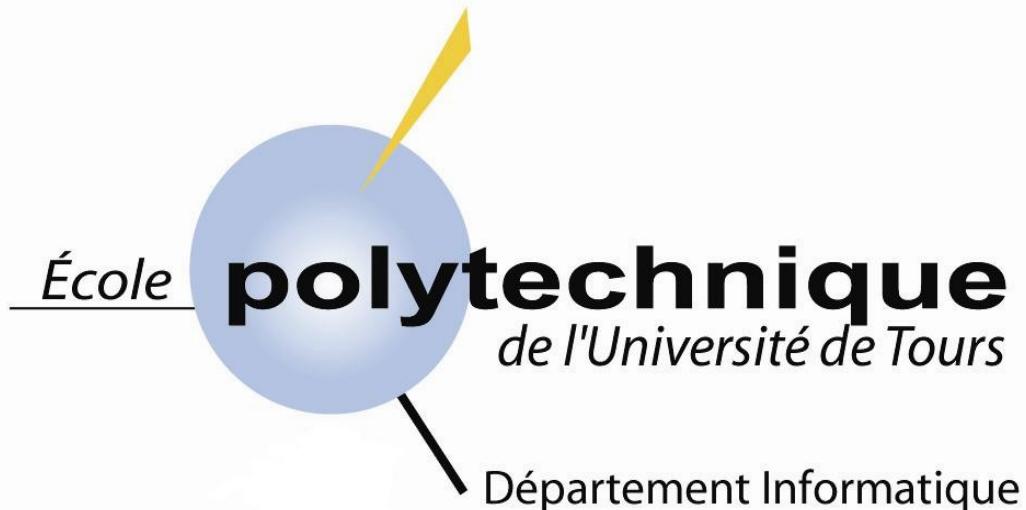


Université François Rabelais - Tours  
École Polytechnique de l'Université de Tours - Département Informatique  
64, avenue Jean Portalis  
37200 Tours



## Projet de Fin d'Etude

---

# PFE : calcul d'itinéraires touristiques pour vélo

---

Encadrants :

Gaël Sauvanet  
Emmanuel Néron

Etudiante :

ZHANG Juan

Année 2008-2009



## Remerciements

Je souhaite tout d'abord remercier beaucoup Gaël Sauvanet pour m'avoir donné toutes les informations nécessaires et tous les conseils. Il m'a aidé tout au long du projet de fin d'étude. Son point de vue m'a été d'une grande utilité pour la compréhension du fonctionnement du système existant. Il m'a expliqué clairement et m'a guidé pour les réflexions concernant les algorithmes.

Un merci à Emmanuel Néron pour ses instructions et ses connaissances. Il m'a proposé des bonnes idées concernant ce projet.

Enfin, merci à tous ceux qui m'ont aidé durant la période du projet de fin d'étude.

# Table des matières

|   |    |
|---|----|
| Remerciements .....   | 3  |
| Introduction .....  | 6  |
| Chapitre 1 Présentation du projet .....                     | 7  |
| 1.1 Présentation du sujet .....                             | 7  |
| 1.2 Le système existant .....                               | 8  |
| 1.3 Les missions du projet .....                            | 9  |
| 1.4 L'analyse du besoin .....                               | 9  |
| 1.5 L'environnement du poste de travail.....                | 10 |
| 1.6 Les technologies utilisées.....                         | 11 |
| 1.6.1 Les technologies pour la base de données .....        | 11 |
| 1.6.2 La technologie pour XMLRPC.....                       | 11 |
| 1.6.3 Les technologies pour le côté de client.....          | 12 |
| 1.6.4 Les technologie pour le côté serveur .....            | 12 |
| Chapitre 2 Les méthodes.....                                | 14 |
| 2.1 Présentation du problème .....                          | 14 |
| 2.1.1 Le problème multi-objectif.....                       | 14 |
| 2.1.2 Le problème de prétraitement.....                     | 15 |
| 2.1.3 Le problème de permutation .....                      | 15 |
| 2.1.4 Le problème de la densité de solution .....           | 16 |
| 2.2 Les méthodes utilisées .....                            | 17 |
| 2.2.1 L'algorithme de Dijkstra.....                         | 17 |
| 2.2.2 L'optimisation multi-objectif .....                   | 20 |
| 2.2.3 La permutation .....                                  | 25 |
| 2.2.4 La distance de crowding.....                          | 25 |
| Chapitre 3 La base de données .....                         | 29 |
| 3.1 La présentation de la base de données .....             | 29 |
| 3.2 Les outils Postgresql et Postgis .....                  | 29 |
| 3.3 Le modèle conceptuel des données (MCD) .....            | 31 |
| 3.4 Le modèle logique des données (MLD) .....               | 33 |
| 3.5 La réalisation de la base de données .....              | 33 |
| Chapitre 4 La réalisation .....                             | 36 |
| 4.1 L'architecture .....                                    | 36 |
| 4.1.1 Fonctionnement général.....                           | 36 |
| 4.1.2 La structure de données .....                         | 37 |
| 4.2 Le prétraitement .....                                  | 39 |
| 4.3 XMLRPC.....   | 39 |
| 4.4 La réalisation de calcul d'itinéraire touristique ..... | 40 |
| 4.4.1 La mise en ordre de points à visite .....             | 41 |
| 4.4.2 La combinaison.....                                   | 42 |
| 4.4.3 La permutation.....                                   | 43 |
| 4.4.4 La limitation du temps .....                          | 45 |

|  |    |
|--|----|
| 4.4.5 La distance de crowding .....          | 45 |
| 4.5 La réalisation de site web.....          | 47 |
| 4.5.1 Le choix de points .....               | 47 |
| 4.5.2 Le choix de solution.....              | 49 |
| 4.5.3 Le choix de limitation du temps .....  | 50 |
| Chapitre 5 Les problèmes rencontrés.....     | 53 |
| Chapitre 6 Conclusion.....                   | 54 |
| Références bibliographiques.....             | 55 |
| Annexe A Installation et configuration ..... | 56 |
| Annexe B Cahier des charges .....            | 60 |

# Introduction

Dans le cadre de ma cinquième année du cycle d'ingénieur de l'Ecole Polytechnique Département Informatique de l'Université de Tours, j'ai effectué un projet de fin d'étude (PFE). J'ai travaillé deux jours par semaine entre le mois d'octobre 2008 et de mai 2009.

Le sujet du projet de fin d'étude a été proposé par le doctorant Gaël Sauvanet. L'objectif de ce projet est de calculer l'itinéraire touristique adapté aux vélos en fonction de lieux touristiques sélectionnés par l'utilisateur, il permet de guider l'utilisateur pour trouver un itinéraire touristique. Le service est proposé sous la forme d'un site web.

Dans ce rapport, je vais commencer par présenter le contexte du projet. Ensuite nous allons voir les méthodes et les algorithmes utilisés, la spécification de la base de données et la réalisation de ce projet. Enfin, nous verrons les difficultés rencontrées et les évolutions futures du projet.

