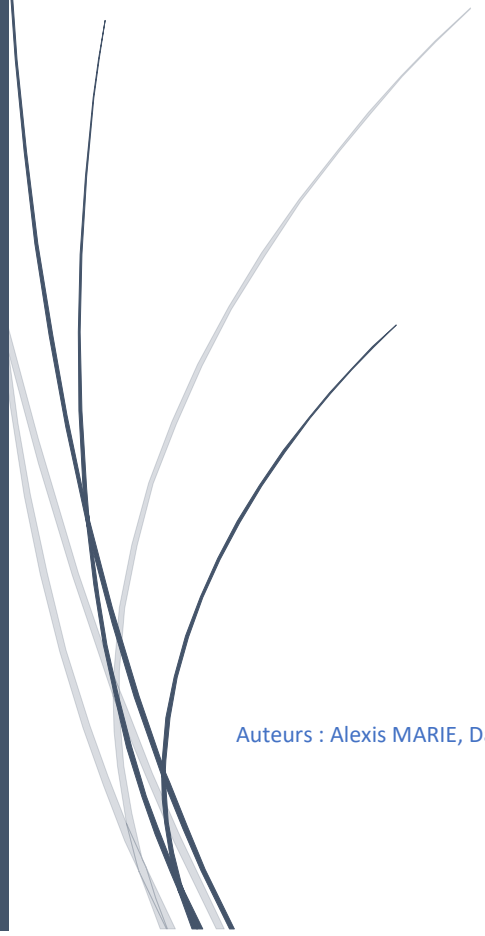
Optimisation

Cahier de laboratoire : comparaison des temps de trajets

Génération d'un plan de navigation à partir d'une scène #PFE23-R-198



Auteurs : Alexis MARIE, David MARCHES, Luca BANKOFSKI, Martial BROSTIN, Théo HELLER, Thomas MABILLE

Sommaire

Création de graphe orienté de réseau routier urbain

Objectif de l'expérience

Le but de cette première expérience est de réussir à créer le programme informatique nous servant à réaliser nos simulations sur nos graphes orientés du réseau routier de villes réelles. Il s'agit donc dans un premier temps de recréer à partir des deux fichiers CSV des nœuds et des arcs d'une ville (dont on peut retrouver les étapes de construction dans le protocole expérimental *PFE23-R-198_PE_Optimisation_Génération-graphes).*

Protocole expérimental

Nous utilisons les deux CSV [1] de la ville de Bischoffsheim pour recréer un graphe orienté. Pour cela, nous créons deux fonctions, "*getNodes*" et "*setEdges*", nous permettant d'une part de créer un graphe avec des nœuds placés à des coordonnées précises grâce aux informations de latitude et longitude ; et d'autre part, nous créons des arcs grâce au second CSV nous indiquant les nœuds de départ et d'arrivée de chaque arc. Nous avons aussi ajouté pour chaque arc la vitesse maximale qui était renseignée dans le CSV ou 50 si elle ne l'était pas.

Une fois ce graphe créé, le but est de pouvoir le visualiser pour confirmer que le graphe formé ait du sens. Bischoffsheim n'a pas été pris au hasard, il s'agit en fait d'un petit village connu de l'un des membres du groupe permettant de reconnaître à l'affichage si le graphe correspond bien à la réalité.

Observations

Nous nous sommes très vite rendu compte que la création d'un tel graphe était rapide mais que l'affichage était lui très lent. Nous n'allions donc pas pouvoir visualiser les résultats de nos simulations futures sur de grandes villes, ce qui posera par la suite des soucis d'interprétation de résultats.

Données

Les données des 102 CSV des 51 villes choisies pour réaliser nos expériences sont disponibles sur le Git [1] dans le fichier "Sources". Voici, pour Bischoffsheim, le résultat obtenu :

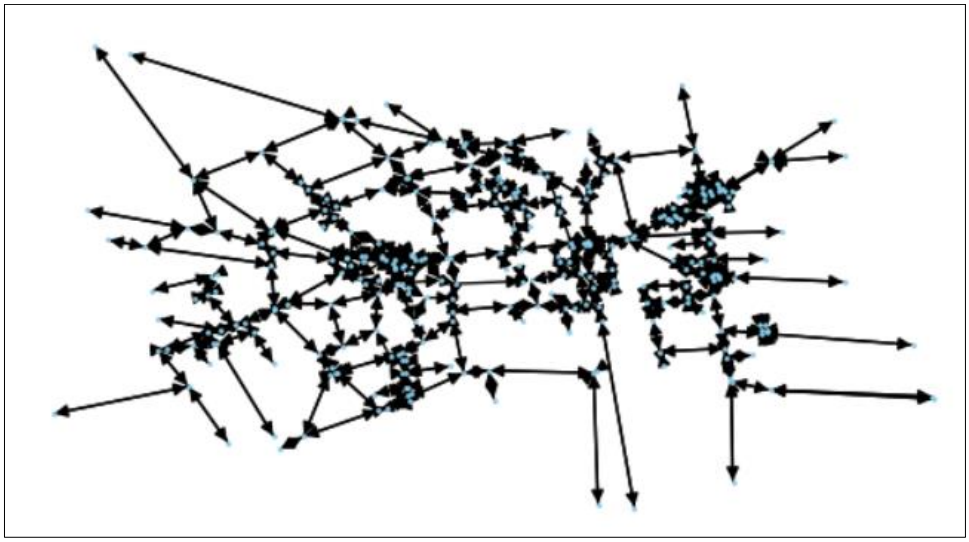


Figure 1 - Graphe du réseau routier de la ville de Bischoffsheim.

