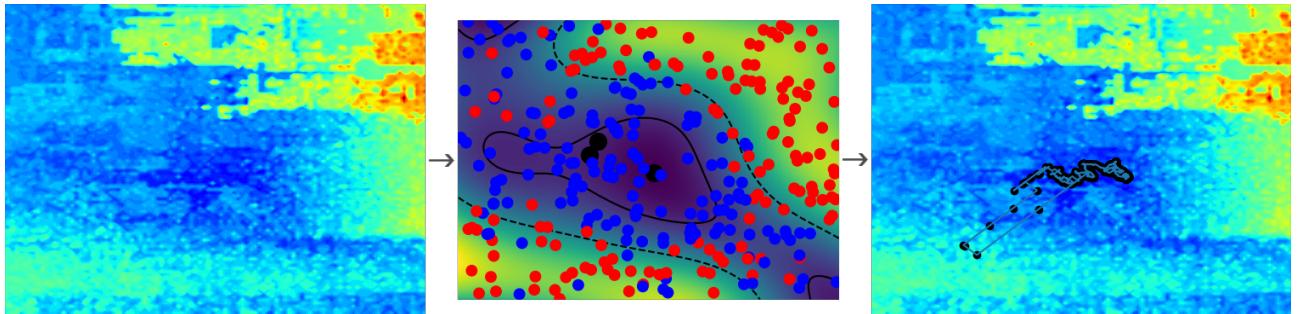


Modélisation et prédition du trafic routier

24 Avril 2017



Lucas BROUX, Lu LIN, Louis GUO, Elia MAZZONI, Jean-Baptiste MEMBRADO, Benjamin PETIT
(lucas.broux@polytechnique.edu)

Résumé : En partenariat avec *Transport Simulation Systems* (TSS), entreprise produisant un logiciel de simulation de trafic routier, nous cherchons à automatiser la calibration des paramètres d'un modèle. Pour cela, nous réalisons d'abord une interface Python qui nous permet d'introduire des fonctions de perte à minimiser pour répondre au problème. Nous constatons l'inefficacité des algorithmes classiques d'optimisation - Nelder-mead, BFGS, ... - ; une méthode s'avérant plus précise est le SPSA (*Simultaneous perturbation stochastic approximation*), mais la sensibilité au point de départ nous contraint à réaliser un pré-traitement. Nous introduisons pour ce faire une solution génétique issue des idées du *machine learning* pour rechercher des "zones d'intérêt" dans lesquelles nous choisissons notre point de départ pour le SPSA, ce qui nous fournit des résultats plus précis en limitant le nombre d'appels à la fonction de perte.



MAP/INF 01 C

MODÉLISATION ET PRÉDICTION DU

TRAFFIC ROUTIER

Projet Scientifique Collectif
Rapport Final

Pour le 21 Avril 2017

Lucas BROUX, Louis GUO, Elia MAZZONI
Jean-Baptiste MEMBRADO, Lu LIN, Benjamin PETIT



Table des matières

I Présentation détaillée du problème	5
1 Le logiciel Aimsun et l'entreprise TSS	5
2 Modélisation du problème et cadre de l'étude	5
II Techniques d'interfaçage développées et aspects techniques	8
1 Scripting Aimsun et architecture de l'API	8
2 Problèmes techniques et utilisation du modèle client-serveur	10
3 Problématique du temps de calcul et solutions expérimentales	11
4 Exploitation des données des bases SQLite	12
III Comparaison des données : former une fonction de perte	14
1 Introduction au problème : vers une boîte noire	14
2 Expérimentation et analyse des résultats	14
IV Analyse de méthodes d'optimisation	19
1 Problématique et Modélisation	19
2 Algorithmes étudiés en première phase	19
3 Algorithme SPSA	22
V Recherche de zones d'intérêt	26
1 Introduction à nos travaux	26
2 Organisation du code	26
3 Résultats expérimentaux de la méthode proposée	33
4 Conclusion scientifique	34
VI Bilan managérial et retour d'expérience	36

Remerciements

Nous souhaitons adresser nos remerciements les plus sincères aux personnes nous ayant aidé et accompagné dans la réalisation de ce projet.

En premier lieu, nous remercions Mme Aurore Rémy, directrice adjointe du groupe TSS. Tutrice du projet, elle a tout au long de l'année été très à l'écoute de nos demandes et nos interrogations, servant ainsi de relais entre le groupe et l'entreprise.

Nous remercions également Mme Annique Lenorzer, spécialiste de la modélisation du trafic routier au sein de TSS pour l'apport scientifique et technique dont elle a fait profiter le groupe.

Nous remercions en outre MM. Jordi Casas et Josep Perarnau, ingénieurs de TSS nous ayant partagé leurs connaissances scientifiques, contribuant ainsi à l'avancement du projet.

Nous remercions de surcroît nos coordinateurs pour ce projet, Mme Flore Nabet et M. Frank Nielsen, pour leurs précieux conseils.

Enfin, nous remercions le Lieutenant de Vaisseau Sere, référent du groupe, pour ses conseils organisationnels et managériaux.

Introduction

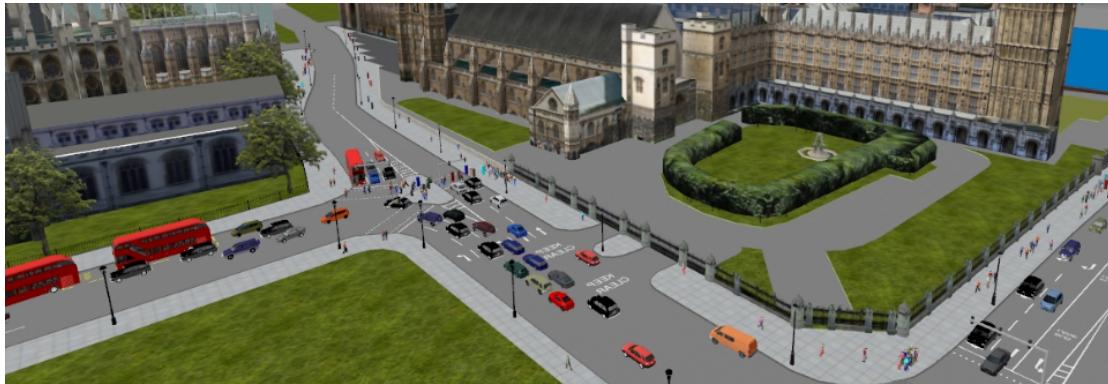


FIGURE 1 – Visualisation d'une simulation du trafic londonien par le logiciel Aimsun

Dans le monde entier, l'explosion du nombre de véhicules automobiles sur les routes depuis les dernières décennies induit de grandes problématiques de gestion et de prévention des congestions routières, sources d'accidents et de pollution. Afin de mieux appréhender ces problématiques, les grandes villes mondiales font de plus en plus appel à des solutions de modélisation. L'une d'entre elles, développée par l'entreprise Transport Simulation Systems (TSS), avec laquelle nous avons noué un partenariat, est le logiciel Aimsun, employé par les municipalités de certaines des plus grandes métropoles (Paris, New York, Singapour...).

Aimsun permet de simuler des situations de trafic routier, et d'en déduire une configuration optimale des infrastructures de transport d'une ville. Cependant, la modélisation mathématique nécessite que l'utilisateur entre manuellement des paramètres dépendant du réseau routier modélisé et de la zone géographique (temps de réaction, distance de visibilité, matrices de demande de trafic...) qu'il est nécessaire de calibrer pour que le modèle renvoie des résultats cohérents avec les données réelles mesurées à partir de capteurs. Cette phase de calibration est actuellement l'une des plus longues et coûteuses pour TSS, car elle est réalisée manuellement par les ingénieurs trafic, qui s'appuient autant sur des indicateurs quantitatifs, que sur leur longue expérience du trafic routier. Notre projet est d'étudier la possibilité d'une automatisation de cette phase de travail.

Pour cela, notre idée est de former une “fonction de perte”, quantifiant les différences et similitudes entre le trafic simulé et le trafic réel mesuré, puis de minimiser celle-ci à l'aide de techniques d'optimisation adaptées afin d'obtenir le jeu de paramètres réalisant la plus grande similitude. Cela nécessite de savoir réaliser une interface permettant, grâce à des scripts Python, d'utiliser les fonctionnalités d'Aimsun, et d'étudier certaines techniques d'optimisation non-conventionnelles adaptées aux contraintes de ce problème (important temps de calcul de la fonction cible, faible régularité de celle-ci et grand nombre de paramètres). En particulier, nous avons souhaité étudier certaines méthodes issues du *machine learning*, qui nous ont paru adaptées à ce type d'optimisation “lourde”.

L'important travail de combinaison des différents outils développés, mené par l'ensemble du groupe à la fin de l'étude, nous mène à proposer à TSS un outil d'optimisation que nous considérons comme un premier pas vers une méthode de calibration automatique.

Dans ce rapport final, nous résumons le travail fourni par l'ensemble du groupe, faisons la synthèse des réussites et difficultés rencontrées, et analysons les résultats obtenus.

Première partie

Présentation détaillée du problème

1 Le logiciel Aimsun et l'entreprise TSS



Fondée en 1997 à Barcelone, l'entreprise TSS est aujourd'hui un leader mondial dans le domaine de la modélisation de trafic routier. Comptant aujourd'hui 6 bureaux dans le monde, à Barcelone, Paris, New York, Londres, Portland et Sydney, elle emploie une centaine de personnes. Spécialisée dans l'élaboration des programmes et algorithmes de simulation, elle offre également ses services de conseil en gestion de trafic aux grandes municipalités et institutions chargées de réguler et d'adapter les infrastructures de transport.

Les solutions produites par TSS ont fait leurs preuves depuis plusieurs années, et sont aujourd'hui vendues dans près de 75 pays, pour un total d'environ 4 000 licences. S'adressant presque exclusivement à une clientèle institutionnelle, ces solutions ont été développées en étroite collaboration avec le milieu de la recherche.

TSS développe et commercialise en parallèle deux solutions complémentaires, utilisées notamment par les villes de Singapour, Paris, Montréal et San Diego :

- Aimsun est son produit historique. Il permet de simuler aux échelles microscopique, mésoscopique et macroscopique des situations de trafic routier très diverses, dans le but d'optimiser les infrastructures de transport d'une ville. Ce logiciel est extrêmement paramétrable ; il prend notamment en considération le volume des réseaux de transports en commun, des critères de sécurité et même les normes environnementales. Plusieurs visualisations sont disponibles : une permet de modifier le réseau en lui-même (en ajoutant des routes, ou encore en modifiant leur tracé), une autre de modifier les paramètres de la simulation, et une dernière de visualiser le trafic simulé en 2D ou 3D. C'est pour cet outil que nous souhaitons participer au développement d'un outil de calibration automatique.
- Aimsun Online, plus récent, est une solution de gestion de trafic en temps réel. Utilisé dans des centres de contrôle du trafic, il permet de prédire l'évolution du trafic sur une durée courte à moyenne, et de gérer en conséquence la signalisation de façon dynamique. À l'inverse d'Aimsun, plus souvent utilisé dans le cadre d'études, Aimsun Online est prévu pour un fonctionnement quotidien.

2 Modélisation du problème et cadre de l'étude

Comme indiqué dans l'introduction, la problématique que nous nous sommes proposés d'étudier pour ce projet est la suivante : comment déterminer, connaissant un jeu de données réelles, des paramètres qui produisent une simulation Aimsun qui se rapproche de la réalité ? Répondre à cette

question est très délicat, c'est pourquoi nous l'avons précisée et simplifiée afin de pouvoir réaliser des expériences informatiques qui puissent nous renvoyer des résultats satisfaisants avec les moyens dont nous disposons.

Notamment, nous avons été amenés à réfléchir aux questions suivantes :

- **Quels types de données considérer ?** Il a été mentionné dès le début du projet de comparer les résultats de simulations informatiques à des données réelles. Toutefois, pour des raisons de simplicité tant administratives - pour obtenir ces données - que techniques - pour comparer ces données à des résultats de simulations -, nous avons choisi de n'utiliser que des données générées par le logiciel Aimsun. En particulier, nous avons généré avec le logiciel une base de donnée de référence avec des paramètres connus, qu'il s'agit de retrouver en appliquant des méthodes d'optimisation. Nous sommes conscients de la difficulté de passer de cette approche simplifiée à l'optimisation sur des données réelles. Cependant, ce premier travail, en particulier sur les méthodes d'optimisation et les fonctions de perte, aura toute son importance dans le développement d'un outil de calibration automatique.
- Nous préciserons par la suite la structure des données utilisées et les moyens techniques pour les interpréter.
- **Quel modèle Aimsun choisir ?** L'entreprise TSS nous a confié l'étude d'une petite zone commerciale située aux alentours de Barcelone. Sur nos ordinateurs, chaque simulation demande un temps d'exécution d'environ 4 secondes, ce qui impose par ailleurs une contrainte forte sur les algorithmes d'optimisation à implémenter : le nombre d'appels doit être réduit.

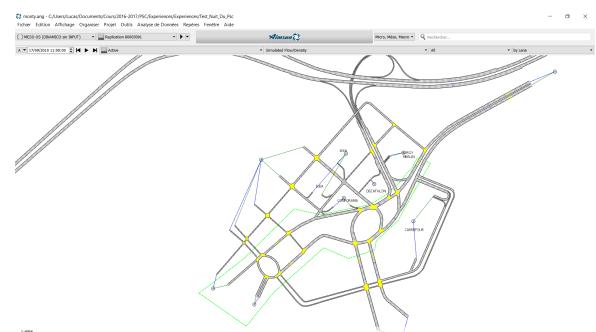


FIGURE 2 – Vue satellite et modèle Aimsun du modèle étudié

- **Quels paramètres calibrer ?** Les conseils de TSS nous ont permis de fixer les paramètres à considérer. Nous avons ainsi choisi d'étudier :
 - Le temps de réaction moyen (*reaction time*) des véhicules du modèle. Nous le recherchons entre 0.9 s et 1.4 s et la valeur de référence usuellement choisie est 1.2 s.
 - La variation du temps de réaction (*reaction time variation*) des véhicules pour chaque section du modèle. Nous le recherchons entre 0.8 s et 1.2 s, et la valeur de référence usuellement choisie est 1.0 s.
 - Le temps de réaction aux feux (*reaction time at traffic light*) moyen pour les véhicules du modèle. Nous le recherchons entre 0.8 s et 1.2 s, et la valeur de référence usuelle est 1.0 s.
 - La distance d'anticipation (*look ahead distance*) pour les véhicules du modèle. Nous le recherchons entre 0 m et 500 m, et la valeur de référence usuelle est 200 m.