# Chapitre 1 Présentation du projet

## 1.1 Présentation du sujet

Le sujet de mon projet de fin d'étude est : calcul d'itinéraires touristiques pour vélo. Un itinéraire touristique est calculé en fonction d'un ensemble de points contenant le point de départ, les points d'intérêts tels que les musées, les jardins, les monuments, ... et le point de destination. Un touriste doit commencer par visiter le point de départ jusqu'au point d'arrivé en passant obligatoirement par tous les points d'intérêts sélectionnés. L'objectif de mon projet est de réaliser les méthodes et les algorithmes pour calculer l'itinéraire touristique mais également toute l'interface web et la base de données.

Il existe donc deux parties sur mon PFE : la construction du site web et la réalisation des algorithmes. En plus, il faut créer la base de données pour stocker tous les points d'intérêts. On peut charger directement les tables de la base de données qui sont déjà fait par le projet existant. Ce sont les fichiers `departement37_v2.sql`, `edge.sql` et `node.sql`. On utilise XMLRPC pour transmettre les données entre le côté de client et le côté de serveur (voir figure 1.1).

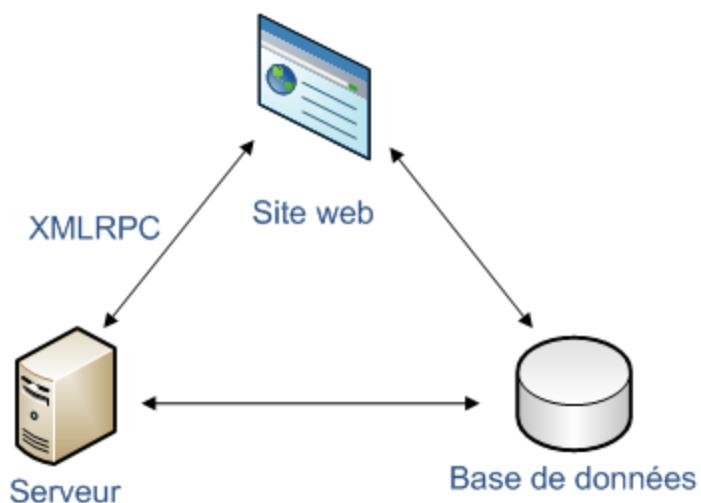


Figure 1.1 L'architecture de mon PFE

La construction du site web se base sur le design existant (xhtml+css), parce qu'il faut assurer la cohérence de type entre toutes les pages web. Le site web offre plusieurs fonctionnalités pour aider l'utilisateur à trouver un itinéraire touristique. Pour réaliser le site web, il faut utiliser plusieurs technologies tels

que Javascript, HTML, Xajax, PHP, CSS, notamment l'API de Google Maps qui nous donne beaucoup de fonctions.

La réalisation de l'algorithme est effectuée du côté de serveur. L'algorithme de Dijkstra est un algorithme principal pour ce sujet. Une adaptation a déjà été réalisée par le doctorant Gaël Sauvanet. Le serveur reçoit les requêtes et les données à traiter et ensuite retourne le résultat à l'utilisateur du côté client (site web). Le résultat contient un ensemble de solutions au lieu d'une solution. Ce résultat est engendré car il s'agit d'un problème multi-objectif. L'optimisation multi-objectif est le problème le plus important dans mon PFE. Pour faire du tourisme en vélo, la distance et la sécurité sont les deux critères les plus importants qui intéressent les touristes. Ils peuvent préparer leur itinéraire en fonction de ces deux critères avant de faire du tourisme. En effet, les touristes souhaitent trouver un chemin à la fois court et sécurisé. J'ai donc besoin de considérer ces deux critères en même temps au cours du calcul. C'est le problème multi-objectif. Il faut donc chercher les méthodes et les algorithmes pour trouver les solutions optimales.

## 1.2 Le système existant

Mon encadrant Gaël Sauvanet a déjà créé le site web [www.geovelio.fr/proto.php](http://www.geovelio.fr/proto.php) (voir Figure 1.2) pour calculer l'itinéraire en fonction du point de départ et du point de destination. Dans ses travaux, il a considéré plusieurs critères comme la distance, la difficulté et la sécurité. Il a utilisé une adaptation de l'algorithme de Dijkstra pour trouver les solutions non dominées. Une solution par défaut est quand même retournée à l'utilisateur.



Figure 1.2 Le site web existant géovelio

## 1.3 Les missions du projet

Pour les missions de mon PFE, non seulement je dois créer l'interface pour le site web, mais aussi chercher comment calculer un itinéraire touristique.

D'abord, il faut installer les outils et configurer la poste de travail.

Ensuite, il faut récupérer la latitude et la longitude pour chaque point d'intérêt sur la carte. Je dois sauvegarder toutes les informations concernant la latitude, la longitude, le nom, le type, la description, l'image, ...dans la base de données. Donc il faut considérer la modélisation de la base de données.

On a besoin d'utiliser l'API de Google Maps pour créer le site. Cette API permet d'afficher tous les points avec les marqueurs et d'afficher tous les tronçons sous la forme de lignes sur la carte. L'utilisateur peut choisir le point de départ, les points d'intérêts et le point de destination sur le site web.

Je dois chercher les méthodes et les algorithmes pour trouver un ensemble de solutions en fonction des deux critères : la distance et la sécurité. Après avoir été calculées, les informations sont retournées à l'utilisateur sur la carte. Ces informations contiennent un ensemble de solutions, la distance, le temps estimé, les tronçons à passer et le type d'aménagements tels que piste cyclable, bande cyclable,... en fonction de la couleur sur la carte. A la fin, l'utilisateur peut indiquer une solution parmi toutes les solutions.

## 1.4 L'analyse du besoin

Ce projet permet à l'utilisateur de sélectionner les points d'intérêts et consulter les informations sur la fenêtre pour chaque point d'intérêt, choisir et déplacer le départ et la destination. L'utilisateur peut indiquer une solution parmi un ensemble de solutions sur la barre. Il peut aussi limiter le temps de transport en vélo.



Figure 1.3 Le diagramme de cas d'utilisation

## 1.5 L'environnement du poste de travail

Ce projet est effectué sous le système d'exploitation Ubuntu 8.

On utilise la base de données Postgresql et son module Postgis. Il faut installer l'outil pgadmin3 pour gérer la base de données et l'autre outil qgis pour visualiser la base des données géographique.

Pour le site web, il faut utiliser PHP5 avec apache2.

Ce projet doit être réalisé en C++. Donc on installe l'outil codeblocks comme environnement de développement et on installe les paquets concernant la librairie de communication XMLRPC.

Vous pourrez consulter le manuel de l'installation et la configuration de l'environnement dans l'annexe.

