenne de pps est faible, bien plus faible que pour Clock. En contrepartie, le temps que le robot a besoin pour prendre une décision est bien plus supérieure que les autres stratégies. Déjà Rui avait une mauvaise complexité par rapport aux autres mais, pour une modélisation dans ce cas assez simple ($I = 1000$, $C = 3$ et $N = 6$) Rui le fais quasi instantanément tandis que le r.n. prend plusieurs secondes.

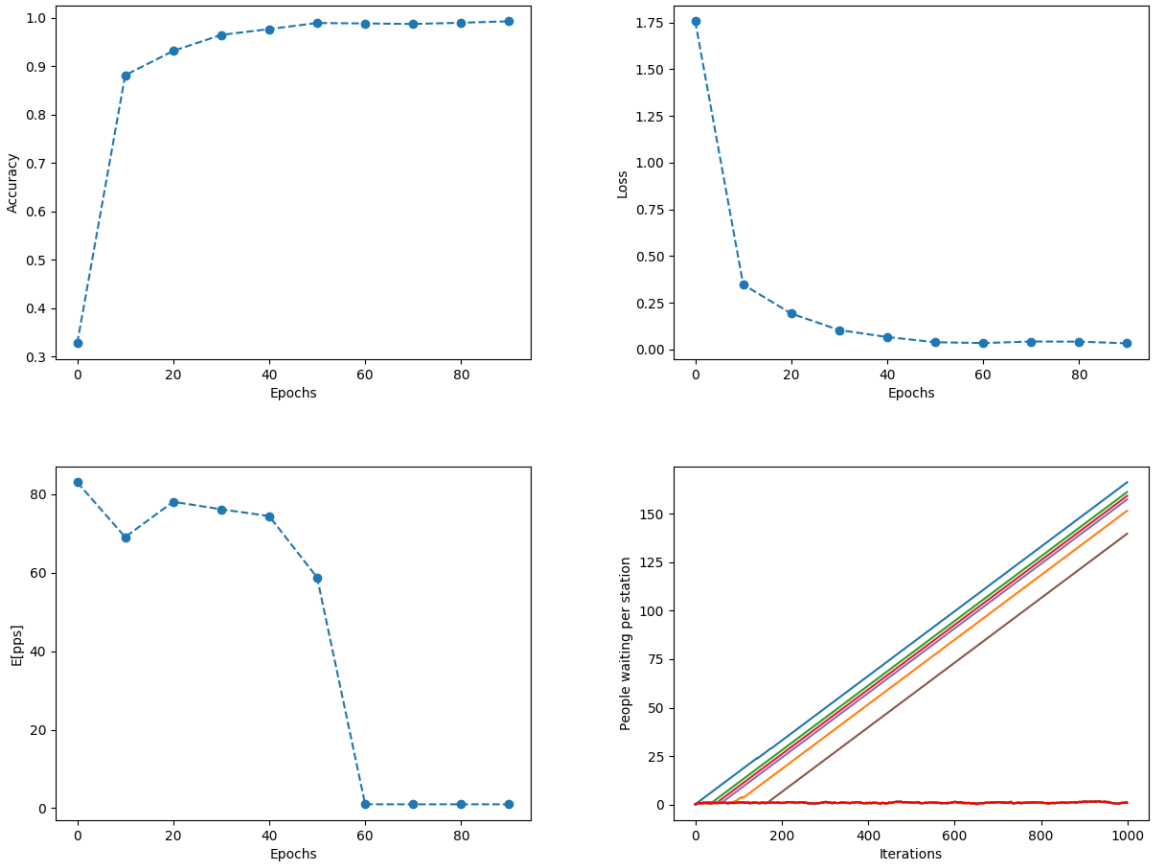

FIGURE 9 – Trois courbes montrant l'évolution de l'accuracy, loss, $\mathbb{E}$[pps] au cours des Epochs d'apprentissage offline, une mesure est faite toute les 10 Epochs. La dernière courbe montre comment a convergé la d'évolution du pps au cours du temps au cours des époques. La courbe ayant les plus petits pps est produite par le r.n. ayant appris a imiter parfaitement la stratégie de Rui (la stratégie de Rui aurait produite la même courbe dans ce cas). [10]

