



MASTER 1 AGENTS DISTRIBUÉS, ROBOTIQUE, RECHERCHE
OPÉRATIONNELLE, INTERACTION, DÉCISION

Projet ANDROIDE

Rapport

Auteurs :

Ella DIJKSMAN

Osmane EL MONTASER

Tony HU

Lou MOULIN-ROUSSEL

N° d'étudiant :

21205961, 21201287,

21212455, 3676580

Olivier Spanjaard

23 mai 2023

Table des matières

1	Présentation de notre projet	1
2	État de l'art	2
3	Contribution	3
4	Génération et exploitation du graphe	4
4.1	Génération	4
4.2	Exploitation	4
5	Algorithmie	6
5.1	Problème multi-objectif, dominance de pareto	6
5.2	Algorithme A* multi-objectif (MOA*)	6
5.2.1	Choix de l'heuristique	7
5.2.2	Queue de priorité	8
5.2.3	Détermination du front de Pareto	8
5.3	Méthodes d'optimisation	8
5.3.1	Réduction du nombre de nœuds à explorer	8
5.3.2	Assouplissement de la dominance de Pareto, méthode par quadrillage $\log(1 + \epsilon)$ [8]	9
5.3.3	Distance maximum	11
6	Interface et application	12
7	Comparaison avec GeoVelo	14
8	Conclusion	16
A	Annexe 1 : Rappel du cahier des charges	18
B	Annexe 2 : Manuel utilisateur	27
B.1	Installation de Python et des pré-requis	27
B.2	Démarrage de l'application sous Windows	27
B.3	Génération de l'itinéraire	28

1 Présentation de notre projet

Notre projet se déroule dans le cadre de notre formation de master informatique parcours ANDROIDE à l'université Paris VI. Il porte sur la planification d'itinéraires cyclistes protégés. Plus exactement, il vise à développer une application permettant de fournir un itinéraire sécurisé pour cyclistes tout en prenant en compte les contraintes fournies par l'utilisateur.

Contrairement au problème du plus court chemin classique, le contexte est ici multi-objectifs puisque plusieurs critères doivent être pris en compte dans le calcul des itinéraires, à savoir la distance et la sécurité. Il n'existe pas qu'une seule solution qui domine strictement toutes les autres, mais un ensemble de solutions qui ont des avantages sur d'autres en certains points et des inconvénients sur d'autres points. La difficulté est donc de trouver un bon compromis sous une contrainte de temps de quelques secondes.

Afin de mener à bien ce projet, nous avons pu établir trois grandes étapes de développement : la **génération du graphe** de Paris *intra-muros*, graphe qui sera exploité par l'**algorithme A* multi-objectifs**, afin d'ensuite procéder à la **création de l'application**.

Ce projet est encadré par Olivier Spanjaard, maître de conférence dans notre université.

Pour accéder à l'ensemble de notre travail, voici le lien vers le dépôt git du projet : <https://github.com/Osmane-EL-MONTASER/PROJET-ANDROIDE>.

2 État de l'art

Avant toute chose, faisons un état de l'art sur les outils déjà existants. La planification de trajet est désormais très répandue et beaucoup d'applications offrent ce type de service tout en faisant des distinctions résidant par exemple dans leurs priorités. Nous nous sommes focalisés sur certains d'entre eux afin de s'inspirer de leurs spécificités. Ces outils sont les suivants : GeoVelo, Google Maps, Komoot et Strava. Bien que la finalité soit la même, ils ont chacun leurs spécificités, les rendant plus ou moins bien adaptés à un certain utilisateur selon le profil de celui-ci et selon ses besoins. Décrivons-en de façon non-exhaustive les fonctionnalités principales.

GeoVelo est certainement l'outil le plus proche de celui qui a été développé durant le projet puisqu'il met en avant la planification d'itinéraires cyclistes. Ce dernier est probablement l'un des plus spécifiques. Il fait la distinction entre différents types de vélos afin de s'adapter au mieux, mais ne supporte pas d'autre moyen de transport. Cet outil offre également la possibilité de choisir un itinéraire sécurisé.

Google Maps, quant à lui, possède une grande variété en termes de moyens de transports disponibles, couvrant quasiment tous les moyens de transport. De plus, l'outil est aussi très informatif en ce qui concerne les commodités (possibilité d'afficher sur la carte les restaurants, les parkings à vélos, les hôtels, les parcs etc.). S'ajoute à cela le fait qu'il dispose d'une capacité d'actualisation permettant d'informer ses utilisateurs de problèmes de trafic en direct, tels que des accidents, des bouchons ou encore des travaux. Google Maps est certainement l'outil le plus développé parmi ceux que l'on compare. Cependant, ce qui nous intéresse ici est la variété de fonctionnalités disponibles pour les vélos, qui elle reste plutôt limitée.

Komoot possède par exemple des spécificités non-disponibles sur Google Maps, comme la possibilité d'entrer sa condition physique afin qu'il s'y adapte. Cette particularité va de pair avec le fait qu'il soit spécialement créé pour les activités sportives puisqu'il ne propose aucun moyen de transport motorisé. Il existe également une option de live tracking.

Le live tracking est d'ailleurs une des spécificités qui caractérisent particulièrement **Strava**, qui a misé sur un aspect "réseau social" où les utilisateurs peuvent suivre les activités mises en ligne par les autres.

Pour ce qui est de l'algorithmie, plusieurs articles et thèses ont été publiés sur la construction de chemins avec plusieurs objectifs. On trouve parmi eux [1] mais aussi des articles plus récents comme [2] et [3].

3 Contribution

Notre projet est étroitement lié avec la thèse [4] du co-fondateur de **Geovelo**. Notre travail se limitera cependant à moins d'objectifs puisque l'on ne s'occupera que de la distance ainsi que de la sécurité du trajet proposé.

L'application utilise un nouvel algorithme présenté dans une thèse récente [5] qui opte pour une approche différente pour le calcul des solutions optimales.

Enfin, nous tenons à profiter de cette occasion pour exprimer nos sincères remerciements à notre encadrant de projet, M. Spanjaard. Sa contribution inestimable et son soutien constant ont grandement enrichi notre expérience dans la réalisation de ce projet.

Tout au long de ce parcours, nous avons été inspirés par l'expertise et les connaissances approfondies de M. Spanjaard. Ses conseils éclairés, sa disponibilité et sa patience ont été d'une valeur inestimable. Grâce à ses orientations judicieuses, nous avons pu surmonter les obstacles et progresser de manière significative dans la réalisation du projet.

4 Génération et exploitation du graphe

4.1 Génération

Dans un premier temps, nous avons pour objectif de couvrir toute l'Île-de-France à partir d'un graphe que l'on générerait à la volée. Cependant, nous avons rencontré des problèmes en termes temps de traitement des requêtes et d'exécution. En effet, parcourir un nœud avec un algorithme tel que A* prenait environ 0.1s, or un trajet de plusieurs kilomètres est constitué de milliers de nœuds à explorer. Par exemple, parcourir 60 000 nœuds nous aurait pris 1h40. Il nous a donc fallu trouver une autre solution.

Afin de remédier à cela, nous avons utilisé les bibliothèques **NetworkX** et **OSMnx** pour la génération du graphe que l'on a ensuite enregistré. Nous nous sommes limités à Paris *intra-muros*. Par la suite, nous nous sommes servis de **pickle** afin de charger le graphe. Le fait de se servir d'un graphe pré-généré nous a permis un grand gain de temps.

