

Projet Cassiopée n°52:

Synchronisation des infrastructures routières et diminution des émissions polluantes

Milan Foglia

Tuteur: Nel Samama



Table des matières:

PARTIE I: Analyse de la problématique

A) Problématiques liées au trafic routier.	3
B) L'importance des feux	3
1) Cycle optimal.	3
2) Optimisation des phases.	4
3) Synchronisation des feux et Onde Verte.	4

PARTIE II: Etudes expérimentales

- Intérêt de simuler.	5
1) Présentation de SUMO.	5
- Les étapes de modélisation.	5
- Python et traitement des données	10
2) Simulations effectuées et résultats des comparatifs.	
- 1. Etude 1: Onde Verte	15
- 2. Etude 2: Phase Verte Maximale	17
- 3. Un cas idéal et réaliste?	20

PARTIE III: Conclusions sur les études, généralisations et ouvertures

- La consommation personnelle	24
- La consommation à l'échelle d'une société.	25
Conclusion	26
Remerciements	26
Annexe	27

PARTIE I: Analyse de la problématique

A) Problématiques liées au trafic routier.

Il est évident que le trafic routier est au centre de nombreux enjeux, écologiques comme politiques, sanitaires comme économiques. A travers sa dépendance aux énergies fossiles et à la pollution atmosphérique, il se place au coeur de la problématique écologique, les tensions sociales à l'instar des gilets jaunes qui se sont déclenchés lors de l'augmentation du prix du carburant est l'exemple parfait de l'aspect central de cette question chez le citoyen français puisque à défaut de se voir présenter un réseau de transports en commun parfait, toute son activité est dépendante du carburant. Toute tentative d'optimiser cette consommation doit donc être considérée et l'objet de cette étude et de caractériser le manque d'optimisation du réseau routier actuel, notamment à travers l'analyse des feux de route.

B) L'importance des feux:

1) cycle optimal

La régulation du trafic a fait apparaître l'importance d'étudier les cycles de phases des feux de signalisation (chaque cycle étant composé de plusieurs phases: typiquement R,J,V)

Les cycles mis en oeuvre actuellement sont compris entre 45 et 90s par cycle et découlent d'études datant des années 50/60. On peut donc se demander s'ils sont obsolètes face à l'évolution du trafic.

En effet, la notion même de cycle perd du terrain face à de nouvelles solutions dynamiques:

[Traffic Signal Timing Manual: Table of Contents - Office of Operations](#)

2) Optimisation des phases

La phase verte est fixe dans de nombreux cas mais des études montrent qu'une phase maximale doit être envisagée sur une route au trafic potentiellement important, il n'est pourtant pas évident de montrer quel temps maximal doit être envisagé.

30-50s pour Kell et Fulkerton

1,3 fois la longueur moyenne de la file concernée pour Orcutt.

(Orcutt F. L. "The Traffic Signal Book. Englewood Cliffs, NJ : PrenticeHall". Dans : (1993).)

Une phase Vert max élevée n'a pas d'impact dans un système dynamique d'après Courage

(Courage K. G., Luh J. Z. et Wallace C. E. "Development of guidelines for implementing computerized timing designs at traffic actuated signals, Two Volumes on Arterial System Implementation, Transp. Res. Center, Gainesville". Dans : (1989).)

C'est justement le but de la simulation du Boulevard des coquibus, étudier l'impact de la phase de feu vert maximal sur la consommation des utilisateurs de la voie principale.

3) Coordination des feux et onde verte

Cette méthode consiste à synchroniser les feux de façon à faire avancer un flot de voiture fluide de manière uniforme sur un réseau, d'où la notion d'onde verte qui se propage. On utilise donc un offset différent pour chaque feu de l'artère principale. La première simulation considérée en partie II tente de mettre en oeuvre l'idée d'onde verte avec un calcul préalable des vitesses et temps considérés dans la simulation afin de contenir le train d'onde dans les offsets. On détaillera les calculs réalisés dans la partie concernée.

PARTIE II: Etudes expérimentales

Intérêt de simuler:

Les données prises en compte dans une expérimentation réelle sont trop volumineuses et demandent de nombreux capteurs pour obtenir des résultats aussi larges qu'une simulation SUMO.

La répétabilité des simulations en font l'outil optimal pour analyser le trafic, tout en gardant les approximations des résultats obtenus.

1)Présentation de Sumo et des configurations utilisées.

généralités

c'est un programme de simulation open source qui permet de modéliser le trafic à toutes les échelles (micro comme macro).

Il permet à la fois de créer des réseaux (net) et des simulations en associant un réseau à un fichier .rout (qui représente le trafic: il contient les différents véhicules et leur route programmées).

Toutes les explications suivantes permettent de dégrossir la documentation Sumo mais il faut tout de même l'étudier avec attention car de nombreux éléments nécessitent de creuser pour les trouver.

[Documentation - SUMO Documentation](#)

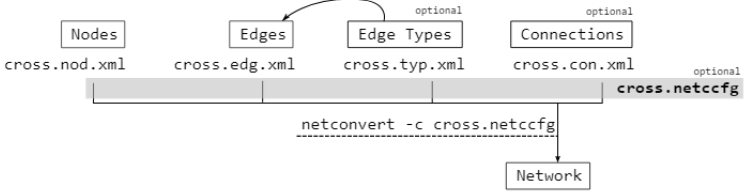
Réseaux (.net):

Plutôt que d'étudier le réseau créé pour le boulevard, considérons l'exemple fourni par Sumo, plus complet, comme il est fait dans ce tutoriel:

[A Programmer's Note on TraCI_tls, TraCI, and SUMO | Traffic Control Test Bed](#)

[Documentation](#)

Ces réseaux sont composés de plusieurs briques élémentaires:

<pre> +-- data +-- cross.nod.xml: +-- +-- cross.edg.xml +-- [cross.typ.xml] +-- cross.con.xml +-- +-- cross.netccfg +-- cross.net.xml +-- +-- cross.det.xml </pre>	<p>nodes: points de base du réseau, ils ont chacun un id, des coordonnées (x,y) et un type.</p> <p>edges: permettent de qualifier les nodes et notamment de leur associer une priorité (utilisée pour les priorités de croisements)</p> <p>Le .typ permet de standardiser un type d'edge afin de simplifier le code (voir exemples du github)</p> <p>.con: connections entre edges (voir graph git)</p> <p>équivalent à la commande: <code>netconvert -n cross.nod.xml -e cross.edg.xml -x cross.con.xml -o cross.net.xml</code></p> <p>output de la commande précédente: il s'agit du réseau, il reste simplement à lui associer une route.</p> <p>fichier optionnel: permet de créer un détecteur.</p>  <pre> graph TD Nodes[Nodes cross.nod.xml] --> netconvert Edges[Edges cross.edg.xml] --> netconvert ET[Edge Types cross.typ.xml optional] --> netconvert Conns[Connections cross.con.xml optional] --> netconvert netconvert --> Network[Network cross.net.xml] netccfg[cross.netccfg optional] -- "netconvert -c cross.netccfg" --> netconvert </pre>
<p>Simulation(.net + .rou)</p> <pre> +-- cross.rou.xml +-- cross.sumocfg </pre>	<p>On ajoute ici un fichier .rou qui contient les types de véhicules, leurs routes associées et leur entrée dans la simulation.</p>

