PROJET SCIENTIFIQUE COLLECTIF
RAPPORT FINAL

MAP04 - Optimisation du contrôle du trafic routier

23 Avril 2018

Mohammed Amine BENNOUNA
Maxime BUYSE
Yassine EL MAAZOUZ
Maximilien LORDON
Ahmed TAHA EDDAHBI
Léo REITZMANN
Tanguy VERNET

Table des matières

Introduction

Le problème du trafic routier est devenu un enjeu majeur à maîtriser pour les grandes métropoles durant ces dernières décennies en raison de l'augmentation du nombre de véhicules et de l'urbanisation accélérée.

Les enjeux sont multiples, car la congestion du réseau routier a un coût économique et environnemental très important au vingt-et-unième siècle. En effet, diminuer les temps de trajet ne permet pas seulement aux usagers du réseau routier d'utiliser plus efficacement leur temps ; il s'agit également de réduire les émissions de gaz d'échappement qui peuvent causer des problèmes environnementaux et sanitaires catastrophiques dans certaines villes.

C'est pourquoi de nombreux scientifiques s'intéressent au problème de l'optimisation du trafic. Il s'agit d'adapter le réseau routier (organisation, plan de feux, etc.) aux flux de véhicules pour obtenir un trafic plus fluide. Pour cela, de nombreuses pistes sont explorées par les chercheurs. La plupart se basent sur des simulations de trafic sur lesquelles on cherche les paramètres du réseau qui donnent le trafic le plus fluide.

Les feux tricolores sont un des principaux outils utilisés par les stratégies de gestion du trafic puisque ce sont eux qui contrôlent et restreignent le trafic aux intersections dans les zones urbaines. Durant les vingt dernières années, la recherche dans ce domaine a donné lieu à des avancées intéressantes, tant pour les séquences fixées à l'avance, que pour les systèmes de micro-régulation qui s'adaptent en temps réel au trafic grâce à des capteurs situés à proximité des intersections. Cependant, de nombreuses agglomérations n'utilisent pas encore ces techniques avancées.

Même si ces méthodes de contrôle peuvent bénéficier de l'expérience d'une large variété d'incidents routiers et de schémas de congestion de trafic, elles sont incapables de s'adapter à des situations inédites. De plus, elles ne permettent pas de coordonner les différentes intersections entre elles et de disposer d'une stratégie globale qui optimiserait les séquences des feux sur tout le réseau.

Le développement de nouvelles théories en informatique et mathématiques (les récents progrès sur les problèmes d'optimisation, liés au Machine Learning, par exemple) a conduit à de très nombreuses recherches sur ce sujet ; le principal problème à contourner est la dimension de l'espace des plans de feux de signalisation, qui rend les temps de calcul extrêmement importants.

Ainsi, la construction de modèles d'optimisation applicables à des réseaux routiers de taille conséquente reste aujourd'hui largement théorique ; les méthodes fonctionnant en temps réel aussi, à cause du manque d'information disponibles (acquises notamment grâce à des capteurs fixes ou des véhicules connectés).

Nous avons immédiatement été attirés par ce thème, proposé par Aimsun (ex TSS, Transport Simulation Systems), entreprise espagnole qui développe et commercialise un logiciel de simulation de trafic routier, adopté par plus de 4000 utilisateurs au sein d'agences

gouvernementales et de centres de recherche, notamment la mairie de Paris.

Sur le plan pratique, notre PSC consiste à implémenter un algorithme qui optimise le temps de trajet total (somme des temps de trajet de tous les véhicules) sur un réseau routier, une durée et une demande d'usagers données. Pour cela, l'entreprise Aimsun met à notre disposition son logiciel de simulation de trafic ainsi que des exemples de réseaux routiers numériques (Paris, Barcelone, etc.).

Algorithme d'optimisation

1 Introduction : Simulation-Based Optimisation (SO)

Le but de notre PSC est de mettre en place une méthode pour trouver le(s) "meilleur(s)" plan(s) de feux pour un réseau routier donné, à l'aide du simulateur d'Aimsun. Pour cela, il faut d'abord exprimer ce problème dans un cadre mathématique, puis algorithmique : on va formuler un problème d'optimisation.

On cherche à minimiser le temps de trajet total des véhicules dans un réseau et une période donnés, en fonction du plan de feux choisi. Il faut également tenir compte des contraintes sur le plan de feux (durée des cycles de feux, durée minimale du feu vert... etc.). On parle de problème d'optimisation sous contrainte.

Ce problème a une autre particularité : la fonction objectif (temps de trajet total) est calculée par Aimsun, via une simulation complexe (on n'a pas de formule exacte). De plus, les simulations prennent en compte certains effets aléatoires. La fonction objectif n'est donc pas déterministe et on n'a aucune information sur sa régularité et ses dérivées. On parle de derivative-free (pour l'absence d'informations sur les dérivées) simulation based optimisation.

Il s'agit donc de minimiser l'espérance d'une fonction aléatoire. On peut obtenir une approximation de cette espérance en faisant des moyennes sur plusieurs simulations. C'est là qu'apparaît une difficulté caractéristiques des problèmes SO : le coût des simulations. Une simulation sur Aimsun dure au moins quelques dizaines de secondes. Comme un algorithme d'optimisation nécessite de nombreuses évaluations de la fonction objectif, un enjeu majeur dans la conception de l'algorithme est de réduire au minimum le nombre de simulations effectués.

C'est pourquoi une méthode envisageable est d'utiliser des métamodèles (littéralement modèle du modèle c'est-à-dire une fonction qui approche la fonction objectif d'Aimsun - le modèle initial). L'idée est de construire une approximation (le métamodèle) de la fonction objectif à partir d'un nombre restreint de simulations et d'appliquer les techniques d'optimisation sur ce métamodèle (donc sans simuler).

Nous nous sommes inspirés d'un algorithme de Carolina Osorio [1]. L'algorithme d'Osorio emprunte à un article de Conn et al. [2] le cadre théorique pour l'optimisation sans dérivée, à région de confiance (quoique [2] traite d'optimisation sans contrainte).

2 Méthode d'optimisation : derivative-free, trust-region optimisation [2]

La technique d'optimisation employée par Osorio est donc inspirée d'un article de Conn. et al. [2]. Les auteurs y établissent la convergence d'une classe d'algorithmes d'op-

timisation : on parle de derivative-free, trust-region optimisation.