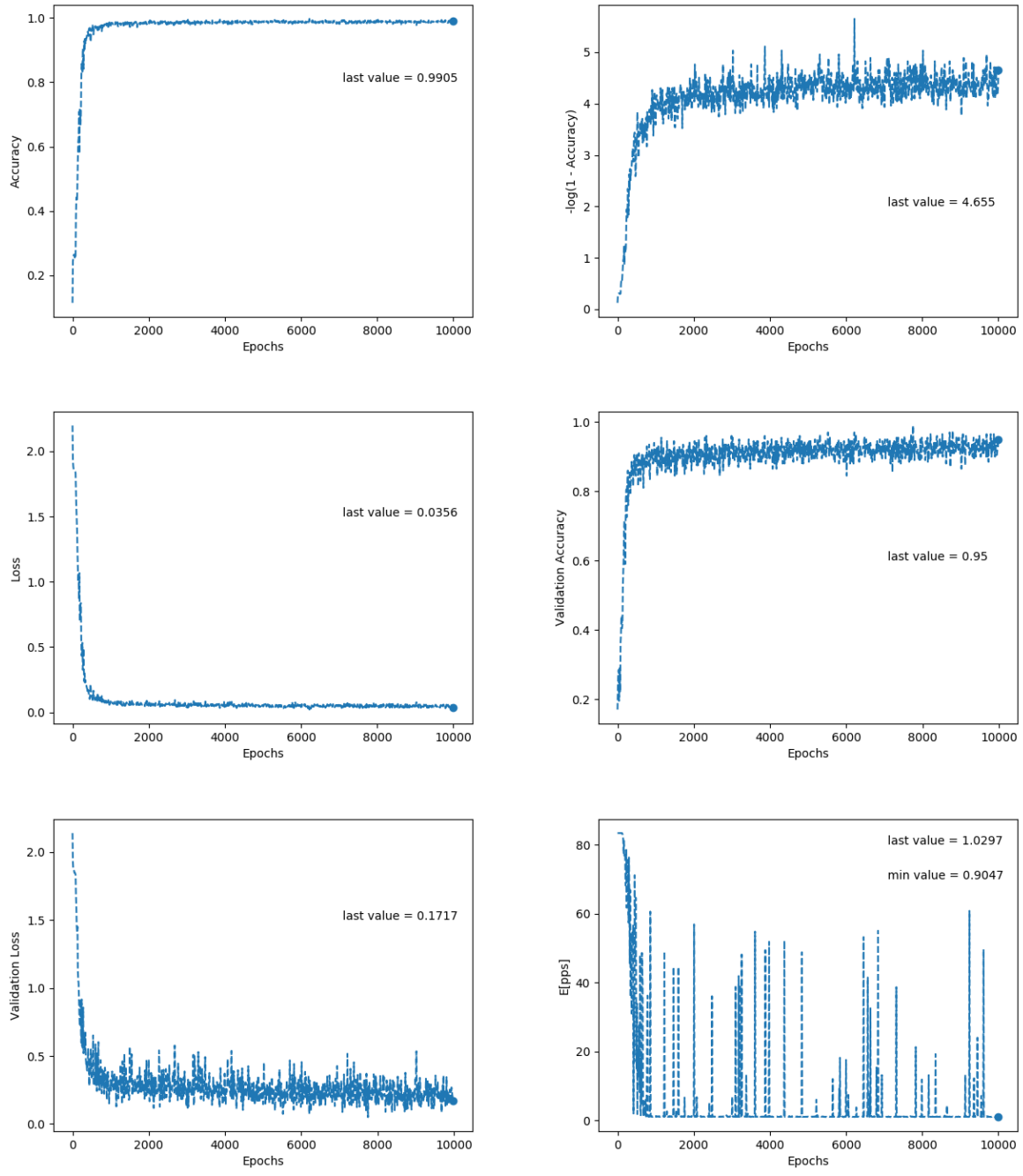

FIGURE 10 – Évolutions de l'accuracy, loss, validation accuracy, validation loss et $\mathbb{E}$[pps] au cours des Epochs d'apprentissage online, une mesure est faite toute les 10 Epochs. [11]

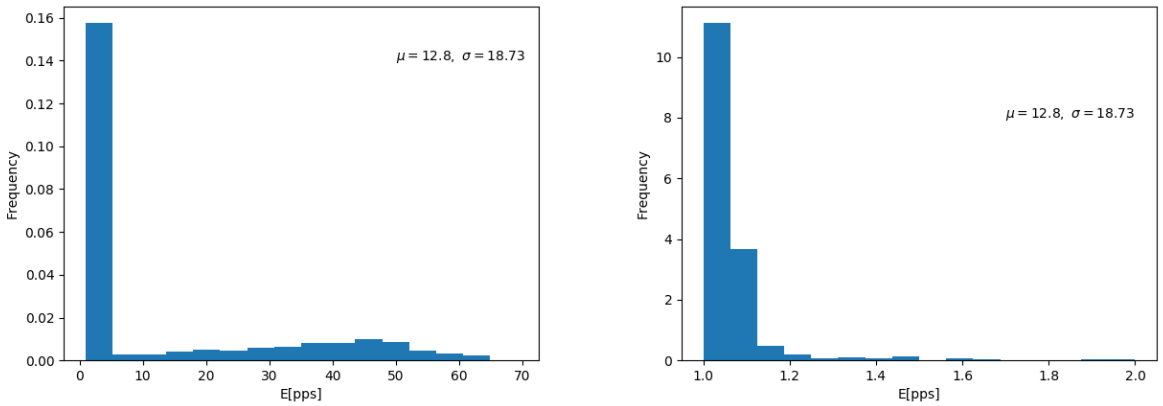

FIGURE 11 – Histogramme des moyennes du pps au cours de $I = 1000$ itérations, avec un zoom sur l'intervalle $[1, 2]$ à droite. [11]