4.2 Exploitation

Une fois que le graphe était généré, nous avons pu passer à la classification des routes. Le graphe créé par **OSMnx** a été parcouru et nous avons regardé les spécificités de chaque arête grâce à **Overpass**. Cela nous a permis de classer les routes et de mettre un poids sur chaque arête. Voici comment nous avons décidé de délimiter les niveaux de sécurité :

- **Très bon** : Pistes cyclables protégées, *i.e.* séparées de la chaussée (chemin spécialement conçu pour les cyclistes et interdit au reste).
- **Bon** : Bandes cyclables non protégées le long de la route, axes partagés (entre piétons/vélos/voitures), ou routes dont la limite de vitesse n'excède pas les 50km/h.
- **Mauvais** : Routes où la limite de vitesse est de plus de 50km/h, routes sans bande cyclable ni signalisation.
- **Très mauvais** : Le restant des routes.

Il va sans dire que les autoroutes ne pourront pas être empruntées, mais notre carte se limitant à Paris *intra-muros*, cela ne pose pas de problème.

Cette classification donne par la suite lieu à la possibilité d'étiqueter les nœuds au fur et à mesure de l'exploration du graphe. Une étiquette est alors représentée par un vecteur ligne de quatre composantes. La première composante correspond au nombre de kilomètres à pédaler sur une **très mauvaise** route ou mieux dans l'itinéraire en cours de construction. Pareillement, la deuxième composante correspond au nombre de kilomètres sur une **mauvaise** route ou mieux, puis la troisième

au nombre de kilomètres sur une **bonne** route ou mieux et enfin, la dernière, au nombre de kilomètres sur une **très bonne** route ou mieux (c'est-à-dire sur une piste cyclable).

Pour récupérer les nœuds de départ et d'arrivée à partir de l'adresse, nous avons utilisé l'API **Nominatim**, qui utilise les données d'OpenStreetMap. Cela nous a permis de trouver les coordonnées GPS (latitude et longitude) d'une adresse fournie. Ainsi, le premier nœud du trajet correspond au point de coordonnées correspondant à l'adresse de départ, et le deuxième nœud correspond au nœud le plus proche de celui-ci dans le graphe. Le fait de relier directement le nœud de départ au nœud le plus proche révèle un léger manque de précision auquel il nous aurait été trop compliqué de pallier. En effet, il aurait fallu récupérer les coordonnées et les numéros des nœuds qui correspondent à chaque bâtiment avec **Overpass** (avec le nombre de bâtiment qu'il y a dans Paris, il nous a paru préférable de laisser cette imperfection).

5 Algorithmie

5.1 Problème multi-objectif, dominance de pareto

Contrairement à un problème de décision mono-objectif, pour un problème de décision multi-objectif, il n'existe en général pas une seule alternative optimale mais plusieurs. Dans notre cas, ces différentes alternatives s'expliquent par le fait que l'on ne recherche pas uniquement à minimiser la distance parcourue, mais également le taux de sécurité d'un trajet. On peut donc arriver à des cas où l'on doit se décider à choisir entre un trajet plus court ou un trajet plus sécurisé. Pour établir quelles sont ces différentes alternatives optimales que l'on doit choisir, nous utilisons la relation de la dominance de Pareto.

Dominance de Pareto

Pour toute paire de solutions $(x, y) \in X^2$, on dit que x domine y au sens de Pareto si $y_i \geq x_i$ pour $i = 1, \dots, n$ et il existe $k \in \{1, \dots, n\}$ tel que $y_k > x_k$. On note cette relation $x \succ_P y$.

L'ensemble des solutions optimales est alors l'ensemble des solutions non-dominées au sens de Pareto, c'est-à-dire l'ensemble des $x \in X$ tel qu'il n'existe pas de $y \in X$ tel que $y \succ_P x$. Cet ensemble est appelé le front de Pareto du problème et est noté X_{Par} .

Il est possible d'établir d'autres relations de dominance pour simplifier le problème, comme par exemple la dominance lexicographique ou la dominance induite par une somme pondérée des valeurs des critères. Mais elles ne permettent pas de représenter l'ensemble des compromis possibles entre les objectifs. C'est pourquoi nous utiliserons la dominance de Pareto.

5.2 Algorithme A* multi-objectif (MOA*)

Pour trouver un plus court chemin dans un problème mono-objectif, l'algorithme A* est l'un des algorithmes classique afin de résoudre ce type de problème. L'idée de cet algorithme est d'utiliser une fonction heuristique minorante permettant d'estimer la distance de chacun des nœuds du graphe au nœud but.

Pour un nœud n , en additionnant cette heuristique au coût du meilleur chemin déjà connu pour arriver à ce nœud, on obtient une estimation du coût d'un chemin du nœud initial au nœud but passant par ce nœud. Si cette estimation est supérieure au coût d'un chemin déjà trouvé, il n'est pas nécessaire de poursuivre l'exploration de ce chemin. Cet algorithme permet donc d'orienter la recherche du plus court chemin et de réduire le nombre de chemins à explorer.

Afin de pouvoir utiliser cet algorithme pour notre problème, une recherche de plus court chemins multi-objectifs, nous en avons utilisé une adaptation, la méthode MOA* (*Multi-Objective A**), pour laquelle nous avons des chemins optimaux au sens de Pareto.

Algorithm 1: A* Multiobjectif

```

Input:  $G = (V, E)$ ,  $startNode$ ,  $endNode$ ,  $h$ 
PriorityQueue  $Q \leftarrow startNode$ ;
 $P \leftarrow dict()$ ;
while not  $Q.isempty()$  do
     $current \leftarrow Q.pophead()$  ;      /* Le nœud au cout le plus faible
    lexicographiquement */
    if  $endNode$  has been found then
        if  $current$  is strictly dominated by any cost in  $P[endNode]$ 
        then
            continue;
        end
    end
    for each  $neighbor$  in  $current$  do
        if  $d(neighbor)$  not dominated in  $P[neighbor]$  then
             $P[neighbor].append(neighbor)$ ;
             $Q.add(neighbor)$ 
        end
    end
end
return  $P$ 

```

5.2.1 Choix de l'heuristique

Pour garantir l'optimalité des solutions trouvées, il est nécessaire que l'heuristique utilisée soit minorante, c'est-à-dire que la valeur de cette fonction à un nœud soit inférieure ou égale au coût du chemin optimal de ce nœud au nœud but. Dans le cas d'un problème d'optimisation multi-objectif, cela signifie que le vecteur de coût retourné par l'heuristique en un nœud doit être non-dominé au sens de Pareto par tous les coûts des chemins de ce nœud au nœud but.

Nous avons donc fait le choix d'utiliser une heuristique fondée sur la distance à vol d'oiseau d'un nœud au nœud but. En notant $d(n)$ cette distance à vol d'oiseau du nœud n au nœud but, nous avons utilisé la fonction $h(n) = [d(n), 0, 0, 0]$ comme heuristique. En effet, la longueur d'un chemin partant d'un nœud n et arrivant au nœud but est forcément plus longue que $d(n)$, or le premier objectif de nos vecteurs

de coût correspond à la distance totale parcourue. Les valeurs des vecteurs de coût sur les autres objectifs étant positives ou nulles, étant des distances, cette heuristique est bien minorante.

5.2.2 Queue de priorité

Afin de pouvoir déterminer quel est le prochain nœud ouvert à explorer, nous utilisons une implémentation par queue de priorité avec les distances et heuristiques obtenues.

Dans notre cas, nous souhaitons explorer en priorité les nœuds ouverts qui possèdent le coût total (distance parcourue totale + heuristique en ce nœud la) le plus faible.

Afin de pouvoir réaliser la queue de priorité, nous utilisons le module **heapq** [6] de Python, qui fournit une implémentation de l'algorithme de queue de priorité avec une structure de données arborescente.

5.2.3 Détermination du front de Pareto

L'algorithme A* repose sur la détermination du plus court chemin allant du nœud initial à chaque nœud exploré. Dans l'algorithme MOA*, cela se traduit par la détermination de l'ensemble des chemins partant du nœud initial non-dominés au sens de Pareto, et ce, pour chaque nœud exploré. Il faut donc calculer et mettre à jour le front de Pareto pour chaque nœud exploré au cours des itérations de l'algorithme. Pour optimiser la rapidité des calculs, nous avons décidé de faire appel à une bibliothèque Python prévue à cet effet : la bibliothèque **paretoset** [7].