Comme mentionné plus haut, au vu du caractère non déterministe de la fonction objectif et de la grande dimension de l'espace des plans de feux, un algorithme de type descente de gradient est inenvisageable. Les techniques développées et prouvées par Conn et al. sont donc tout indiquées.

Trust-region désigne une famille d'algorithmes itératifs. Les algorithmes itératifs calculent une suite de points (les itérées) qui approchent de mieux en mieux le minimum de la fonction à optimiser. Les algorithmes à région de confiance (trust-region) calculent les itérées successives en créant une approximation de la fonction objectif - la fonction modèle - que l'on considère fiable sur une petite région entourant l'itérée en cours : la région de confiance.

L'article de Conn et al. donne des conditions suffisantes sur le type de fonctions modèles pour que l'algorithme proposé fonctionne. En bref, il faut connaître une méthode permettant, sur n'importe quelle région de confiance : de vérifier que la fonction modèle est "fiable" (elle doit vérifier des conditions de proximité avec la fonction objectif au premier ordre) ; le cas échéant, de proposer, en un nombre fini d'étapes, une fonction modèle "fiable" (quitte à réduire la taille de la région de confiance).

Le fonctionnement (simplifié) de l'algorithme est détaillé ci-dessous :

- **Initialisation** :
  - Choisir x_0 (première itérée) et Δ_0 (rayon de confiance initial).
  - Choisir une classe de fonctions modèles.
  - Choisir une méthode de minimisation des fonctions modèles.
  - Choisir une première fonction modèle m_0.
- **Itération** k :
  1. Test critique :
     Si le gradient de la fonction modèle est trop faible (seuil fixé), tester et (le cas échéant) améliorer la fonction modèle. Sinon, conserver la fonction modèle.
  2. Calcul de la nouvelle itérée :
     Calculer le minimum x_k de la fonction modèle dans la région de confiance.
  3. Test du nouveau point : Si la fonction objectif décroît suffisamment (seuil fixé) entre l'ancienne itérée et la nouvelle, accepter x_k et choisir une nouvelle fonction modèle sur une région de confiance centrée en x_k. Sinon, rejeter x_k.
  4. Mise à jour du métamodèle :
     Tester et (le cas échéant) améliorer la fonction modèle.
  5. Mise à jour du rayon de confiance :
     Si x_k est accepté, augmenter le rayon de confiance. Sinon, diminuer le rayon de confiance.
- **Test d'arrêt** :
  L'algorithme s'arrête dès que le budget computationnel (le nombre de simulations autorisées) est dépassé.

Le fonctionnement de l'algorithme est représenté sur la figure 1.

A partir de cette formulation générale de l'algorithme, il faut choisir, pour l'implémenter : une classe de fonctions modèles et une méthode pour les calculer, une méthode de minimisation des fonctions modèles, une méthode pour tester et améliorer les fonctions modèles, un ensemble de constantes diverses (non citées ci-dessus), déterminant les seuils des différents tests effectuées dans l'algorithme de Conn et al.

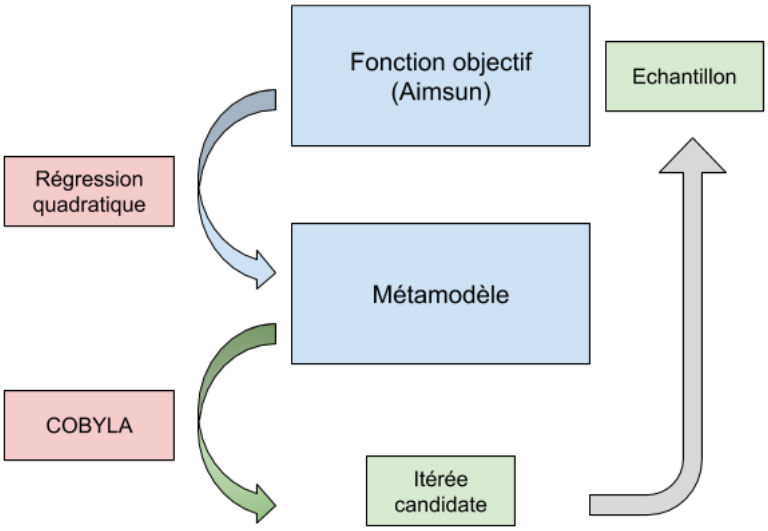

Figure 1: Fonctionnement schématique de l'algorithme d'optimisation

Ce sont ces choix que nous allons détailler dans les sections suivantes.

3 Choix du métamodèle (Osorio, 2013)

D'abord, intéressons nous au choix des fonctions modèles (i.e. des métamodèles) développés dans Osorio [1]. C. Osorio propose essentiellement deux types de métamodèles : un métamodèle de type "fonctionnel" - c.a.d. une fonction modèle polynomiale, calculée sans tenir compte de la nature spécifique du problème (trafic routier), un métamodèle de type "physique" - c.a.d. une fonction modèle calculée à l'aide d'un modèle de files d'attente (queuing model).

Osorio propose un métamodèle mélangeant ces deux classes de fonctions modèles, afin de "coller au mieux aux données".

$$m(x, y; \alpha, \beta, q) = \alpha T(x, y; q) + \Phi(x; \beta)$$

Avec les notations suivantes :
x est le vecteur indiquant les durées des séquence de feux tricolores
T est la composante physique du métamodèle
Φ est la composante fonctionnelle du métamodèle
q est un paramètre du réseau
y représente des variables liées au réseau

Nous développerons dans cette partie uniquement la composante fonctionnel du métamodèle. La composante physique (queuing model) fera l'objet d'une partie ultérieure.

Décrivons, pour commencer, l'argument de la fonction modèle. Il s'agit d'un vecteur réel x à d composantes. Chaque composante correspond à la durée d'une phase d'un cycle de feux tricolores. Par exemple, pour un réseau contenant 2 intersections avec feux tricolores, chacune possédant respectivement 3 phases et 6 phases, $d = 9$.