Dû au grand nombre de sections dans le modèle considéré, cela nous permet de nous concentrer sur un nombre de paramètre allant de 1 jusqu'à 100. En pratique, nous avons fait le choix de nous concentrer sur un nombre réduit de paramètres (entre 2 et 4) : non seulement cela permet des visualisations graphiques plus aisées, mais grâce à un système de stockage et d'interpolation des données, cela permet aussi de multiplier les essais de nos méthodes. Nous pensons que la méthode est intéressante pour un nombre plus élevée de paramètres. Cependant, la puissance de calcul réduite de nos ordinateurs ne nous a pas encouragés à poursuivre dans cette voie, préférant effectuer un nombre plus important d'expériences sur un nombre réduit de paramètres afin d'éprouver autant que possible la robustesse de la méthode proposée.

Ainsi, la problématique se trouve précisée : nous devons appliquer des algorithmes d'optimisation pour des paramètres donnés et sur un modèle concret afin de retrouver des valeurs de référence connues.

Lors de notre réflexion quant à cette problématique, nous avons dégagé 4 sous-problèmes à traiter :

- Réaliser une **interface** de scripting permettant d'appeler les fonctions de modification des paramètres et de simulation d'Aimsun depuis Python.
- Proposer et implémenter différentes fonctions de perte pour **comparer** les bases de données produites par Aimsun.
- Proposer, étudier et analyser différents algorithmes d'**optimisation** pour obtenir les paramètres voulus.
- Proposer, étudier et analyser différents algorithmes issus du **machine learning** pour obtenir des "zones d'intérêt" dans lesquels chercher les paramètres, en pré-traitement de l'optimisation.

Nous présentons par la suite, et dans cet ordre, les solutions que nous avons apportées pour répondre à ces sous-problèmes, ainsi que les résultats que nous obtenons.

Deuxième partie

Techniques d’interfaçage développées et aspects techniques

Dès le début du projet, un point majeur a été de mettre en place une manière de faire communiquer nos scripts Python avec le logiciel Aimsun, afin de produire des simulations, et d’en exploiter les résultats.

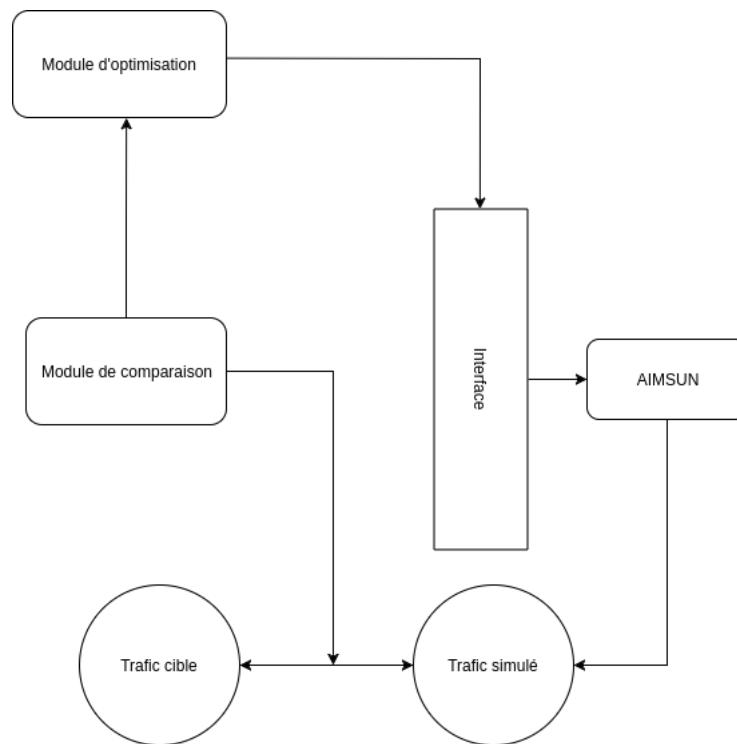


FIGURE 3 – Place de la partie interfaçage au sein de notre projet

1 Scripting Aimsun et architecture de l’API

Aimsun, la plupart du temps utilisé par l’intermédiaire de son interface graphique utilisateur, est également doté de fonctionnalités très avancées de scripting, qui peuvent s’exécuter au choix au sein de l’interface graphique, ou dans la version console du logiciel. Ces fonctionnalités permettent, entre autres, d’effectuer des simulations, d’exporter des données et de modifier les paramètres du modèle. Ceci peut sembler assez simple au premier abord. Néanmoins, pour comprendre le fonctionnement de l’API Python Aimsun, il est nécessaire de comprendre l’agencement global des différents objets dans le logiciel lui-même [1]. En effet, Aimsun est muni d’une structure largement orientée objet, et aussi bien les différentes portions de route, que les véhicules et même le simulateur lui-même sont des objets héritant de diverses classes dans le logiciel, ce afin de leur donner la plus grande versatilité possible.

1.1 Le système

L’objet de type *GKSystem* est en quelque sorte le “conteneur” du kernel Aimsun. Il permet de créer des instances de la plupart des objets que nous verrons plus bas (modèle, simulateur...).

1.2 Le modèle

Le modèle, issu de la classe *GKModel*, est, dans notre cas, l’objet principal auquel nous ferons appel. Il nous permet, via le catalogue (classe *GKCatalog*), d’accéder à tous les autres objets des autres classes présents dans le réseau modélisé. Il est initialisé en début de script, juste après le chargement du réseau.

1.3 L’expérience

L’expérience, du type *GKExperiment*, est un paramétrage de la simulation réalisée. Elle contient les divers paramètres de demande, ainsi que les paramètres proprement dits du réseau étudié (temps de réaction des conducteurs...).

1.4 La réPLICATION

La réPLICATION, du type *GKReplication*, est une instance de l’expérience, qui sera simulée par le simulateur. Elle est obtenue à partir du modèle.

1.5 Les classes graphiques du réseau

Par l’intermédiaire du catalogue, nous accédons à tous les objets héritant du type *GKSection*, qui nous permettent d’interagir avec les différents paramètres de la partie physique du modèle, section de route par section de route.

1.6 Les attributs

Chaque objet utilisé par le logiciel Aimsun possède différents attributs (*ie* paramètres), héritant de la classe *GKAttribute*. En utilisant sur un objet les fonctions *GetAttributeByID* et *SetAttributeByID*, il est possible d’obtenir leur valeur et/ou de la modifier.

1.7 Le plugin et le simulateur

A partir du système, il est possible de créer un objet de type *GKSimulator*, obtenu lui-même à partir d’un objet de type *GKPlugin*, qui représente le plugin choisi pour réaliser la simulation (micro, méso ou macro). Le cadre de notre étude se limite aux simulations du module mésoscopique ; nous n’utiliserons que le plugin “*AMesoPlugin*”. En ajoutant au simulateur des tâches sous la forme d’un objet de type *GKSimulationTask*, il est possible d’effectuer des simulations en appelant la méthode *simulate*.

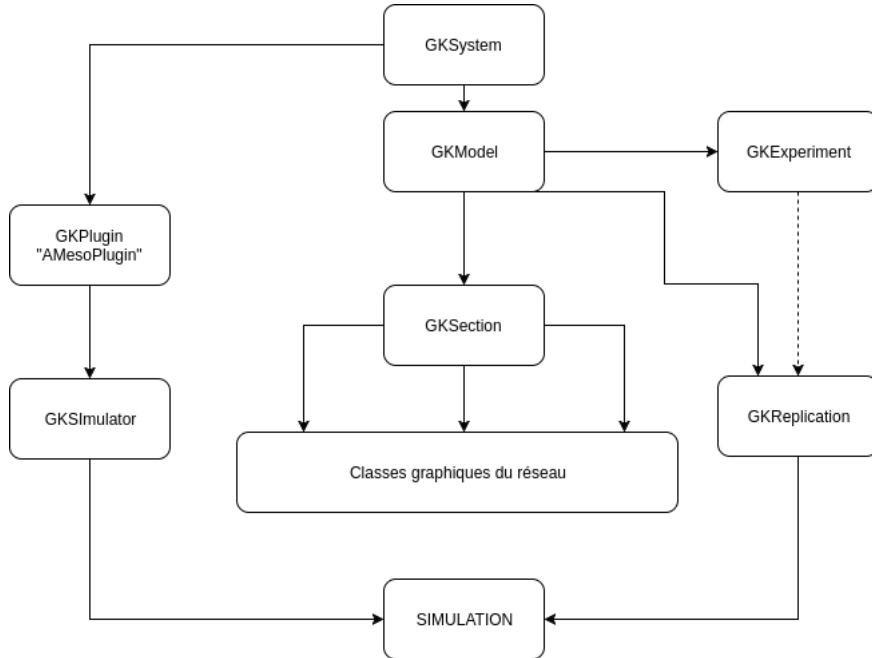


FIGURE 4 – Diagramme simplifié du fonctionnement de l'API Python Aimsun

2 Problèmes techniques et utilisation du modèle client-serveur

La version de Python présente au sein d'Aimsun est certes munie de tous les modules nécessaires pour utiliser le logiciel, mais elle est complètement “nue” ; elle est dépourvue de tous les modules mathématiques (NumPy, Scipy) et graphiques (Matplotlib). Il nous est par ailleurs difficilement possible de la modifier sans modifier les fichiers internes de Aimsun. De surcroît, il nous est complètement impossible de nous passer de l'interpréteur interne d'Aimsun, car il est le seul à disposer des modules de simulation. Aussi, nos tentatives d'exécuter des simulations depuis un interpréteur Python externe se sont révélées vaines.

Pour pallier cette impossibilité, la solution que nous avons choisi de retenir est basée sur le modèle client-serveur [2]. Un script serveur est lancé au moyen de l'interpréteur présent dans Aimsun (sous sa forme console), et exécute les ordres qui lui sont envoyés par un autre script lancé dans un interpréteur Python externe, sur le modèle client-serveur. La communication entre le module client et le module serveur utilise ici le protocole TCP/IP, habituellement utilisé pour communiquer entre plusieurs ordinateurs. Nous l'utilisons ici, grâce au module *socket* de Python [2], pour communiquer entre plusieurs processus au sein d'un même ordinateur. Aimsun est donc piloté de l'extérieur comme une boîte noire via le protocole TCP/IP.

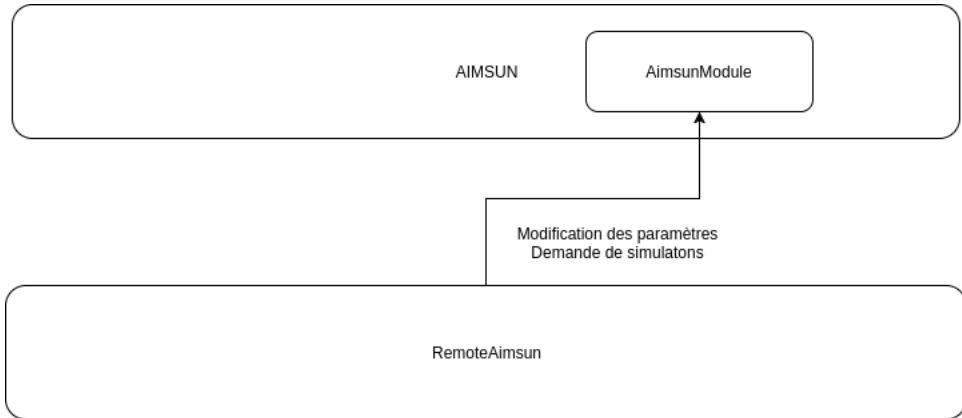


FIGURE 5 – Schéma simplifié du procédé d'interfaçage

3 Problématique du temps de calcul et solutions expérimentales

L'un des problèmes clés auxquels nous avons été confrontés lors de ce projet est celui du temps de calcul. En effet, dans des cas classiques d'utilisation d'algorithmes d'optimisation, il est possible d'utiliser simplement une méthode de gradient (ou autre en fonction des caractéristiques de la fonction objectif) pour optimiser une fonction d'une centaine de paramètres sans trop se soucier du nombre d'appels qui y seront faits. Malheureusement ici, le calcul d'une simulation Aimsun peut prendre plusieurs secondes, jusqu'à plusieurs minutes dans le cas de très grands réseaux.

Aussi, il est inenvisageable d'effectuer, lors de notre processus d'optimisation, plusieurs millions d'appels à une fonction dont le temps de calcul est d'environ une seconde. Par ailleurs, le simple fait de devoir attendre une heure à chaque essai de nos techniques dans la phase d'expérimentation ralentirait grandement notre travail.

Nous avons donc pris le parti d'utiliser une technique simple d'interpolation des résultats expérimentaux. Une fois les fonctions de perte fixées, nous générerons, pour différentes valeurs de 2 à 4 paramètres, des tables de données au format CSV contenant les valeurs de ces fonctions. Ensuite, grâce à un procédé d'interpolation linéaire, nous pouvons obtenir très rapidement des valeurs approchées des fonctions de perte pour d'autres jeux de paramètres. Cela nous permet, dans notre phase de comparaison des différentes méthodes d'optimisation, d'éviter de faire appel à une simulation Aimsun pour le calcul de chaque valeur demandée par l'algorithme.

Fichier utilisé	Ordre de grandeur du temps de calcul d'une simulation
test_meso_parameters.ang	0.5 s
monty.ang	4 s
victoria.ang	60 s

TABLE 1 – Temps moyen de calcul d'une simulation sur un ordinateur portable récent

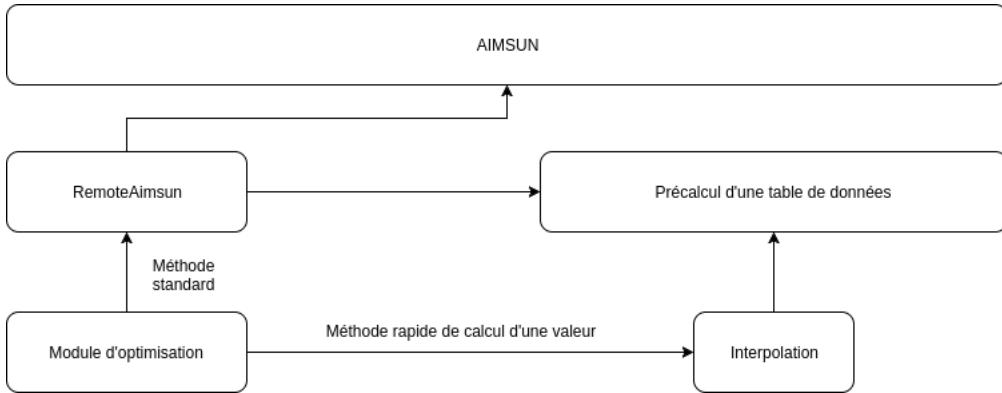


FIGURE 6 – Diagramme des différentes méthodes de calcul

4 Exploitation des données des bases SQLite

4.1 Récupération de la sortie de Aimsun : bases de données SQLite

Afin de visualiser la base de données produite par la simulation, nous utilisons le logiciel DB SQLite qui permet d'ouvrir les fichiers de format SQLite. La base de données est répartie en tables qui correspondent aux différents tronçons de la route modélisée.