5.3 Méthodes d'optimisation

5.3.1 Réduction du nombre de nœuds à explorer

Une des grandes différences entre l'Astar mono-objectif et le multiobjectif est la multiplication des nœuds à explorer. En effet, en augmentant le nombre d'objectifs et de critères lors de l'exploration, il n'est plus évident de savoir quel chemin est le plus optimal. Cela a pour conséquence que dès que la distance entre le nœud initial et le nœud but est supérieure à quelques centaines de mètres, le nombre de chemins Pareto-optimaux est très grand.

On arrive parfois à des cas où l'on a un nombre élevé de ces chemins qui ont des coûts très proches et seront quasiment équivalents pour un cycliste, nous pouvons donc considérer qu'ils sont redondants. Pire, un chemin peut être légèrement meilleur selon un objectif mais beaucoup moins bon selon les autres et quand même être renvoyé par l'algorithme alors qu'il est évidemment moins "bon" que d'autres. Ce grand nombre de chemins ralentit considérablement l'exécution de l'algorithme et rend difficile l'interprétation des résultats. Afin d'optimiser l'algorithme, il est donc nécessaire de chercher à réduire le nombre de chemins à explorer, tout en gardant une certaine qualité dans les chemins retournés.

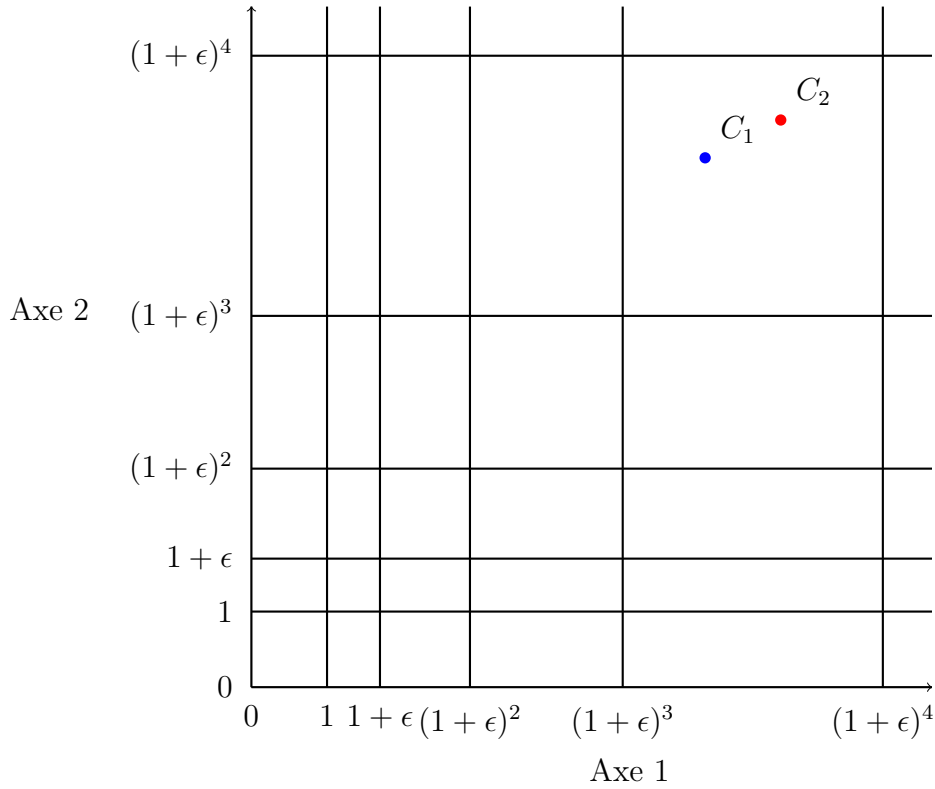
5.3.2 Assouplissement de la dominance de Pareto, méthode par quadrillage $\log(1 + \epsilon)$ [8]

Une des méthodes afin d'accomplir cette tâche est de modifier la relation de dominance utilisée entre les vecteurs de coûts. Le principe est d'assouplir la relation de Pareto en considérant un quadrillage en dimension 4 du domaine des vecteurs coûts. Deux vecteurs sont alors considérés comme équivalents s'ils appartiennent au même hypercube dans ce quadrillage.

Nous voulons utiliser cette nouvelle relation à chaque étape de l'algorithme afin de réduire le nombre de nœuds à découvrir durant tout le long de l'exploration en éliminant les coûts/chemins qui sont très proches, et donc, redondants.

Cependant, un problème survient lorsque l'on utilise un quadrillage de taille fixe pour toutes les grilles : un pas fixe trop petit ne permettra pas d'éliminer assez de chemins lorsqu'on arrivera à des nœuds éloignés du nœud initial mais un pas trop grand pourrait mener à une perte de qualité, éliminant des chemins qui font partie des différents chemins optimaux, dont notamment pour les nœuds initiaux. Par exemple, 10m peut représenter une différence significative pour des chemins arrivant en un nœud proche du nœud initial mais pour un chemin de plusieurs centaines de mètres ce n'en est pas une. En éliminant dès le début des nœuds qui pourraient faire partie de chemins optimaux, on a une grande perte de qualité dans le nombre de chemins possibles, pouvant arriver à des situations où qu'un seul chemin est retourné à la fin de l'algorithme, si notre distance fixe est trop grande.

Il est donc nécessaire de trouver un moyen pour que le quadrillage et la taille des grilles soit adapté pour que l'on puisse garder seulement les chemins importants, tout en éliminant ceux qui sont redondants. Afin de pouvoir résoudre ce problème, nous avons voulu établir une relation de proportionnalité entre le paramètre ϵ et la distance déjà parcourue en un nœud, passant donc par une autre méthode pour calculer la taille des différentes grilles du quadrillage, celle de $\log(1 + \epsilon)$



Représentation de l'échelle utilisée en deux dimensions

Par exemple, avec cette relation de dominance et ce quadrillage, les coûts C_1 et C_2 représentés ci-dessus ne se dominent ni l'un ni l'autre et sont considérés équivalents. Nous n'ajoutons au tas des chemins à explorer pour un nœud que les chemins dont les coûts sont non-dominés et non-équivalents à ceux des chemins déjà trouvé pour ce nœud. Pour un nœud n , s'il existait deux chemins y arrivant de coût C_1 et C_2 et que l'un de ces coûts se trouvait déjà dans le tas des chemins à explorer, l'autre n'y serait pas rajouté.

Implémentation

Pour pouvoir continuer à utiliser la bibliothèque *pareto*set, nous modifions les coûts en amont de la mise à jour du front de Pareto en un nœud de la manière suivante :

$\forall x$ vecteur coût, en notant \tilde{x} le coût modifié,
 $\forall i \in \{1, \dots, 4\}, \tilde{x}_i = \lfloor \frac{\log(x_i)}{\log(1+\epsilon)} \rfloor$

Notre nouvelle relation de dominance revient alors à la dominance de Pareto sur ces vecteurs modifiés. Nous pouvons donc utiliser la bibliothèque *pareto*set avec ces coûts modifiés.

De plus, le choix d' ϵ est très important. Plus il est grand, plus l'algorithme s'exécutera rapidement, car plus de chemins seront éliminés. On court alors le

risque d'en éliminer trop et de renvoyer une solution non-représentative de l'ensemble du front de Pareto. Mais un ϵ trop petit ne permettra pas d'améliorer le temps d'exécution de l'algorithme.

Limites

Même si cette méthode permet en outre d'accélérer l'exploration en ne prenant pas en compte les chemins redondants, elle ne nous garantit pas de trouver tous les chemins Pareto optimaux. En effet, comme on l'a vu sur le graphique précédent, si un chemin de coût équivalent a déjà été trouvé, un nouveau chemin ne sera pas ajouté au tas des chemins à explorer, et ce, même si ce nouveau chemin domine l'ancien au sens de Pareto. En éliminant des chemins au cours de l'algorithme, nous risquons de ne pas explorer un chemin Pareto optimal.