Conclusion

Cette première étape s'est très bien passée puisque nous avions toutes les données nécessaires à la recréation de ces graphes dans les CSV. Le but est ensuite de créer les simulations S1, S2, S3 et S4 détaillées dans le protocole expérimental (voir document *PFE23-R-198_PE_Simulation-graphes)*.

Utilisation de l’algorithme Dijkstra sur nos graphes orientés

Objectif de l'expérience

Le but de cette expérimentation est de réussir à partir de nos graphes régénérés précédemment à appliquer l’algorithme de Dijkstra (voir document *PFE23-R-198_EDA_Algorithmes*) d’un point A à un point B.

Protocole expérimental

Pour cette expérience très rapide, nous avons utilisé la bibliothèque Python NetworkX qui fournissait une fonction *dijkstra_path* nous permettant simplement de trouver le chemin le plus court entre un point A et un point B d’un graphe orienté. [2]

Données

Voici un visuel des résultats que l’on a pu obtenir sur la ville de Bischoffsheim avec en vert le chemin le plus court obtenu :

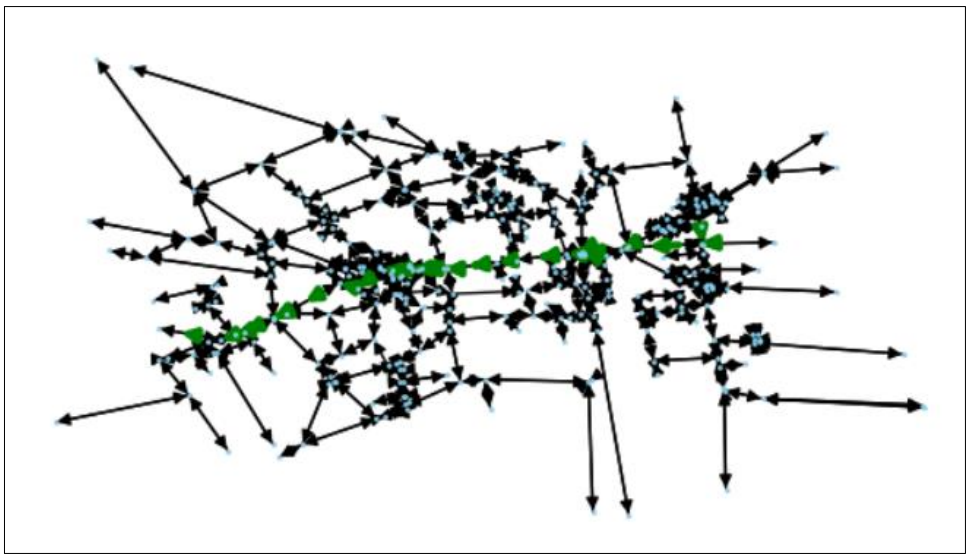


Figure 2 - Exemple d'itinéraire le plus court entre deux sommets de Bischoffsheim.

Analyses préliminaires

La mise en place d’un tel programme était simple mais l’algorithme de Dijkstra étant déterministe, nous nous sommes posé la question du temps de calcul de notre fonction. Heureusement, sur des petits parcours ces temps étaient très négligeables mais des temps de calcul bien plus longs sur les grandes villes présageaient de longues périodes de calcul pour les simulations de nos différentes situations.

Conclusion

Nous avions réussi à faire tourner un algorithme de plus court chemin entre deux points d'un graphe représentant le réseau routier d'une ville ce qui représentait le premier gros défi de ce projet. Le but était désormais de prendre en compte le poids des arêtes en créant un graphe pondéré par le temps de trajet de chaque arc du graphe.

Ajout d’une pondération à nos graphes orientés

Objectif de l'expérience

Le but de cette nouvelle expérience est de faire de nos graphes orientés des graphes orientés pondérés. Cette pondération sera égale au temps de trajet en seconde de cette voie à vitesse maximale. Nous vérifierons ensuite que notre algorithme de plus court chemin prenne en compte ces poids dans son calcul d’itinéraire.

Protocole expérimental

Nous avons repris notre fonction *setEdges* pour y rajouter le fait que chaque arête du graphe sera égale au temps en seconde nécessaire à sa traversée. Pour cela, nous avons utilisé les informations de vitesse maximale et de longueur de la route présentes dans le CSV qui permet de reconstruire les arcs du graphe. Dans l’étude préalable de nos graphes, nous avions remarqué qu’un bon nombre de voies n’avaient pas de spécification de vitesse maximale. Comme nous étudions des villes et comme nous avions des vitesses moyennes d’environ 50 km/h sur la moyenne globale des limitations de vitesse, nous avons utilisé une vitesse par défaut de 50 km/h pour nos calculs.

Nous avons ensuite testé à nouveau l’algorithme de Dijkstra afin de vérifier s’il prenait bien en compte la pondération ajoutée au graphe. Pour cela, nous avons manuellement vérifié les réponses de l’algorithme sur l’affichage du graphe avec le chemin le plus court ainsi que la pondération des voies.