De nombreux attributs, qui correspondent à l'ensemble des résultats de la simulation, y sont présents. Ceci n'étant pas documenté, nos échanges avec les ingénieurs de TSS nous ont progressivement permis de leur donner une signification [3]. Remarquons que l'enchaînement des lignes ne correspond pas à l'avancement temporel de la simulation (chaque ligne n'est pas un nouveau pas de la discrétisation temporelle). Toute cette information est déjà indiquée dans les attributs ; notons les plus importants qui nous permettront d'extraire efficacement les données :

- la discrétisation spatiale est mentionnée par l'attribut *oid* qui permet de se situer spatialement (chaque objet modélisé est représenté par un identifiant *oid*) ;
- la discrétisation temporelle est contenue dans *ent*.

Les attributs principaux significants que nous allons comparer entre deux simulations sont :

- *count / input_count* : comptage à la sortie et à l'entrée de la section (nombre de véhicules) ;
- *flow / input_flow* : indicateur classique du débit à la sortie et à l'entrée des sections (veh/h) ;
- *density* : densité (veh/km) ;
- *speed* : vitesse des véhicules (km/h) ;
- *qmean* : longueur moyenne des bouchons (km) ;
- *ttime* : le temps de trajet (secondes).

La liste n'est pas exhaustive afin de ne pas dérouter le lecteur. Remarquons que souvent pour des cases sans cohérence (un rond-point n'a par exemple pas de vitesse), la valeur attribuée est de -1.

4.2 Réflexion sur les attributs récupérés

Il convient de se demander quels sont les degrés de liberté parmi les attributs, à savoir combien d'attributs sont nécessaires pour décrire entièrement la sortie. Par exemple, la donnée de la longueur

Table : MELANE													
	did	oid	eid	sid	ent	lane	count	count_D	flow	flow_D	input_count		
2	518431	111	NULL	0	1	2.0	0.0	-1.0	0.0	-1.0	0.0		
3	518431	111	NULL	0	1	3.0	0.0	-1.0	0.0	-1.0	0.0		
4	518431	113	NULL	0	1	1.0	0.0	-1.0	0.0	-1.0	0.0		
5	518431	113	NULL	0	1	2.0	0.0	-1.0	0.0	-1.0	0.0		
6	518431	113	NULL	0	1	3.0	0.0	-1.0	0.0	-1.0	0.0		
7	518431	113	NULL	0	1	4.0	0.0	-1.0	0.0	-1.0	0.0		
8	518431	114	NULL	0	1	1.0	0.0	-1.0	0.0	-1.0	0.0		
9	518431	114	NULL	0	1	2.0	0.0	-1.0	0.0	-1.0	0.0		
10	518431	115	NULL	0	1	1.0	0.0	-1.0	0.0	-1.0	0.0		
11	518431	115	NULL	0	1	2.0	0.0	-1.0	0.0	-1.0	0.0		
12	518431	115	NULL	0	1	3.0	0.0	-1.0	0.0	-1.0	0.0		
13	518431	116	NULL	0	1	1.0	0.0	-1.0	0.0	-1.0	0.0		
14	518431	116	NULL	0	1	2.0	0.0	-1.0	0.0	-1.0	0.0		
15	518431	117	NULL	0	1	1.0	0.0	-1.0	0.0	-1.0	0.0		
16	518431	117	NULL	0	1	2.0	0.0	-1.0	0.0	-1.0	0.0		
17	518431	121	NULL	0	1	1.0	0.0	-1.0	0.0	-1.0	0.0		
18	518431	121	NULL	0	1	2.0	0.0	-1.0	0.0	-1.0	0.0		
19	518431	121	NULL	0	1	3.0	0.0	-1.0	0.0	-1.0	0.0		

FIGURE 7 – Vision de la base de données SQLite du tableau MELANE

de la portion de route considérée et du *input_count* et du *count* nous permet de recalculer la densité. Une analyse plus poussée des données disponibles des attributs suffisants pour décrire entièrement l'état serait possible mais pas indispensable. Nous avons choisi de ne pas rentrer à ce niveau de détails car cela n'était pas nécessaire, la redondance de l'information peut être contrôlée par une pondération de chaque attribut.

4.3 Récupération et traitements des données simulées sour Python

Le module *sqlite3* de la bibliothèque standard de Python permet d'ouvrir une connexion entre Python et SQL et à l'aide d'un objet curseur ; il est possible de communiquer à travers des requêtes sous langage SQL dans la console Python grâce à la commande *cursor.execute* [4]. Ainsi, par des requêtes simples, nous sommes capables de sélectionner la liste des densités et du flux et de les récupérer sous forme de *array* sous Python pour ensuite les exploiter plus facilement.

Fixons nous un réseau routier modélisé sur Aimsun. L'appel à Aimsun calcule à partir de paramètres rentrés par l'utilisateur la simulation et retourne la base de données sus-mentionnée. Un outil primordial qui servira pour tous les algorithmes est donc la comparaison de la base de données de référence *A* et une base de données quelconques en sortie de la fonction Aimsun. Pour cela il est nécessaire de comparer la structure des deux bases SQLite. Deux aspects à considérer sont la taille de la base de données sortie (en nombre de lignes) et l'ordre d'apparition de ces lignes. Nous avons remarqué que ces deux caractéristiques restent invariantes pour toutes les bases considérées pour un même réseau routier. Après nous en être assuré auprès de TSS, nous avons mis en place une structure d'extraction des données, utilisée avec les différents paramètres précédemment mentionnés.

Troisième partie

Comparaison des données : former une fonction de perte

1 Introduction au problème : vers une boîte noire

La première étape vers notre processus d'optimisation est donc de proposer une fonctions de perte, représentant une sorte de “distance” entre simulation et “réalité”, pertinente. Celle-ci doit en particulier satisfaire certaines propriétés afin de maximiser sa compatibilité avec les algorithmes d'optimisation que nous souhaitons appliquer. Dans l'idéal, elle serait suffisamment régulièr(e) (ou, au moins, ne pas présenter un trop grand nombre d'irrégularités), possèderait un unique minimum global si possible suffisamment “creusé”, et serait suffisamment monotone autour de celui-ci, sans trop osciller lorsque l'on fait varier les paramètres. Nous avons donc expérimenté différentes familles de fonctions de perte, qui présentent chacune leurs avantages et leurs inconvénients. Dans notre cas, nous les appliquons sur six colonnes de la table MESECT - qui contient les données des simulations section par section -, qui sont *flow*, *ttime*, *density*, *count*, *dtime*, *speed*. Présentons donc dans un premier temps ces fonctions.

- **Normes L^1 et L^2** : Nous appliquons les formules de la norme L^1 et L^2 aux colonnes considérées.
- **Norme GEH** : Un indicateur couramment utilisé en analyse du trafic routier est le GEH (du nom de son inventeur, Geoffrey E. Havers) [5], dont la justification est purement empirique. En divisant la norme L^1 par la racine de la somme des normes L^1 , on obtient une normalisation qui minimise les différences d'échelle entre les différentes grandeurs.

$$GEH = \frac{|X - Y|}{\sqrt{(|X| + |Y|)}}$$

Initialement pensé pour être appliqué au flux de véhicules (nombre de voitures traversant la section considérée par unité de temps), nous proposons une amélioration qui consiste à utiliser le même type d'indicateur sur d'autres types de statistiques (notamment les vitesses moyennes, temps de trajet, etc...) afin de construire un indicateur agrégeant plusieurs colonnes de la table SQLite.

- **Coefficient de régression linéaire** : Nous considérons la version simulée d'une colonne ainsi que la colonne de référence, et nous formons le nuage de points formé, en abscisses, par les valeurs de la colonne de référence, et en ordonnées par celles de la colonne simulée. Nous en calculons le coefficient de régression des moindres carrés qui est retranché à 1 pour former une norme.

2 Expérimentation et analyse des résultats

Nous avons appliqué et représenté ces fonctions de perte pour différents choix de deux paramètres parmi ceux retenus. Pour chaque expérience, nous avons conservé les résultats de 10000 simulations (sur une grilles 100*100) dans un fichier .csv à réutiliser pour constituer via interpolation des fonctions de perte générale. Le modèle choisi est monty.ang, modèle principal étudié au cours de ce projet ; et les deux paramètres choisis sont respectivement :

- Temps de réaction et distance d'anticipation.

- Temps de réaction et temps de réaction aux feux.
- Temps de réaction aux feux et distance d'anticipation.

Nous obtenons alors dans cet ordre les représentations suivantes pour les lignes de niveau des fonctions de perte, dont les valeurs ont été ramenées entre 0 et 1 par une transformation affine pour des raisons d'uniformité.

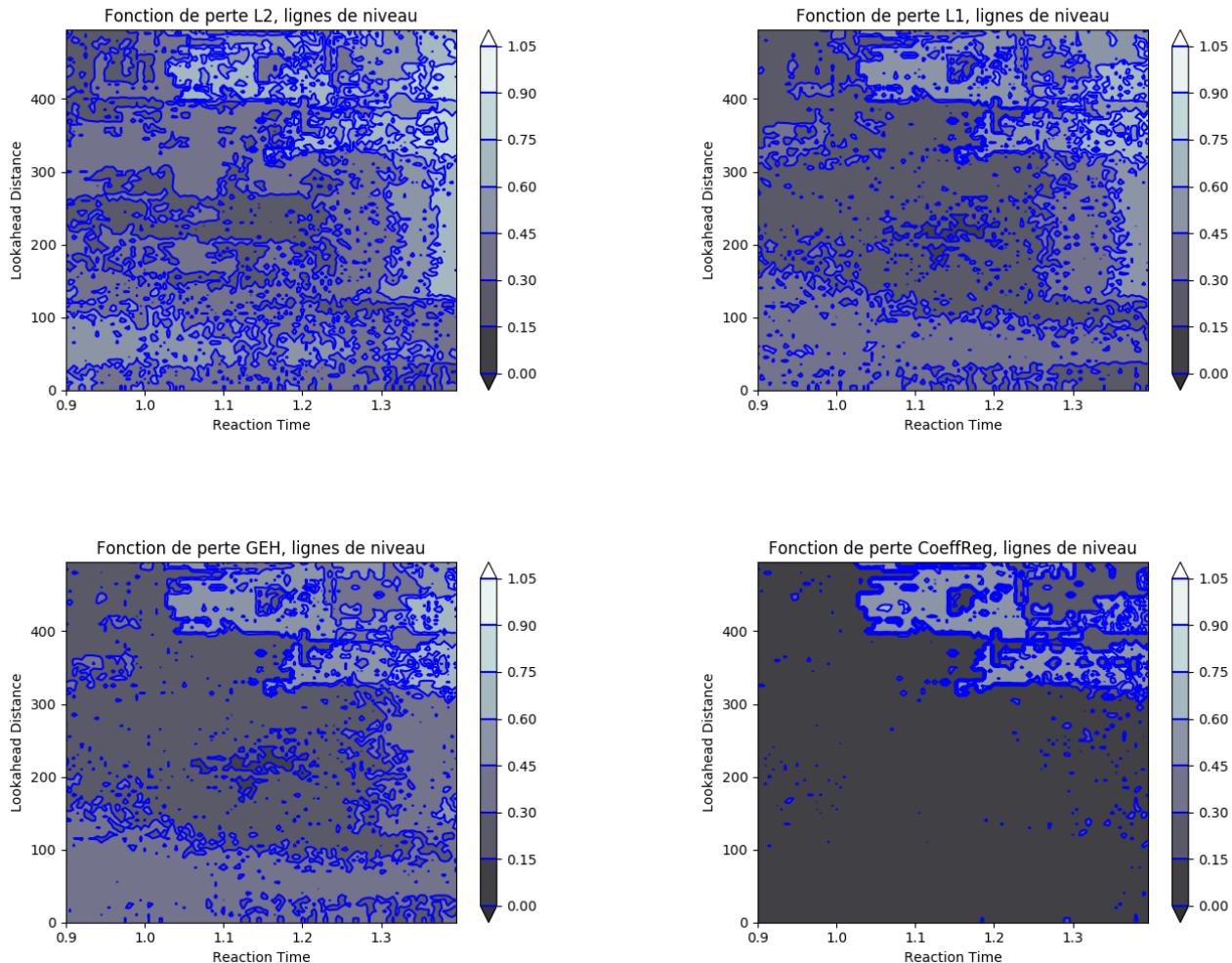


FIGURE 8 – Temps de réaction et distance d'anticipation

Sur ces premières figures, les paramètres recherchés sont 1.2 pour le temps de réaction et 200 pour la distance d'anticipation. Globalement, chacune des fonctions de perte étudiées prend ses valeurs les plus faibles au voisinage du point de référence : cela est d'autant plus visible pour la norme GEH. En revanche, la fonction de perte par coefficient de régression linéaire souffre d'un effet de vallée important qui la rend peu praticable. La distinction entre les trois autres est plus minime, mais nous remarquons que le creux du GEH est plus marqué et régulier.