## 1.6 Les technologies utilisées

Ce projet nous demande plusieurs technologies. Pour la création de site web, on utilise l'API de Google Maps, HTML, CSS, Javascript, Xajax, PHP. Les technologies Postgresql et Postgis sont utilisées pour la base de données. XMLRPC permet la communication entre le client et le serveur. La réalisation des méthodes et des algorithmes est effectuée en C++ et LEDA. Je vais détailler l'utilisation et les fonctionnalités de ces technologies.

### 1.6.1 Les technologies pour la base de données

Postgresql est un système de gestion de base de données relationnelles (SGBDR). Il possède plusieurs caractéristiques. Il peut être utilisé sous beaucoup de langages tels que (C, C++, Java, ...) pour ajouter, supprimer et consulter les enregistrements dans la base de données.

Postgis est un module spatial. Il peut être ajouté sur une base de données Postgresql. Cela permet d'ajouter des nouvelles structures de données (de type géographique comme le point, le polygone...) et des nouvelles fonctions.

### 1.6.2 La technologie pour XMLRPC

XMLRPC est un protocole RPC (remote procedure call). Il existe 8 types de données : string, int, boolean, double, dateTime, base64, array et struct. Le processus de XMLRPC utilise XML pour la communication. Un serveur RPC reçoit la requête XML transférée par le côté de client et retourne le résultat traité au client par XML. Le client analyse XML et obtient les données (voir Figure 1.4).

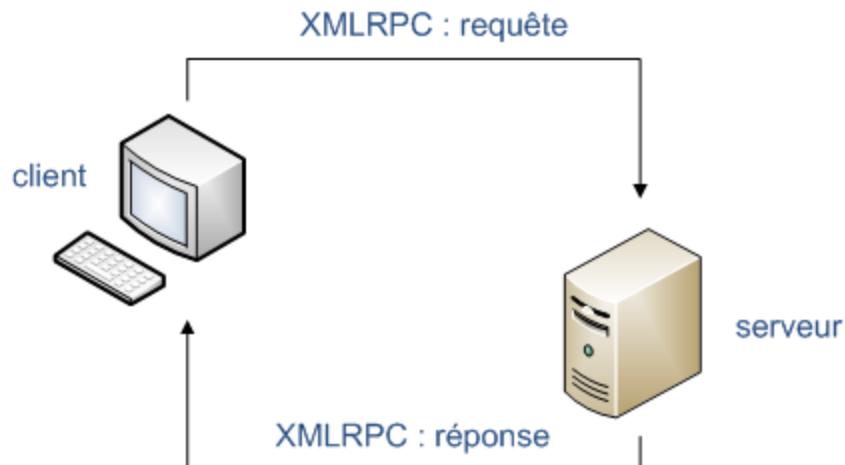


Figure 1.4 XMLRPC

### 1.6.3 Les technologies pour le côté de client

API Google Maps offre beaucoup de fonctionnalités à l'utilisateur comme le chargement de Google Maps avec Javascript, l'affichage de marqueur positionné et l'affichage de lignes. Il nous permet d'ajouter des fonctionnalités sur notre site web.

HTML et CSS permettent de concevoir l'apparence visuelle des pages web.

Javascript est un langage de script côté client. Il sert à traiter les formulaires dans la page web et à faire appel aux fonctions de Xajax qui sont écrit dans un fichier en PHP.

Xajax permet de mettre à jour le contenu d'une page web sans recharge de toute la page. Il fait appel à des fonctions de PHP de manière asynchrone.

PHP est un langage qui est exécuté côté serveur. On peut l'utiliser par exemple pour se connecter à la base de données.

### 1.6.4 Les technologie pour le côté serveur

LEDA est une librairie C++ contenant beaucoup de fonctions mathématiques. Il fournit des structures de données et des algorithmes très

performants (notamment pour les graphes). Dans ce projet, il faut utiliser des structures de données et des fonctions pour construire le graphe sur la carte et trouver le plus court chemin.

Tout ce qui concerne le calcul d'itinéraire touristique est donc réalisé en C++ sous l'IDE codeblocks.

## Chapitre 2 Les méthodes

Pour calculer l'itinéraire, j'ai utilisé plusieurs méthodes et algorithmes. Dans ce chapitre, les problèmes du sujet, plusieurs méthodes et algorithmes que j'ai utilisé seront présentés.

### 2.1 Présentation du problème

Le calcul d'itinéraire touristique est la plus importante partie dans mon PFE. Il s'agit plusieurs méthodes et plusieurs algorithmes pendant le calcul.

Comme le problème du voyageur de commerce, étant donné  $n$  villes et les distances pour chaque paire de noeud, un voyageur de commerce doit trouver un circuit de longueur totale minimale qui visite chaque ville exactement une fois. Pour calculer itinéraire touristique de mon côté, un touriste doit partir du point de départ, visiter une et une seule fois les points d'intérêts (châteaux, musées, etc.), jusqu'à arriver à la destination. La complexité de ce problème combinatoire est NP-difficile.

#### 2.1.1 Le problème multi-objectif

Par rapport aux problèmes mono-objectif, la difficulté de problème multi-objectif est qu'on ne peut pas trouver la solution unique. Puisque les objectifs sont toujours conflictuels dans ce problème, il nous obtient plusieurs solutions. On doit donc trouver les solutions de bons compromis.

Pour faire du tourisme en vélo, la distance et la sécurité sont les deux critères les plus importants qui intéressent les touristes. En effet, les touristes souhaitent trouver un chemin à la fois court et sécurisé. J'ai donc besoin de considérer ces deux critères en même temps au cours du calcul. Dans mon PFE, la distance et la sécurité sont toujours indépendantes sur chaque chemin. Il n'y a pas de relation entre elles. On ne peut pas seulement considérer un seul des deux critères. De plus, les deux critères ne sont pas combinés en un seul critère pour faciliter le calcul.

On peut utiliser le calcul d'itinéraire bicritère existant (fourni au début de mon PFE) qui se base sur l'algorithme de Dijkstra pour calculer un itinéraire intermédiaire entre deux points adjacents à visiter. L'algorithme de cette fonction est différent de l'algorithme classique de Dijkstra. Pour l'algorithme classique, on peut obtenir une solution concernant le plus court chemin. Mais le

résultat de l'algorithme qu'on utilise retourne beaucoup de solutions intermédiaires entre ces deux points adjacents. Chaque solution est représentée par un couple {distance, sécurité} correspondant aux deux critères. Par exemple :