Observations

On remarquait sur certains tests que cette pondération était absolument cruciale pour le réalisme de l’expérience. En effet, sur certaines vérifications visuelles faites, l’algorithme préférait « prendre l’autoroute », nous indiquant que les résultats attendus étaient cohérents par rapport au monde réel.

Données

Voici un visuel des résultats que l’on a pu obtenir sur la ville de Bischoffsheim avec pour chaque arête une pondération en secondes de temps de trajet :

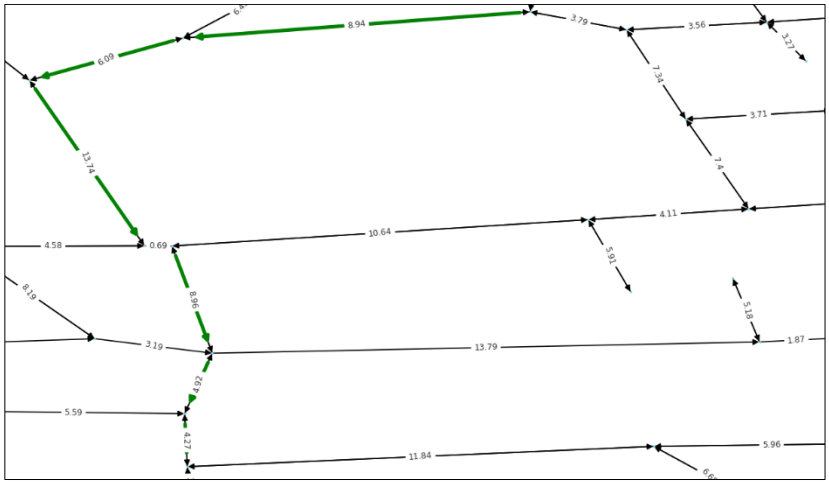


Figure 3 - Zoom sur le graphe du réseau routier de Bischoffsheim pondéré en temps de trajet par arc en seconde.

Conclusion

Nous avons donc réussi à construire tous les briques nécessaires à la réalisation de nos 4 situations de simulations en créant un graphe orienté représentant le réseau routier d'une ville pondéré par le temps de trajet de chaque route.

Création des programmes pour les situations S1 et S4

Objectif de l'expérience

L'objectif de cette expérience était de mettre en place les simulations des situations S1 et S4 de notre étude détaillée dans le protocole expérimental *PFE23-R-198_PE_Optimisation_Simulation-graphes*.

Protocole expérimental

Dans un premier temps, nous avons créé une fonction *add_perturbations* nous permettant de rajouter des perturbations d'une durée définie sur un nombre donné d'arcs du graphe sélectionnés aléatoirement. Il s'agissait donc simplement de rajouter sur certaines arêtes un nombre de secondes de perturbation. Nous avons utilisé 60 secondes par défaut pour avoir des perturbations courtes qui sont très fréquentes en ville.

Pour la situation S1, le but était donc de trouver le chemin le plus court sur notre graphe comme s'il ne voyait pas les perturbations (voir document *PFE23-R-198_PE_Simulation-graphes*). L'enjeu était donc dans un premier temps de trouver le chemin le plus court d'un point A à un point B. Ensuite, nous avons rajouté le poids des perturbations sur le graphe. Enfin, nous avons sommé le poids de toutes les arêtes de notre chemin le plus court pour obtenir le temps de trajet total pour cette situation.

Pour S4, le procédé est le même cependant les perturbations sont ajoutées avant de réaliser le calcul du plus court chemin pour que l'algorithme puisse éviter les arcs trop lourds dès le départ. S4 est omniscient de ce qui se passe sur le graphe.

Observations

Plusieurs modifications ont été apportées à ces différentes fonctions. Lorsque nous avons commencé nos tests sur des plus grandes villes, il nous a paru plus cohérent de fixer une proportion de routes perturbées et non plus un nombre fixe. Cela nous a permis de faire des études par rapport à un pourcentage de routes perturbées souvent situé entre 0 et 5%.

Le but étant de comparer les trajets des situations S1, S2, S3 et S4 sur des graphes ayant les mêmes perturbations aux mêmes endroits, nous avons corrigé l'algorithme du programme S1 pour que les perturbations rajoutées après calcul du plus court chemin soient bien les mêmes perturbations qui seront appliquées à S1, S2, S3 et S4.

Données

Ici aussi la vérification du fonctionnement de ces algorithmes s'est faite manuellement sur de nombreux graphes créés nous permettant de tester de nombreuses situations imaginées pour être piégeuses. Les situations S1 et S4 étant assez simples à réaliser, la vérification était assez rapide.

Voici par exemple le genre de données que nous avons vérifiées avec le chemin du trajet le plus court, sa longueur ainsi que son poids:

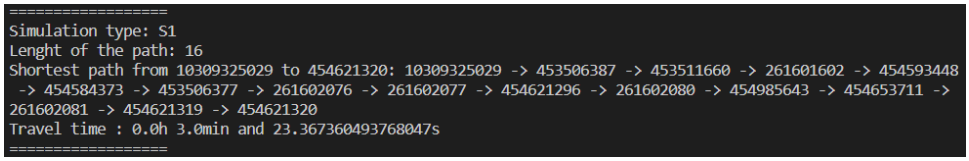


Figure 4 - Exemple d'informations collectées lors d'une simulation de chemin le plus court avec la situation S1.