Il est important de noter que, contrairement à l'algorithme proposé par Conn et al. [2], le vecteur x obéit à des contraintes (composantes positives, somme des composantes constantes pour une intersection donnée). Cela va influencer le choix de la méthode de minimisation. La composante fonctionnelle retenue par Osorio est un polynôme de degré 2 sans termes croisés (uniquement des termes diagonaux). Il s'agit donc de choisir $2d+1$ coefficients.

Ces coefficients sont calculés par régression quadratique en les points où la fonction objectif a déjà été évaluée. C'est pourquoi il est important, étant donné le coût de la simulation, de garder en mémoire toutes les valeurs connues de la fonction objectif.

La régression quadratique est formulée ainsi :

$$min_{\alpha,\beta} \sum_{i=1}^{n_k} (w_{ki}(f(x^i,p) - m(x^i,y^i;\alpha,\beta,q))^2 + (w_0(\alpha-1))^2 + \sum_{i=1}^{2d+1} (w_0\beta_i)^2$$

Le poids w_i permet de quantifier l'importance de chaque point x_i en fonction de sa distance par rapport au point actuel : sa valeur est inversement proportionnelle à sa distance par rapport au point actuel.

Enfin, il nous faut une méthode pour tester la validité et améliorer les fonctions modèles polynomiales. Une telle méthode est fournie par Conn et al. dans un article antérieur [3] pour les polynômes calculés par régression quadratique.

Il suffit pour cela de vérifier une propriété de l'échantillon de points à partir duquel est effectué la régression quadratique : la "Λ - poisedness". Cette propriété permet de contrôler les erreurs d'ordre 0 et 1 entre la fonction modèle et la fonction objectif. Conn et al. fournissent dans [3] une méthode pour garantir la "Λ - poisedness" de l'échantillon. Cette méthode repose sur la factorisation LU d'une matrice de Vandermonde.

4 Calcul des itérées : méthode COBYLA (Powell 1994)

Il reste ensuite à choisir une méthode de minimisation des fonctions modèles sur une région de confiance. Comme on le verra dans une prochaine section, la nature du méta-modèle physique empêche d'accéder facilement aux dérivées du métamodèle. De plus, la minimisation doit se faire sous contrainte.

Contrairement à C. Osorio, nous avons codé notre algorithme en Python. Nous avons donc cherché une méthode de minimisation répondant à ces critères dans la bibliothèque SCIPY.OPTIMIZE.

Notre choix s'est porté sur la méthode COBYLA, issue d'un article de Powell [4]. Cette méthode est itérative et repose sur l'évolution d'un simplexe dans l'espace des arguments de la fonction objectif. Les itérées successives sont calculées en fonction des sommets du simplexe et de leur image par la fonction objectif, augmentée des contraintes. Nous n'entrerons pas plus dans les détails, la méthode étant déjà entièrement codée dans le package Python choisi.

5 Implémentation de l'algorithme

Pour finir, donnons quelques détails sur notre implémentation de l'algorithme.

Comme mentionné précédemment, nous l'avons intégralement codé en Python. Nous avons repris les choix de constantes de l'article de C. Osorio (seuils pour les tests, nombres de simulations autorisées... etc.).

Nous avons également choisi de ne pas implémenter le test critique (première étape de la boucle), pour deux raisons : parce que le gradient de la composante physique du métamodèle est difficile à calculer (et probablement coûteux), et parce que C. Osorio n'a observé aucun déclenchement du "mode conservatif" de son algorithme c.a.d. de la méthode d'amélioration du métamodèle et qu'elle donne peu de détails sur celle-ci.

Nous attendions de faire tourner notre algorithme pour choisir quelle attitude adopter vis-à-vis de ce test critique. Une option envisageable aurait été de mesurer le gradient de la composante fonctionnelle uniquement (très facile à calculer). De plus, au vu de la technicité de la méthode proposée par Conn et al. dans [3], nous voulions être certains que cette étape soit indispensable avant de la coder.

Queuing Network Model

Le Queuing Network Model est une modélisation du trafic routier dans un réseau reposant sur la théorie des files d'attente. Osorio en propose plusieurs déclinaisons dans ses articles successifs. Il fallait donc d'abord choisir le modèle adéquat. Nous avons en particulier opté pour la forme la plus simple du modèle, exposé dans son article de 2015 [5].

Cette forme est la plus simple en ce qu'elle comporte moins de variables que les modèles précédemment développés par Osorio, et permet donc une résolution numérique par l'ordinateur plus rapide et moins gourmande en ressources informatiques, afin de pouvoir l'appliquer à des réseaux routiers de plus grande taille. Il faut toutefois garder à l'esprit que cette simplification s'accompagne d'une diminution de la précision du modèle. Pour autant, il est intéressant pour le méta-modèle d'avoir une composante physique qui privilégie le temps de calcul plutôt que la précision des résultats, étant donné que le simulateur microscopique nous fournit des résultats très réalistes (grâce à une modélisation des comportements des véhicules individuels, prenant en compte des effets aléatoires) mais coûteux en ressources informatiques.

Nous présentons ici les résultats de la théorie sans rentrer dans les détails. Dans ce qui suit, nous adopterons les notations suivantes :

γ_i external arrival rate,
λ_i total arrival rate,
μ_i service rate,
ρ_i traffic intensity
P_i^f probability of being blocked at queue i,
k_i upper bound of the queue length,
N_i total number of vehicles in queue i,
$P(N_i = k_i)$ probability of queue i being full, also known as the blocking or spillback probability,
p_{ij} transition probability from queue i to queue j,
D_i set of downstream queues of queue i.