Il est donc nécessaire de choisir un ϵ approprié à nos besoins. Si les nœuds de départ et d'arrivée sont relativement proches, nous pouvons choisir un ϵ faible, qui permettra de réduire la taille des grilles et donc avoir des solutions de meilleure qualité. Cependant, si la distance est trop élevée, un ϵ trop petit aura pour conséquence une réduction trop faible du temps d'exécution, ne coupant pas assez de chemins lors de l'exploration. Il est donc nécessaire, s'il l'on souhaite exécuter le programme en un temps convenable, d'augmenter ϵ , en perdant en contrepartie en qualité.

5.3.3 Distance maximum

Afin de réduire le nombre de nœuds à explorer, il est aussi possible de choisir une distance maximale entre le chemin le plus court trouvé en termes de distance total et tout autre nouveaux chemins. En effet, il n'est pas nécessaire généralement de trouver des chemins qui pourraient être plus sécurisés sur quelques mètres en plus, mais qui est plusieurs de kilomètres plus longs que le chemin le plus court que l'on aurait trouvé.

Sur des distances plus longues, le nombre de nœuds restants dans la queue de priorité peut être très élevé, et il n'est pas toujours intéressant de chercher encore des chemins qui seraient beaucoup plus longs que le plus court que l'on a déjà trouvé, quelle que soit l'amélioration en termes de sécurité.

6 Interface et application

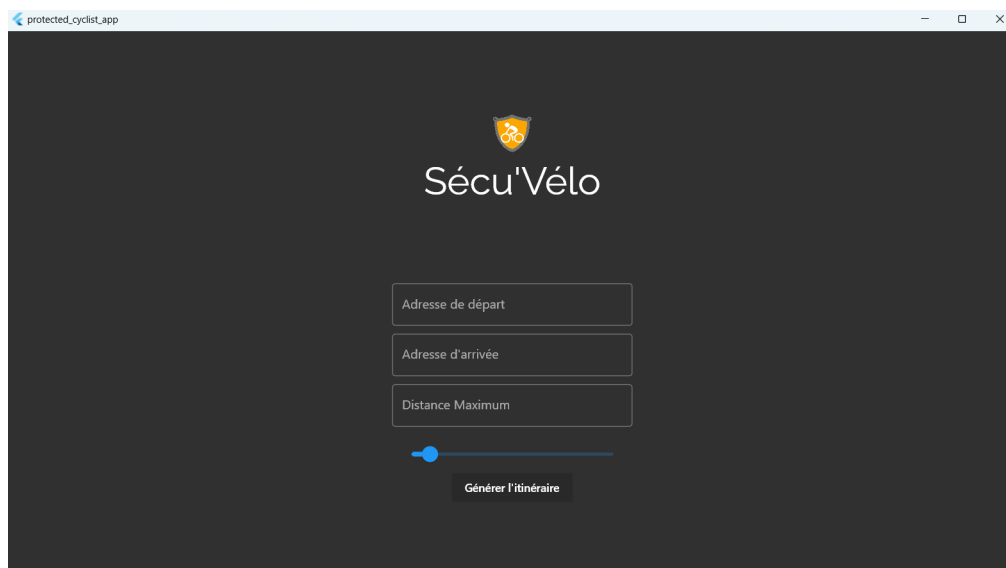
Pour la création de l'application, nous avons fait le choix de passer par **Flutter** qui est un kit de développement logiciel d'interface utilisateur open-source créé par **Google**. Son utilisation est simple et multi-plateforme : il permet de développer des applications sur Windows, Linux, macOS, iOS, Android etc. sans que l'on ait nous-même à s'adapter à chacun des systèmes d'exploitation.

Afin d'éviter de recoder A* pour la génération des itinéraires, nous avons pu également nous servir d'une API. Une API est un serveur qui va effectuer des tâches pour l'utilisateur, donc dans notre cas, qui va effectuer des itinéraires.

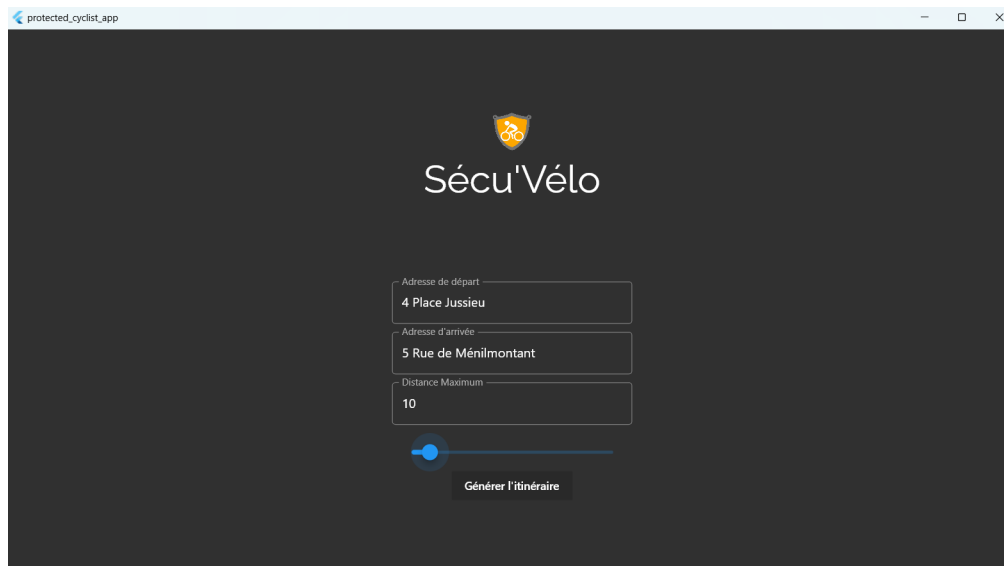
Nous avons fait une requête **http** sur le serveur en lui envoyant les adresses de départ et d'arrivée fournies par l'utilisateur. L'application n'effectue aucun calcul mais fait seulement office d'interface, puisque l'API s'occupe alors de la génération de l'itinéraire, du calcul de la distance et du calcul du temps de trajet en supposant que la vitesse moyenne d'un cycliste dans Paris soit de 15km/h. Les adresses sont converties en points dans le graphe multi-objectifs. L'API renvoie alors ses résultats à l'application sous forme de dictionnaire. Nous le convertissons ensuite en **Json** pour nous permettre de le parser plus simplement en **dart**.



Pour l'affichage, *i.e.* l'interface, nous utilisons la bibliothèque **flutter_map** car elle nous permet d'aisément afficher une carte et d'interagir avec.