Analyses préliminaires

Notre hypothèse de départ concernant nos 4 situations S1, S2, S3 et S4 était que le temps de trajet de S1 ≥ S2 ≥ S3 ≥ S4. Durant nos tests préliminaires, nous avons effectivement obtenu que S4 était toujours au moins aussi bon que S1 sinon meilleur.

Conclusion

Nous avons réussi à créer nos 2 situations les plus simples et avons pu tester sur quelques exemples l'efficacité de S4 par rapport à S1 confirmant les hypothèses formulées dans le protocole expérimental *PFE23-R-198_PE_Optimisation_Simulation-graphes*.

Création du programme pour la situation S2

Objectif de l'expérience

L'objectif de cette expérience est de mettre en place la simulation de la situation S2 de notre étude détaillée dans le protocole expérimental *PFE23-R-198_PE_Simulation-graphes*.

Protocole expérimental

Nous avons repris la fonction créée dans l'expérience précédente pour ajouter une proportion donnée de perturbation dans un graphe donné.

Nous avons pour cette situation créé un algorithme calculant le chemin le plus court comme pour les situations S1 et S4, cependant ici les poids perturbés ne sont pas mis à jour ni au début ni à la fin mais en temps réel. Chaque fois que nous ajoutons un nouveau nœud dans notre chemin final le plus court, l'algorithme doit vérifier si les arcs voisins de ce nœud sont des arcs perturbés. Il recalcule ensuite le nouveau chemin le plus court avec ces nouvelles informations jusqu'à la fin. Il est donc ici nécessaire de lancer notre algorithme de plus court chemin chaque fois qu'un nœud est ajouté à notre chemin final.

Observations

S2 nécessitait de calculer beaucoup plus de chemins le plus court que S1 et S4 ce qui nous a amenés à l'optimiser le plus possible. Si dans son voisinage aucune route n'était perturbée, on ne recalculait pas de chemin le plus court et on continuait simplement sur celui précédemment calculé. Cela nous a permis de gagner beaucoup de temps de calcul pour nos simulations.

Données

Ici aussi la vérification du fonctionnement de cet algorithme s'est faite manuellement sur de nombreux graphes créés nous permettant de tester de nombreuses situations imaginées pour être piégeuses.

Voici par exemple le genre de données que nous avons vérifié avec le chemin du trajet le plus court, sa longueur ainsi que son poids.

```
==================
Simulation type: S2
Lenght of the path: 21
Shortest path from 115805262 to 458852661: 115805262 -> 261601759 -> 453506375 -> 2848397722 -> 454593448 -> 454
602080 -> 454985643 -> 454653711 -> 261602081 -> 454621319 -> 454621320 -> 454621309 -> 454593446 -> 458852639 -
Travel time : 0.0h 2.0min and 8.342815861147727s
==================
```

Figure 5 - Exemple d'informations collectées lors d'une simulation de chemin le plus court avec la situation S2.

Analyses préliminaires

Notre hypothèse de départ concernant nos 4 situations S1, S2, S3 et S4 était que le temps de trajet de S1 ≥ S2 ≥ S3 ≥ S4. Durant nos tests préliminaires, nous avons effectivement obtenu que S2 était toujours au moins aussi bon que S1 sinon meilleur, du moins sur de petites villes. Plus tard, nous

verrons que sur nos simulations à grande échelle, nous avons eu des contre-exemples venant du nombre plus important de situations qu'offrait une plus grande ville.

Conclusion

Nous avons réussi à créer notre troisième situation et avons pu tester sur quelques exemples l'efficacité de S2 par rapport à S1 mais sa faiblesse face à S4, confirmant les hypothèses formulées dans le protocole expérimental *PFE23-R-198_PE_Simulation-graphes*.

Création du programme pour la situation S3

Objectif de l'expérience

L'objectif de cette expérience est de mettre en place la simulation de la situation S3 de notre étude détaillée dans le protocole expérimental *PFE23-R-198_PE_Optimisation_Simulation-graphes*.

Protocole expérimental

Pour cette simulation, les choses sont plus complexes à mettre en place. Tout d'abord, il faut créer une fonction permettant de calculer la distance entre deux points dont on connaît les coordonnées géographiques (latitude et longitude). Nous avons donc créé une fonction « haversine » utilisant les équations d'haversine [3] pour calculer une telle distance entre deux points en prenant en compte la courbure terrestre.

Dans un second temps, il faut créer une fonction "edges_within_distance_from_id" utilisant cette première fonction créée. Elle a pour but de renvoyer tous les arcs perturbés que nous avions préalablement placés situés à l'intérieur d'un cercle de centre le nœud voulu et de rayon le rayon voulu. Pour rappel, S3 va pouvoir mettre à jour les arcs perturbés dans un certain rayon autour de lui.

Nous pouvons ensuite, de la même manière que S2, chercher le chemin le plus court, suivre ce chemin en mettant à jour les arcs perturbés autour de lui et recalculer le chemin le plus court si des perturbations ont été trouvées. Ce procédé est répété jusqu'à arriver au point final.

Observations

S3 est de loin la plus gourmande des situations puisqu'elle demande pour chaque sommet parcouru de revérifier dans un rayon autour de lui l'ensemble des arcs pouvant être perturbés. De plus, ce rayon de vision implique qu'il est beaucoup plus probable que S3 « voit » de nouveaux arcs perturbés impliquant le calcul d'un nouveau plus court chemin que S2.