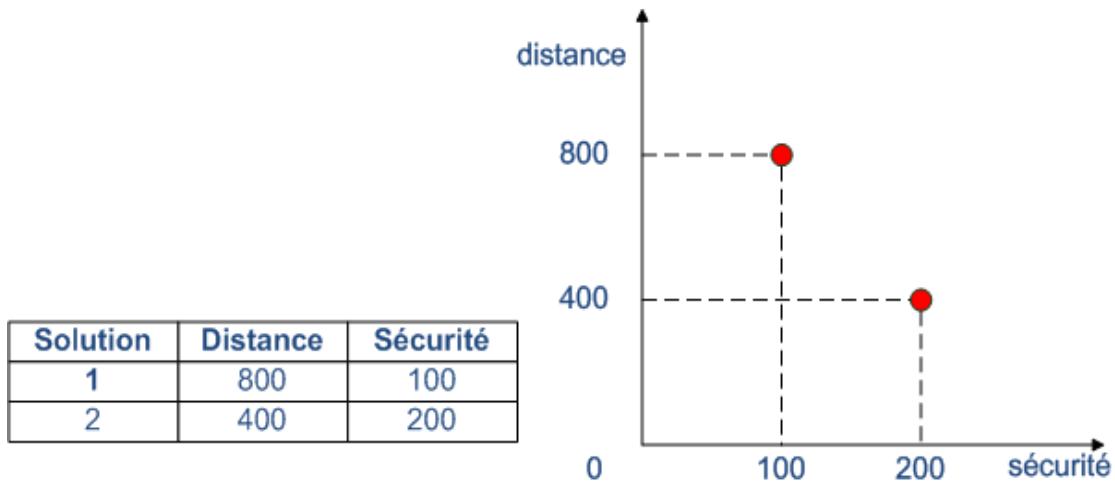


Figure 2.1 deux solutions

On ne peut pas dire quelle est la meilleure solution, parce que la première solution est plus sécurisée que la deuxième solution, mais la distance de la première solution est plus longue. Donc les deux solutions ne sont pas comparables. Si un touriste s'intéresse beaucoup à la sécurité, il peut prendre la première solution, en revanche il peut choisir la deuxième solution. Il faut donc considérer la distance mais aussi la sécurité pour calculer l'itinéraire touristique. On appelle ces problèmes des problèmes « multi-objectif ». Ici, nous avons deux critères, il s'agit d'un problème « bicritère ».

### 2.1.2 Le problème de prétraitement

Au cours de calcul d'itinéraire touristique, on a toujours besoin de calculer les solutions entre les deux points en fonction de l'algorithme existant de Dijkstra. Il mène la dépense de temps à l'utilisateur. Donc on peut créer l'autre travail pour sauvegarder toutes les solutions potentielles dans la base de données.

### 2.1.3 Le problème de permutation

Il existe beaucoup d'ordre de visite qui a déjà choisi par l'utilisateur. Il est impossible de calculer tous les ordres de visite, parce qu'il faut faire

attention la temps de réponse. Il faut trouver un ordre de visite par défaut. Ensuite on peut échanger les points adjacents pour chercher les meilleures solutions. Il s'agit le problème de permutation.

### 2.1.4 Le problème de la densité de solution

Étant donné un ordre de visite tel que le départ S, les points d'intérêts P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub> et la destination T, il faut trouver toutes les solutions intermédiaires pour chaque paire de points adjacents jusqu'à ce que tous les points soient parcourus. Quand on calcule un itinéraire de S à P<sub>1</sub>, il peut nous donner un ensemble qui consiste à moins de 100 solutions. Quand on calcule un itinéraire de P<sub>1</sub> à P<sub>2</sub>, la taille d'ensemble de solutions peut aussi atteindre 100 solutions. Après avoir combiné ces deux ensembles de solutions, il existe peut-être des milliers et des milliers de solutions. Le nombre de solutions sera trop grand quand on termine la combinaison de toutes les solutions.

Comment sélectionner les bonnes solutions après la combinaison aux utilisateurs est le problème plus important. Parmi les solutions trouvées, il existe beaucoup de solutions similaires. Si on donne toutes les solutions qui sont proches à l'utilisateur, il n'y a pas de signification, parce que la variation de ces solutions est très petite. (voir Figure 2.2)



Figure 2.2 Les solutions similaires

Comme la figure 2.2 ci-dessus, les solutions 1, 2 et 3 sont plus proches, on choisit la première solution qui peut remplacer les trois. Donc la première et la quatrième sont des bonnes solutions représentatives. Je dois donc chercher une méthode pour sélectionner de bonnes solutions qui sont significatives pour l'utilisateur. Cette méthode doit permettre d'augmenter la variation de solutions. Donc j'ai besoin de considérer la densité de solution. La distance de crowding est une méthode de l'estimation de densité. Elle nous permet de sélectionner les

solutions les plus représentatives.

Par la suite, je vais présenter plusieurs méthodes qui permettent de résoudre les problèmes ci-dessus dans la suite de cette partie.

## 2.2 Les méthodes utilisées

Dans cette section, je vais expliquer les différentes méthodes utilisées qui comprennent l'algorithme de Dijkstra, l'optimisation multi-objectif, la distance de crowding et la permutation.

### 2.2.1 L'algorithme de Dijkstra

Edsger Wybe Dijkstra est un mathématicien et informaticien néerlandais qui a proposé l'algorithme de Dijkstra en 1959. « L'algorithme de Dijkstra sert à résoudre le problème du plus court chemin entre deux sommets d'un graphe connexe dont le poids lié aux arêtes est positif ou nul. »<sup>[1]</sup> On peut l'utiliser chercher un plus court chemin d'un à tous les autres (one to all).

#### Le principe de l'algorithme classique de Dijkstra

Initialisation :

On affecte 0 au sommet et infini aux autres sommets. On choisit le sommet de départ.

Chaque étape :

D'abord, étant donné le dernier sommet sélectionné, on parcourt toutes les arêtes sortant du dernier sommet sélectionné et arrivant à un sommet non sélectionné. On ajoute le poids de l'arête pour le sommet non sélectionné.

Si le résultat est inférieur au poids actuel du sommet que l'arête arrive, on remplace le poids ancien par ce résultat. On choisit le sommet de poids minimal entre les sommets non sélectionnés.

On s'arrête jusqu'à ce que tous les sommets soient parcourus.

L'algorithme de Dijkstra est un algorithme de type de glouton. Il faut chercher un nouveau sommet à chaque étape.

#### L'algorithme classique de Dijkstra

Définition :

$G=[X, U]$  est un graphe.  $X$  est un ensemble de sommet à parcourir et  $U$  est un ensemble d'arête. La taille de  $X$  est  $N$ .

$X = \{1, 2, 3, 4, \dots, N\}$ ,  $\forall u = (i, j) \in U$ , c'est une arête sortant du sommet  $i$  et arrivant au sommet  $j$ . La longueur de chaque arête supérieur ou égale zéro.

On note  $D(i)$  qui représente la longueur du plus court chemin du sommet de départ au sommet  $i$ . S'il n'existe pas d'arête sortant du sommet de départ qui peut arriver au sommet  $i$ ,  $D(i)$  égale zéro.

On note  $P(i)$  qui représente la longueur du plus court chemin du sommet de départ au prédecesseur de sommet  $i$ .