Avec ces notations, et après quelques simplifications, le Queuing Model est formulé comme un système d'équations dont les paramètres d'entrée (exogènes, caractérisant le réseau) sont : γ_i, p_{ij}, k_i, D_i et les inconnues de l'équation (paramètres endogènes) sont : λ, ρ_i, $P(N_i = k_i)$. Le système d'équations du modèle est le suivant :

$$\begin{cases} \lambda_i = \gamma_i(1 - P(N_i = k_i)) + \sum_j p_{ij}\lambda_j \\ \rho_i = \frac{\lambda_i}{\mu_i} + (\sum_{j \in D_i} p_{ij} P(N_j = k_j))(\sum_{j \in D_i} \rho_j) \\ P(N_i = k_i) = \frac{1-\rho_i}{(1-\rho_i)^{k_i+1}} \rho^{k_i} \end{cases}$$

La résolution de cette série d'équations nous permet d'avoir tous les paramètres décrivant le trafic routier dans le réseau. A partir de ces paramètres, on peut calculer la composante physique $T(x, y, q)$ qui représente l'espérance du temps de trajet total avec la formule suivante :

$$\frac{\sum E[N_i]}{\sum_i \gamma_i(1 - P(N_i = k_i))}$$

où E[Ni] est l'espérance du nombre de véhicules dans la ligne i, qui s'exprime selon la formule :

$$E[N_i] = \rho_i(\frac{1}{1-\rho_i} - \frac{(k_i+1)\rho_i^{k_i}}{1-\rho_i^{k_i+1}})$$

1 Variables du modèle

Le rôle du pôle Queuing Model (Léo Reitzmann, Amine Bennouna) était double : d'une part adapter algorithmiquement le modèle pour l'implémenter dans le langage Python ; d'autre part extraire d'Aimsun les données relatives au modèle. Certaines variables n'étaient pas directement disponibles sur Aimsun, il a donc fallu trouver une méthode afin de les calculer à partir de variables extraites intermédiaires. Voici la liste des des variables qu'il fallait déterminer, sur lesquelles nous nous sommes penchés :

- k_i *upper bound of the queue length* : il s'agit du nombre maximal de voitures dans une "queue" (une queue correspond à une voie, et est appelée "lane" dans le logiciel Aimsun). Ce paramètre est approximé en divisant la longueur de la voie considérée par la longueur moyenne d'un véhicule. Les longueurs de voies ne sont pas renvoyées pas Aimsun et sont donc calculées à la main.

- p_{ij} *transition probability from queue i to queue j* : il s'agit de la probabilité pour un véhicule quelconque d'emprunter la queue j juste après avoir emprunté la queue i. Ces probabilités dépendent du réseau routier, du plan de feu utilisé, et de la demande en véhicule, et sont représentées dans une matrice carrée. Ces probabilités ne sont pas disponibles directement dans le logiciel Aimsun, il faut les estimer expérimentalement en faisant une étude statistique sur des simulations dans le logiciel.
  La méthode que nous avons mise au point et codée en Python et SQL est la suivante : pour une simulation Aimsun, nous récupérons dans une base de données SQL, les "queues" empruntées par chacun des véhicules à des instants régulièrement espacés. On peut alors obtenir pour chaque véhicule sa trajectoire (la liste des queues successives empruntées). Pour chaque transition d'une queue à une autre

par un véhicule, on incrémente la case correspondante dans la matrice des probabilités d'une unité. On normalise ensuite chaque ligne afin d'obtenir la matrice stochastique voulue.
Il faut noter que cette méthode est imprécise, pour les raisons suivantes entre autres :

— Cette détermination expérimentale varie de simulation en simulation (pour les mêmes paramètres) : en effet le logiciel Aimsun est très détaillé et comporte des éléments aléatoires.

— La durée de la simulation utilisée pour estimer la matrice de probabilités de transition (une minute) de queues est plus faible que celle utilisée dans le reste de l'algorithme (une heure). En effet, les positions de chacune des voitures est mesurée chaque seconde, ce qui crée une base de données à plusieurs centaines milliers de lignes pour le nombre total de voitures (de l'ordre du millier) : le logiciel Aimsun plante avant de pouvoir remplir la base de données pour la durée totale (une heure) de la simulation.

— Ces probabilités dépendent du plan de feu : par exemple, un plan de feu peut forcer toutes les voitures à emprunter un certain chemin, ce qui modifie fortement la matrice de probabilités. Cependant, nous n'évaluons cette matrice qu'au début de l'algorithme. En effet, la raison d'être de la composante fonctionnelle du méta-modèle est de fournir une estimation de la fonction objectif indépendamment du logiciel Aimsun, afin d'être relativement facile à calculer. Nous acceptons donc le compromis de garder la même matrice de probabilités malgré le changement de plan de feu à chaque itération de l'algorithme.

- D_i *set of downstream queues of queue i* : il s'agit de l'ensemble des queues accessibles depuis la queue i. De même que pour les probabilités de transitions de queues, nous représentons ces paramètres dans une matrice carrée où la ligne correspond à la queue i. Pour l'obtenir, nous remplaçons simplement les coefficients non nuls de la matrice de probabilité de transitions de queues par des coefficients égaux à 1. Les limites qui s'appliquent à la matrice de probabilités sont aussi valables ici.

- γ_i *external arrival rate* : ce paramètre correspond au nombre de voitures par unité de temps arrivant dans la queue i. Nous n'avons pas réussi à l'obtenir de façon automatique dans une base de données renvoyée par le logiciel. Il est toutefois possible de l'obtenir à la main dans l'interface graphique du logiciel Aimsun, via un calcul utilisant la matrice de demande de véhicules et le réseau routier étudié.

- μ_i *service rate* : il s'agit du nombre de voitures par unité de temps quittant la queue i. Ce paramètre est lié au plan de feu. Nous n'avons pas réussi à l'obtenir à partir des données renvoyées par Aimsun. Cela peut cependant se calculer à la main à partir du plan de feux et de la carte du réseau.

Dans cette partie du travail, de même que dans la partie concernant l'interface entre notre console Python et le logiciel Aimsun, nous nous sommes rendus compte que notre utilisation du logiciel Aimsun était assez inhabituelle, en comparaison de l'utilisation qu'en font les principaux clients d'Aimsun, comme par exemples les collectivités territoriales.

En effet, nous avons dû extraire des données qui ne sont pas renvoyées simplement par le logiciel, ce qui nous a menés à créer des méthodes de calcul approché adaptées aux possibilités du logiciel. Il a fallu faire un usage inventif de la documentation d'Aimsun ; notre tutrice de l'entreprise Aimsun, Madame Lenorzer, nous a aidés sur ce point en nous fournissant de la documentation complétant le guide utilisateur disponibles dans l'interface graphique, et en nous dirigeant vers des méthodes possibles. Cependant, certains paramètres nous échappent encore et doivent être calculés à la main.

Le pôle Algorithme du groupe de PSC a codé un script Python traduisant l'algorithme écrit par le pôle Queuing Model. La résolution du système d'équations non-linéaires se fait grâce à la méthode nsolve de la bibliothèque sympy. Nous n'avons pas encore testé le queuing model sur un réseau routier, mais cela est sur le point d'être réalisé.