Conclusion

À travers de cette étude on a pu remarquer que dans ce modèle simplifié des stratégies de navigation primitives ne fonctionnent pas, bien qu'avec quelques modifications , augmenter la capacité ou le nombre de bus, elles pourraient le faire. On a également étudié, comment grâce à une application qui permet de connaître la destination des personnes qui attendent dans les stations on peut produire une bien meilleure stratégie (la stratégie de Rui). De plus on a implémenté du Supervised learning pour qu'un r.n. apprenne la stratégie de Rui et pouvoir connaître la complexité nécessaire d'un tel apprentissage. Il reste cependant à :

1. Implémenter du Reinforcement learning afin de pouvoir trouver la "meilleure" stratégie.
2. Désigner un modèle plus complexe et plus proche de la réalité.
3. Établir d'analyses mathématiques plus approfondis afin de connaître les barrières minimum des pps qu'on peut avoir dans le cas général, comme on à fait pour la stratégie Clock avec une plus grande capacité.

Remerciements

Ce stage a eu lieu à l'Université Paris 7 Diderot (UP7D), plus précisément au Laboratoire Jacques-Louis Lions (LJLL), pour cela je suis très reconnaissant à tous les membres de l'équipe. En plus de cela je veux donner des remerciements spéciales à :

- Yves Achdou : Pour avoir accepter d'être mon tuteur formel.

- Nathalie Bergame : Pour avoir pris en charge tous les procédures administratives.

- Le groupe de thésards du LPMA : Pour avoir été aussi accueillants, surtout mes compagnons de bureau qui m'ont aidé avec toutes mes doutes et inquiétudes.

- Roman Andreev : Pour avoir été un tuteur très patient et toujours présent, m'aidant à progresser tantôt en matière d'écriture et de codage. Sans lui la production de ce document n'aurait pas été possible.

Références

[1] Page de la compétition du busybus. [1]
`https://verybusybus.wordpress.com`
Auteur : Roman Andreev

[2] Principaux programmes python, contenant les stratégies de Greedy, Clock et la simulation du busybus. Son exécution permet d'obtenir les Figures 2, 3 et 4.
`https://github.com/alejox10/Verybusybus/tree/master/primitives%20strategies`
Auteur : Roman Andreev et Alejandro Caicedo

[3] Programme python pour simuler plusieurs bus. Son exécution permet d'obtenir la Figure 5.
`https://github.com/alejox10/Verybusybus/tree/master/k%20buses`
Auteur : Alejandro Caicedo

[4] Programme python de la stratégie de Rui. Il permet d'obtenir les Figures 6, 7 et 8.
`https://github.com/alejox10/Verybusybus/tree/master/AI_RV`
Auteur : Rui Viana

[5] Les bases et différents types de machine learning.
`https://medium.com/@machadogj/ml-basics-supervised-unsupervised-and-reinforcement-le`
Auteur : Gustavo Machado

[6] Introduction au deep learning avec un exemple.
`http://karpathy.github.io/2016/05/31/rl`
Auteur : Andrej Karpathy

[7] Texte introductive aux réseaux de neurones.
`https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-rn.pdf`
Auteur : Université de Toulouse

[8] Page officielle du alphaGo.
`https://deepmind.com/research/alphago/`
Auteur : Google Deepmind

[9] Documentation officielle de la libraire keras.
`https://keras.io/`
Auteur : François Chollet

[10] Programmes python, un auxiliaire et un autre principale, pour modéliser la stratégie d'apprentissage offline. Son exécution permet d'obtenir la Figure 9.
`https://github.com/alejox10/Verybusybus/tree/master/learning1/offline`
Auteur : Roman Andreev et Alejandro Caicedo

[11] Programmes pythons, un auxiliaire et un autre principale, pour modéliser la stratégie de apprentissage online. Son exécution permet d'obtenir les Figures 10 et 11.
`https://github.com/alejox10/Verybusybus/tree/master/learning1/online`
Auteur : Roman Andreev et Alejandro Caicedo

1. Pour toutes les pages web la date d'accès est la même: 30/07/2017.