On note  $S$  qui représente l'ensemble des sommets pour lesquels on connaît la longueur du plus court chemin du sommet de départ au sommet  $i$ .

### Algorithme de Dijkstra<sup>[2]</sup> :

/\* Initialisations \*/

- (a)  $S = \{1\}$  et  $D(1) = 0$  ;  
 $\forall i, (1, i) \in U, P(i) = L_{ij}$ ;  
 $\forall i, (1, i) \notin U, P(i) = +\infty$ ;

/\* Itérations \*/

(b) Tant que  $S \neq X$  Faire

Sélectionner  $k \in X \setminus S$ ,  $P(k) = \min P(i)$ ,  $i \in X \setminus S$ ;

$D(k) = P(k)$ ;

$S = S \cup \{k\}$ ;

Si  $S \neq X$  Alors

Pour arc  $(k, l)$  avec  $l \in X \setminus S$  Faire

$D(l) = \min(D(l), P(k) + L_{kl})$  ;

FinPour

FinSi

FinTQ

Exemple :

On crée un graphe comme la figure ci-dessous. Le noeud 1 est un sommet de départ, le noeud 6 est un sommet de la destination. On peut chercher le plus court chemin de 1 à 6.

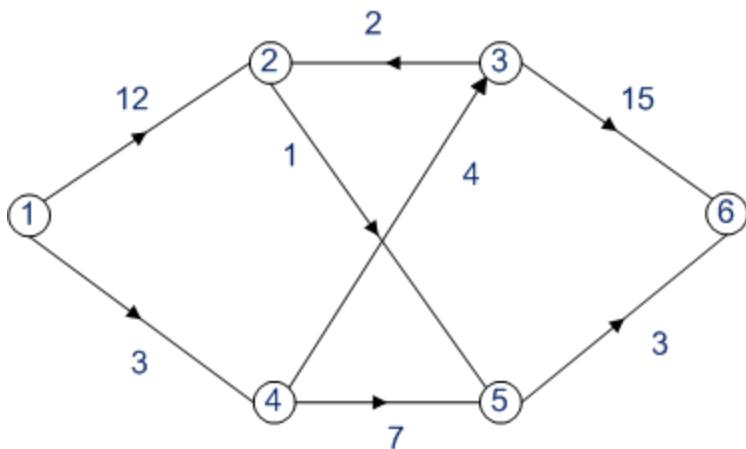


Figure 2.3 un graphe à traiter

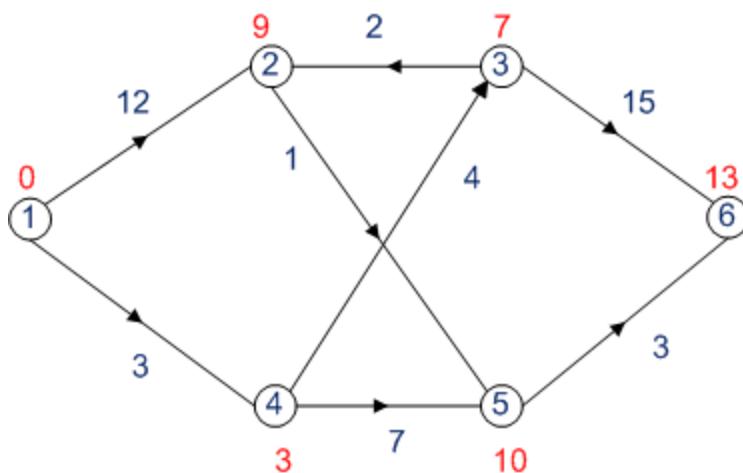


Figure 2.4 le plus petit poids de sommet de départ au sommet i

On utilise cette table pour montrer le résultat du calcul chaque fois.

|      | 1 | 2  | 3         | 4 | 5         | 6         |
|------|---|----|-----------|---|-----------|-----------|
| D(0) | 0 | 12 | $+\infty$ | 3 | $+\infty$ | $+\infty$ |
| D(1) |   | 12 | 7         | 3 | 10        | $+\infty$ |
| D(2) |   | 9  | 7         |   | 10        | 22        |
| D(3) |   | 9  |           |   | 10        | 22        |
| D(4) |   |    |           |   | 10        | 13        |
| D(5) |   |    |           |   |           | 13        |

Figure 2.5 Le résultat du calcul

Donc le plus court chemin est 1 – 4 – 3 – 2 – 5 – 6

Il n'existe qu'un critère sur l'algorithme classique de Dijkstra, mais il faut pourtant considérer deux critères pour mon PFE. J'utilise donc l'algorithme existant (fourni pour mon PFE) qui se base sur l'algorithme classique. A cause des deux critères, soit on considère seulement la distance, soit on considère seulement la sécurité, on peut donc obtenir les deux extrémités. Il existe encore plusieurs solutions entre ces deux extrémités. A la fin, il nous revoie un ensemble de solutions non dominées. La figure 2.6 nous montre qu'il existe 4 solutions entre S et T au lieu d'une solution.

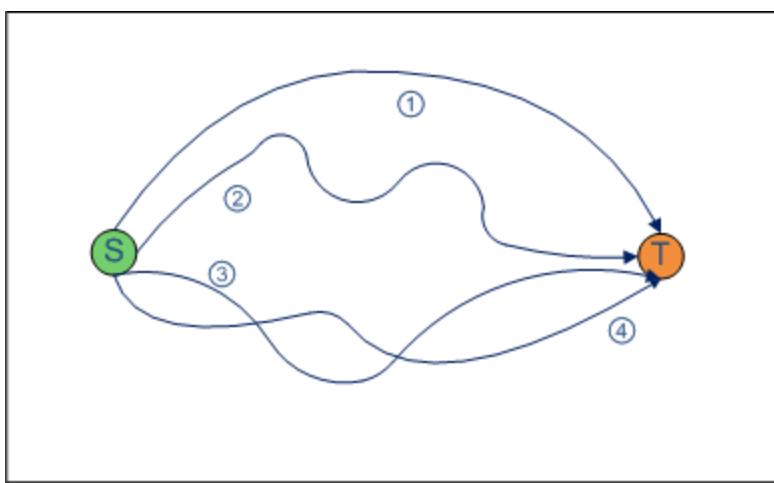


Figure 2.6 Plusieurs solutions entre deux points

### 2.2.2 L'optimisation multi-objectif

La méthode d'une optimisation multi-objectif est différent de l'approche mono-objectif. Elle est plus importante que la méthode mono-objectif en pratique par suite de l'application étendue. La méthode mono-objectif n'a qu'un objectif (voir Figure 2.7). Cette méthode est toujours recherchée pour l'optimisation combinatoire classique. A l'égard de la méthode mono-objectif, on peut obtenir la solution optimale globale, soit la valeur minimale, soit la valeur maximale (voir Figure 2.8). Par contre, il n'existe pas une solution qui peut satisfaire tous les objectifs sur la méthode d'optimisation multi-objectif. On obtient seulement un ensemble de solutions satisfaites. A la fin, le décideur peut choisir une solution comme le résultat final.