Il y a eu de nombreux problèmes avec ce programme et nous avons dû faire de nombreux tests pour vérifier son bon fonctionnement. Par exemple, pendant un temps, c'est un simple problème d'unité qui nous a fait perdre beaucoup de temps, la fonction haversine renvoyant une longueur en kilomètres alors que nous la voulions en mètres.

Données

Ici aussi la vérification du fonctionnement de cet algorithme s'est faite manuellement sur de nombreux graphes créés nous permettant de tester de nombreuses situations imaginées pour être piégeuses.

Voici par exemple le genre de données que nous avons vérifiées avec le chemin du trajet le plus court, sa longueur ainsi que son poids.

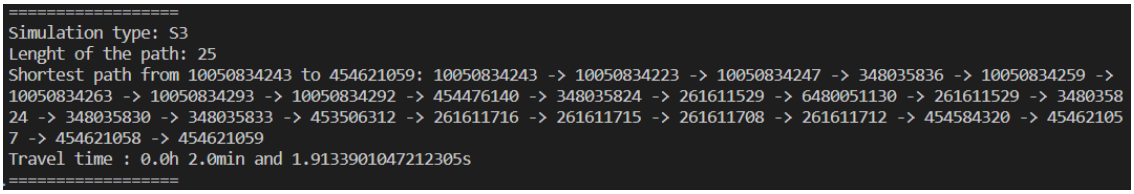


Figure 6 - Exemple d'informations collectées lors d'une simulation de chemin le plus court avec la situation S3.

Analyses préliminaires

Notre hypothèse de départ concernant nos 4 situations S1, S2, S3 et S4 était que le temps de trajet de S1 ≥ S2 ≥ S3 ≥ S4. Durant nos tests préliminaires, nous avons effectivement obtenu que S3 était toujours au moins aussi bon que S1 et S2 sinon meilleur mais que S4 restait le meilleur des 4.

Conclusion

Nous avons réussi à créer notre quatrième situation et avons pu tester sur quelques exemples l'efficacité de S3 par rapport à S1, S2 et S4 confirmant les hypothèses formulées dans le protocole expérimental *PFE23-R-198_PE_Optimisation_Simulation-graphes*.

Maintenant que nos 4 situations pouvaient être simulées, nous pouvions lancer nos simulations sur tous les paramètres voulus sur nos 51 villes pour avoir des données précises quant aux performances de chaque situation.

Simulations des 4 situations S1, S2, S3 et S4 sur les villes étudiées

Objectif de l'expérience

L'objectif de cette expérience est de comparer l'efficacité de nos 4 situations S1, S2, S3 et S4 entre elles sur des villes de toutes tailles. Le but est, dans un premier temps, de tester ces situations avec un rayon de S3 fixe, des temps de perturbation fixes et une proportion de routes perturbées variable.

Protocole expérimental

Cette expérience se découpe en plusieurs petites expériences. Pour rappel, les temps d'exécution de S2 et plus particulièrement de S3 sont longs et il est difficile de lancer nos simulations d'une seule traite. C'est pourquoi les simulations doivent être lancées sur une ville après l'autre de la manière suivante :

Dans un premier temps, nous chargeons la ville dans le programme grâce aux fonctions créées dans la première expérience et aux 2 fichiers CSV que l'on a construits en suivant les étapes du protocole expérimental *PFE23-R-198_PE_Optimisation_Génération-graphes*.

Ensuite, nous effectuons une boucle allant de 0 à 5 par pas de 0.5, représentant le pourcentage de perturbations que nous allons ajouter à notre ville.

Nous effectuons une seconde boucle dans laquelle nous plaçons les perturbations de manière aléatoire dans notre ville. Nous avons conservé des perturbations de 60 secondes pour nos expérimentations, représentant de petites perturbations que l'on trouve fréquemment dans les villes.

Enfin, nous effectuons une troisième boucle dans laquelle nous trouvons un point de départ et un point d'arrivée aléatoires.

Dans cette troisième boucle, nous simulons nos 4 situations avec un rayon de S3 égal à 300 mètres et enregistrons dans un fichier CSV les résultats suivants :

- Le temps de trajet de la situation S1 ;
- Le temps de trajet de la situation S2 ;
- Le temps de trajet de la situation S3 ;
- Le temps de trajet de la situation S4 ;
- La longueur du trajet S1 en nombre d'arcs ;
- La longueur du trajet S2 en nombre d'arcs ;
- La longueur du trajet S3 en nombre d'arcs ;
- La longueur du trajet S4 en nombre d'arcs ;
- Le numéro d'avancement de la boucle dans laquelle on génère les perturbations ;
- Le pourcentage de perturbation appliqué sur notre ville.

On analyse ensuite les performances de S1, S2, S3 et S4 entre elles dans un autre programme informatique. Pour cela, il suffit de faire la moyenne des résultats de chaque situation pour chaque pourcentage de perturbation appliqué et de comparer les 4 valeurs entre elles pour vérifier l'hypothèse formulée dans le protocole expérimental. Pour vérifier que nos moyennes sont

exploitables, on vérifie aussi l'écart type de chacune de nos séries de données, le but étant qu'il soit le plus faible possible.

Pour visualiser nos résultats, on peut aussi les représenter sous forme de graphique, comme vous pouvez le voir Figure 7 ci-dessous.