	Elément final de la configuration, permet de lancer l'exécutable Sumo ou Sumo-gui (avec interface utilisateur) en faisant le lien entre le réseau (.net) et le trafic (.rou)
--	--

Remarque: output de la simulation

"sumo-gui", "-c", "coquibus_conf.sumocfg", "--**emission-output**", "coquibusemissions.xml"

dans cette commande, on récupère un fichier xml contenant les émissions de toutes les voitures de notre simulation, c'est donc ces éléments que j'ai utilisés pour analyser la consommation et les émissions. La démarche sera détaillée plus tard.

Comment créer un réseau?

1. From scratch:

- La méthode permettant de comprendre les éléments précédents consiste à les créer à la main et à utiliser **netgenerate** afin de créer le réseau.
- plus simple et efficace: utiliser l'outil interfacé **netedit** de Sumo qui permet de créer des réseaux complexes rapidement grâce à un UI de qualité, le **.net** généré sera le même qu'un réseau créé à la main, voire mieux optimisé. On peut dire que netgenerate est obsolète à ce niveau.

2. Par import OSM

Utilisation de **netconvert** --osm-files berlin.osm.xml -o berlin.net.xml

avec les paramètres: --osm-files et le fichier **.osm** téléchargé sur OpenStreetMap:

<https://www.openstreetmap.org/#map=6/46.449/2.210>

-o pour output permet d'écrire le fichier généré.

De nombreux paramètres doivent être pris en compte pour réaliser un import de qualité:

[Networks/Import/OpenStreetMap - SUMO Documentation](#)

On connecte ensuite le réseau .net créé à un fichier **.rou** afin d'obtenir un fichier **.sumocfg** permettant de lancer la simulation comme présenté dans le tableau.

Je conseille cependant d'utiliser un script python dans le but d'automatiser les routes et les lancements de simulations. Ce qui explique l'intérêt de présenter Traci.

Traci:

Traci permet de relier une simulation Sumo à un programme python permettant de communiquer avec la simulation en cours (à travers les feux de signalisation notamment: il permet de modifier les phases à chaque pas de la simulation).

Le guide présenté précédemment permet de bien comprendre la mise en place de Traci: [A](#)

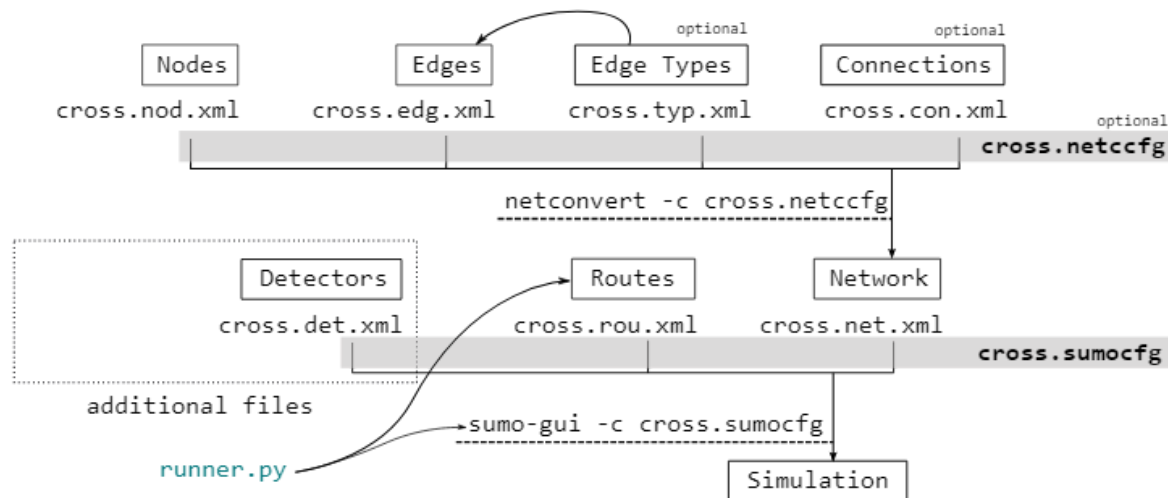
[Programmer's Note on TraCI_tls, TraCI, and SUMO | Traffic Control Test Bed Documentation](#)

Le script python runner.py permet de créer une route et de l'associer à un réseau préalablement créé.

generateroute(): permet de créer automatiquement un fichier route comme on l'aurait conçu à la main dans un fichier .rou.

run(): Permet d'interagir avec la simulation: changer les états des feux etc

Bilan: Cette image récapitulative permet d'avoir une vue d'ensemble d'une simulation lancée avec traci:



L'output:

Le supplément de commande "**--emission-output**", "coquibusemissions.xml" permet de créer un fichier xml contenant les éléments suivant:

Sur toute la simulation: chaque timestep est renvoyé:

```

<timestep time="1.00">
  <vehicle id="0" eclass="HBEFA3/PC_G_EU4" CO2="2624.72"
  CO="164.78" HC="0.81" NOx="1.20" PMx="0.07" fuel="1.13"
  electricity="0.00" noise="55.94" route="route0" type="Car"
  waiting="0.00" lane="1to2_0" pos="2.10" speed="0.00" angle="90.00"
  x="2.10" y="-4.80"/>
</timestep>

```

Il contient chaque véhicule dans la simulation avec ses paramètres à l'instant considéré, notamment "fuel" et "CO2" qui nous intéressent.

```

<timestep time="17.00">
  <vehicle id="car_0" eclass="HBEFA3/PC_G_EU4" CO2="2414.97"