Aimsun/Python

1 Utilité de l'interface

Le simulateur de Aimsun est équipé d'un interpréteur Python interne qui permet de lancer des scripts. Cependant ce dernier ne contient pas toutes les librairies et les modules Python qui nous sont utiles dans notre étude du trafic routier (numpy, scipy, sympy, random ....etc). Il est donc nécessaire de traiter les données renvoyées par le simulateur Aimsun par un interpréteur Python externe. L'algorithme que nous avons implémenté étant itératif, il était de plus nécessaire d'envoyer les paramètres de simulation à l'interpréteur interne.

Le but de l'interface Aimsun/Python est justement de faire communiquer les deux interpréteurs en créant une connexion TCP/IP, afin de pouvoir simuler sur le logiciel Aimsun et lancer le processus d'optimisation en parallèle sur l'interpréteur Python externe. Pour cela nous avons organisé une réunion avec le groupe PSC X2015 qui a travaillé avec Aimsun l'an dernier afin de comprendre l'architecture de l'interface qu'il ont développée.

Nous avons procédé dans un premier temps à débugger l'interface en question et à résoudre quelques problèmes liés à l'établissement de la connexion " Bridge TCP/IP " entre les deux interpréteurs Python interne et externe. Dans ce processus, nous avons rencontré de nombreux problèmes d'implémentations et d'organisation du fonctionnement de l'interface que l'ancien groupe n'a pas eu le temps de résoudre.

L'interface ne répondant pas à nos besoins, nous étions aussi obligés de l'adapter en tenant compte des résultats que nous cherchions à obtenir et aux fonctionnalités dont nous aurions besoin pour lancer notre algorithme d'optimisation. Nous avons également rencontré plusieurs bugs dans l'implémentation de l'interface, ce qui nous a forcé à changer quelque parties de celle-ci et d'automatiser son fonctionnement au maximum.

Pour ce faire, nous avons utilisé une architecture orientée objet simple sur Python :

- La partie connexion était assurée par deux objet : Connector et Listening_Thread le premier créée par l'algorithme et l'autre par Aimsun en exécutant le script AimsunModule.
- La partie concernant l'envoi et la réception des commandes est assurée par un objet Communicator et par l'objet Connector qui possède des méthodes d'envoi de requêtes au simulateur Aimsun.

2 Fonctionnement de l'interface

L'interface consiste essentiellement en trois objets Python : *Communicator*, *Listening_Thread* et *Connector*.

Le fonctionnement de l'interface s'articule comme suit :

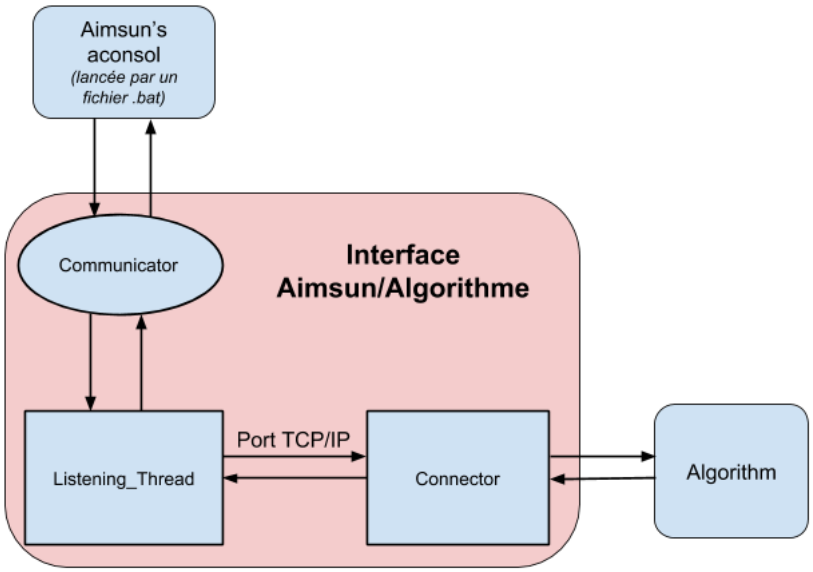

Figure 2: Fonctionnement de l'interface Python-Aimsun

- L'algorithme crée une instance de la classe Connector. Celui-ci réserve un port TCP/IP et attend qu'un objet Listening_Thread se connecte au même port pour échanger.
- Un fichier système crée une instance de la classe Listening_Thread dans la console Python de Aimsun. Ce Listening_Thread prend comme paramètre une instance de la classe Communicator et se connecte au même port que l'objet Connector a réservé et attend les requêtes de l'algorithme.
- Une requête envoyée par l'algorithme via l'objet Connector, est passée à l'objet Listening_Thread via le port TCP/IP. Celui-ci l'envoie ensuite à l'objet Communicator qui communique la requête au simulateur de Aimsun.
- Aimsun est alors mis en contact avec l'algorithme d'optimisation (via la aconsole) et les instructions de l'algorithme sont envoyées via le bridge TCP/IP sous forme de requêttes.

Les requêtes qu'on peut exécuter sont :

- SET + Parameters : permet de modifier les paramètres d'une simulation, notamment le plan de feux que l'on modifie grâce aux méthodes setFrom et setDuration de l'objet controlPhase de Aimsun.
- SIMULATE : qui permet de lancer une simulation sur le simulateur via la méthode simulate de l'objet simulator.
- INTERRUPT : qui permet de fermer la connexion et de libérer le port TCP/IP réservé auparavant.

Pour rendre le fonctionnement de l'interface plus "user-friendly", on a opté pour une exécution en parallèle de notre algorithme d'optimisation et d'Aimsun, en utilisant des "Thread" d'exécution. Cela permet de lancer en même temps les simulations et les calculs d'optimisation, de gagner en rapidité et rend l'exécution de l'algorithme plus facile et automatique.

Résultats, problèmes scientifiques et techniques

La mise en œuvre de l'algorithme s'est révélée bien plus complexe que prévu, principalement à cause de difficultés dans l'utilisation du logiciel et de bugs dans l'interface entre le logiciel aimsun et notre algorithme Python. Ces problèmes nous ont retardé pendant plusieurs mois, et ce n'est que récemment que nous avons pu faire fonctionner avec succès le programme.