Observations

Nous avons eu des problèmes au lancement de certaines grosses simulations, puisque le hasard a fait qu'il n'existait pas toujours de chemin entre notre point de départ et d'arrivée. Nous avons donc rajouté une fonction *try catch* lors du calcul de S1 pour simplement vérifier si un chemin existait. Cette fonction permet de ne pas faire stopper notre programme même si un chemin n'existe pas.

Lors de nos premières expérimentations, nous avons obtenu des résultats qui n'étaient pas très concluants ni très réalistes pour nos simulations dans des petites villes. En effet, l'aléatoire créait souvent un point de départ et un point d'arrivée beaucoup trop proches l'un de l'autre, ce qui n'est pas très réaliste. En effet, on prend rarement la voiture pour faire un trajet de 100 mètres dans nos villes. C'est pourquoi nous avons aussi ajouté une condition lors du calcul de S1, qui était de s'assurer que la longueur du nombre d'arcs obtenue soit supérieure à 20.

Nous avons aussi remarqué, en analysant nos données de manière plus poussée, que de temps en temps, la situation S2 était moins efficace que la situation S1, ce qui allait à l'encontre de notre hypothèse. Il était difficile de comprendre pourquoi, car cette situation se produisait principalement dans les grandes villes, ce qui rendait l'affichage dur, et nous avons donc réfléchi à la possibilité que notre code soit faux. C'est après des recherches plus poussées que nous avons mis en lumière des cas où cet événement se produisait. Cela ne venait pas de notre code, mais bien de cas rares de situations où S2 était effectivement très mauvais. Cela s'explique par la nécessité pour certaines voitures de prendre un chemin qu'elles pensent être meilleur, mais qui est en fait moins bon. D'une certaine manière, on pourrait dire que pour que notre solution fonctionne, il faut nécessairement une voiture qui signale la perturbation aux autres et qui prendra rarement bien plus de temps que si elle avait suivi l'itinéraire initialement prévu. C'est pourquoi notre solution finale ne peut fonctionner de manière optimale avec simplement un module caméra, mais que la communication entre ces modules est essentielle. Ce sont les résultats des situations S3 et S4 qui, elles, sont toujours meilleurs que S1 et S2.

Données

Les données de nos différentes observations sont disponibles sur le git de notre projet [4].

Nous avons pu tracer le graphique ci-dessous qui résume les résultats obtenus à la suite de cette longue expérience :

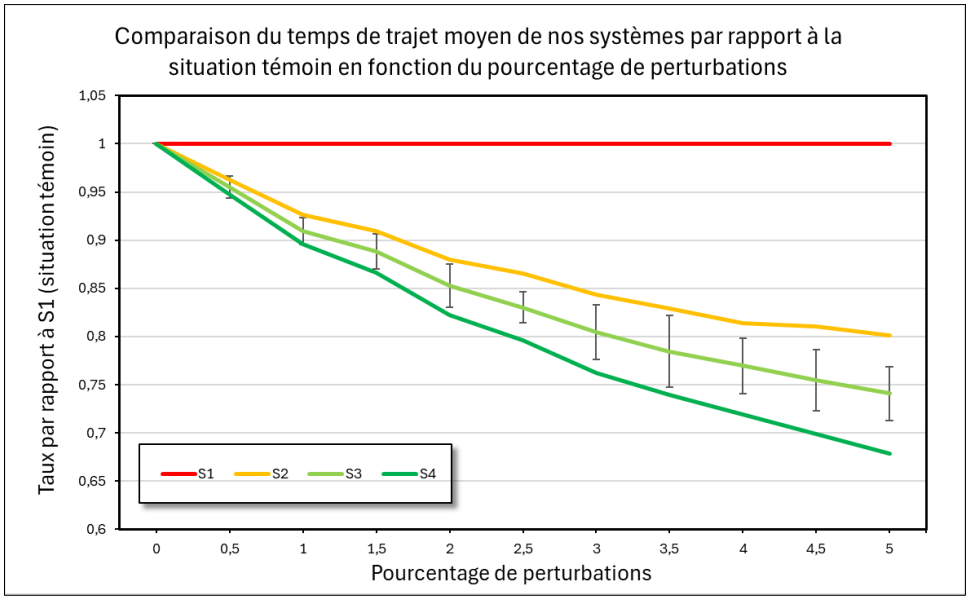


Figure 7 - Comparaison des performances de S1, S2, S3 et S4 par rapport au pourcentage de perturbations ajouté à nos villes.

Analyses préliminaires

Malgré le cas rare où S2 est moins bon que S1, nous obtenons tout de même que S1 est pratiquement toujours moins bon ou égal à S2, lui-même toujours moins bon ou égal à S3, lui-même toujours moins bon ou égal à S4, ce qui vérifie nos hypothèses de départ.

Conclusion

Nous avons réussi à vérifier notre hypothèse de départ, à savoir que le temps de trajet de S1 est supérieur ou égal à celui de S2, qui est lui-même supérieur ou égal à celui de S3, qui est à son tour supérieur ou égal à celui de S4. Cependant, il ne faut pas oublier de prendre en compte que les résultats sont peut-être biaisés par le pourcentage de voies à sens unique qui, rappelons-le, est une variable qui fluctue énormément entre les villes. C'est pourquoi une expérience sera menée sur l'influence de la proportion de routes à sens unique sur nos résultats.

Etude de l’impact de la proportion de route à sens unique sur nos simulations

Objectif de l’expérience

Cette expérience a pour but de vérifier l’influence de la proportion de route à sens unique sur nos simulations de S1, S2, S3 et S4.