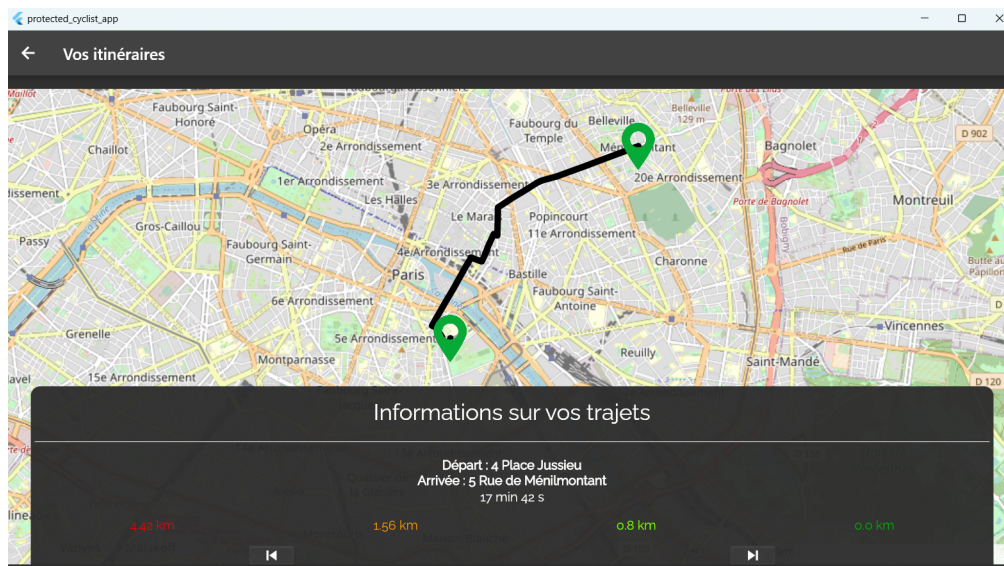
Procédons à un exemple du fonctionnement de l'application **Sécu'Vélo**. A l'ouverture de l'application, on arrive sur une fenêtre comme celle-ci :



On entre les adresses de départ et d'arrivée, puis on saisit la distance maximale que l'on est prêt à parcourir. Ensuite, on peut également déplacer le curseur à notre guise. Celui-ci correspond à ϵ dans notre heuristique de quadrillage. Plus ϵ est élevé, plus l'algorithme va élaguer, et donc moins il proposera de chemins (et plus la génération de l'itinéraire sera rapide). Dans cet exemple, voici la saisie de l'utilisateur :



Celle-ci nous propose trois itinéraires différents, que l'on peut voir en cliquant sur les flèches  et . Un itinéraire est proposé comme ceci :



On peut voir le temps de trajet ainsi que la distance totale du trajet (la valeur en rouge). Le détail des distances à pédaler sur des chemins de chaque niveau de sécurité est donné grâce au même code couleur explicité en 4.2.

7 Comparaison avec GeoVelo

Nous avons décidé de comparer Geovelo à notre application en les exécutant sur les mêmes adresses de départ et d'arrivée. L'image ci-dessous illustre les résultats renvoyés par Geovelo.

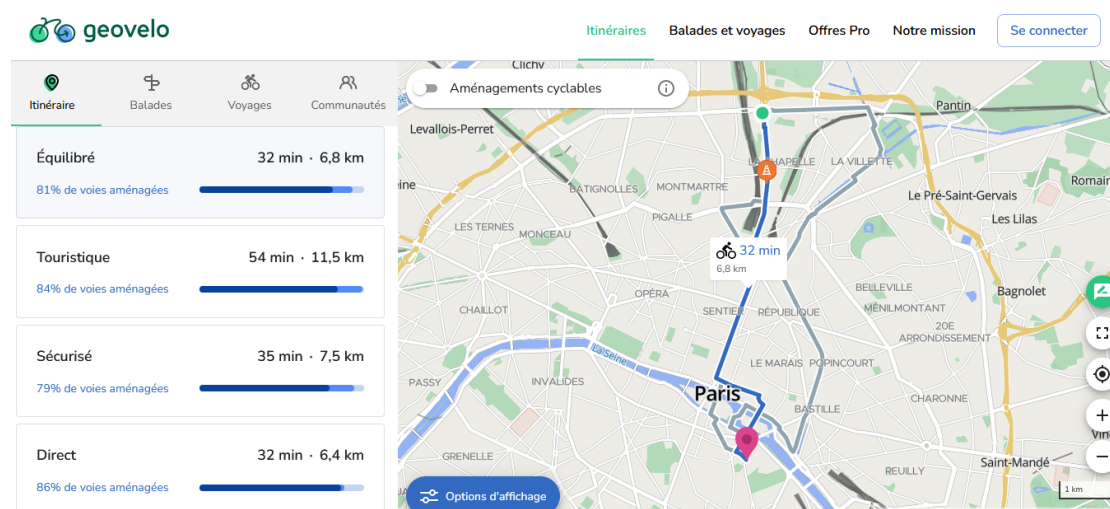


FIGURE 1 – Itinéraires renvoyés par Geovelo

Distance totale en km	Distance parcourue en km			
	Niveau 4	Niveau 3	Niveau 2	Niveau 1
6.65	0.29	4.82	1.54	0
6.71	0.54	4.9	1.27	0
6.76	1.46	4.17	1.13	0
6.77	1.46	4.26	1.05	0
6.78	1.46	4.37	0.95	0
6.8	1.46	4.53	0.81	0
6.85	1.82	4.25	0.78	0
6.92	1.81	4.45	0.66	0
7.11	2.25	4.16	0.7	0
7.2	2.48	4.17	0.55	0

FIGURE 2 – Itinéraires renvoyés pour $\epsilon = 1$

Distance totale en km	Distance parcourue en km			
	Niveau 4	Niveau 3	Niveau 2	Niveau 1
6.65	0.29	4.82	1.54	0
6.71	0.54	4.9	1.27	0
6.76	1.46	4.17	1.13	0
6.77	1.46	4.26	1.05	0
6.78	1.46	4.37	0.95	0
6.8	1.46	4.53	0.81	0
6.85	1.82	4.25	0.78	0
6.92	1.81	4.45	0.66	0
7.11	2.25	4.16	0.7	0
7.2	2.48	4.17	0.55	0

FIGURE 3 – Itinéraires renvoyés pour $\epsilon = 0.3$

Il est difficile de pouvoir comparer le niveau de sécurité entre les trajets obtenus avec **Geovelo** et ceux de notre application. En effet, nous avons établi nos propres critères pour déterminer le niveau de sécurité de chaque voie et nous ne connaissons pas les critères utilisés par **Geovelo**. Nous n'utilisons donc pas les mêmes critères afin de déterminer le niveau de sécurité que **Geovelo** et la route sécurisée de **Geovelo** ne précise pas pourquoi cet itinéraire est sécurisé.

Nous remarquons cependant que notre itinéraire le plus court est plus long que celui trouvé par **Geovelo** : il est de 6.65 km contre 6.4 km pour **Geovelo**. Cela

est peut-être dû au fait que la méthode par quadrillage $\log(1 + \epsilon)$ n'est pas une méthode exacte.

Nous avons ensuite testé notre algorithme sur la même entrée mais en modifiant le paramètre ϵ . En modifiant le paramètre ϵ de $\epsilon = 0.3$ à $\epsilon = 1$, nous n'observons pas de différence dans les chemins renvoyés. Il faut prendre une valeur qui est plus faible, telle que 0.2 pour améliorer les résultats obtenus. Enfin, les temps d'exécution de notre algorithme sont beaucoup trop longs ; ils sont de plusieurs minutes pour le chemin donné en exemple dans cette partie alors que le temps d'exécution de **Geovelo** est de l'ordre de la seconde.

8 Conclusion

Dans ce rapport, nous avons exposé les différentes étapes de développement ainsi que les résultats obtenus lors de notre projet. Même si l'on peut considérer ces résultats comme satisfaisants au vu du temps alloué à la réalisation du projet, plusieurs pistes d'améliorations restent possibles.

En effet, malgré les efforts faits pour accélérer l'exploration de nos nœuds lors de l'exécution de l'algorithme, celui-ci fournit des résultats à des vitesses bien plus faibles que les autres alternatives qui sont déjà existantes. En effet, le temps de calcul pour trouver des chemins relativement long est bien plus conséquent que lorsque l'on utilise notre algorithme que lorsque l'on utilise **Geovelo** par exemple.

C'est pourquoi il est toujours possible de rechercher et d'implémenter d'autres méthodes qui permettraient d'accélérer l'exécution de notre algorithme de recherche tout en gardant une qualité dans les solutions.

Toutefois, malgré les difficultés et obstacles que nous avons pu rencontrer, nous avons réussi les principaux objectifs du projet qui était d'implémenter la génération de graphes multi-objectif présenté dans la thèse et la génération d'itinéraires à partir de ces graphes.