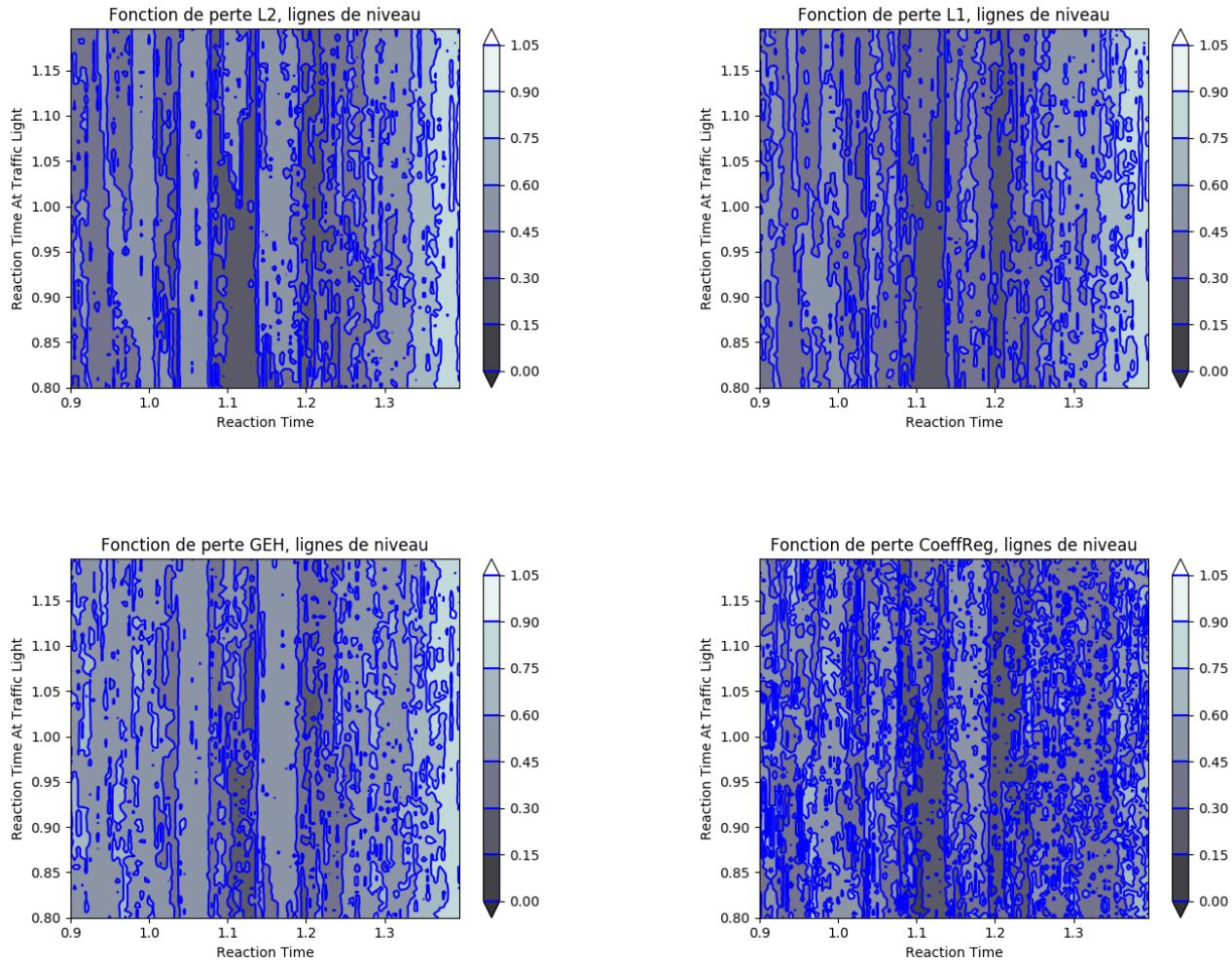


FIGURE 9 – Temps de réaction et temps de réaction aux feux

Sur ces figures, les paramètres recherchés sont 1.2 pour le temps de réaction et 1.0 pour le temps de réaction aux feux. Remarquons tout d'abord que le temps de réaction est bien plus influent que le temps de réaction aux feux : nous distinguons encore une zone de vallée mais celle-ci s'étend sur tout l'axe vertical. Ici, contrairement aux premières expériences, nous remarquons la présence de deux vallées principales, ce qui remet quelque peu en cause l'idée d'un unique minimum global. Il nous semble que pour cette expérience, la fonction de perte la plus pertinente est donc la fonction L^1 , pour laquelle les zones de faibles valeurs sont les plus marquées.

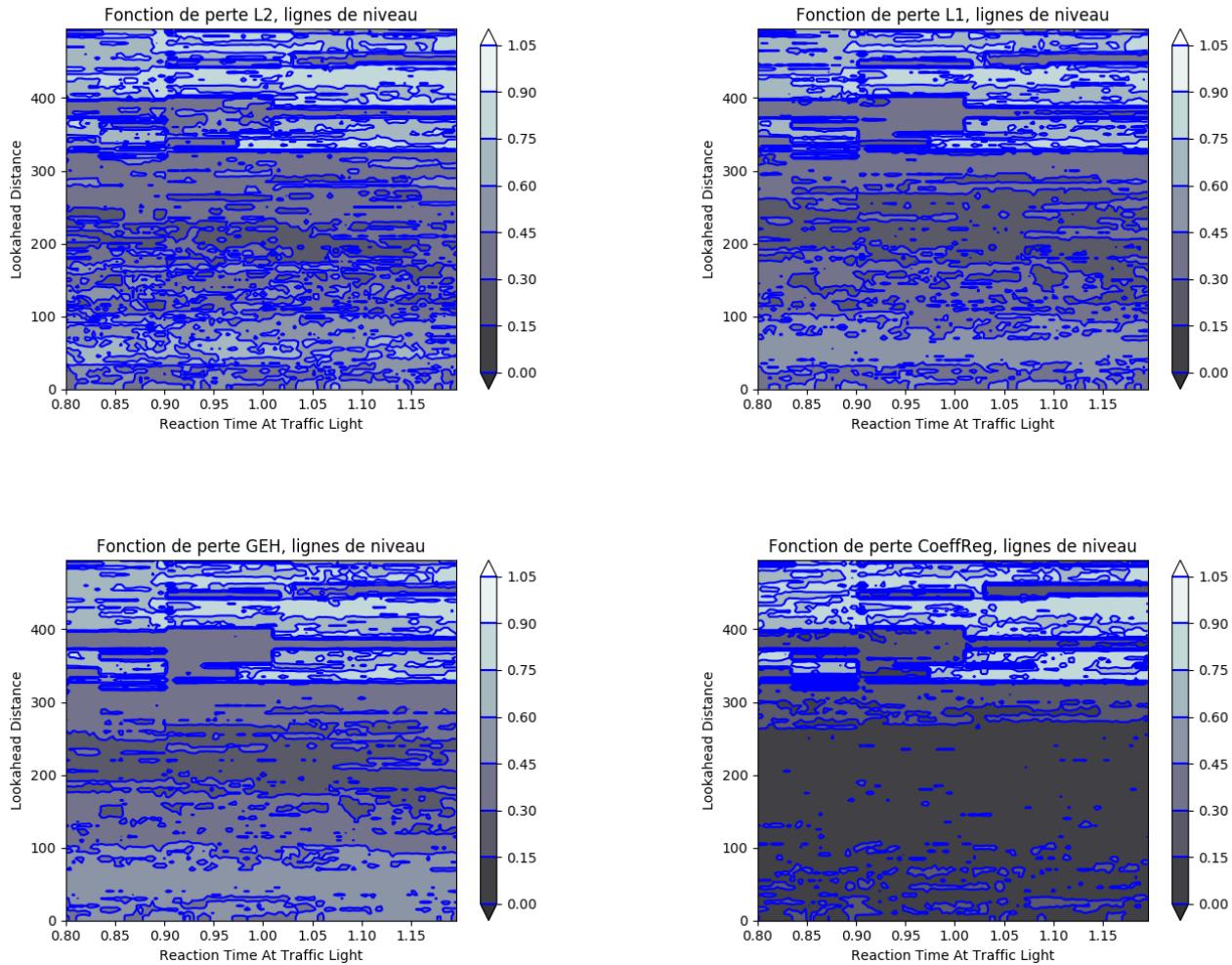


FIGURE 10 – Temps de réaction aux feux et distance d'anticipation

Sur ces figures, les paramètres recherchés sont 1.0 pour le temps de réaction aux feux et 200 pour la distance d'anticipation. Nous remarquons comme pour l'expérience précédente l'existence d'une vallée de minima horizontale qui confirme la faible influence du temps de réaction aux feux sur le modèle choisi. De même que dans l'expérience précédente, la fonction de perte que nous jugeons la plus pertinente pour cette expérience est la perte L^1 , qui fait encore une fois apparaître une claire zone de minima.

Ainsi, de cette étude, nous tirons les conclusions suivantes :

- Les paramètres du modèle ont des influences très variées sur la valeur des fonctions de perte, et donc a fortiori sur les simulations réalisées : ici, le temps de réaction aux feux se montre peu influent par rapport au temps de réaction et à la distance d'anticipation ; mais ces deux derniers paramètres partagent une influence similaire, comme le prouve la forme "en bol" des fonctions de perte correspondant à cette expérience.
- Sur cette étude, nous voyons que globalement toutes les fonctions de perte construites remplissent leur contrat de présenter une zone de minima aux abords des valeurs de référence. Nous voyons toutefois que cette zone est plus marquée pour certaines fonctions que pour d'autres.

— Ici, se distinguent en particulier la fonction de perte L^1 ainsi que le GEH, qui sont intrinsèquement très proches.

Pour la suite de nos expériences, nous avons donc choisi de ne considérer que les fonctions de perte L1.

Quatrième partie

Analyse de méthodes d'optimisation

1 Problématique et Modélisation

Reformulons notre problématique de recherche de paramètres en des termes plus mathématiques. Nous souhaitons déterminer des solutions approchées (une suffit) θ , d'une équation de la forme $f(\theta) = a$, où f est une fonction de \mathbb{R}^p dans \mathbb{R}^m . Nous nous intéressons au cas où f peut être évaluée explicitement, mais l'évaluation de f est très coûteuse. Il faut ainsi déterminer θ avec le moins d'évaluations possible de la fonction f . Ce problème se retrouve dans de nombreuses applications industrielles, où il est nécessaire d'optimiser un processus, mais où les expériences sont très lourdes et parfois onéreuses¹.

Dans le cadre de notre projet, f correspond aux simulations du logiciel Aimsun proposé par notre partenaire *TSS*, p est le nombre de paramètres du modèle et m le nombre de valeurs présentes dans le fichier SQL de sortie. Suite à notre travail de recherche et d'analyse de fonctions de perte, nous disposons de fonctions $g : \mathbb{R}^p \rightarrow \mathbb{R}$ de façon à ce que la composition $\tilde{f} = g \circ f$ ramène notre problème au cas de l'optimisation d'une fonction scalaire. (\tilde{f} sera notée f par la suite pour simplifier les notations). On cherche donc à minimiser la fonction $f : \mathbb{R}^p \rightarrow \mathbb{R}$ i.e. trouver un point minimum au sein d'un espace de dimension inférieure ou égale à p (plus petite dans le cas où certains paramètres sont corrélés). La fonction f n'étant pas explicite, le calcul d'un seul gradient numérique avec les méthodes classiques requiert $O(p)$ évaluations de f . Nous avons déjà constaté que cela n'est pas idéal au vu du temps de calcul d'une seule itération de la fonction et du nombre de paramètres que nous souhaitons optimiser à terme.

Dans une première phase du projet, nous avons étudié divers algorithmes déterministes, mais nous avons constaté leur inefficacité notamment en termes de nombre d'appels à la fonction. Nous nous sommes alors tournés vers un algorithme stochastique qui permet de réduire ce nombre d'appels, l'algorithme SPSA (*Simultaneous Perturbation Stochastic Approximation*).

2 Algorithmes étudiés en première phase

2.1 Organisation générale

Nous avons commencé notre travail de recherche en nous intéressant aux algorithmes les plus classiques, déjà implémentés dans le package *Scipy*. Les algorithmes alors considérés sont les suivants :

- Méthode de Nelder-Mead
- Méthode du Gradient Conjugué
- Méthode BFGS
- Méthode Dogleg

Leur principe ayant déjà été donné dans le rapport intermédiaire, nous ne reviendrons pas sur leur fonctionnement précis.

Toutes les méthodes étudiées lors de cette première phase du projet ont été étudiées avec [6].

1. Dans le cas d'un test de vaccins ou d'un médicament par exemple.

2.2 Méthode de Nelder-Mead

Au début du projet, nous n'avions pas les résultats expérimentaux sur les fonctions de pertes présentés dans la partie précédente. Toutefois nous savions, au vu de la complexité des modèles en jeu, qu'une hypothèse de différentiabilité de la fonction serait trop demander. La méthode Nelder-Mead a été la première à avoir été envisagée, puisqu'elle ne repose pas sur une telle hypothèse.

Cette méthode consiste à utiliser des simplexes qui évoluent par contraction, expansion et réflexion au cours des itérations successives afin d'approcher le minimum recherché [6].

2.3 Du gradient à l'algorithme BFGS

Nous nous sommes ensuite attachés à étudier l'algorithme BFGS, et globalement les méthodes dérivées de la méthode Newton. Le principe de la méthode de Newton repose sur un développement de Taylor à l'ordre 2 de la fonction à minimiser :

$$f(x + \Delta x) \approx f(x) + \Delta x^T \nabla f(x) + \frac{1}{2} \Delta x^T (\nabla^2 f(x)) \Delta x$$

A chaque itération, nous choisissons un vecteur Δx qui nous permettra de construire le prochain vecteur $x_{n+1} = x_n + \Delta x$. Dans le but de trouver le pas qui minimise la valeur de la fonction en x_{n+1} , on différentie le développement de Taylor par rapport à Δx , ce qui donne :

$$g_n + H_n \Delta x = 0, \text{ avec } g_n = \nabla f(x_n) \text{ et } H_n = \nabla^2 f(x_n)$$

On doit donc inverser la matrice Hessienne H_n pour trouver la meilleure direction vers laquelle se diriger pour minimiser la fonction f . On en déduit le pas

$$\Delta x = -H_n^{-1} g_n$$

L'intérêt des méthodes quasi-Newton et en particulier de la méthode BFGS est donc d'approximer la matrice Hessienne, ou bien directement son inverse toutes deux difficiles à calculer. Diverses méthodes telles que la méthode DFP, BFGS ou Dogleg choisissent d'approximer différemment l'inverse de cette matrice Hessienne.

Dans le cas de l'algorithme BFGS, l'inverse de la matrice Hessienne est approchée selon la méthode suivante : pour calculer la matrice de l'itération n , on cherche la matrice la plus proche en norme L^2 de la Hessienne approximée à l'itération $n - 1$ qui satisfait les deux conditions suivantes.

- Il faut que la matrice soit symétrique, tout comme la matrice Hessienne réelle, et donc son inverse aussi.
- Il faut qu'elle satisfasse la condition sécante suivante, qui assure que l'approximation de la fonction prend les mêmes valeurs en gradient que g_n en x_n :

$$H_n(x_n - x_{n-1}) = (g_n - g_{n-1})$$

Ces deux conditions et la minimisation de $\| H^{-1} - H_{n-1}^{-1} \|_F^2$ donnent une expression pour l'approximation de la Hessienne d'une itération à l'autre :

$$H_{n+1}^{-1} = (I - \rho_n y_n s_n^T) H_n^{-1} (I - \rho_n s_n y_n^{-1}) + \rho_n s_n s_n^{-1} \text{ avec } \rho_n = (y_n^T s_n)^{-1}$$

Cette méthode d'optimisation est très efficace pour de faibles dimensions.

2.4 Résultats expérimentaux

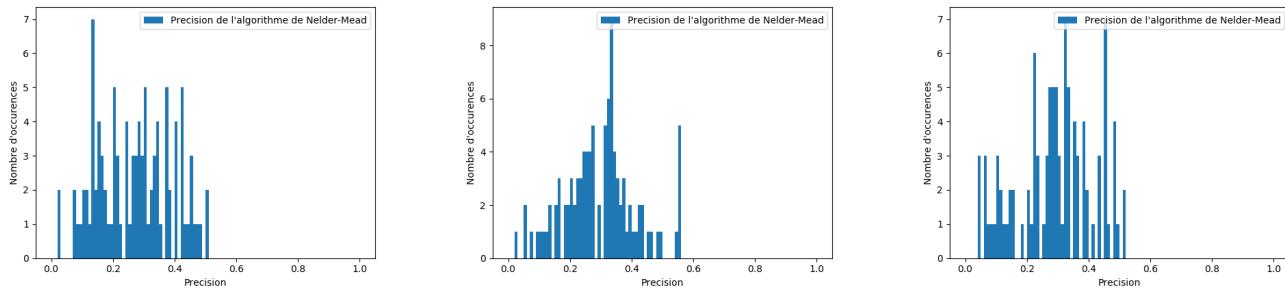
Nous avons cherché à tester l'efficacité des différentes méthodes mises en œuvre dans ce projet. Pour cela, voici l'expérience que nous avons mise en œuvre : pour chaque méthode, nous l'appliquons à la fonction de perte L^1 conçue dans la partie précédente pour notre réseau principal *monty.ang*, et pour les trois jeux de paramètres pour lesquels nous avons construit des tables de valeurs. Pour uniformiser les calculs, nous avons normalisé par une transformation linéaire les paramètres d'entrée de la fonction de perte, de sorte que pour nous, $f : [0; 1]^2 \rightarrow [0; 1]$.