Protocole expérimental

Dans cette expérience, nous reprendrons les résultats de l'expérience précédente, mais cette fois-ci en les comparant par rapport à la proportion de routes à sens unique de chaque ville. Cette analyse ne se fait donc plus par rapport à la proportion de routes perturbées, qui restera fixée à 5%, car c’est la situation où les écarts sont les plus grands et où une influence de la proportion de routes à sens unique sera la plus visible.

Il s’agit donc ici de comparer les moyennes des résultats de nos 4 situations pour chaque pourcentage de routes à sens unique. Il ne faut pas oublier de vérifier si notre moyenne a un sens en calculant l’écart type de chaque série, que l’on veut le plus petit possible.

Observation

Pour avoir plus de données, nous avons également effectué ces tests sur des graphes représentant le réseau routier d'une ville fictive. En effet, nous pensions que recréer des graphes de réseau routier réalistes pourrait nous permettre de tester bien plus de paramètres et de mettre en valeur des variables qui affecteraient beaucoup l'une ou l'autre situation. Le programme reprenant l'étude préliminaire sur ces graphes de réseau routier et permettant d'en générer des réalistes a été créé, mais sera finalement très peu utilisé, puisque les variables telles que la proportion de routes à sens unique n'avaient aucun effet sur les performances de nos 4 situations entre elles.

Données

Les données de nos différentes observations sont disponibles sur le git de notre projet [4].

Nous avons pu tracer le graphique ci-dessous qui résume les résultats obtenus à la suite de cette expérience :

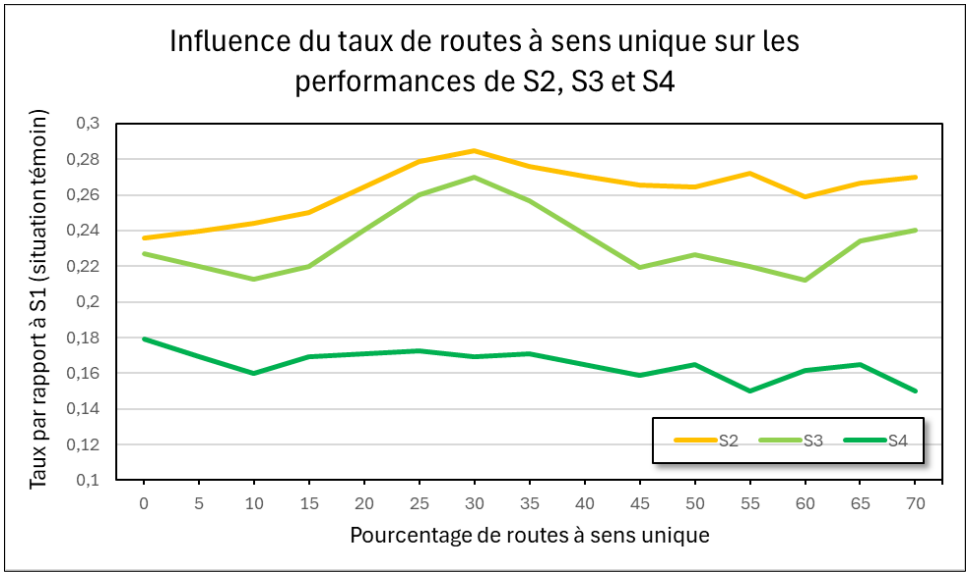


Figure 8 - Influence de la proportion de route à sens unique sur les performances de S1, S2, S3 et S4 entre elles.

Conclusion

Nous avons conclu de cette expérience que la proportion de routes à sens unique dans le réseau routier n'avait pas d'influence sur les performances de nos situations entre elles. Cependant, il nous restait à tester l'influence du rayon de S3 sur ses performances par rapport à S1, S2 et S4, car cette variable n'étant pas dépendante d'une ville, il y avait de très fortes chances que celle-ci ait une influence sur nos simulations.

Analyse de l'influence du rayon de vision de S3 sur ses performances

Objectif de l'expérience

Cette expérience a pour but de vérifier l'influence du rayon de vision de S3 sur ses performances par rapport à S2 et S4.

Protocole expérimental

Pour cette expérience nous avons repris le protocole expérimental de l'expérience *PFE23-R-198_PE_Optimisation_Simulation-graphes*. Cependant la boucle faisant varier le pourcentage de perturbation de 0 à 5% sera remplacé par une boucle faisant varier le rayon de vision de S3 de 0 à 1000 mètres avec un pas de 50 mètres. La proportion de perturbation sera elle fixe à 5% car c'est la situation où les écarts sont les plus grands.

De la même manière que dans l'expérience de la simulation des 4 situations, nous avons utilisé les CSV créé et analysé les résultats des moyennes de chaque situation pour chaque rayon différent de S3. Encore une fois il ne faut pas oublier de calculer l'écart type de chaque série pour s'assurer que nos moyennes ont un sens.

Observation

Nous avons observé que le temps de trajet de S3 est en fait strictement compris entre celui de S2 et celui de S4. C'est assez intuitif lorsque l'on y pense, car S2 représente en fait la situation où S3 a un rayon de 0 mètre de distance, et la situation S4 est celle où S3 a un rayon infini de distance.

Données

Les données de nos différentes observations sont disponibles sur le git de notre projet [4].