1 objectif :  $f$



Figure 2.7 Le mono-objectif

1 objectif :  $f$

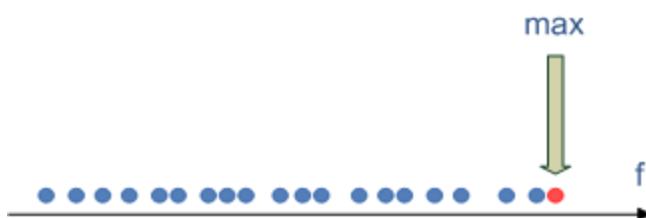


Figure 2.8 La solution optimale globale

L'optimisation multi-objectif est une branche de l'optimisation combinatoire. Elle est utilisée dans nombreux secteurs de l'industrie et beaucoup de domaines tels que le transport, l'aéronautique, l'environnement... Cette méthode vise à optimiser simultanément plusieurs objectifs d'un même problème pendant le processus d'optimisation, parce qu'il existe des conflits entre ces objectifs (voir Figure 2.9). On ne peut pas donc simplifier ces objectifs en un objectif unique. Donc l'objectif d'un problème multi-objectif est de trouver de bons compromis parmi toutes les solutions au lieu d'une solution. Les problèmes multi-objectif peuvent être NP-complets. Donc quand on résout les problèmes multi-objectif, il faut diminuer les objectifs en minimum.

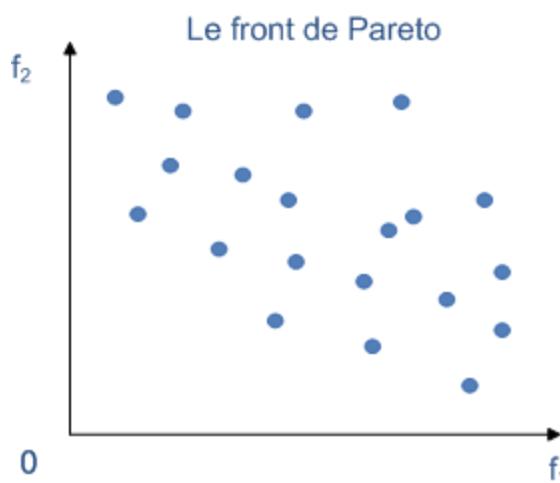


Figure 2.9 L'optimisation multi-objectif

Pour mon PFE, je dois considérer les deux critères : la distance et la sécurité. Les deux critères sont conflictuels, parce que si la distance est diminuée, la sécurité est diminuée aussi, et vice versa. C'est un problème d'optimisation bicritère. Les évaluations de ces deux critères sont respectivement la somme de toutes les distances parcourues et la somme de toutes les sécurités parcourues de chaque paire de points adjacents.

Je vais présenter quelque définition par la suite.

### **Définition 1 : Le problème multi-objectif**

Etant donné  $X = (x_1, x_2, \dots, x_n)$  est le vecteur des variables de décision. Il satisfait les contraintes ci-dessous :

$$g_i(X) \geq 0, i = 1, 2, \dots, k \quad (1)$$

$$h_i(X) = 0, i = 1, 2, \dots, l \quad (2)$$

Supposons  $r$  objectifs, et tous ces objectifs sont conflictuels, la fonction optimale est :

$$f(X) = (f_1(X), f_2(X), \dots, f_r(X))$$

Trouver  $X^* = (x_1^*, x_2^*, \dots, x_n^*)$ , optimiser  $f(X^*)$  sous les contraintes (1) et (2). Les solutions qu'on obtient sont les solutions Pareto optimal.

### **Définition 2 : Pareto optimal**

Pareto optimal a été proposé par un économiste italien Vilfredo Pareto. L'optimalité de Pareto est utilisé dans l'économie, l'ingénierie et la science sociale et de nombreux autres domaines. En économie, un optimum de Pareto est un état de l'attribution de ressources. On ne peut pas détériorer la situation d'individu. Dans ce cas-là, il est aussi impossible d'améliorer la situation d'individu.

On dit que le vecteur  $X^* \in F$  est un optimum de Pareto si, pour chaque  $X \in F$ , soit  $\forall i \in I (f_i(X) = f_i(X^*))$ ,  $I = \{1, 2, \dots, r\}$ , soit  $\exists j \in I, f_j(X) > f_j(X^*)$ .

Sachant que  $F = \{X \in R^n \mid g_i(X) \geq 0, i = 1, 2, \dots, k ; h_j(X) = 0, j = 1, 2, \dots, l\}$

En général, l'optimisation multi-objectif donne un ensemble de solutions, plus d'une solution. Dans un ensemble de solutions  $P$ , on sélectionne un ensemble de solutions non dominées. Dans l'espace de recherche, les solutions non dominées sont les solutions Pareto optimal. L'ensemble de solutions non dominées forme « le front de Pareto ».

L'objectif d'optimisation multi-objectif est de trouver un ensemble de solutions non dominées et proche du « front de Pareto ».

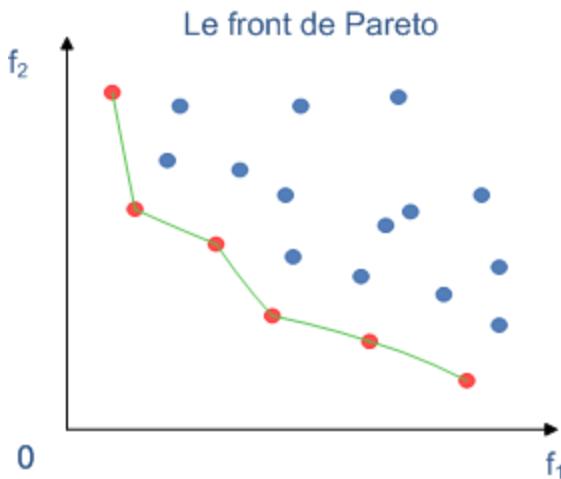


Figure 2.10 Le front de Pareto

### Définition 3 : La relation de dominance

$P$  est un ensemble de solutions, il existe  $k$  objectifs :  $f_1()$ ,  $f_2()$ , ...,  $f_k()$  pour chaque individu dans l'ensemble  $P$ . Il y a trois relations entre les deux solutions  $x$  et  $y$ . Pour  $x$  et  $y$ , soit  $x$  domine  $y$ , soit  $y$  domine  $x$ , soit  $x$  et  $y$  ne se dominent pas leur même. On peut donc définir la relation ci-dessous :

#### - La dominance :

Une solution  $x$  et une solution  $y$  appartiennent à l'ensemble  $P$ ,  $x$  domine  $y$  si et seulement si :  $\forall i \in \{1, 2, \dots, k\} : f_i(x) \leq f_i(y)$  et  $\exists j \in \{1, 2, \dots, k\} : f_j(x) < f_j(y)$ . On note  $y \prec x$ .