```

```

CO="0.00" HC="0.01" NOx="0.72" PMx="0.01" fuel="1.04"
electricity="0.00" noise="65.14" route="route0" type="Car"
waiting="0.00" lane="3to4_0" pos="53.45" speed="14.71" angle="90.00"
x="213.45" y="-4.80"/>
    <vehicle id="car_1" eclass="HBEFA3/PC_G_EU4" CO2="2266.39"
CO="0.00" HC="0.04" NOx="0.69" PMx="0.01" fuel="0.97"
electricity="0.00" noise="63.96" route="route0" type="Car"
waiting="0.00" lane="1to2_0" pos="114.27" speed="13.02" angle="90.00"
x="114.27" y="-4.80"/>
    <vehicle id="car_2" eclass="HBEFA3/PC_G_EU4" CO2="0.00"
CO="0.00" HC="0.00" NOx="0.00" PMx="0.00" fuel="0.00"
electricity="0.00" noise="64.19" route="route0" type="Car"
waiting="0.00" lane="1to2_0" pos="92.68" speed="13.73" angle="90.00"
x="92.68" y="-4.80"/>
    <vehicle id="car_3" eclass="HBEFA3/PC_G_EU4" CO2="6472.19"
CO="72.72" HC="0.49" NOx="2.63" PMx="0.12" fuel="2.78"
electricity="0.00" noise="68.15" route="route0" type="Car"
waiting="0.00" lane="1to2_0" pos="52.13" speed="15.03" angle="90.00"
x="52.13" y="-4.80"/>
    <vehicle id="car_4" eclass="HBEFA3/PC_G_EU4" CO2="11707.31"
CO="212.24" HC="1.25" NOx="5.21" PMx="0.26" fuel="5.03"
electricity="0.00" noise="75.72" route="route0" type="Car"
waiting="0.00" lane="1to2_0" pos="23.10" speed="10.50" angle="90.00"
x="23.10" y="-4.80"/>
</timestep>

```

On remarque ici l'ajout de chaque véhicule de la simulation, chacun avec un id différent.

Fonctions Python de traitement de données:

Ces fichiers XML sont évidemment volumineux pour des grandes simulations, j'ai donc codé des fonctions python permettant de les traiter:

importxmltocsvAndsum() permet d'écrire le fichier xml dans un .csv afin de le traiter en excel et renvoie la somme de la consommation ("fuel" ou "CO2") de tous les véhicules dans la simulation.

importxmlAndsum() est utilisé pour les fichiers trop volumineux pour générer un csv, il fait uniquement la somme des consommations.

```
import xml.etree.ElementTree as ET

import csv
def importxmltocsvAndsum():

    consototale=0

    with open('coquibusemissions.csv', 'w',newline='') as g:

        writer=csv.writer(g)
        # Ouverture du fichier en mode lecture

        f = open("coquibusemissions.xml","r");

        # On lit le contenu du fichier

        contents = f.read();

        # On parse le fichier

        xml = ET.fromstring(contents)

        # On itère sur chaque élément timestep

        for timestep in xml:

            # Pour chaque element vehicle on récupère l'attribut 'fuel'

            for vehicle in timestep:

                fuel=float(vehicle.attrib['fuel'])

                #test arrêt:

                if (fuel==0):

                    fuel=0.2778
```

```

consototale+=fuel

writer.writerow([vehicle.attrib['fuel']])

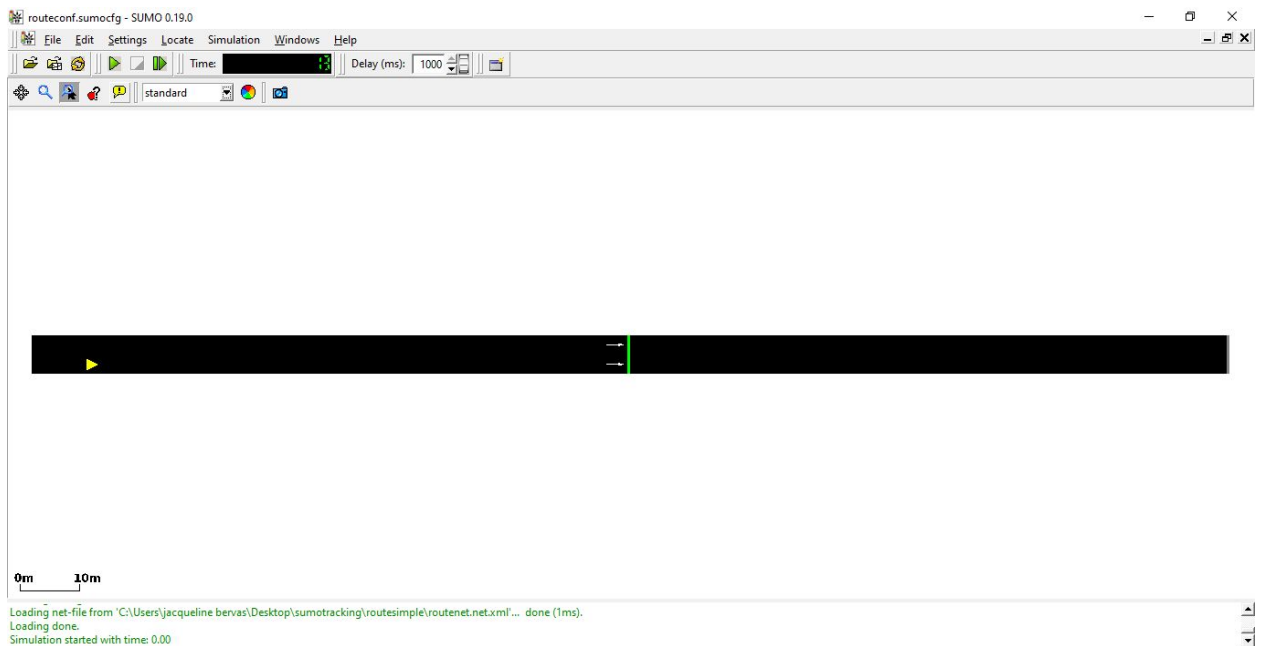
return consototale