Références

1. GUDAITIS, M. S., LAMONT, G. B. & TERZUOLI, A. J. *Multicriteria vehicle route-planning using parallel A * search* in *Proceedings of the 1995 ACM symposium on Applied computing* (Association for Computing Machinery, New York, NY, USA, février 1995), 171-176. <https://dl.acm.org/doi/10.1145/315891.315949> (2023).
2. MANDOW, L. A New Approach to Multiobjective A* Search. en.
3. MACHUCA, E. & MANDOW, L. Multiobjective heuristic search in road maps. en. *Expert Systems with Applications* **39**, 6435-6445. ISSN : 0957-4174. <https://www.sciencedirect.com/science/article/pii/S0957417411016939> (2023) (juin 2012).
4. SAUVANET, G. *Recherche de chemins multiobjectifs pour la conception et la réalisation d'une centrale de mobilité destinée aux cyclistes* thèse de doct. (Université François Rabelais de Tours, avril 2011).
5. KLAMROTH, K., STIGLMAYR, M. & SUDHOFF, J. *Ordinal Optimization Through Multi-objective Reformulation* arXiv :2204.02003 [math]. avril 2022. <http://arxiv.org/abs/2204.02003>.
6. *heapq — Heap queue algorithm — Python 3.10.11 documentation* <https://docs.python.org/3.10/library/heapq.html> (2023).
7. TOMMY. *paretoset* original-date : 2020-03-18T10 :33 :22Z. Mai 2023. <https://github.com/tommyod/paretoset> (2023).
8. PERNY, P. & SPANJAARD, O. Near Admissible Algorithms for Multiobjective Search. en.

A Annexe 1 : Rappel du cahier des charges



MASTER 1 AGENTS DISTRIBUÉS, ROBOTIQUE, RECHERCHE
OPÉRATIONNELLE, INTERACTION, DÉCISION

Projet ANDROIDE

Cahier des charges

Auteur :

Ella DIJKSMAN
Osmane EL MONTASER
Tony HU
Lou MOULIN-ROUSSEL

N° d'étudiant :

21205961, 21201287,
21212455, 3676580

Professeur:

Olivier Spanjaard

9 février 2023

Table des matières

1	Présentation de notre projet	1
2	Description de la demande	3
2.1	Algorithmique	3
2.1.1	Créer des itinéraires sécurisés	3
2.1.2	Accélération de la création d'itinéraires sécurisés	3
2.1.3	Affinage de la liste des itinéraires sécurisés proposés	3
2.2	Traitement des données	4
2.2.1	Récupérer des données cartographiques	4
2.2.2	Modélisation des données cartographiques	4
2.3	Interfaçage	4
2.3.1	Accéder à l'application depuis le navigateur web	4
2.3.2	Accéder à l'application depuis le smartphone	4
3	Solutions envisagées ?	5
3.1	Algorithmique	5
3.1.1	Créer des itinéraires sécurisés	5
3.1.2	Accélération de la création d'itinéraires sécurisés	5
3.1.3	Affinage de la liste des itinéraires sécurisés proposés	5
3.2	Traitement des données	5
3.2.1	Récupérer des données cartographiques	5
3.2.2	Modélisation des données cartographiques	5
3.3	Interfaçage	6
3.3.1	Accéder à l'application depuis le navigateur web	6
3.3.2	Accéder à l'application depuis le smartphone	6
4	Résultats attendus	6

1 Présentation de notre projet

Notre projet porte sur la planification d'itinéraires cyclistes protégés. Il vise plus exactement à développer une application permettant de fournir un itinéraire sécurisé pour cyclistes tout en prenant en compte les contraintes fournies par l'utilisateur.

Avant toute chose, nous avons commencé par comparer les outils existants (puis à l'outil qui aura été développé), en étudiant leurs fonctionnalités. Nous avons décidé de se focaliser sur les outils suivants : GeoVelo, Google Maps, Komoot, Strava, Mappy, OSRM. Bien que la finalité soit la même, ils ont chacun leurs spécificités, les rendant plus ou moins bien adaptés à un certain utilisateur selon le profil de celui-ci et selon ses besoins.

Voici un tableau comparatif des caractéristiques majeures de chacun de ces six outils :

Outil	Types de carte	Détails de carte	Modes de transport supportés	Plus
GeoVelo	<ul style="list-style-type: none"> - Standard - Photos aériennes - IGN 	<ul style="list-style-type: none"> - Relief - Parkings à vélo - Contributions - A faire, à voir - Restaurants - Hébergements - Lieux pratiques pour cyclistes 	<ul style="list-style-type: none"> - Vélo classique - Vélo libre-service - VTC - Vélo cargo 	<ul style="list-style-type: none"> - Choix du type d'itinéraire (équilibré, sécurisé, touristique, direct) - Feuille de route pour l'itinéraire avec étapes du trajet et types de route - Profil altimétrique/dénivelés - Durée du trajet avec calories dépensées + CO2 non émis
Google Maps	<ul style="list-style-type: none"> - Standard - Satellite 	<ul style="list-style-type: none"> - Transports en commun - Trafic - Pistes cyclables - Relief - Street view - Incendies - Qualité de l'air - Restaurants - Hôtels - Activités à découvrir - Musées - Pharmacies - Distributeurs de billets 	<ul style="list-style-type: none"> - Voiture - Transports en commun - Piéton - Vélo - Avion 	<ul style="list-style-type: none"> - 3 types d'itinéraires : moins de trafic, moins de virages, "meilleur itinéraire" - Spécification de la densité du trafic et du dénivelé - Possibilité de définir des étapes - Vitesse estimée en fonction des données des utilisateurs
Komoot	<ul style="list-style-type: none"> - Standard - Satellite - Open Street Map - Open Cycle Map 	<ul style="list-style-type: none"> - Chemins de randonnée - Pistes cyclables - Pistes VTT - Trail 	<ul style="list-style-type: none"> - Randonnée - Vélo - VTT - Cyclisme sur route - Course à pied - Gravel - VTT enduro - Alpinisme 	<ul style="list-style-type: none"> - Possibilité d'entrer sa condition physique avec un curseur - Possibilité de choisir entre aller simple et boucle - Possibilité de définir des étapes - Météo du tour pour anticiper - Live tracking
Strava	<ul style="list-style-type: none"> - Standard - Satellite - Hybride 	<ul style="list-style-type: none"> - Points de repos - Heatmap - Lieux fréquentés 	<ul style="list-style-type: none"> - Vélo - Course à pied 	<ul style="list-style-type: none"> - Réseau social - Tracking et analyse des performances sportives - Possibilité d'entrer le dénivelé souhaité et la surface (bitume ou terre) - Système de recommandation de parcours personnalisés basé sur les 3 milliards d'activités réalisées sur Strava - Estimation de la durée de sortie basée sur la vitesse moyenne de l'utilisateur sur les 4 dernières semaines - Carte de chaleur offrant une visualisation des endroits plébiscités - Amélioration des itinéraires grâce aux données des utilisateurs
Mappy	<ul style="list-style-type: none"> - Standard - Simple - Nature - Satellite - Nuit 	<ul style="list-style-type: none"> - Trafic - Crit'air - Pistes cyclables - RER/métro - Hôtels - Parkings - Supermarchés - Stations-service - Bars/café - Banques - Hôpitaux - Bureaux de poste - Fontaines à eau 	<ul style="list-style-type: none"> - Voiture - Moto - Transports en commun - Vélo - Trottinette - Piéton 	<ul style="list-style-type: none"> - Possibilité de définir des étapes - Possibilité de choisir une vitesse de déplacement (lent, normal, rapide) - Possibilité de choisir l'itinéraire le plus court ou l'itinéraire avec le plus de voies cyclables
OSRM	Standard	<ul style="list-style-type: none"> - Pistes cyclables - Chemins de randonnée 	<ul style="list-style-type: none"> - Voiture - Vélo - Piéton 	<ul style="list-style-type: none"> - Moteur de calcul d'itinéraire open source très léger et très efficace - Se démarque des autres moteurs de calcul d'itinéraires en utilisant l'algorithme de contractions hiérarchiques - Choix entre route la plus courte et route la plus rapide

Ces outils ont tous pour format d'exportation entre autres le format GPX, sauf pour OSRM qui exporte ses itinéraires au format GeoJSON.