1 Problèmes techniques

Adaptation au logiciel aimsun

Pour commencer, nous devions prendre en main Aimsun. Celui-ci s'est avéré difficile à utiliser du fait du grand nombre d'options et de fonctionnalités qu'il propose. De plus, le but de ce logiciel est de permettre de modéliser le trafic routier afin d'analyser l'impact de stratégies de gestion routière. Ainsi il n'est pas toujours adapté pour l'usage que nous en avons fait.

Par exemple, certaines données dont nous avions besoin pour le métamodèle n'étaient pas disponibles dans la liste des paramètres proposées par Aimsun, comme la probabilité de changement de voie pour les véhicules, que nous avons dû estimer nous-mêmes en nous basant sur des statistiques faites sur un grand nombre de simulations.

Par ailleurs, l'algorithme était aussi un peu difficile à adapter pour exploiter le "queuing model", vu le grand nombre de paramètres exogènes du modèle et de variables qui entrent en jeu. La difficulté résidait principalement dans l'extraction et l'exploitation de ces paramètres d'Aimsun (sous forme de base de données SQLite) d'une part, et dans la résolution des systèmes d'équations qui les régissent d'autre part, surtout que celles-ci sont non linéaires et nécessitent un temps de calcul important.

2 Problèmes informatiques

Dans un premier temps nous avons pu faire tourner l'algorithme tout seul, sans utiliser l'interface avec Python, et constater qu'il fonctionnait correctement.

Néanmoins, la principale difficulté de ce projet a été de faire fonctionner l'interface. Nous avons rencontré de nombreux problèmes purement informatiques qui empêchaient l'algorithme de tourner ; tout d'abord, un problème de système d'exploitation, puisqu'au départ l'interface ne fonctionnait que sur linux, et nous renvoyait une erreur sur Windows.

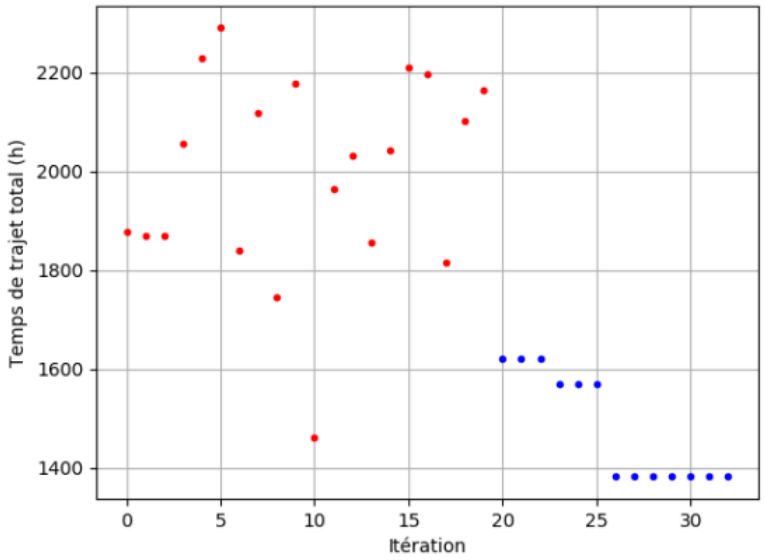

Figure 3: Résultat d'une mise en œuvre de l'algorithme.

Nous avons également rencontré des problèmes de socket dans l'interface, et de requêtes aimsun erronées que nous avons pu résoudre grâce à Annique Lenorzer et les experts d'Aimsun.

3 Résultats

Nous avons mis en œuvre l'algorithme sur un réseau routier qui modélise la place de la Catalogne à Barcelone.

Le graphique Figure 3 représente le résultat d'une mise en œuvre de l'algorithme. A chaque itération du programme (en abscisses), on calcule le temps de trajet total des véhicules sur le réseau, et on le représente en ordonnées. On effectue tout d'abord les premières itérations (représentées en rouge) avec un certain nombre de points distincts afin d'initialiser le métamodèle avec une première régression. C'est pourquoi les points sont répartis de façon aléatoire.

Les points en bleus représentent les itérations qui ont été effectuées à partir du moment où débute l'optimisation.

On constate que le temps de trajet évolue comme prévu par l'algorithme : soit le nouveau point donné par la fonction de minimisation est accepté et le temps de trajet diminue de façon notable, soit on garde le même point si le test est négatif.

Le deuxième graphique, Figure 4, représente le même résultat, mais pour un nombre d'itération maximal moins important.

On constate bien dans les deux cas une amélioration du temps de trajet total par rapport au plan de feux initial qui est aléatoire, et que les résultats sont meilleurs avec un nombre plus important d'itérations.

En raison de la puissance limitée de nos ordinateurs, nous nous sommes limités pour l'instant à un faible nombre maximal d'itération. En effet, l'algorithme reste assez lent à tourner. Ce qui est dû aux fonctions Python utilisées pour la régression quadratique et

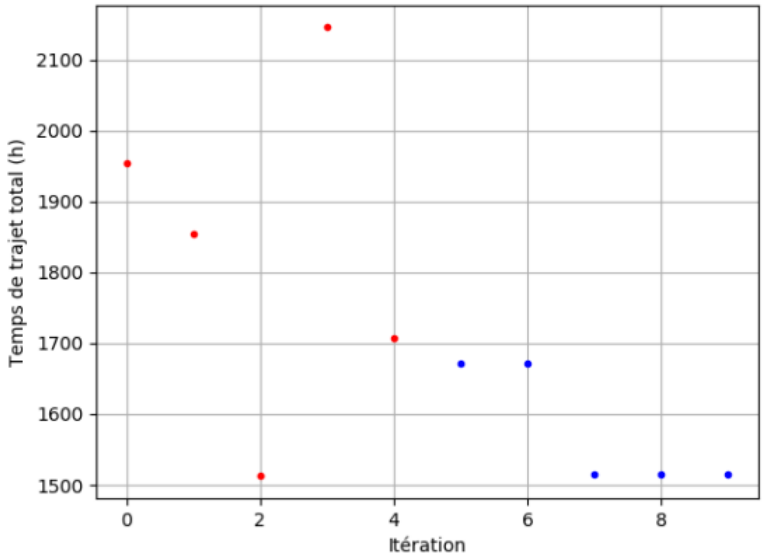

Figure 4: Résultat d'une mise en œuvre de l'algorithme, nombre d'itération maximal moins important.

la minimisation de la fonction sur une région de confiance, ainsi qu'aux simulations du logiciel aimsun (chacune prend plusieurs dizaines de secondes). Pour poursuivre le travail, il faudrait tester l'algorithme en ajoutant le "queuing model", et en comparant les résultats avec ceux que nous avons déjà obtenu en utilisant uniquement la composante fonctionnelle du métamodèle.