Nous avons pu tracer le graphique ci-dessous qui résume les résultats obtenus pour la ville de Montpellier à la suite de cette expérience :

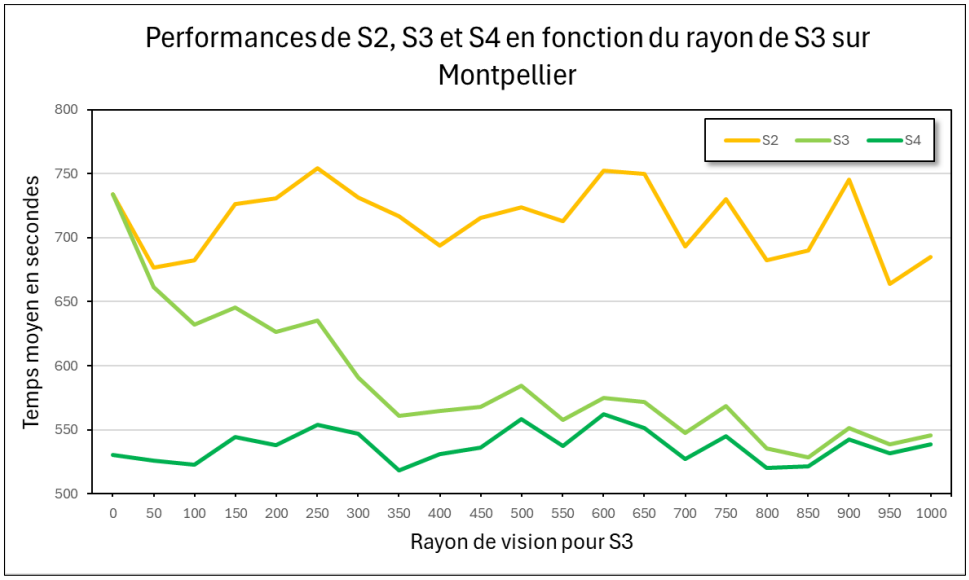


Figure 9 - Influence du rayon de vision de S3 sur ses performances par rapport à S2 et S4 pour la ville de Montpellier.

Analyses préliminaires

Nous nous sommes donc rendu compte que peu importe la taille de la ville, le temps de trajet de S3 est à mi-chemin entre celui de S2 et de S4 à partir de 300 mètres de rayon. Il se rapproche beaucoup des performances de S4 à 500 mètres de rayon. Enfin, S3 converge vers S4 autour des 1000 mètres de rayon.

Conclusion

Le rayon de S3 exerce donc une grande influence sur ses performances, et plus il est élevé, plus cette situation est performante, approchant la perfection autour du kilomètre de rayon.

Nous sommes cependant conscients que d'un point de vue réaliste, il faudrait des voitures dans toutes les rues, connectées entre elles, pour que S3 fonctionne comme nous l'avons prévu. Dans la réalité, avec la situation S3, nous n'aurions pas accès à tous les changements dans un rayon autour de nous, mais seulement à certains. C'est pourquoi on pourrait approfondir le sujet en implémentant nos algorithmes dans des simulations avec plusieurs voitures communiquant entre elles.

Conclusion globale

Nous avons réussi à valider notre hypothèse à l'aide de nos simulations. En effet, nous observons bien cette inégalité en termes de temps de trajet : S1 ≥ S2 ≥ S3 ≥ S4.

Nous avons découvert qu'il existe des exceptions avec notre situation S2 qui, dans de rares cas, produit des chemins plus longs que S1. Cela met en valeur la nécessité de notre système à communiquer sur un rayon large pour éviter ce genre d'exception au maximum, en misant sur la prévoyance.

Nous avons mis en évidence dans les résultats de notre protocole expérimental *PFE23-R-198_PE_Optimisation_Génération-graphes* que la taille d'une ville n'avait pas d'influence sur sa structure intrinsèque et que la seule véritable variable fluctuante d'une ville à l'autre était sa proportion de routes à sens unique.

Nous avons pu voir à travers nos diverses simulations qu'effectivement, comme on s'y attendait, la taille d'une ville n'avait pas d'influence sur les performances de nos situations entre elles (bien que la taille d'une ville puisse bien évidemment faire augmenter ou diminuer le temps de trajet de chaque situation, mais en les comparant les unes aux autres, cela n'a pas d'influence). De manière plus inattendue, nous avons aussi remarqué que la proportion de routes à sens unique n'avait pas d'influence non plus sur les performances de nos situations entre elles.

Enfin, nous avons pu mettre en lumière l'importance qu'il y avait à faire communiquer nos 3 systèmes, puisque plus le rayon de communication et donc de vision de S3 est grand, plus il est performant, se rapprochant de notre situation idéale S4 autour des 1000 mètres de rayon.

Voilà qui termine notre étude sur cette partie optimisation, montrant clairement qu'un tel système sur nos voitures pourrait grandement diminuer nos temps de trajet sur la route, et plus particulièrement dans nos villes où le trafic tend à devenir de plus en plus dense avec l'augmentation du parc automobile.

Références

[1] https://github.com/alexismarieece/PFE_RECHERCHE_VOITURE/tree/theo/Sources/map

[2] https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.weighted.dijkstra_path.html

[3] https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128

[4] https://github.com/alexismarieece/PFE_RECHERCHE_VOITURE/tree/theo/Sources/simulations