Nous réalisons 100 calculs et nous notons pour chaque essai la précision L^2 obtenue dans le carré $[0; 1]^2$ vis à vis des paramètres de référence, que nous divisons par $\sqrt{2}$, longueur de la diagonale du carré, afin de normaliser les valeurs obtenues. Nous projetons sur $[0, 1]^2$ les résultats éventuels qui sortiraient du carré.

Nous obtenons les résultats suivants, présentés sous forme d'histogramme, et toujours dans le même ordre pour ne pas surcharger les légendes (respectivement temps de réaction et distance d'anticipation, temps de réaction et temps de réaction aux feux, temps de réaction aux feux et distance d'anticipation).

2.4.1 Nelder-Mead

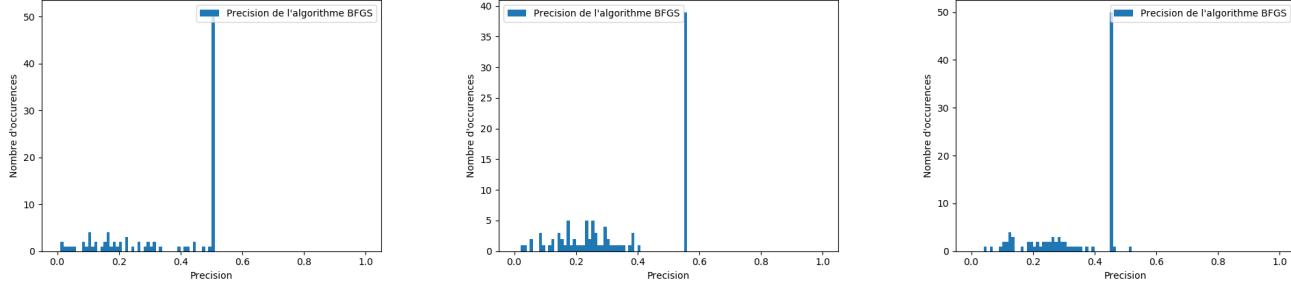
Nous appliquons l'algorithme de Nelder-Mead en choisissant comme point de départ un point aléatoirement tiré dans $[0; 1]^2$ selon une loi uniforme. Nous obtenons les histogrammes :



Nous constatons que l'algorithme de Nelder-Mead renvoie des résultats relativement peu précis et dont les valeurs de précision obtenues sont très dispersées, ce que nous pensons dû à la présence de nombreux minima locaux, dans lesquels l'algorithme a tendance à se perdre, pour les fonctions de perte étudiées.

2.4.2 BFGS

De même, nous appliquons l'algorithme BFGS en partant d'un point de départ tiré suivant une loi uniforme, et nous obtenons les histogrammes :



Nous constatons que l'algorithme BFGS est encore moins précis que celui de Nelder-Mead dans les conditions considérées. Cela est sûrement dû aux hypothèses très restrictives quant à la régularité de la fonction à minimiser. Notons que l'histogramme fait apparaître une valeur très souvent atteinte : il s'agit de la projection sur le carré $[0; 1]^2$ des valeurs sortant du cadre.

Les deux algorithmes que nous venons de tester ne sont pas très satisfaisants dans le cadre de notre étude. C'est pourquoi nous avons décidé de nous tourner vers des algorithmes stochastiques, qui permettent non seulement de contourner le problème des minima locaux, mais aussi de réduire éventuellement le nombre d'appels à la fonction de perte².

3 Algorithme SPSA

3.1 Fonctionnement de l'algorithme

La publication de James C. Spall dans *Encyclopedia of Electrical and Electronics Engineering* [7], qui nous a servi de base pour établir notre version de l'algorithme développe longuement cet aspect (p. 534-536). L'inconvénient majeur des algorithmes du gradient classiques présentés ci-dessus est la nécessité de calculer une approximation de chaque coordonnée du gradient, soit un total de $O(p)$ évaluations de notre fonction. L'algorithme SPSA propose de résoudre ce problème par un calcul différent du gradient approximé :

$$\widehat{g}_k(\widehat{\theta}_k) = \frac{y(\widehat{\theta}_k + c_k \Delta_k) - y(\widehat{\theta}_k - c_k \Delta_k)}{2c_k} \begin{bmatrix} \Delta_{k1}^{-1} \\ \Delta_{k2}^{-1} \\ \vdots \\ \Delta_{kp}^{-1} \end{bmatrix}$$

La perturbation se fait via la direction du vecteur Δ_k , et non plus coordonnée par coordonnée. On ne doit donc évaluer que 2 fois la fonction et non plus $O(p)$ fois. Cela permet de limiter le temps de calcul à chaque itération, ce qui dans notre cas est essentiel.

Une autre idée à proposée est de ne perturber notre vecteur θ_k que selon une direction choisie aléatoirement à chaque étape. Cela ne demande que 2 calculs par itération, et propose tout de même une approximation du gradient de notre fonction. Cette idée est au coeur de la méthode FDSA [8], (Approximation Stochastique avec Différences Finies). Cet algorithme dérive directement de l'algorithme de gradient classique tout en réduisant le nombre de calcul à chaque itération. Le calcul du

2. Notamment en dimension plus élevée comme nous allons voir au paragraphe suivant.

gradient approximé se fait selon la formule :

$$\widehat{g}_{ki}(\widehat{\theta}_k) = \frac{y(\widehat{\theta}_k + c_k e_i) - y(\widehat{\theta}_k - c_k e_i)}{2c_k}$$

Où e_i représente le vecteur unité selon la i^{eme} direction.

En pratique, l'algorithme FDSA n'est pas utilisé, car la perturbation selon une seule direction fait qu'il faut en moyenne p fois plus itérations que pour le SPSA. Dans [8], on aboutit en effet à :

$$\frac{\text{Nombre d'évaluations de } y \text{ avec SPSA}}{\text{Nombre d'évaluations de } y \text{ avec FDSA}} \rightarrow \frac{1}{p}$$

L'algorithme SPSA est donc privilégié pour de grandes dimensions, par rapport à l'algorithme FDSA. On se donne en plus de la fonction à minimiser un vecteur de départ θ_0 , des suites (a_k) et (c_k) et une suite de vecteurs aléatoires (Δ_k) indépendants et identiquement distribués, de taille p (p étant la dimension de l'espace considéré).

Partant de θ_0 , on itère sur le vecteur θ_k selon la formule suivante :

$$\widehat{\theta}_{k+1} = \widehat{\theta}_k - a_k \widehat{g}_k(\widehat{\theta}_k)$$

Cette formule simple est commune aux algorithmes de type gradient, dont le FDSA.

Le choix des suites (a_k) et (c_k) est primordial, et l'on constate que selon celui-ci, l'algorithme peut être plus ou moins efficace, voire même diverger. Il existe des théorèmes qui garantissent la consistance de la méthode SPSA ([6],[8]), notamment la convergence en probabilité sous certaines hypothèses. Ici, nous choisissons :

$$a_k = \frac{a}{(A+k)^\alpha}, \quad c_k = \frac{c}{k^\gamma} \quad \text{pour } k \geq 1$$

où $A \geq 0$, $a > 0$, $c > 0$, et $\gamma \in [\frac{1}{6}, \frac{1}{2}]$. (Dans nos tests, nous avons utilisé $A = 0$, $a = 1$, $c = 0.5$, et $\gamma = \frac{1}{3}$.)

En outre, nous calculons (Δ_k) comme une suite de vecteurs dont les composantes sont des variable aléatoire de moyenne nulle, et de moment inverse fini (dans notre cas des variables de *Rademacher*).

Dès lors, si les suites satisfont :

$$\lim_{l \rightarrow +\infty} a_k \rightarrow 0 \text{ et } \sum_{k=1}^{+\infty} a_k = +\infty$$

$$\gamma \leq \frac{1}{2} \text{ et } \sum_{k=1}^{+\infty} \left(\frac{a_k}{c_k} \right)^2 < +\infty$$

alors les hypothèses du théorème sont satisfaites et la méthode converge en probabilité.

3.2 Avantages de la méthode

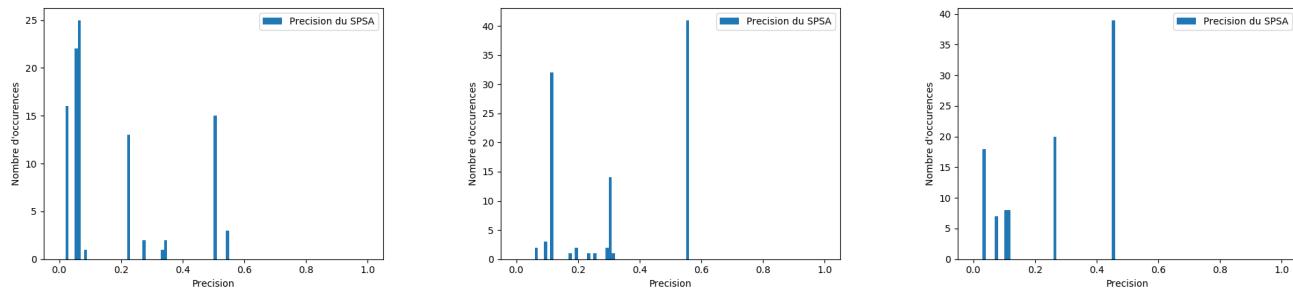
Nous avons donc, pour la phase de réel développement du projet, retenu la méthode SPSA comme algorithme d'optimisation principal. Les raisons de notre choix peuvent se résumer comme suit :

- Les contraintes de temps mentionnées dans une section précédente rendent extrêmement attractive la méthode SPSA, qui ne demande qu'un faible nombre d'appels à la fonction objectif.

- Les gains potentiels de précision que pourraient apporter des méthodes de type BFGS ou Dogleg ne sont pas réellement possibles au vu des faibles propriétés de régularité des fonctions de perte étudiées.
- La tendance à une convergence globalement assez rapide des algorithmes de gradient les rend également particulièrement attractifs.

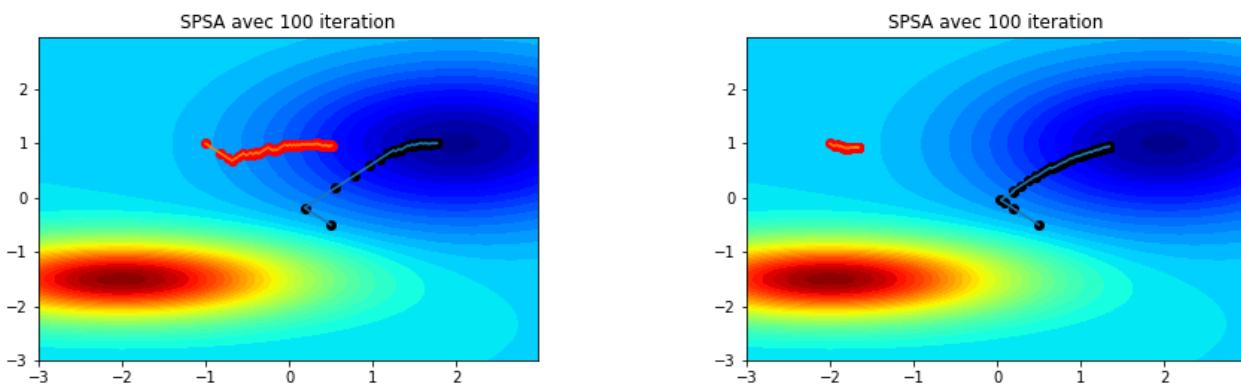
3.3 Résultats expérimentaux

Réalisant la même expérience que précédemment en utilisant l'algorithme SPSA sur un point de départ choisi aléatoirement (suivant une loi uniforme) pour 10000 itérations - ce qui correspond à 20000 appels à la fonction de perte -, nous obtenons les histogrammes suivants :



Nous constatons que la méthode renvoie très régulièrement des résultats très proches des valeurs de références, mais que les précisions obtenues sont très dispersées, puisque de nombreuses valeurs de précision restent encore grandes.

Ceci est en réalité dû à un phénomène de dépendance avec le point de départ choisi : si celui-ci est trop éloigné du point recherché, l'algorithme risque de ne pas converger assez vite. Nous illustrons ce phénomène avec les graphiques ci-dessous pour une différence de deux gaussiennes : dans certains cas, l'algorithme s'éloigne même un peu de la zone d'intérêt. Comme le pas est choisi rapidement décroissant, accroître le nombre d'itérations n'aura alors guère d'effet pour améliorer la convergence.



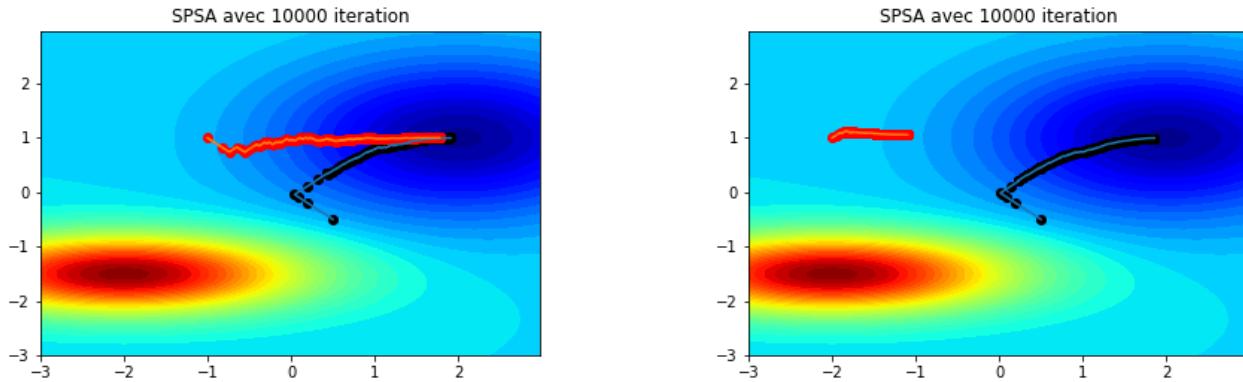


FIGURE 11 – Sensibilité de l'algorithme au point de départ

Nous avons en outre constaté sur l'exemple de fonctions de perte Aimsun, que seuls **5%** des points de départ sélectionnés au hasard (loi uniforme) mènent à une convergence suffisante vers le minimum global recherché, c'est à dire aboutissant à une valeur correcte à 10% près. Ainsi, il parait intéressant de chercher à réduire la dépendance du SPSA au point de départ ; c'est l'objet de la section suivante.