On pourrait également effectuer un grand nombre de simulations en augmentant à chaque fois le nombre maximal d'itération afin de comprendre comment le temps de trajet minimal varie en fonction du nombre de simulations.

Il reste encore à comparer nos résultats avec d'autres types de méthodes. Ceci est rendu difficile puisque nous n'avons pas accès à des résultats qui auraient été effectué sur un réseau complètement identique.

Organisation

Outre la dimension scientifique, le PSC comporte une dimension collective. Le bon fonctionnement du travail en groupe est indispensable à la réalisation du projet et nécessite donc une organisation rigoureuse. Les aspects organisationnels de notre projet étaient doubles : la communication et l'organisation au sein de notre groupe, et l'interaction avec l'entreprise Aimsun (anciennement TSS).

1 Organisation interne

Nous avons pris contact avec Aimsun en juin dernier via le groupe de PSC de la promotion 2015 qui avait travaillé avec eux. Plusieurs sujets différents nous ont alors été proposés. Après discussion, nous avons choisi de travailler sur l'optimisation du temps de trafic sur un réseau en modifiant le plan de feux.

Une fois cette étape passée, une phase de recherche documentaire a commencé : nous avons étudié des articles de recherche (d'abord proposés par nos référents à Aimsun, puis trouvés au fil de nos recherches). Pour ce faire, nous tenions un tableau Google Sheet partagé avec les liens des articles à lire et les commentaires et observations de ceux d'entre nous qui les avaient lus.

Au terme de cette phase de documentation, nous sommes arrivés à un état de l'art sur les techniques d'optimisation du trafic. Les travaux de Carolina Osorio ont particulièrement attiré notre attention et nous avons choisi de nous inspirer d'un de ses algorithmes [5].

Nous avons présenté cet état de l'art ainsi que l'orientation retenue pour notre PSC dans la proposition détaillée que nous avons rendue fin septembre. Nous avons également présenté ces éléments à Aurore Rémy et Flore Nabet (Annique Lenorzer nous suivait en visioconférence) lors de notre réunion de cadrage.

A partir d'octobre, nous sommes entrés dans la phase de réalisation du projet. Pour ce faire nous avons identifié trois tâches différentes à effectuer et nous avons donc réparti notre équipe en trois pôles :

- Pôle algorithme (Amine et Léo) : en charge de l'adaptation de l'algorithme (et notamment du queuing model)
- Pôle programmation (Maximilien, Taha et Tanguy) : en charge de l'implémentation de l'algorithme en Python
- Pôle interface (Yassine et Maxime) : en charge de la réalisation de l'interface entre Python et Aimsun

Nous avons alors mis en place notre organisation pour cette phase de travail : nous nous retrouvions le mercredi à 13h pour appeler nos référents Aimsun (quand ils étaient disponibles), discuter de l'avancement du projet et demander de l'aide sur les points qui nous posaient problème. Ensuite, nous en profitions pour discuter du projet avec tout le

groupe avant de nous séparer selon nos pôles pour travailler sur nos tâches respectives. A l'intérieur de chacun des pôles, le travail pouvait encore être divisé (écriture de parties différentes de manière indépendante pour le pôle programmation par exemple) ou alors fait en groupe.

Le pôle interface a également organisé une rencontre avec Benjamin Petit, notre camarade X2015 qui s'était occupé de l'interface Python/Aimsun que nous avons partiellement reprise, il nous a alors expliqué le principe de fonctionnement de cette interface.

Cette organisation a bien fonctionné, les tâches ont été bien divisées et cela permettait à chacun de se concentrer sur quelque chose de précis en évitant l'inertie du groupe. Le seul problème lié à cette division du travail était que certains passaient parfois un après-midi à essayer de résoudre un problème (par exemple lié à la maîtrise du logiciel Aimsun), alors que d'autres auraient pu leur donner la solution s'ils avaient eu connaissance du problème rencontré. Cependant nous réglions en général ce genre de problèmes grâce aux réunions de la semaine suivante.

Ainsi, nous avons pu nous rendre compte de l'importance de l'organisation et de la division du travail dans tout projet impliquant une équipe, ainsi que des forces et des faiblesses du travail en groupe.

Néanmoins, vers la fin du mois de janvier le pôle programmation a fini d'écrire l'algorithme. C'est pourquoi, après avoir rendu le rapport intermédiaire, notre organisation devenait obsolète. C'est peut-être à ce moment que nous avons eu le plus de difficultés sur le plan organisationnel.

En effet, nous n'avons pas réellement défini de nouvelle organisation à ce moment-là et la répartition s'est alors faite naturellement. Nous avions constaté que la partie posant le plus de difficulté était l'interface et les membres du pôle programmation sont donc venus en renfort du pôle interface. Cependant ça n'était pas facile pour eux puisqu'ils n'avaient pas participé à son élaboration depuis le début. Le pôle algorithme a, quant à lui, continué à travailler de son côté. En effet, ce qui leur posait problème était la réalisation du queuing model qui n'est pas indispensable pour faire tourner l'algorithme (contrairement à l'interface).

L'idéal aurait peut-être été d'avoir positionné trois personnes sur l'interface et deux sur la programmation dès le départ. Mais il nous était difficile à ce moment-là de savoir quelle tâche serait la plus délicate. Ce que nous retenons de cela, c'est que l'organisation que l'on choisit au départ n'est pas toujours la bonne, car il est difficile de prévoir les difficultés que l'on va rencontrer. Néanmoins, il est important, lorsqu'on s'en rend compte, de remettre les choses à plat et de choisir une nouvelle organisation, bien définie et mieux adaptée à la suite du projet.

Ce projet aura donc été très instructif sur les manières d'organiser une équipe pour la réalisation d'un projet (et complète en ce sens les connaissances que nous avons reçues en cours de MIE) et cela nous servira tous dans notre vie professionnelle à venir.