```

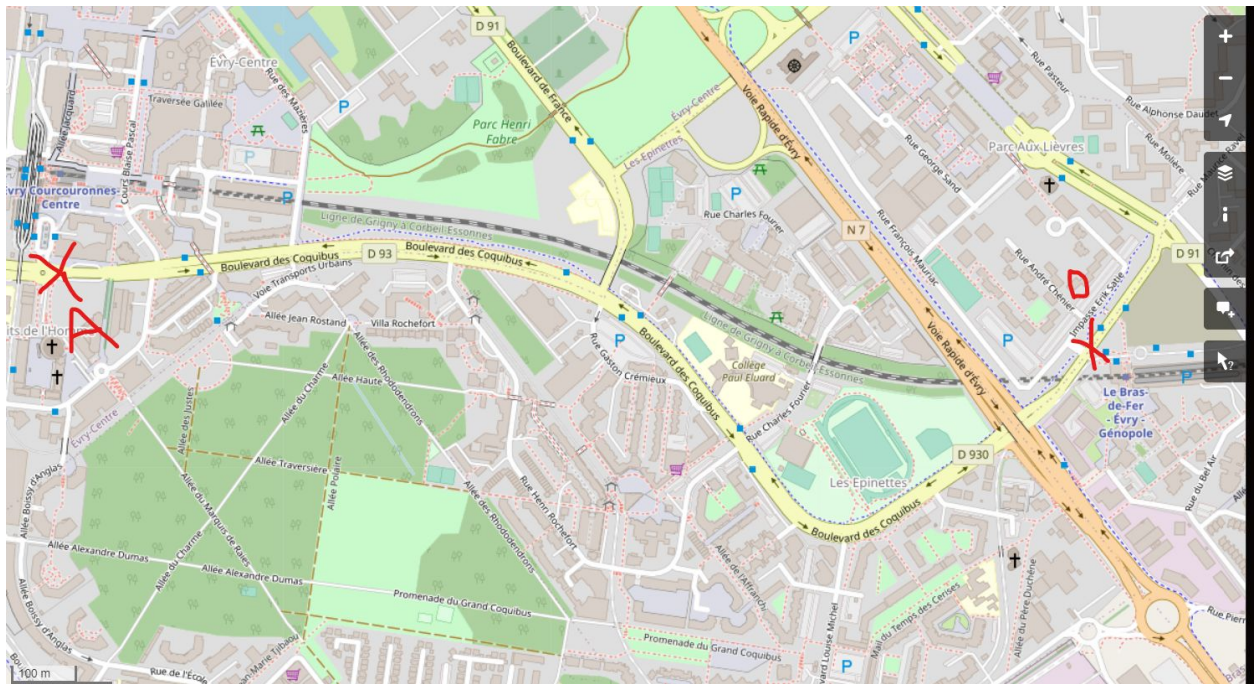
2) Simulations effectuées et résultats des comparatifs:

1. Premiers pas:

La première simulation effectuée est celle d'une route simple séparée par un feu en son milieu, son but était de s'habituer aux éléments de configurations de SUMO. Cette simulation a été réalisée avec une ancienne version de Sumo (0.19)



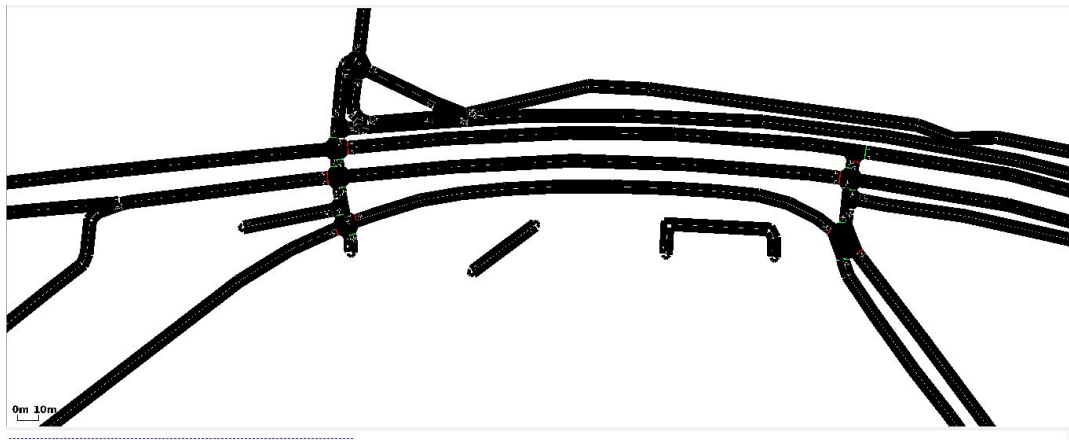
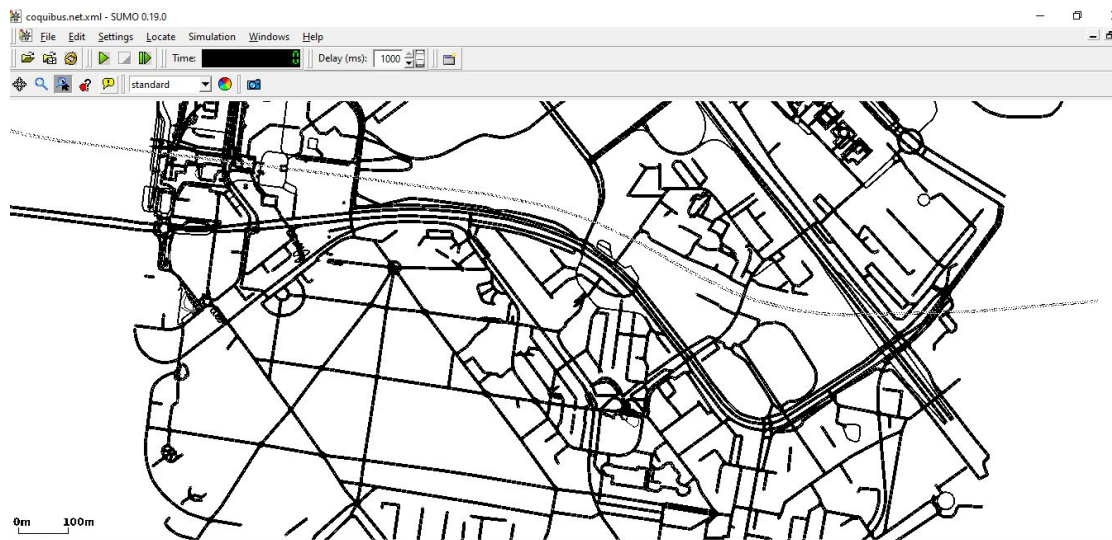
2. La deuxième modélisation est celle du **boulevard des coquibus**:



D'une longueur totale de 1690m, il contient 9 feux espacés comme suit:

120m/40m/130m/140m/200m/330m/240m/240m/200m/50m (mesures réalisées grâce à OSM et Google Maps)

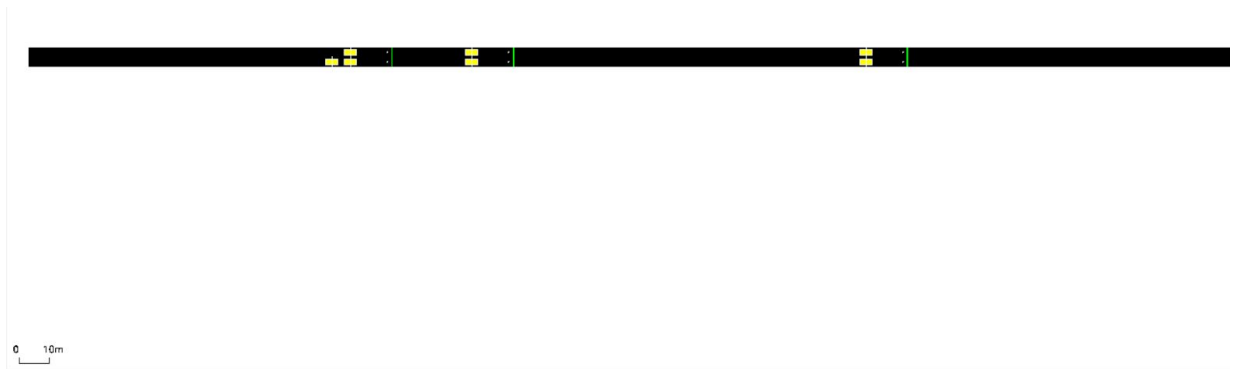
Pour la modélisation, plusieurs pistes furent considérées: (import OSM ou création directe): l'import OSM amenait trop de complexité au réseau



Cependant, cela permet d'observer le potentiel de Sumo en macro, une ville entière peut ainsi être modélisée.