Cinquième partie

Recherche de zones d'intérêt

1 Introduction à nos travaux

Nous avons ainsi constaté que si la méthode *SPSA* possède le double avantage de requérir peu d'appels à la fonction de perte et de ne pas se perdre dans les minima locaux, elle est très sensible au point de départ choisi.

Cela a motivé le développement d'une méthode efficace de recherche de point de départ utilisant le moins d'évaluations d'Aimsun possible. Nous avons tout d'abord pensé - dans un procédé très orienté Monte-Carlo - à tirer des points au hasard dans l'espace des paramètres, puis parmi ces points, à prendre celui ayant la valeur de perte minimale comme point de départ de l'algorithme *SPSA*. Néanmoins, ce n'est pas toujours efficace, car il y a une forte probabilité de tomber dans un minimum local, biais auquel les méthodes de gradient sont particulièrement sensibles. En nous inspirant des algorithmes de classification avancés comme les *SVM*, les réseaux de neurones et la clusterisation (*K-moyennes*, notamment), nous développons une technique de recherche du point de départ afin d'appliquer le *SPSA* efficacement. Cette technique tente d'exploiter au maximum les données du passé (les anciennes évaluations de f) pour réduire la zone de recherche.

2 Organisation du code

Le choix de Python pour notre projet nous permet d'accéder facilement à de puissants *packages* tels que *Scikit-learn* ou encore *Tensorflow*, spécialisés dans les techniques de *machine learning*, dont font partie les algorithmes de classification. Nous avons souhaité, afin que le travail du groupe soit efficace, garantir la modularité du projet, afin que chacun des membres puisse travailler indépendamment sur une partie du projet et d'interfacer facilement les différents modules développés.

Nous insistons sur le fait que, dès le début, nous avons pensé à la généralisabilité de nos codes, notamment en faisant appel aux fonctionnalités de calcul vectoriel de Python. Celles-ci sont en effet utilisées par la plupart des modules (ce qui permettrait en particulier d'implémenter, par la suite, d'autres méthodes d'optimisation plus avancées), et favorisent le *High Performance Computing* (structure notamment adaptée aux calculs *CUDA*).

Le diagramme ci-dessous résume la relation entre nos différents scripts :

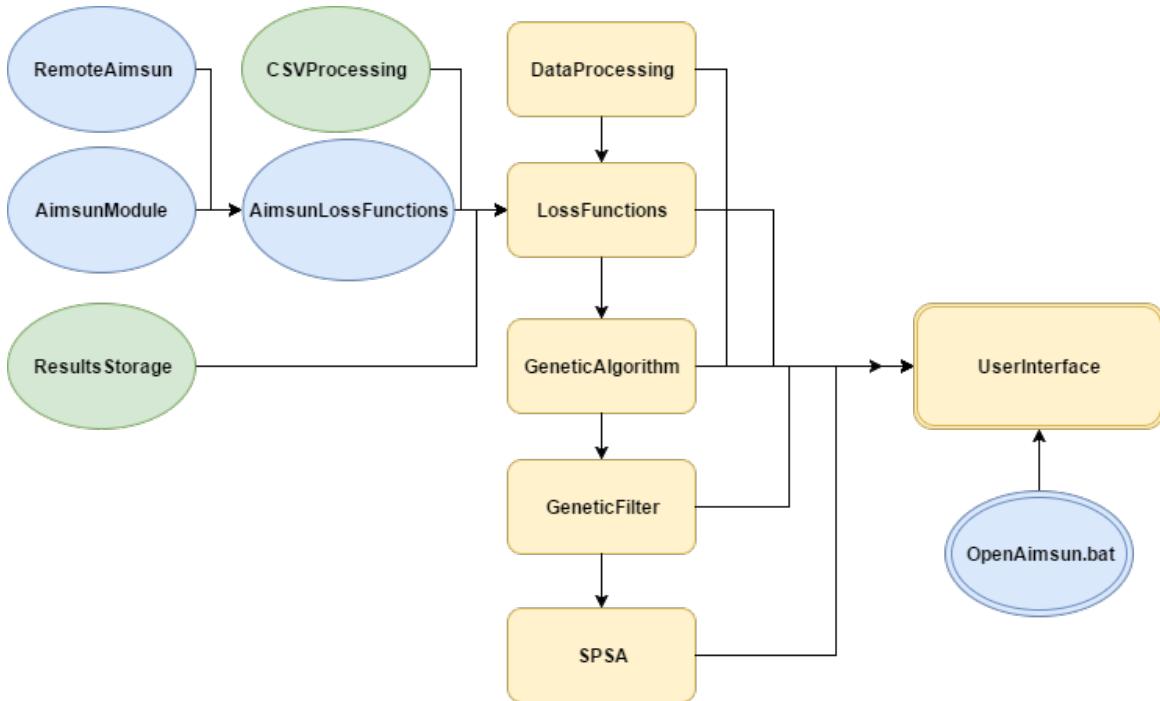


FIGURE 12 – L’organisation du code

Chaque case représente un fichier script Python. Les cases bleues représentent les scripts des parties d'**Interface** et de **Comparaison**, et les jaunes ceux des parties d'**Optimisation** et de **Recherche des zones d’intérêt**.

2.1 DataProcessing

Ce script est chargé de l’ensemble des fonctionnalités de pré-traitement : normalisation des données (normes \mathbb{L}^1 , \mathbb{L}^2 , centrage et réduction, transformations linéaires et non-linéaires expérimentées...). Sachant que les variables n’ont pas la même amplitude de variation (la variance), ni la même dimension, cela n’aurait en effet eu aucun sens de les comparer et de les appliquer directement dans les calculs (en physique, on accorde par exemple une grande importance à l’homogénéité des grandeurs).

Dans le cadre de notre projet, on utilise principalement la méthode dite "Centrée-Réduite" pour chaque paramètre afin d’assurer que les différents paramètres sont de moyenne nulle et de variance 1 (de même "portée"). Soit $x = (x_1, \dots, x_m)$ un point dans l'espace \mathbb{R}^m , et N le nombre de points dans l'espace. On stocke ces données dans une matrice X de taille $N * m$ en respectant la convention internationale, c'est-à-dire :

$$X = \begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_m^1 \\ x_1^2 & \diagdown & \diagdown & \vdots \\ \vdots & \diagdown & \diagdown & \vdots \\ x_1^N & x_2^N & \cdots & x_m^N \end{pmatrix}$$

où chaque ligne représente un point, l’indice en haut signifiant k -ième point. Pour chaque paramètre x_i , on retranche la moyenne de ce paramètre sur les N données et on le divise par son écart type pour

le normaliser.

$$\tilde{X} = \frac{X - \text{moyenne}(X)}{\text{écart-type}(X)} \stackrel{\text{noté}}{=} \frac{X - m}{\sigma}$$

2.2 LossFunctions

Ce script regroupe l'ensemble des fonctions de perte que nous implémentons, que ce soit les fonctions de perte décrites précédemment dans la partie comparaison, afin d'exploiter les simulations produites par Aimsun, ou encore certaines fonctions explicitement définies (gaussiennes, polynomiales ou plus exotiques) que nous avons aussi utilisées pour tester nos différentes méthodes d'optimisation.

2.3 GeneticAlgorithm

Ce script joue un rôle central dans notre algorithme de recherche des zones d'intérêt. Il porte le nom de *GeneticAlgorithm*, mais ce n'est pas réellement un algorithme génétique, car il n'utilise pas le principe de mutation. Cependant, il s'inspire largement de cette idée. En effet, l'idée principale est de faire des classifications binaires récursives, s'améliorant au cours des différentes itérations, en utilisant certaines techniques de *machine learning*. L'objectif est de déterminer les zones de recherche les plus intéressantes dans l'espace des paramètres. Nous lançons tout d'abord des points dans l'espace au hasard. Le programme fait ensuite appel à *LossFunctions* pour évaluer la valeur de perte de chaque point tiré. Puis, l'algorithme de classification SVM permet d'obtenir une "prédiction" (sous la forme d'une fonction $f : \mathbb{R}^m \rightarrow \mathbb{R}$ qui est beaucoup plus rapide en évaluation) des zones les plus intéressantes dans lesquelles rechercher les paramètres. Ce procédé est itéré plusieurs fois pour réduire considérablement la zone de recherche. Le diagramme ci-dessous illustre le principe de notre algorithme :

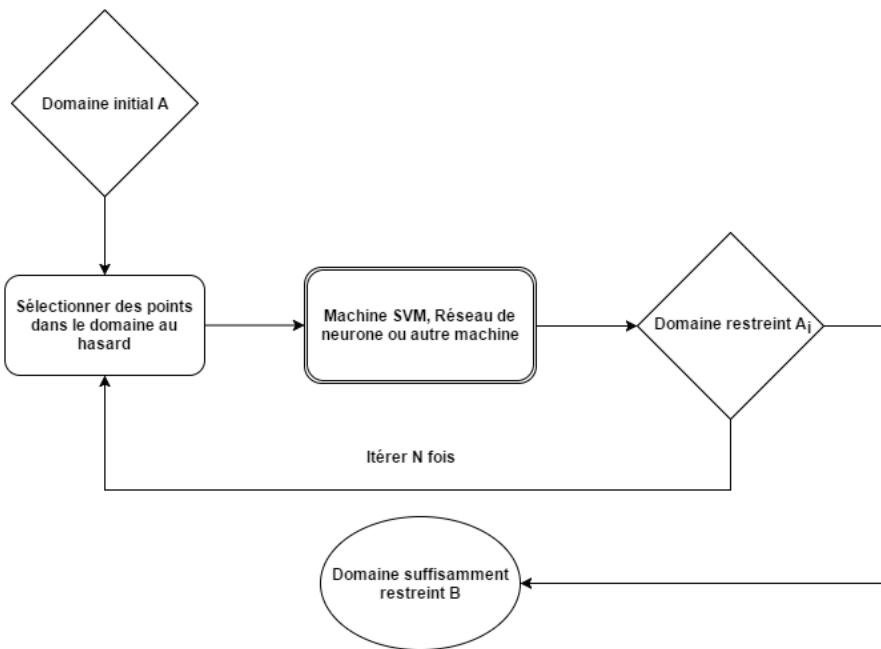


FIGURE 13 – Diagramme pour la cellule *GeneticAlgorithm*.

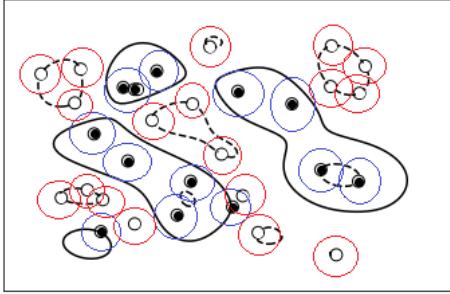


FIGURE 14 – La fonction originale

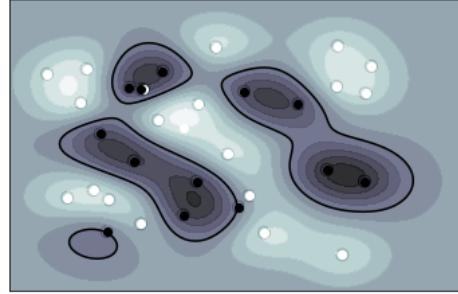


FIGURE 15 – 0 itération de l'algorithme

Pour notre projet, nous utilisons le paquet Python *Scikit-learn*, [9], qui comprend notamment un module **SVM** très performant, écrit en C/C++. Voyons rapidement le fonctionnement des machines à vecteurs de support (on s'appuie ici sur [10]). Le principe est simple : il consiste à trouver la frontière optimale séparant deux groupes de données sous la forme d'un hyperplan, en maximisant la distance entre les vecteurs de données et l'hyperplan trouvé. Ici, nous utilisons au préalable une transformation non-linéaire (de type noyau gaussien) de l'espace permettant des configurations plus complexes que simplement linéaires. La fonction de décision peut alors, par exemple, prendre la forme :

$$f(x) = \text{signe} \left(\sum_{i=1}^N w_i y_i K(x_i, x) \right)$$

où w est le vecteur définissant l'hyperplan séparateur, et K le noyau gaussien ($K(x, y) = e^{-\frac{\|x-y\|^2}{2}}$).

L'algorithme **SVM** possède en particulier deux hyperparamètres qu'on peut modifier manuellement pour calibrer la précision de la classification, C et γ . L'hyperparamètre C est un paramètre de pénalisation pour traiter le problème d'*overfitting*, c'est-à-dire d'un trop grand degré d'adaptation aux données de la machine, réduisant fortement les capacités de prédiction de l'algorithme. Si le résultat de la classification est *overfitté*, il peut être nécessaire de diminuer C , et vice versa. L'hyperparamètre γ peut quant à lui être considéré comme l'inverse du rayon de la boule (resp. cercle en 2D) qui délimite les échantillons considérés comme positifs des échantillons négatifs. Aussi, plus le rayon est petit (ie plus γ est grand), plus la classification sera précise.

L'idée de notre algorithme peut donc être résumée en une phrase : **transformer un problème d'optimisation lourd en un autre problème d'optimisation beaucoup plus simple (ajustement du noyau gaussien de la machine SVM au lieu de descente de gradient à fort coût unitaire)**.