2 Communication avec Aimsun

Ce PSC a été réalisé en collaboration avec l'entreprise Aimsun. Nous avons notamment été en contact avec Aurore Remy, notre tutrice, directrice adjointe de la filiale française d'Aimsun, et Annique Lenorzer, développeuse. Aimsun étant basée à Barcelone, nous avons communiqué principalement par skype ainsi que par e-mail. Nous organisions une réunion skype le mercredi à 13h lorsque c'était possible (une semaine sur deux en moyenne) et

nous posions nos questions par e-mail à Annique.

Cette organisation permettait que notre travail soit suivi de manière régulière et que nous ne soyons pas bloqués sur des problèmes techniques. En effet, nous avons rencontré de nombreux problèmes dans notre utilisation d'Aimsun et ils ont été résolus par Annique et ses collègues.

Mais notre collaboration avec Aimsun ne s'arrête pas là. En effet, le logiciel est payant mais Aimsun nous a procuré des licences gratuites pour que nous puissions réaliser notre projet. Malheureusement, Yassine a rencontré un problème avec son PC en janvier ce qui l'a obligé à demander une nouvelle licence qui a un peu tardé à venir car le service client était très occupé. Cela nous a ralenti un peu notre avancée car à ce moment-là, notre interface fonctionnait uniquement sous linux et Yassine est le seul d'entre nous à avoir un PC équipé de ce système d'exploitation. Nous avons rencontré ce problème une deuxième fois car toutes nos licences expiraient le 30 mars, nous avons donc dû en demander de nouvelles et le service était toujours surchargé. Ces épisodes nous ont permis de réaliser la charge de travail importante que rencontrent les employés d'une entreprise à certaines périodes.

Nous en avons également fait l'expérience avec Annique, qui était très occupée à une période et ne pouvait donc plus répondre à nos questions. Nous saurons donc dans notre vie professionnelle future qu'il est important de prévoir des délais lorsqu'on travaille avec d'autres acteurs qui peuvent rencontrer une charge de travail trop élevée et donc être indisponibles pendant un certain temps. De manière plus générale, notre collaboration avec Aimsun nous a permis de rencontrer des acteurs du milieu des transports et d'avoir un contact avec le monde de l'entreprise. Nous tenons pour cela à renouveler nos remerciements à Aurore, Annique et toute l'entreprise Aimsun.

Conclusion

Travailler sur ce sujet nous a permis de développer beaucoup de connaissances dans plusieurs domaines, ce qui est intéressant dans le cadre de notre cursus académique qui allie entre autres mathématiques appliquées à l'industrie et informatique. De plus, ce sujet a été très motivant parce qu'il constitue un problème non résolu ; en effet, si l'on a déjà trouvé des solutions optimales pour une seule intersection gérée par des feux tricolores, dès que l'on veut coordonner plus de feux entre eux, il devient beaucoup plus complexe de trouver des solutions satisfaisantes.

L'aspect concret de ce problème industriel nous a bien motivés pour travailler sur ce projet. Les méthodes scientifiques sont avant tout appliquées pour résoudre un problème réel, qui relève du bien commun. Nous avons aussi pu bénéficier de la présence d'Aimsun à nos côtés qui est une entreprise spécialisée dans ce domaine, pour nous guider et nous aider tout au long de la réalisation de ce projet.

Par ailleurs, ce projet nous a appris beaucoup de choses aussi bien sur le plan académique que sur le plan humain. Travailler au sein d'un groupe tout au long de l'année n'a pu être qu'enrichissant et bénéfique pour nous tous. Nous avons pu développer notre sens de l'organisation, du dialogue et de la communication lors de nos réunions hebdomadaires, tout cela dans un cadre de respect qui nous facilitait beaucoup le travail et nous permettait de nous concentrer encore plus sur l'avancement de notre PSC.

L'occasion de travailler à sept sur un projet pendant toute une année était une expérience exceptionnelle, et différait beaucoup des autres projets que nous avions en parallèle au cours de l'année (seuls ou en binôme). L'enjeu était de taille, et nous en étions conscients, par conséquent, tous les membres du groupe avaient un sentiment d'engagement et d'implication pour le projet et faisaient de leur mieux afin de rendre le meilleur des résultats en fin d'année.

Vu la diversité de nos profils et de nos provenances, nous avons pu nous compléter les uns les autres, et profiter des points de force de chacun de nous. Le travail en groupe nous a permis d'apprendre à écouter autrui, à admettre nos erreurs et accepter d'autres points de vue. Nous avons vraiment senti la différence avec le système académique pur des classes préparatoires, l'Ecole polytechnique offrant l'occasion de mener des projets scientifiques innovants.

Finalement, nous tenons à remercier l'entreprise Aimsun qui nous permet d'utiliser son logiciel sans lequel notre PSC ne pourrait exister. Nous remercions particulièrement Mme Aurore Remy, Mme Annique Lenorzer et M. Josep Perarnau pour leur aide précieuse et leur support tout au long de cette année. Nous remercions également notre coordinatrice Mme Flore Nabet pour son soutien et ses conseils qui nous ont fort aidé pendant notre travail.

Bibliographie

[1] Carolina Osorio, Michel Bierlaire (2013) A Simulation-Based Optimization Framework for Urban Transportation Problems. Operations Research 61(6) :1333-1345. http ://dx.doi.org/10.1287/opre.2013.1226

[2] Conn AR, Scheinberg K, Vicente LN (2009a) Global convergence of gen- eral derivative-free trust-region algorithms to first- and second-order critical points. SIAM J. Optim. 20(1) :387–415.

[3] Conn AR, Scheinberg K, Vicente LN (2008b) Geometry of sample sets in derivative-free optimization : Polynomial regression and underde- termined interpolation. IMA J. Numerical Anal. 28 :721–748.

[4] M.J.D Powell, 1994, A direct Search Optimization method that models the objective and constraint functions by linar interpolation

[5] Carolina Osorio, Linsen Chong (2015) A Computationally Efficient Simulation-Based Optimization Algorithm for Large-Scale Urban Transportation Problems. Transportation Science 49(3) :623-636. http ://dx.doi.org/10.1287/trsc.2014.0550