Dans l'étude menée ici, le choix d'une création from scratch semblait plus cohérent avec les objectifs.

Ainsi, le boulevard est modélisé comme suit: rectiligne en respectant les distances mesurées sur OSM.



Représentation des 4 premiers tronçons du boulevard

A partir de cette modélisation, plusieurs simulations ont été menées:

1. Etude n°1: Onde verte.

L'**onde verte** consiste à gérer l'offset des feux afin d'assurer le déplacement uniforme d'un train d'onde de voiture avec des feux verts sur tout le boulevard.

Les temps optimaux d'offset ont été calculés grâce au modèle suivant:

A chaque départ, la voiture a une vitesse nulle et une accélération maximum ($a=2,6\text{m.s}^{-2}$) ce jusqu'à ce qu'elle atteigne la vitesse max autorisée: $v_{\text{max}}=13,88\text{ m.s}^{-1}$, en intégrant deux fois notre accélération, on obtient donc un tps de 0 à v_{max} de **$t0_v_{\text{max}}=5,34\text{s}$** et une distance parcourue de **$d0_v_{\text{max}}=37\text{m}$** . Il reste donc $t_n=(D_n-d0_v_{\text{max}})/V_{\text{max}}$ de temps de parcours. avec $D_n=x_{f_n+1}-x_{f_n}$ soit la distance entre le départ et le premier feu.

On a finalement **$\text{offset_1} = t0_v_{\text{max}} + t_n$** .

Ensuite on ajoute simplement $d_{\text{feux}}/v_{\text{max}}$ à chaque offset.

On considère une approximation à l'arrivée dans la mesure où on ne considère pas de freinage (ie le feu est vert pour le conducteur lorsqu'il approche du feu) ce qui est théoriquement vrai à quelques secondes prêt si l'on considère une onde verte.

D'où le tableau des offset optimisés suivant:

lanes:	tps avec acc max	tps + tps_offset précédent
120	11,31982709	11,31982709
40	2,88184438	14,20167147
130	9,365994236	23,56766571
140	10,08645533	33,65412104
200	14,4092219	48,06334294
330	23,77521614	71,83855908
240	17,29106628	89,12962536
240	17,29106628	106,4206916
200	14,4092219	120,8299135
50	3,602305476	124,432219
1690		



Les voitures se suivent bien avec les feux verts à chaque intersection.

On obtient alors avec ces résultats une consommation par voiture de **8L/100km** et une pollution atmosphérique de **313g de CO2** sur le trajet dans le cas idéal ou un seul groupe de voiture passe avec le bon timing sur l'onde verte.

Dans un cas aléatoire, modélisé par une moyenne sur 700 cas, on a une consommation de **13,12L/100km** et de **473,6g de CO2** par trajet.

Il semblait intéressant d'étudier un cas où tous les feux sont rouges, on voit évidemment que la consommation monte en flèche dans ce cas. On pourrait l'associer à des bouchons où le conducteur doit s'arrêter à chaque feu.

Les résultats sont dans le tableau ci-dessous.

Cas	Au pire	Aléatoire	Au mieux	Delta (aléatoire-mieux)
Fuel (L/100km)	18,34	13,12	7,99	5,13
Start/Stop (L/100km)	17,27	12,73	7,99	4,74
pollution CO2 (g)	678,52	473,6	313,98	159,62*

Aléatoire: 1 véhicule toutes les 15s pendant 5000s (avec vérification qu'il n'y avait pas de congestion du trafic)

*la pollution à l'arrêt n'est pas prise en compte.

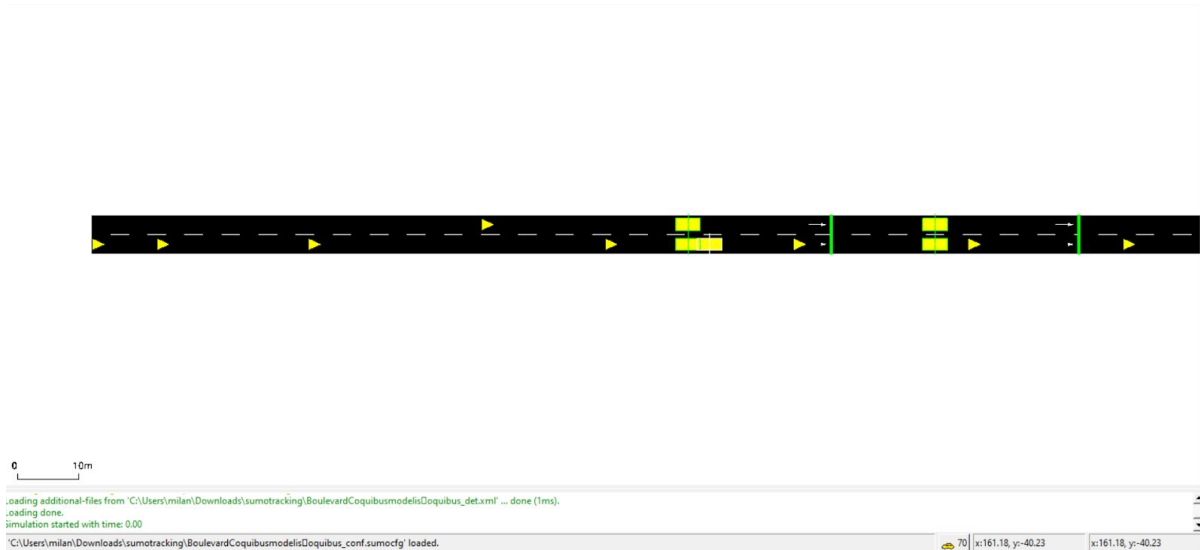
2. Etude n°2: Phase verte maximale.

Etude d'un paramètre "**phase verte maximale**" permettant d'augmenter la durée de la phase verte jusqu'à une certaine valeur à déterminer, cette phase verte est contrôlée par des capteurs placés en amont de chaque feu. L'augmentation de phase verte, comme l'onde verte permet de privilégier une voie principale, ce qui explique le choix de garder le boulevard sans croisement .