Intéressons-nous tout d'abord à une fonction de perte très simple en dimension 2, pour laquelle on peut facilement visualiser toutes les étapes du calcul, que l'on représente ci-dessous :

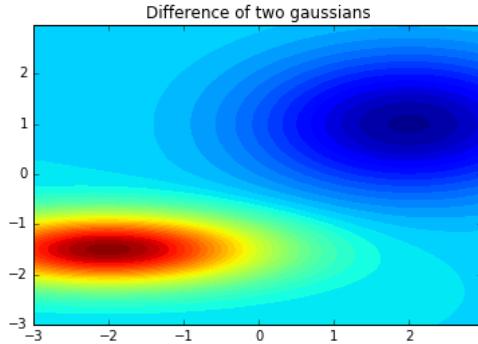


FIGURE 16 – La fonction originale

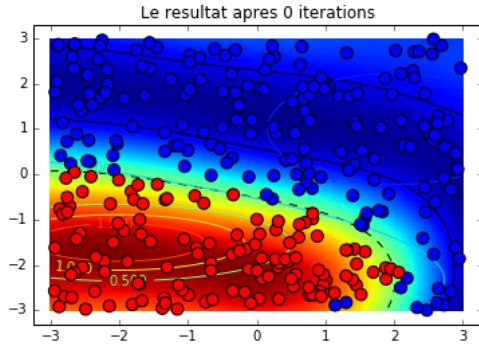


FIGURE 17 – 0 itérations de l'algorithme

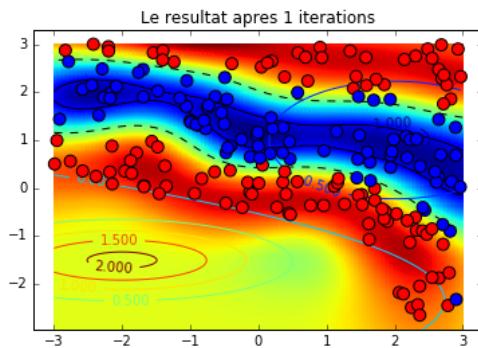


FIGURE 18 – Résultat après 1 itération

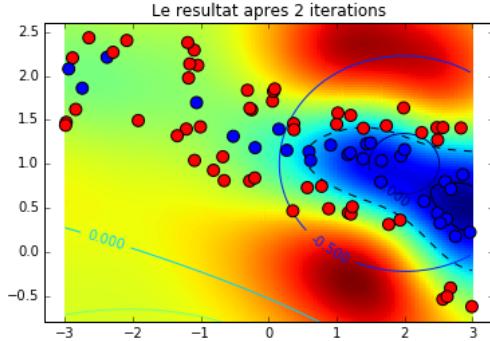


FIGURE 19 – Résultat après 2 itérations (zoomé)

La Figure (16) représente la fonction étudiée. Son minimum global se situe au point $(x, y) \simeq (1.5, 1.0)$. La couleur bleu correspond donc à la "vallée" et la rouge correspond à la "montagne". À l'itération 0, la machine fait la classification et elle infère la forme de la surface avec les données (les points) qu'on lui fournit. Nous voyons que sur la Figure (17) que la machine a réussi à approcher, dans ce cas simple, la forme de la surface originale. (c.f. Figure (16)) Ensuite, en relançant le tirage au hasard, on ne conserve que les points dans la zone bleue de la Figure (17). La machine relance alors une classification, qui mène à la Figure (18), et ainsi de suite. Avec 2 itérations, on obtient enfin la Figure (19) qui nous indique que le minimum global de la fonction étudiée se situe certainement dans la zone bleue. Cela correspond très bien à notre attente : $(1.5, 1.0)$. Ici, on a pris $C = 1.0$ et $\gamma = 0.09$.

Le lecteur prudent pourrait alors légitimement poser la question suivante : À l'itération 0, on a déjà la forme de la fonction étudiée imitée par la machine. Pourquoi ne chercherait-on pas directement le minimum global au sein des valeurs aléatoires lancées par la machine ? Cette question est en fait plus épiqueuse qu'il n'y paraît. Cependant, la non-perfection des fonctions de perte étudiées en pratique avec Aimsun fait qu'on pourrait facilement tomber par malchance dans un minimum local, et au passage rater le véritable minimum global. Utiliser la machine SVM permet donc d'éviter le biais classique d'*overfitting*, particulièrement fort ici à cause de l'importante probabilité de présence d'*outliers*.

Nous sommes donc très attentifs concernant l'utilisation de l'algorithme SVM. Un point crucial est notamment le quantile de filtrage des données lancées au cours des différentes phases de l'algorithme.

Un juste équilibre est à trouver entre un *overfitting* certain (seuil trop exigeant) et une non-éfficience de l'algorithme liée à un seuil trop faible. Nous utilisons donc une boucle *while* pour augmenter petit à petit ce seuil, afin ce que la proportion d'échantillons positifs représente entre 20% et 80% de la distribution totale. Nous observons, par exemple sur les figures (15) à (18), que cela mène à une décroissance exponentielle du volume de la zone d'intérêt obtenue.

La sortie de ce script est un fichier en `.pk1` qui caractérise la zone de recherche (bleu)³. En pratique, nous utilisons le script `GeneticFilter` pour en obtenir les informations d'intérêt.

NB : D'autres approches, comme le réseau de neurones ou les k-moyennes, ont également été tentées pour effectuer la classification. Cependant, nous avons estimé qu'il serait plus judicieux de se concentrer sur une seule forme de classificateur, ici celle qui nous semblait la plus prometteuse.

2.4 GeneticFilter

Rappelons que l'algorithme génétique précisé précédemment retourne une fonction de décision qui permet de délimiter des zones d'intérêt où on estime que la fonction de perte est sous un certain seuil. L'étape suivante est donc de déterminer un point de départ pour les algorithmes d'optimisation qui se situerait à l'intérieur de ces zones d'intérêt. Il est ainsi nécessaire de réaliser d'abord un filtrage de l'espace en ne gardant que les zones que nous estimons assez intéressantes puis de choisir un point de départ pour l'algorithme SPSA. Afin de rester flexible, le *GeneticFilter* propose de coder ce filtrage en question et choisit judicieusement non pas un seul point de départ mais plusieurs (nombre déterminé par `num_cluster`). En effet, étant donné que le SPSA est fortement sensible au point de départ, cela lui permet ici d'effectuer le parcours pour différentes valeurs de départ, l'utilisateur peut ainsi choisir le parcours qui lui donne le meilleur résultat.

2.4.1 Filtrage par seuil.

Afin de délimiter les zones d'intérêt, l'idée est de tirer uniformément sur l'espace de travail un nombre N de points. A l'aide de la fonction d'estimation retournée par l'algorithme génétique (on rappelle que le calcul est ici direct et qu'il n'y pas d'appel à Aimsun effectué ici), on parvient à évaluer chaque point tiré et de ne sélectionner que les points sous un certain seuil défini par l'utilisateur. La valeur par défaut du seuil est naturellement 0 étant donné que la fonction de perte retenue par l'algorithme génétique est une classification binaire. Si l'utilisateur souhaite varier le seuil, il doit garder en tête que ce dernier doit rester entre $[-1, 1]$ tout simplement parce que la valeur des points tirés par la fonction de décision sont dans cet intervalle.

La figure à gauche représente les points tirés par l'algorithme génétique précédemment décrite qui ont permis d'obtenir la fonction de décision (dont la carte de chaleur en est la représentation). La figure de droite représente quant à elle les points tirés par le filtrage. Il est important de ne pas confondre ces deux séries de points ! Ici on souhaitait sélectionner l'ensemble des points sous le seuil représenté par la ligne de niveau dans la figure à gauche.

3. En fait, on a concaténé ce script avec le script `GeneticFilter` qui fournit un "barycentre de la zone d'intérêt" (voir le paragraphe suivant), donc la sortie est plutôt un(des) point(s).

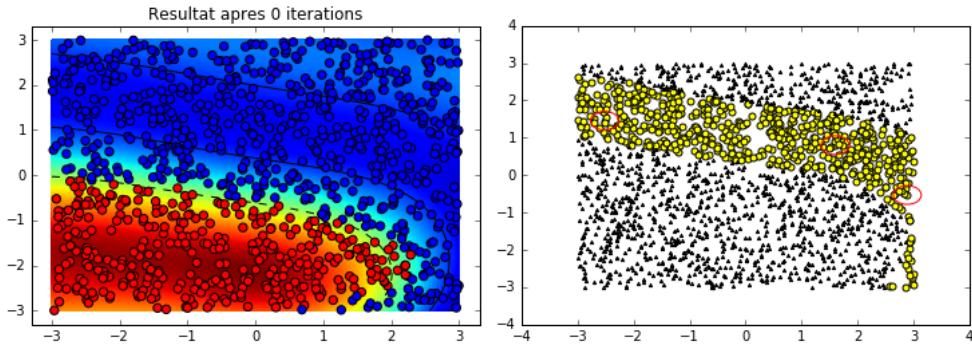


FIGURE 20 – Filtrage par selection des points tirés sous un certain seuil

2.4.2 Sélection des points de départ du SPSA

La dernière étape consiste à sélectionner les points de départs pour l'algorithme génétique. Naïvement, nous pouvons tout simplement choisir directement parmi les points tirés et filtrés. Cette solution conviendrait dans le sens où le SPSA partirait bien d'un point de la zone d'intérêt. Toutefois, nous n'exploitons pas toute l'information à notre disposition à savoir que tous les autres points filtrés se retrouveraient inutiles et que l'utilisateur connaît peu de propriétés sur les points choisis. Ainsi il est possible qu'un point choisi soit proche de la ligne de niveau du seuil limite du filtre. Cela aurait pour conséquence non seulement d'alourdir la tâche de l'algorithme d'optimisation mais ce dernier risque aussi de partir dans la “mauvaise direction” à savoir vers l'extérieur de la zone d'intérêt.

Ainsi l'idée est de s'éloigner de la ligne de niveau du seuil limite du filtre. Pour cela, pour former un *cluster*, nous choisissons un point issu du filtre aléatoirement et nous saisissons tous les autres points issus du filtre à un certain rayon. En exploitant un argument de convexité local, le point de sortie de l'algorithme est alors le barycentre de tous ces points. En tirant un nombre de points assez important pour couvrir uniformément l'espace de travail, cette méthode permet notamment de s'éloigner des lignes de niveau limite.

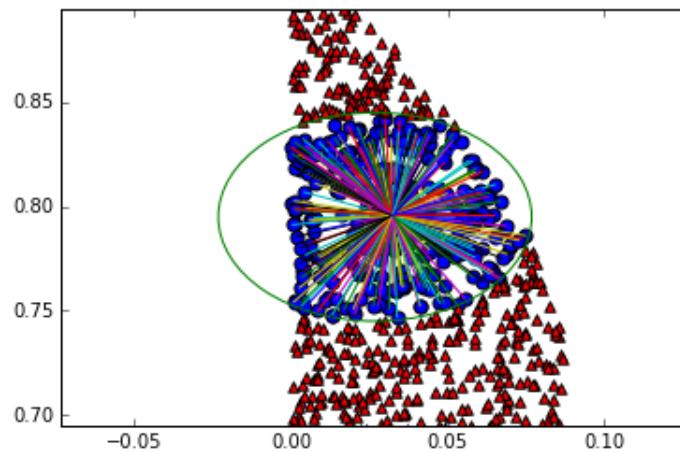


FIGURE 21 – Sélection des points de départ

Une problématique se pose quant au choix du rayon de sélection du *cluster*. Dans le cas d'un rayon trop peu élevé, le barycentre ne parvient pas vraiment à s'éloigner du bord et le *cluster* risque de

ne contenir qu'un seul point à cause d'une sélection trop stricte ce qui rendrait tout cela inutile. Toutefois, un rayon trop grand perd de son intérêt et la convexité n'est généralement pas assuré auquel cas le barycentre ne serait pas une mesure appropriée.

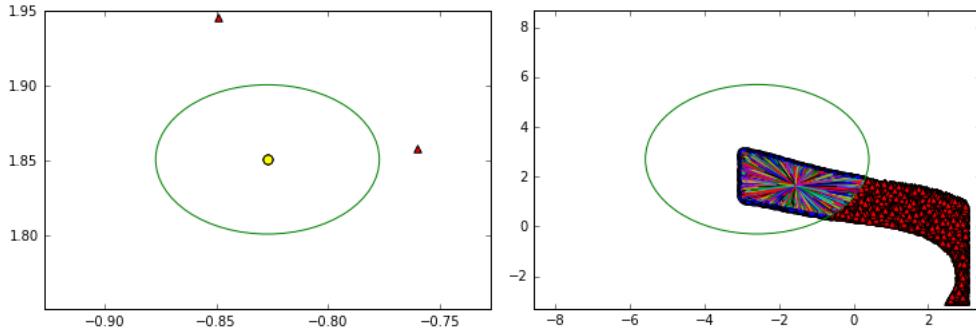


FIGURE 22 – Cas pathologiques selon le rayon de sélection (a) trop faible (b) trop important

Il est donc nécessaire de bien calibrer le rayon pour rendre ce processus plus efficace.

Nous remarquons que finalement nous ne sommes pas si loin de la méthodologie employée par l'algorithme génétique. Finalement il aurait été envisageable de programmer un algorithme de classification qui situerait la ligne de niveau limite et s'en éloignerait le plus possible.

2.5 UserInterface

L'objectif de ce script, plutôt cosmétique, est de permettre à l'utilisateur de faire appel à notre processus de calibration en un seul script. Ce programme nous permet également d'effectuer différentes tâches de vérification des résultats : enregistrement des données, affichage de graphes et de messages automatiques, et stockage en cours de calcul pour permettre une grande tolérance aux pannes qui, au vu du temps de calcul, peuvent facilement survenir.

Notre intérêt étant d'évaluer le moins de points possible pour retrouver une forme proche de celle de la fonction étudiée, puis d'appliquer l'algorithme SPSA pour obtenir une solution plus précise, il nous faut donc trouver un compromis entre le nombre de points de tirage par `GeneticAlgorithm` (noté N) et le nombre d'itérations de SPSA (noté $Iter$). Le facteur déterminant de la complexité de notre programme d'optimisation étant bien entendu le nombre d'appels effectués à Aimsun, notre complexité vaut ici :

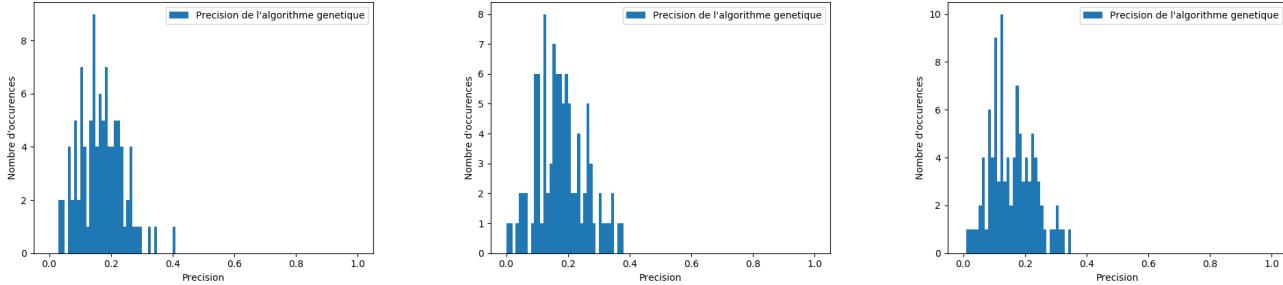
$$\boxed{\text{Appels Aimsun} = N + 2 * Iter}$$

Minimiser ce nombre d'appels tout en conservant une précision de convergence est trop délicat dans le cadre de notre étude, mais il est intéressant de pouvoir jouer sur ces deux paramètres afin de limiter le nombre d'appels à la fonction de perte réalisés.