Si la solution  $x$  domine la solution  $y$ , on dit que  $x$  est la solution non dominée et  $y$  est la solution dominée par  $x$ . Le symbole «  $\prec$  » représente la relation dominée.

#### - Non relation :

Une solution  $x$  et une solution  $y$  appartiennent à l'ensemble  $P$ , on dit que la solution  $x$  et la solution  $y$  sont non relation si et seulement s'il n'existe pas la relation dominée entre elles.

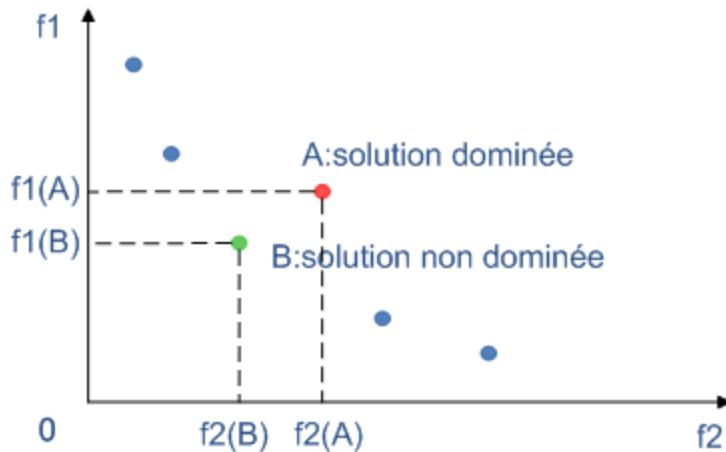


Figure 2.11 B domine A

Dans la figure ci-dessus, on peut savoir  $f_1(A) > f_1(B)$  et  $f_2(A) > f_2(B)$ . En fonction de la définition de dominance, le point B domine le point A. On note  $A \prec B$ . Donc le point B est une solution non dominée et le point A est une solution dominée. Le point A n'appartient pas à l'ensemble de « le front de Pareto », parce qu'il est dominée par le point B.

### **L'algorithme de la mise en ordre non dominée que j'utilise:**

$P =$  mettre en ordre en fonction de la distance pour toutes les solutions dans l'ensemble  $P$

$n = |P|$  ;

$dis = P[0].distance$  ;

$secu = P[0].securite$  ;

$i = 1$  ;

Tant que  $i < n$  Faire

Si  $P[i].distance$  n'égale pas  $P[i-1].distance$  Alors

Si  $P[i].securite < P[i-1].securite$  Alors

$secu = P[i].securite$ ;

Sinon

Supprimer la solution  $i$ ;

FinSi

Sinon

Si  $P[i].securite < P[i-1].securite$  Alors

Supprimer la solution  $i-1$ ;

Sinon

Supprimer la solution  $i$  ;

FinSi

FinSi

FinTQ

Dans cet algorithme,  $P[i].distance$  et  $P[i].securite$  représentent respectivement la distance et la sécurité de solution  $i$  dans l'ensemble  $P$ .

## 2.2.3 La permutation

J'utilise la méthode de permutation pour trouver les meilleures solutions. Je cherche l'ordre de visite par défaut en fonction de la distance la plus petite entre toutes les paires de points adjacents. Par exemple sachant que l'ordre de visite S – P1 – P2 – P3 – T, il y a 5 points à visiter, le nombre de permutation est donc  $5! = 120$ . C'est impossible de calculer tous les ordres de visite, parce qu'il faut considérer le temps de réponse. Les solutions doivent être retournées à l'utilisateur sur le site web, quand on calcule tous les ordres, il faut prendre beaucoup de temps. A cause de cette raison, j'échange seulement les points adjacents qui s'appuient sur l'ordre de visite par défaut. Donc il existe les deux ordres de visite tels que S – P2 – P1 – P3 – T et S – P1 – P3 – P2 – T. Après je combine toutes les solutions pour trouver les meilleures solutions.

## 2.2.4 La distance de crowding

Comme la première partie concernant la description de problème, la distance de crowding permet d'estimer la densité de solutions non dominées. Cette méthode est utilisée souvent dans le domaine de l'algorithme génétique.

La distance de crowding d'une solution se calcule par la moitié du périmètre de la figure quadrangulaire qui est formé par les voisins les plus proches autour de cette solution. La figure 2.12 montre la distance de crowding de solution  $i$  associé à ses voisins les plus proches, ce sont le point  $i-1$  et le point  $i+1$ . Dans cette figure, on peut savoir que la distance de crowding de la solution  $i$  est la somme de la valeur absolue de différence entre point  $i-1$  et point  $i+1$  sur chaque objectif, donc la distance de crowding de la solution  $i$  =  $|f_1(i-1) - f_1(i+1)| + |f_2(i+1) - f_2(i-1)|$

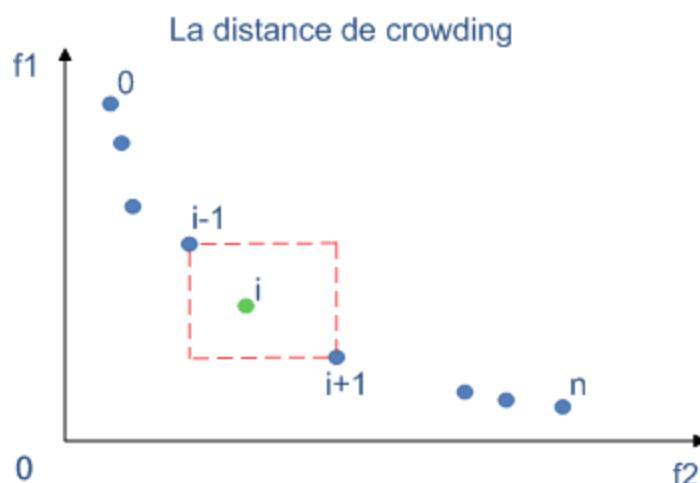


Figure 2.12 La distance de crowding

Je vais représenter l'algorithme du calcul de la distance de crowding de toutes les solutions non dominées. Dans cet algorithme, P est un ensemble de solution non dominée,  $P[i].m$  décrit la valeur de  $m^{\text{ième}}$  objectif lié à  $i^{\text{ème}}$  solution dans l'ensemble P.

### L'algorithme de la distance de crowding<sup>[3]</sup> :

```
/* Nombre de la solution dans l'ensemble P*/
n = |P| ;
/*Initialiser et calculer la distance de crowding*/
Pour chaque i, affecter  $P[i]_{\text{distance}} = 0$  Faire
    Pour chaque objectif m Faire
        /*Faire la mise en ordre en fonction de la valeur de l'objectif m*/
        P = sort(P, m) ;
         $P[1]_{\text{distance}} = +\infty$ ;
         $P[n]_{\text{distance}} = +\infty$ ;
        Pour i = 2 à (n-1) Faire
             $P[i]_{\text{distance}} = P[i]_{\text{distance}} + (P[i+1].m - P[i-1].m)$ 
        FinPour
    FinPour
FinPour
```

Pour mon PFE, je vais trouver le facteur D qui est utilisé à sélectionner les bonnes solutions. Si la distance de crowding est plus petite, en revanche la densité est plus grande. Je supprime les solutions qui ont leur distance de crowding inférieur au facteur D. Après je vais représenter l'algorithme utilisé, le problème existant et l'algorithme amélioré.