Ces capteurs sont implémentés dans la feature **Actuated traffic light**: implémentation interne de SUMO permettant de prolonger la phase actuelle.

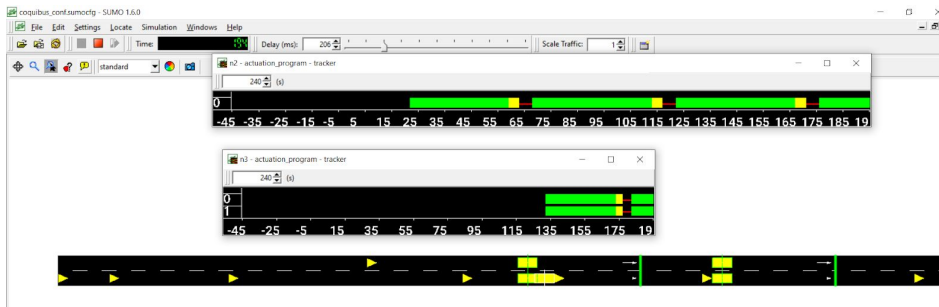
Ils sont appliqués sur les feux aux id: n1,n2,n3, ...,n9

code type:



Comparatifs:

1/P=0,2



Pour 706 voitures sur une heure (Pemission=0,2)

Non Opti:	Opti:
conso= 12.16 L/100km	conso= 9.39L/100km

Soit un delta=2.67L/100km

```

Entrée [31]: print("optimisé avec P=0,2 et Ph_v=90")
              #optimisé

              #total=importxmlAndsum()
              print ("total="+str(total))
              voiture=112021.04/706
              consol_100=voiture/16.9
              print ("conso= "+str(consol_100)+" L/100km")

              optimisé avec P=0,2 et Ph_v=90
              total=112021.04239998611
              conso= 9.38875907269893 L/100km

```

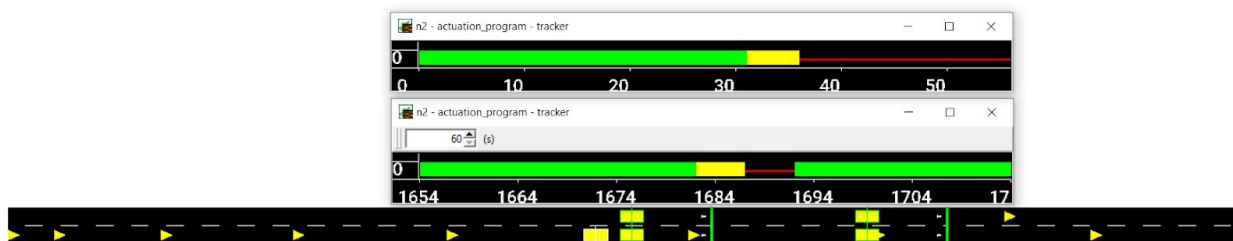
2/P=0,5 soit 1770 véhicules sur une heure

Les résultats sont relativement proches à 0,2 véhicules par seconde, qu'en est-il de 0,5 vh par seconde, cad un trafic plus important, plus proche des heures de pointe du boulevard.

Non opti:	Optimisé:
conso=14.29L/100km	Conso=8.65 L/100km

Soit un delta de :

delta= 5,49L/100km pour chaque conducteur.



Avec une proba d'émission de 5/10, on obtient une circulation fluide grâce à l'extension des phases de feu vert **jusqu'à 90s**.

Testons dorénavant l'influence du paramètre Phase_Verte_Max:
pour

Conso (L/100km)	Phase verte maximale
8.65	90s
8.66	60s
8.66	50s
8.90	40s
9.05	35s

Nous voyons ici que la conso ne varie que très peu (la différence reste écrasée puisqu'on ramène tout à une seule voiture mais la différence est plus faible qu'attendue)

Etudions donc l'influence d'une augmentation de la phase min du rouge. Passons là de $Ph_min_r=5s$ ce qui ne paraît pas très réaliste puisque les routes perpendiculaires se retrouveraient bloquées dans ce cas à $Ph_min_r=10s$.

Entrée [45]: `print("optimisé avec P=0,5 etPh_v=35 et Ph_r_min=10 aulieu de 5")`
`#optimisé`

```
total=importxmlAndsum()
print ("total="+str(total))
voiture=total/1770
print (voiture)
consol_100=voiture/16.9
print ("conso= "+str(consol_100)+" L/100km")
```

```
optimisé avec P=0,5 etPh_v=35 et Ph_r_min=10 aulieu de 5
total=318337.19760028325
179.85152406795663
conso= 10.642102015855423 L/100km
```

On obtient une conso de **10.64 L/100km** à $Ph_min_r=10$ et $Ph_max_v=35s$.

3. Un cas idéal et réaliste?

Imaginons alors le cas idéal et réel ou la phase verte peut augmenter jusqu'à 50s et la phase rouge reste supérieure à 10s toujours avec une proba de 0,5.



```

Entrée [49]: print("optimisé avec P=0,5 etPh_v=50 et Ph_r_min=10")
              #optimisé

              total=importxmlAndsum()
              print ("total="+str(total))
              voiture=total/1770
              print (voiture)
              consol_100=voiture/16.9
              print ("conso= "+str(consol_100)+" L/100km")

              optimisé avec P=0,5 etPh_v=50 et Ph_r_min=10
              total=469257.7220002146
              265.1173570622681
              conso= 15.687417577649002 L/100km

```

conso de 15.69 L/100km

Remarque: Nous avons ici un résultat aberrant, l'augmentation de la phase verte a créé des bouchons ce qui n'aurait pas dû avoir lieu, il s'agit sûrement d'un placement du capteur trop proche du feu, dès qu'il y a 6 voitures alignées, aucune voiture ne peut rentrer, cela peut potentiellement fausser les entrées de la boucle.

Tentative de correction en plaçant le détecteur 2x plus loin:

```

Entrée [53]: print("optimisé avec P=0,5 etPh_v=50 et Ph_r_min=10 et nouveau detector gap x2")
              #optimisé

              total=importxmlAndsum()
              print ("total="+str(total))
              voiture=total/1770
              print (voiture)
              consol_100=voiture/16.9
              print ("conso= "+str(consol_100)+" L/100km")

              optimisé avec P=0,5 etPh_v=50 et Ph_r_min=10 et nouveau detector gap x2
              total=318248.8260002179
              179.8015966102926
              conso= 10.639147728419681 L/100km

```

Avec une conso de **10.64L/100km**, le résultat est enfin satisfaisant, malgré certains moments de grand ralentissement, la globalité du trafic était fluide. Il n'est pas nécessaire d'aller plus loin à ce niveau puisque la simulation ne présente pas de trafic perpendiculaire qui sont des paramètres trop importants pour être négligés à ce niveau de précision.