3 Résultats expérimentaux de la méthode proposée

3.1 Méthode génétique

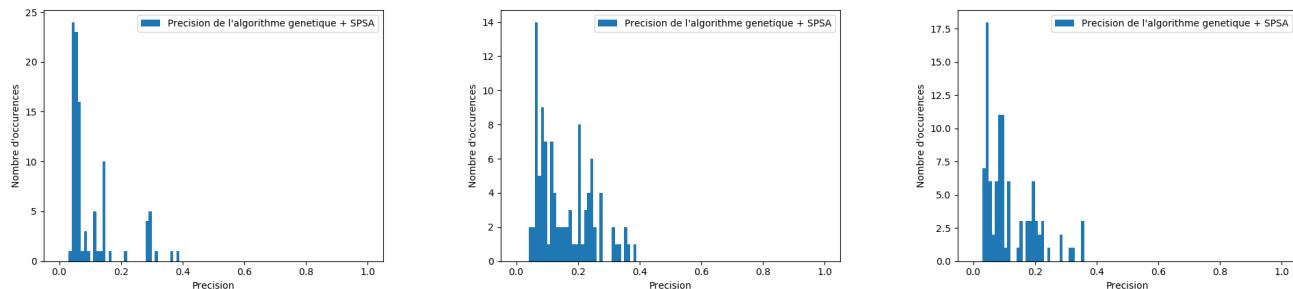
Pour cette expérience, réalisée selon les mêmes conditions que dans la partie précédente, nous utilisons la méthode "génétique" développée ci dessus, pour 500 points de tirages, et une seule itération. Nous obtenons les histogrammes :



Le résultat nous satisfait car pour seulement 500 appels à la fonction de perte, nous obtenons des valeurs plus proches en moyenne des valeurs de référence que pour tous les autres algorithmes essayés jusqu'à présent, sans toutefois que les précisions obtenues ne soient trop dispersées.

3.2 Méthode génétique + SPSA

Pour cette expérience, nous réalisons une application du SPSA avec 100 itérations, partant du point obtenu pour l'expérience précédente. Cela correspond donc à 700 appels à la fonction de perte. Nous obtenons les histogrammes :



Nous constatons que la méthode est encore plus précise que la méthode génétique seule, ce qui nous satisfait relativement.

4 Conclusion scientifique

Nous avons essayé plusieurs méthodes et algorithmes dans ce projet pour résoudre le problème de la calibration des paramètres considéré.

Bien sûr, notre étude ne se prétend pas générale : nous avons restreint le problème à la minimisation d'une fonction interpolée correspondant à une valeur de perte pour un modèle de trafic correspondant à l'étude d'une petite zone commerciale, en ne considérant que deux paramètres. Nous sommes toutefois satisfaits d'être parvenus à élaborer une méthode intéressante et relativement efficace pour traiter notre problème, après avoir constaté l'échec des paradigmes d'optimisation "classiques".

Suite à ce projet, nous identifions plusieurs perspectives d'étude :

- L'étude de notre algorithme en dimension plus élevée, afin de préciser son efficacité.
- L'étude de l'influence des différents nombres d'itération pour notre algorithme génétique et le SPSA.

- L'utilisation d'autre méthodes de *machine learning* plus avancées : réseaux de neurones, optimisation bayésienne, *deep learning*...

Sixième partie

Bilan managérial et retour d'expérience

1 Organisations du groupe au cours du projet

Au cours de l'année, la vision d'ensemble du projet et son organisation ont varié. Revenons donc sur les différentes articulations que le groupe a connues.

Comme en témoigne la proposition détaillée que nous avions constituée au commencement du projet, nous comptions initialement diviser l'effectif du groupe en deux pôles interdépendants, l'un chargé d'étudier les différentes méthodes de comparaison pour former une fonction de perte, l'autre de rechercher et analyser les différentes méthodes d'optimisation évoquées en premier lieu afin de répondre au problème.

Nous nous sommes rapidement rendu compte que nous avions négligé l'importance de certains aspects du projet. Notamment nous pensions initialement pouvoir rapidement réaliser l'interface entre Python et Aimsun, mais cela s'est révélé plus long que prévu suite à certains délais d'obtention des licences et à la mauvaise compréhension de la structure de l'API d'Aimsun ; ce travail d'interfaçage nous a finalement demandé plusieurs mois. De même, la réflexion menée quant aux méthodes d'optimisation à utiliser pour ce projet nous a conduit à rechercher et développer des méthodes de recherche de "zones d'intérêt", aspect dont nous n'avions pas réalisé l'importance initialement.

Le groupe s'est donc naturellement articulé en quatre pôles traitant les quatre problématiques dégagées au début de ce rapport. Cette structure est restée pertinente jusqu'à la complétion de l'interface entre Python et Aimsun et la conception des différentes fonctions de perte : depuis, les membres qui faisaient partie des deux pôles chargés respectivement de réaliser cette interface et de constituer des fonctions de perte ont pu rejoindre les deux autres pôles.

Au sein du groupe, tous les élèves ont à chaque instant été considérés comme égaux. Toutefois, afin d'unifier la direction choisie et de simplifier les problématiques administratives, certaines responsabilités ont été attribuées à différents membres du groupe :

- Lucas BROUX a été désigné comme chef du projet, son rôle étant la charge des relations administratives et des directions de recherche scientifique.
- Benjamin PETIT a été désigné comme chef adjoint du projet, responsable des démarches scientifiques : il a tout au long du projet permis de guider les décisions scientifiques à prendre.
- Louis GUO a été désigné comme responsable du contact avec l'entreprise. Tout au long du projet, il a organisé les réunions Skype avec TSS, ce qui a contribué à l'établissement de très bonnes relations entre le groupe et l'entreprise.
- Lu LIN a été désigné responsable de l'étude et de l'avancement des différentes méthodes d'optimisation et de recherche de zones d'intérêt étudiées.

Ainsi, le groupe a su adapter son organisation à l'avancement du projet. Voici précisément l'organigramme que le groupe a suivi durant le projet.

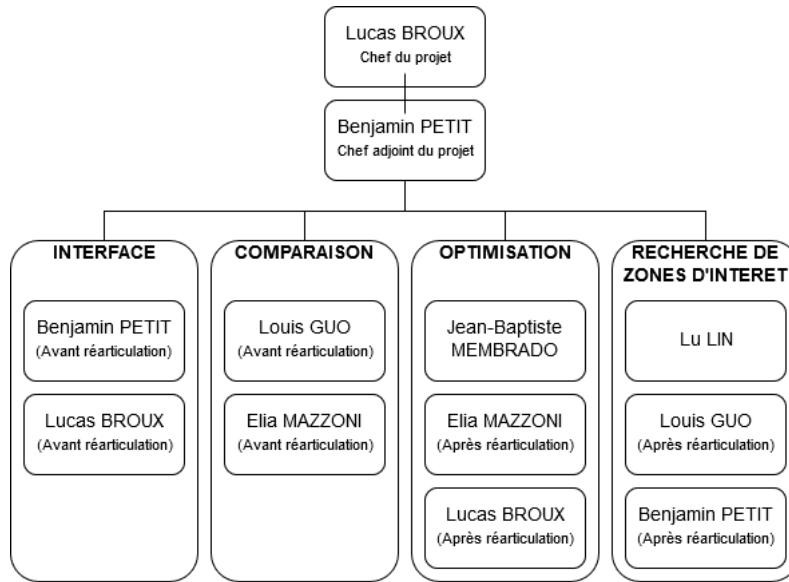


FIGURE 23 – Organigramme du groupe

2 Collaboration avec l'entreprise TSS

Ce Projet Scientifique Collectif n'était pas un coup d'essai pour l'entreprise TSS, qui avait déjà encadré un projet l'année précédente, et qui connaissait donc le format et les exigences du projet. Contrairement au groupe précédent, nous avons tenu à rester le plus possible en contact avec TSS, c'est pourquoi nous avons organisé tout au long de l'année des réunions Skype hebdomadaires. Ces réunions nous ont permis d'effectuer des points de situation réguliers avec notre tutrice, ainsi qu'avec l'équipe scientifique de TSS. Mme Annique Lenorzer, avec qui nous avons pu échanger très régulièrement, nous a particulièrement aidé dans nos recherches, et a pu nous mettre en contact avec différents scientifiques de l'entreprise qui ont pu nous apporter leur expérience sur certaines thématiques plus spécifiques.

En outre, le travail de communication que nous avons tâché de mettre en œuvre cette année a permis un véritable renforcement des liens entre l'École et TSS. En effet, suite à des entretiens avec Jean-Baptiste BORDES, directeur délégué du cycle ingénieur polytechnicien, l'entreprise TSS est en discussion pour participer à la conception d'un MODAL dédié à la problématique des véhicules autonomes et des *Smart Cities*. Ainsi, cette collaboration avec TSS aura été extrêmement fructueuse et satisfaisante. Nous espérons que celle-ci se prolongera pendant plusieurs années, avec les futures promotions de l'École.

3 Retour d'expérience sur la gestion du projet

3.1 Gestion des inattendus

Nous avons dû faire face au cours de ce projet à un certain nombre d'inattendus qui nous ont retardé dans notre travail. Notamment :

- L'obtention des licences a pris plus de temps que prévu. Cela a été en partie dû à des raisons administratives : un document à signer s'étant perdu. Au cours de l'année, nous avons également du faire face à des pertes de validité inopinées de certaines licences, mais le problème fut rapidement réglé. Il est à noter que malheureusement l'un des membres du groupe - travaillant sur un *Mac* - n'a pas pu obtenir de licence.
- D'un point de vue informatique, l'implémentation de l'interface entre Aimsun et Python a demandé plus de temps que prévu, puisque la documentation informatique sur laquelle nous nous sommes appuyés était en réalité obsolète. Grâce au contact avec TSS et les conseils d'Annique Lenorzer, et après de nombreuses relectures de notre code source, nous avons heureusement pu identifier la source du problème à temps et corriger l'erreur.

3.2 Gestion du calendrier prévisionnel

Nous nous sommes très tôt donnés des objectifs ambitieux, dont la réalisation a souvent dû être retardée suite à des inattendus dont la résolution s'est parfois fait attendre, comme témoigné plus haut.

Toutefois, comme le témoignent les différents rapports précédemment réalisés dans le cadre de ce projet, nous avons, à chaque étape, tenu à mettre en place et conserver un calendrier prévisionnel nous guidant dans la réalisation de nos objectifs, quitte à le réarctiquer en cas de retard trop important. Cet état d'esprit, qui consiste à connaître à chaque instant les objectifs qui nous concernent, nous a permis d'améliorer notre potentiel d'efficacité tout au long de l'année.

Nous sommes globalement satisfaits de l'avancement général du projet et des résultats que nous sommes parvenus à obtenir afin de répondre à la problématique définie conjointement avec l'entreprise partenaire TSS.

4 Conclusion quant à l'aspect managérial du projet

L'avancement de ce Projet Scientifique Collectif n'a pas été aussi linéaire et régulier que nous l'aurions espéré lorsque nous nous sommes engagés à étudier la problématique proposée par TSS. Au contraire, nous avons parfois été contraints et bloqués par des problèmes, souvent techniques, qui nous ont réellement ralenti dans la réalisation de notre projet : en particulier, les problèmes de licence que nous avions au début du projet, et les multiples difficultés que nous avons eues pour réaliser l'interface de scripting entre Python et Aimsun.

Ces inattendus ont contribué à la perturbation du calendrier prévisionnel que nous nous étions fixés ; mais les capacités d'adaptabilité de tous les membres du groupe ont permis de gérer ces nécessités de modification et de prendre les décisions qui s'imposaient afin de redéfinir ce calendrier.

Toutefois, nous considérons ce projet comme un succès : nous sommes parvenus à formuler des éléments de réponse pour l'entreprise TSS sur un sujet que nous savions dès le début difficile. Nous avons de surcroît participé au renforcement des liens entre l'entreprise et l'École, ceux-ci s'étant rapproché et actuellement en discussion dans le cadre d'un projet de formation.

Conclusion

Ce projet comportait deux directions majeures, une portée vers de l'informatique, une vers des mathématiques appliquées. Les connaissances acquises grâce aux cours de l'école ont été grandement exploitées et le projet a également permis de pousser notre savoir-faire dans les différents domaines. L'interfaçage du logiciel Aimsun a nécessité une compréhension solide des sorties du logiciel Aimsun. Notre solution basée sur le modèle client-serveur se révèle être assez efficace et nous avons pu générer et enregistrer massivement des tables de données (environ 100 CPU-heures de calcul) pour faciliter le bon déroulement des algorithmes génétiques et d'optimisation. L'agencement et la sauvegarde de ces tables de données nous ont permis de manipuler les fichiers .csv qui sont présents dans tout travail manipulant un peu des données.

Nous avons également porté un nouveau regard sur le problème de minimisation d'une fonction. Nous nous sommes bien aperçus du déphasage entre les théorèmes généraux enseignés en cours et l'application à une fonction qui présente une faible régularité. Toutefois, avec les connaissances acquises, nous avons pu apporter une solution façonnée pour ce problème. Il nous a d'abord fallu trouver une fonction de perte adéquate au problème à savoir assez régulière. Cette partie était intéressante dans le sens où nous devions apporter nos propres idées, les références étant peu nombreuses étant donné la spécifité du problème. Il a été enrichissant de voir comment chacun raisonné à ce niveau là. Par la suite, l'algorithme génétique et le filtrage résultant nous a permis de localiser les zones d'intérêt où nous suspectons la présence de faibles valeurs. A partir de ces zones, nous pouvons lancer les différents algorithmes d'optimisation. Le fait d'avoir partitionné le travail en différentes problématiques a permis de simplifier un problème initial qui paraissait difficilement résolvable.

Ce projet a été une opportunité formidable pour appliquer sur un cas concret et professionnel nos compétences scolaires. Le projet touchant un peu à tout, chaque membre du groupe a pu s'attacher à ce qui l'intéressait le plus. Nous remercions évidemment les équipes de TSS pour nous avoir accompagné tout du long en nous appuyant et nous fournissant un soutien continu. Ils ont été très intéressés par l'avancée de nos recherches.

Bibliographie

- [1] Transport Simulation Systems (2014), Aimsun 8 Scripting Manual, p.10 - Aimsun Architecture.
- [2] Le Goff, V. (2014). Apprenez à programmer en Python. 1st ed. [Paris] : Openclassrooms, p.Communication en réseau dans les programmes Python.
- [3] Communication privée avec A. Lenorzer (Décembre-Février 2016/17).
- [4] Klein, B. (2017). Python Advanced : Python and SQL. [online] Python-course.eu. Available at : http://www.python-course.eu/sql_python.php [Accessed Feb. 2017].
- [5] Smith,, J. and Blewitt, R. (2010), Traffic Modelling Guidelines, Transport for London, [online] Transport for London. Available at : <http://content.tfl.gov.uk/traffic-modelling-guidelines.pdf> [Accessed Feb. 2017].
- [6] Chong, E. and Zak, S. (2011). An Introduction to Optimization. 1st ed. Hoboken : John Wiley & Sons.
- [7] Spall, J. (1992). Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control, 37(3), pp.332-341.
- [8] Gentle, J., Härdle, W. and Mori, Y. (2012). Handbook of computational statistics. 1st ed. Heidelberg : Springer, Stochastic Approximation.
- [9] Varoquaux, G., Buitinck, L., Louppe, G., Grisel, O., Pedregosa, F. and Mueller, A. (2015). Scikit-learn. GetMobile : Mobile Computing and Communications, 19(1), pp.29-33.
- [10] Bishop, C. (2013). Pattern recognition and machine learning. 1st ed. New Delhi : Springer, p.325.

Annexes

Le code Python réalisé est à trouver sur le repository GitHub suivant : <https://github.com/unclebenX/PSC>