2 Description de la demande

Dans cette section nous allons vous présenter les différents besoins auxquels nous devrons répondre dans ce projet.

2.1 Algorithmique

2.1.1 Créer des itinéraires sécurisés

Objectif : Automatiser la recherche d'un trajet sécurisé pour les cyclistes.

Description : On tape l'adresse de départ et l'adresse d'arrivée, l'application renvoie une liste d'itinéraires Pareto optimaux. Certains trajets peuvent être très rapides mais moins sécurisés et d'autres très sécurisés mais très longs et d'autres à la fois sécurisés et rapides.

Niveau de priorité : Obligatoire.

2.1.2 Accélération de la création d'itinéraires sécurisés

Objectif : Accélérer la recherche d'un trajet sécurisé pour les cyclistes.

Description : L'algorithme de Dijkstra multi-objectif sera très certainement lent à s'exécuter car Dijkstra explore un très grand nombre de noeuds.

Niveau de priorité : Moyen.

2.1.3 Affinage de la liste des itinéraires sécurisés proposés

Objectif : Proposer une liste moins exhaustive des itinéraires possibles pour que l'utilisateur ne soit pas noyé parmi des dizaines d'itinéraires à comparer à la main.

Description : L'algorithme de Dijkstra multi-objectif retournera très certainement beaucoup trop de chemins Pareto-optimaux non-dominés.

Niveau de priorité : Moyen.

2.2 Traitement des données

2.2.1 Récupérer des données cartographiques

Objectif : Récupérer des données cartographiques pour pouvoir créer des itinéraires dessus.

Description : Dans un premier temps, il faudra récupérer les données en ligne via une API. Dans un deuxième temps, lorsque nous aurons besoin de faire davantage de requêtes, nous devrons utiliser un serveur de cartographie en local et lui envoyer des requêtes.

Niveau de priorité : Obligatoire.

2.2.2 Modélisation des données cartographiques

Objectif : Convertir les données cartographiques.

Description : Après avoir reçu les données cartographiques, il faudra les convertir de façon à pouvoir utiliser des algorithmes de graphes dessus.

Niveau de priorité : Obligatoire.

2.3 Interfaçage

2.3.1 Accéder à l'application depuis le navigateur web

Objectif : Pouvoir obtenir un itinéraire depuis une interface.

Description : Réaliser un premier prototype fonctionnel sur le navigateur entièrement fonctionnel. Il pourrait convenir en tant que livrable si les contraintes matérielles (serveur) ne peuvent pas être respectées avec le portage sous mobile.

Niveau de priorité : Haut.

2.3.2 Accéder à l'application depuis le smartphone

Objectif : Faire une application mobile.

Description : On lance l'application, les fonctionnalités de l'application doivent rester les mêmes sur le mobile que sur le PC.

Niveau de priorité : Bas.

3 Solutions envisagées ?

3.1 Algorithmique

3.1.1 Créer des itinéraires sécurisés

Solution : Pour la sécurité, il faudra prendre en compte le **type de route** (départementale, rue, autoroute...) ainsi que le **type de revêtement** (pavés, terre, goudron...). Plus tard, on prendra en compte les **grandes intersections** du type Place de l'Étoile à Paris. Pour trouver le chemin optimal, nous nous baserons sur une modélisation par graphe proposée par la thèse *Ordinal Optimization Through Multi-objective Reformulation* de Kathrin Klamroth, Michael Stiglmayra et Julia Sudhoff. Nous devons également dans un premier temps utiliser l'algorithme de Dijkstra multi-objectif pour trouver les chemins Pareto optimaux.

3.1.2 Accélération de la création d'itinéraires sécurisés

Solution : Faire en sorte de récupérer les sommets du graphe à la volée lors de l'exécution du **Dijkstra multi-objectif**. Il faudra aussi utiliser une variante de cet algorithme : le **Dijkstra multi-objectif bidirectionnel**.

3.1.3 Affinage de la liste des itinéraires sécurisés proposés

Solution : Il s'agira de chercher un moyen de trier les chemins Pareto optimaux non-dominés. Par exemple, avec une heuristique ou bien avec d'autres algorithmes après celui de Dijkstra qui permettrait d'obtenir un sous-ensemble de chemins Pareto optimaux non-dominés plus petit.

3.2 Traitement des données

3.2.1 Récupérer des données cartographiques

Solution : Utilisation de l'API d'**OpenStreetMap** pour récupérer les données cartographiques. L'application web qui fera ces requêtes sera codée en Python avec Flask.

3.2.2 Modélisation des données cartographiques

Solution : Il faudra utiliser la bibliothèque **Python-JGraphT** qui permet de modéliser les données cartographiques sous forme de graphe et qui propose une

implémentation de l'algorithme de Dijkstra multi-objectif.

3.3 Interfaçage

3.3.1 Accéder à l'application depuis le navigateur web

Solution : Nous utiliserons une bibliothèque qui permet de créer une application web localement qui s'appelle **Flask**.

3.3.2 Accéder à l'application depuis le smartphone

Contraintes : Il faudra utiliser **Flutter** et son langage **DART** pour accélérer le développement multiplateforme et porter l'application faite au préalable sur le navigateur. Nous devons utiliser aussi un serveur et une API faits-maison pour les requêtes sur OSRM.

4 Résultats attendus

Notre projet comporte plusieurs points importants qui dépendront de la réussite du projet :

- Le projet devra être rendu durant le **mois de mai**.
- L'application doit être capable de générer des itinéraires sécurisés selon les spécifications décrites ci-dessus.
- L'application sera une application web capable de tourner sur un navigateur.
- Le livrable et les documents devront être rendu sur **GitHub** avant la date de rendu.

B Annexe 2 : Manuel utilisateur

Note : Pour le moment, seule la version Windows de l'application est disponible. Mais il est possible de build via Flutter un exécutable pour chaque plateforme avec les bons SDKs installés sur la machine. (Aucun changement de code requis)

B.1 Installation de Python et des pré-requis

Vous devez d'abord installer **Python 3.8.16** et les bibliothèques nécessaires au lancement de l'API :

```
pip install flask scipy six requests networkx paretoset  
geopy matplotlib utm "pandas<2.0.0"
```

Une fois les bibliothèques installées vous pouvez passer directement au lancement de l'application.

B.2 Démarrage de l'application sous Windows

Pour démarrer l'application, il faut d'abord démarrer l'API Python qui se trouve dans le dossier `protected_cyclist_api` avec la commande suivante :

```
python main.py
```

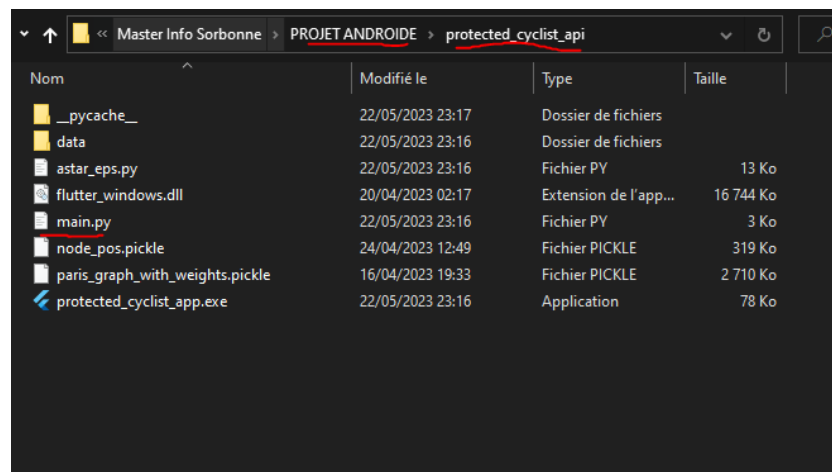


FIGURE 4 – Emplacement du fichier API `main.py`

Note : Il sera toujours nécessaire de lancer l'API sur un PC / Serveur pour faire fonctionner l'application. L'application quant à elle fonctionne sur toutes les plateformes.