Résultat final: **10.64 L/100km** avec 50s de phase verte maximale et 10s de phase rouge min sur le feu rouge, le tout avec des capteurs placés à $10s \times \text{vitessemax} = 130\text{m}$ des feux.

En associant ce résultat à celui de l'onde verte, les deux éléments ne sont pas parfaitement en accord, on obtient tout de même une légère amélioration:

```
#modèle accélération 2.6 m.s-2
print(" super opti croisé avec onde verte")

total=importxmlAndsum()
print ("total="+str(total))
voiture=total/1769
print (voiture)
consoL_100=voiture/16.9
print ("conso= "+str(consoL_100)+" L/100km")

super opti croisé avec onde verte
total=309139.9216002718
174.75405404198517
conso= 10.340476570531667 L/100km
```

soit **10,34L/100km**

Reprenons le cas des offset en considérant un démarrage à chaque feu (on calcule donc tous les offset comme celui du premier feu):

lanes:	tps avec acc max	tps + tps_offset précédent
120	11,31982709	11,31982709
40	5,556138329	16,87596542
130	12,04028818	28,9162536
140	12,76074928	41,67700288
200	17,08351585	58,76051873
330	26,44951009	85,21002882
240	19,96536023	105,175389
240	19,96536023	125,1407493
200	17,08351585	142,2242651
50	6,276599424	148,5008646
1690		

Nouveau tableau des offsets

on obtient le résultat suivant, **conso= 10.36L/100km**

On peut donc en déduire que les offset sont faussés par les changements de phase et que l'amélioration est trop faible pour être considérée. On peut cependant imaginer un système qui corrige les offset à chaque instant mais il s'agirait d'un système parfaitement dynamique donc le terme offset n'a plus vraiment de sens.

Nous conservons donc le résultat précédent sans y ajouter l'offset.

Comparaison finale:

En comparant ce résultat au cas des feux non optimisés, on obtient:

$$\text{DeltaFinal} = 14.29 - 10.64 = 3,54 \text{ L/100km}$$

C'est donc sur ce résultat qu'on fera les généralisations de la partie suivante.

Remarque: Le code d'un tel feu est le suivant:

```
<tlLogic id="n2" type="actuated" programID="actuation_program" offset="0">
  <param key="max-gap" value="3.0"/>
  <param key="detector-gap" value="10.0"/>
  <param key="show-detectors" value="true"/>
  <param key="file" value="NULL"/>
  <param key="freq" value="300"/>
  <phase duration="31" minDur="31" maxDur="50" state="GG"/>
  <phase duration="5" minDur="5" maxDur="5" state="yy"/>
  <param key="max-gap" value="2.0"/>
  <param key="detector-gap" value="10.0"/>
  <param key="show-detectors" value="true"/>
  <param key="file" value="NULL"/>
  <param key="freq" value="300"/>
  <phase duration="20" minDur="10" maxDur="20" state="rr"/>

</tlLogic>
```

avec:

GG=green

yy=yelow

rr=red

Pour avoir des infos sur les paramètres de ce type de feu:

https://sumo.dlr.de/docs/Simulation/Traffic_Lights.html#actuated_traffic_lights

PARTIE III: Conclusions sur les études, généralisations et ouverture à des solutions concrètes.

1. La consommation personnelle

Considérons le trajet du boulevard des coquibus 1,69 km comme le trajet quotidien d'une personne partant au travail, ce trajet est trop faible et devrait être fait à vélo ou en transport en commun évidemment mais imaginons que ce soit une partie du trajet de cet individu. C'est parfaitement réaliste lorsqu'on observe la quantité de voitures en heure de pointe sur le boulevard.

Alors on peut généraliser la consommation sur ce tronçon à une année de trajet du lieu de travail au domicile:

On a 251 jours ouvrés dans l'année soit $\Delta\text{ConsoAnnée} = 251 * 2 * 3.54 / 1.69 = 30.03L$

Cela peut sembler faible mais en extrapolant à des trajets réels, les résultats sont significatifs.

En effet, d'après l'INSEE, <https://www.insee.fr/fr/statistiques/3714237#titre-bloc-10>

Quelle que soit la distance, le trajet en voiture prédomine

Un tiers des salariés, soit 7,5 millions de personnes, vivent et travaillent dans la même commune. Dans 51 % des cas, le trajet domicile-travail est alors effectué en voiture. La marche à pied constitue le deuxième mode de déplacement (18 %), devant les transports collectifs (16 %). Les Franciliens

utilisent nettement moins leur voiture que les autres salariés (22 % contre 57 % sur le reste du territoire national) et utilisent davantage les transports en commun.

Dès que les salariés doivent quitter leur commune de résidence pour aller travailler, la part de la voiture augmente fortement. Elle atteint 47 % pour les Franciliens et 89 % sur le reste du territoire. Dans un cas sur deux, les salariés parcourent en voiture plus de 15 kilomètres pour atteindre leur lieu de travail et dans un cas sur quatre, plus de 26 kilomètres. Les distances s'allongent pour les personnes habitant des territoires peu denses.

Prenons donc un **trajet de 15km**, en sous-estimant la moyenne, on peut se permettre de généraliser un cas relativement mal synchronisé à un trajet complet où nous comparons un cas relativement réaliste au cas aléatoire.

On obtient cette fois **DeltaConsoAnnée=266.56L** ce qui est conséquent.

2. La consommation à l'échelle d'une société

En prenant les 15M de salariés (les deux tiers restants de l'INSEE) qui ne travaillent pas dans la même commune, et en moyennant la part des voitures et des individus concernés par une mauvaise optimisation à 50%, on obtient 7.5M d'individus concernés par les 15km (qui encore une fois ne sont pas représentatifs du cas de la mauvaise optimisation mais on peut on considérer que c'est un cas fréquent).

On obtient alors grossièrement **ConsoSalariés=2 milliards de L par an** soit **2 millions de m3**

Ce qui rapporté à la consommation nationale d'essence (autour de 50M m3 en 2017) et de diesel, est loin d'être négligeable.

3.Remarques sur l'étude et la démarche:

Une étude plus poussée sur le sujet peut présenter des chiffres évidemment plus faibles mais tout de même non négligeables.

Il faut garder à l'esprit l'aspect uniquement simulé de cette étude, qui dépend donc des considérations de Sumo quant à la consommation des voitures. Une comparaison à une étude expérimentale réelle permettraient ainsi de raccrocher ces résultats à la réalité.

Conclusion:

Cette étude, malgré son caractère simulé (donc par nature éloigné de la réalité) permet tout de même d'alerter sur l'impact de la non optimisation des feux sur la consommation de carburant des trajets dits de migration pendulaire en France. La pollution, bien que moins représentée (du fait d'un manque d'info sur la pollution à l'arrêt) dans cette étude est aussi à considérer, notamment dans les grandes villes car il s'agit d'un problème sanitaire et environnemental.