Ensuite, vous pouvez exécuter l'exécutable Windows pour lancer l'application :

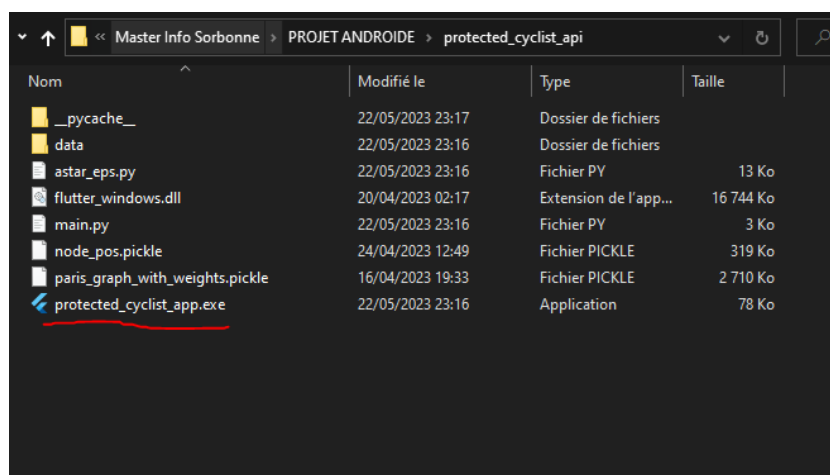


FIGURE 5 – Emplacement de l'exécutable

B.3 Génération de l'itinéraire

Pour générer l'itinéraire, il faut juste remplir les différents champs de texte dédiés avec l'adresse de départ, l'adresse d'arrivée, la distance maximum entre le plus petit chemin trouvé et les autres et le slider qui correspond à un paramètre epsilon qui permet d'assouplir la dominance de Pareto puis de cliquer sur **Générer l'itinéraire** :

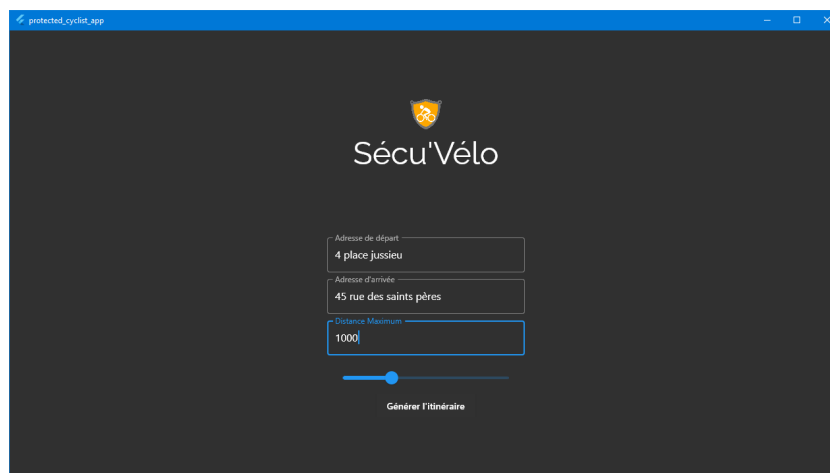


FIGURE 6 – Configuration de l'itinéraire souhaité

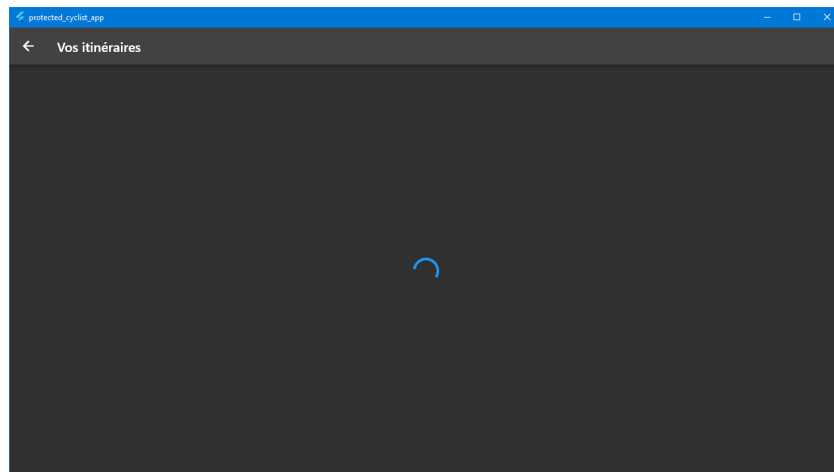


FIGURE 7 – Chargement lors de la génération de l’itinéraire

Après un certain temps, vous pourrez visualiser vos trajets sur la carte. Vous pouvez tirer sur **Informations sur vos trajets** pour avoir les détails du trajet et pour pouvoir afficher les autres trajets avec les boutons fléchés.

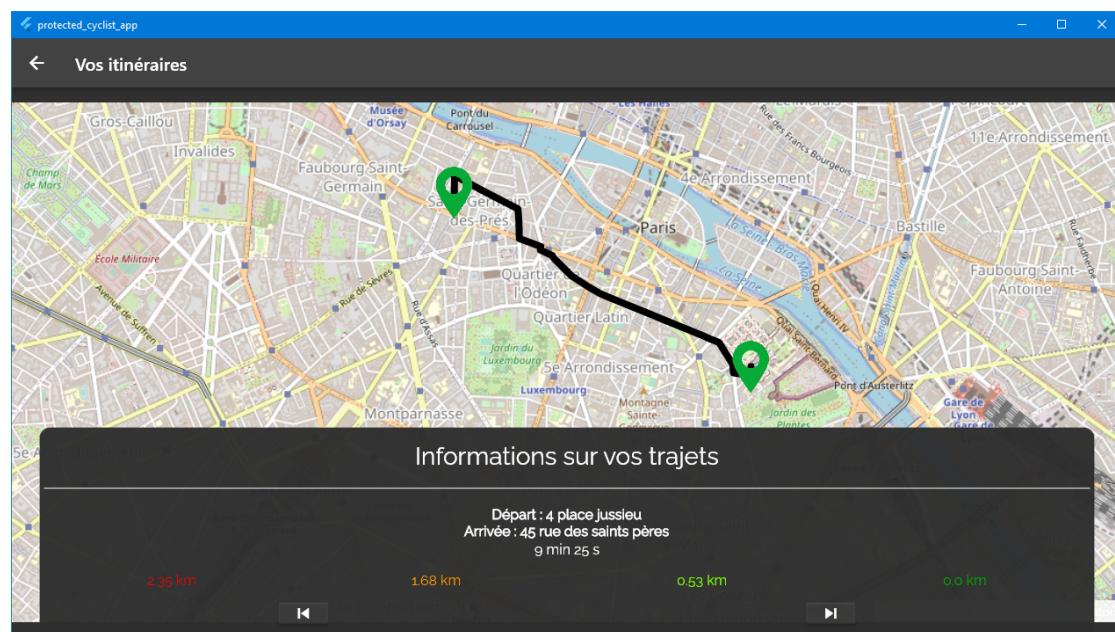


FIGURE 8 – Trajet affiché et liste de trajets

Notez que vous pouvez à tout moment retourner en arrière pour pouvoir générer un autre itinéraire en cliquant sur le bouton fléché en haut à gauche de l’écran. Le détail des distances à pédaler sur des chemins de chaque niveau de sécurité est donné grâce au même code couleur explicité en 4.2.