A propos des choix arbitraires au niveau des statistiques de la partie III, c'est regrettable de ne pas avoir eu plus de temps afin de les analyser avec plus de profondeur. J'ai donc décidé de les laisser tels quels afin d'en discuter plus largement lors de la soutenance.

De nombreuses solutions sont mises en place pour implémenter une optimisation du trafic, elles sont notamment présentées dans cette thèse: [Contrôle et gestion du trafic routier urbain par un réseau de capteurs sans fil](#). Ce sera aussi un point important de la soutenance.

Remerciements

Je tiens à remercier Nel Samama pour sa gentillesse et la justesse de ses remarques tout au long du projet. Je pense que le sujet est réellement au coeur des problématiques actuelles d'optimisation et qu'il peut être creusé bien plus loin que je ne l'ai fait. J'espère que mon travail restera une base correcte pour les prochains étudiants.

Annexe:

Code Tracl fourni par Sumo:

```
#!/usr/bin/env python
# Eclipse SUMO, Simulation of Urban MObility; see
# https://eclipse.org/sumo
# Copyright (C) 2009-2020 German Aerospace Center (DLR) and others.
# This program and the accompanying materials are made available
# under the
# terms of the Eclipse Public License 2.0 which is available at
# https://www.eclipse.org/legal/epl-2.0/
# This Source Code may also be made available under the following
# Secondary
# Licenses when the conditions for such availability set forth in the
# Eclipse
# Public License 2.0 are satisfied: GNU General Public License,
# version 2
# or later which is available at
# https://www.gnu.org/licenses/old-licenses/gpl-2.0-standalone.html
# SPDX-License-Identifier: EPL-2.0 OR GPL-2.0-or-later

# @file    runner.py
# @author   Lena Kalleske
# @author   Daniel Krajzewicz
# @author   Michael Behrisch
# @author   Jakob Erdmann
# @date     2009-03-26

from __future__ import absolute_import
from __future__ import print_function

import os
import sys
```

```

import optparse
import random

# we need to import python modules from the $SUMO_HOME/tools
directory
if 'SUMO_HOME' in os.environ:
    tools = os.path.join(os.environ['SUMO_HOME'], 'tools')
    sys.path.append(tools)
else:
    sys.exit("please declare environment variable 'SUMO_HOME'")

from sumolib import checkBinary # noqa
import traci # noqa

def generate_routefile():
    random.seed(42) # make tests reproducible
    N = 3600 # number of time steps
    # demand per second from different directions
    pWE = 1. / 10
    pEW = 1. / 11
    pNS = 1. / 30
    with open("data/cross.rou.xml", "w") as routes:
        print("""<routes>
            <vType id="typeWE" accel="0.8" decel="4.5" sigma="0.5"
length="5" minGap="2.5" maxSpeed="16.67" \
guiShape="passenger"/>
            <vType id="typeNS" accel="0.8" decel="4.5" sigma="0.5"
length="7" minGap="3" maxSpeed="25" guiShape="bus"/>

            <route id="right" edges="51o 1i 2o 52i" />
            <route id="left" edges="52o 2i 1o 51i" />
            <route id="down" edges="54o 4i 3o 53i" />""", file=routes)
        vehNr = 0
        for i in range(N):
            if random.uniform(0, 1) < pWE:
                print('    <vehicle id="right_%i" type="typeWE"

```

```

route="right" depart="%i" />' % (
    vehNr, i), file=routes)
    vehNr += 1
    if random.uniform(0, 1) < pEW:
        print('    <vehicle id="left_%i" type="typeWE"
route="left" depart="%i" />' % (
    vehNr, i), file=routes)
    vehNr += 1
    if random.uniform(0, 1) < pNS:
        print('    <vehicle id="down_%i" type="typeNS"
route="down" depart="%i" color="1,0,0"/>' % (
    vehNr, i), file=routes)
    vehNr += 1
    print("</routes>", file=routes)

# The program looks like this
#   <tlLogic id="0" type="static" programID="0" offset="0">
# the locations of the tls are      NESW
#   <phase duration="31" state="GrGr"/>
#   <phase duration="6"  state="yryr"/>
#   <phase duration="31" state="rGrG"/>
#   <phase duration="6"  state="ryry"/>
#   </tlLogic>

def run():
    """execute the TraCI control loop"""
    step = 0
    # we start with phase 2 where EW has green
    traci.trafficlight.setPhase("0", 2)
    while traci.simulation.getMinExpectedNumber() > 0:
        traci.simulationStep()
        if traci.trafficlight.getPhase("0") == 2:
            # we are not already switching
            if traci.inductionloop.getLastStepVehicleNumber("0") > 0:
                # there is a vehicle from the north, switch
                traci.trafficlight.setPhase("0", 3)

```

```

        else:
            # otherwise try to keep green for EW
            traci.trafficlight.setPhase("0", 2)
        step += 1
    traci.close()
    sys.stdout.flush()

def get_options():
    optParser = optparse.OptionParser()
    optParser.add_option("--nogui", action="store_true",
                        default=False, help="run the commandline
version of sumo")
    options, args = optParser.parse_args()
    return options

# this is the main entry point of this script
if __name__ == "__main__":
    options = get_options()

    # this script has been called from the command line. It will
    start sumo as a
    # server, then connect and run
    if options.nogui:
        sumoBinary = checkBinary('sumo')
    else:
        sumoBinary = checkBinary('sumo-gui')

    # first, generate the route file for this simulation
    generate_routefile()

    # this is the normal way of using traci. sumo is started as a
    # subprocess and then the python script connects and runs
    traci.start([sumoBinary, "-c", "data/cross.sumocfg",
                  "--tripinfo-output", "tripinfo.xml"])
    run()

```

Modifs apportées pour coquibus:

```
def generate_routefile():
    random.seed(42) # make tests reproducible
    N = 3600 # number of time steps
    # demand per second from different directions
    pWE = 1. / 2
    with open("test.rou.xml", "w") as routes:
        print("""
<routes>

<vType
  vclass="passenger" accel="2.6" decel="4.5" id="Car" length="2.0"
maxSpeed="100.0" sigma="0.0" />

<route id="route0" edges="1to2 2to3 3to4 4to5 5to6 6to7 7to8 8to9
9to10 10to11" />

""", file=routes)
        vehNr = 0
        for i in range(N):
            if random.uniform(0, 1) < pWE:
                print('    <vehicle id="car_%i" type="Car"
route="route0" depart="%i" />' % (
                    vehNr, i), file=routes)
                vehNr += 1
        print("</routes>", file=routes)
```

L'ensemble du code rédigé n'est pas représentable ici et sera transmis sous format zip si nécessaire.