### L'algorithme de coupure qui se base sur la distance de crowding :

Etant donnée k : 10 ;

Dmax : la plus grande distance de crowding parmi toutes les solutions ;

Dmin : la plus petite distance de crowding parmi toutes les solutions ;

/\* Nombre de la solution dans l'ensemble P\*/

n = |P| ;

/\*Affecter la distance de crowding infinie aux deux extrémités \*/

$P[1]_{\text{distance}} = +\infty$ ;

$P[n]_{\text{distance}} = +\infty$ ;

/\*Sélectionner les solutions\*/

Tant que  $n > k$  Faire

Pour chaque i = 2 à (n-1) Faire

Calculer la distance de crowding  $P[i]_{\text{distance}}$ ;

Si  $P[i]_{\text{distance}} > D$  Alors

garder cette solution dans l'ensemble P ;

Sinon

supprimer la solution i dans l'ensemble P ;

*FinPour*

$D \leq D_{\min} + (D_{\max} - D_{\min}) / (n - 2) * 10\% ;$

$n \leq \text{recalculer le nombre d'ensemble } P ;$

*FinTQ*

Cet algorithme que j'utilise sélectionne les solutions dont la distance de crowding est la plus grande. Mais il existe des problèmes. Quand la solution qui a sa distance de crowding la plus petite est supprimée, il influence la distance de crowding des autres solutions dans l'ensemble  $P$ . Donc cet algorithme ne peut pas représenter la densité réelle entre les solutions.

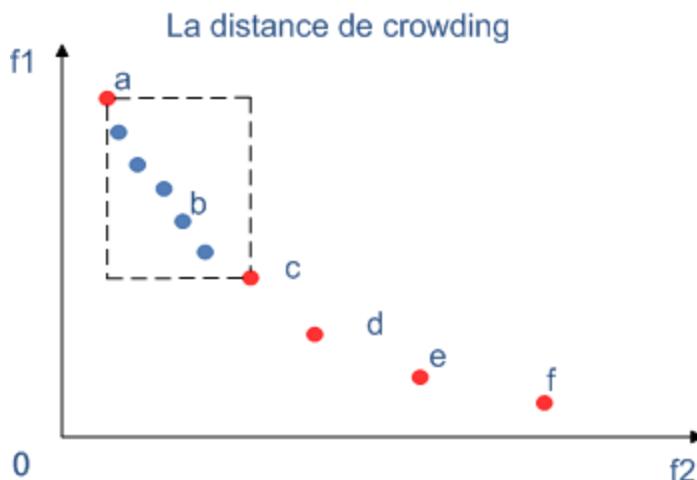


Figure 2.13 Le problème de la distance de crowding

La Figure 2.13 montre que quand on supprime les solutions qui ont une distance de crowding inférieur au facteur  $D$ , on obtient les solutions en rouges dans la figure. En effet, la distribution de ces solutions n'est pas uniforme, il existe un trou et il n'existe pas de solution dans le cadre. Pour résoudre ce problème, on peut utiliser l'algorithme amélioré.

### L'algorithme amélioré :

Etant donné  $k$  : le nombre de la solution qu'on espère

*/\* Nombre de la solution dans l'ensemble  $P$ \*/*

$n = |P| ;$

*/\*Affecter la distance de crowding infinie aux deux extrémités \*/*

$P[1]_{\text{distance}} = +\infty;$

$P[n]_{\text{distance}} = +\infty;$

*/\*calculer la distance de crowding et trier les bonnes solutions\*/*

*Tant que  $n > k$  Faire*

*Pour chaque  $i = 2$  à  $(n-1)$  Faire*

    Calculer la distance de crowding  $P[i]_{\text{distance}}$ ;

*FinPour*

    faire la mise en ordre pour toutes les distances de crowding ;

27 / 68

supprimer la solution que sa distance de crowding est la plus petite dans l'ensemble P;

$n = n-1$  ;

FinTQ

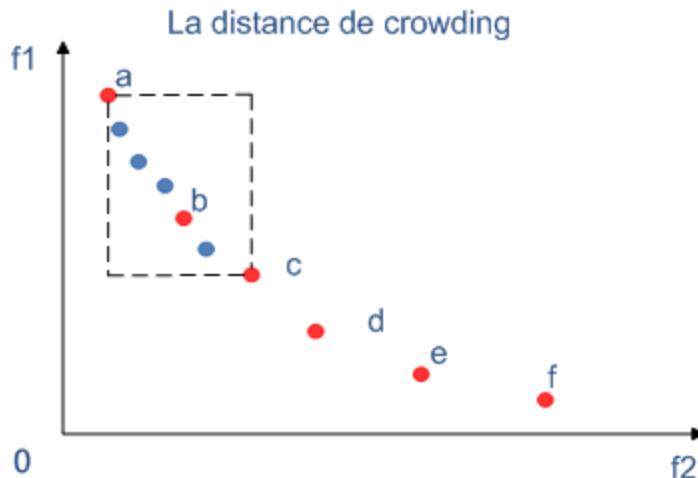


Figure 2.14 la distribution uniforme pour toutes les solutions

Cet algorithme amélioré permet de mettre à jour la distance de crowding de toutes les solutions chaque fois, quand on a supprimé la solution qui a sa distance de crowding la plus petite. Cet algorithme représente la distribution réelle pour toutes les solutions. Ces solutions sont plus significatives. Mais je n'ai pas utilisé cette méthode améliorée, parce que chaque fois, il faut faire la mise en ordre pour la distance de crowding, ce qui prend du temps.

## Chapitre 3 La base de données

Dans ce chapitre, je vais vous présenter les outils, la conception, la mise en place de la base de données géométrique et l'utilisation des fonctionnalités qui sont fournis pour traiter les informations géométriques.

### 3.1 La présentation de la base de données

Pour calculer l'itinéraire, il faut récupérer les informations géométriques comme les coordonnées de départ, de destination et les tronçons à passer. Donc les tables « département37\_v2 », « node » et « edge » sont fournies. Ces tables contiennent beaucoup d'informations géométriques à l'égard des points et des tronçons du département d'Indre et Loire. Elles offrent aussi les caractéristiques pour les points et les tronçons comme la latitude, la longitude des points et la longueur, la sécurité, le nom, la catégorie, la tranquillité, la source et la destination pour chaque tronçon.

Dans mon PFE, pour chercher un point d'intérêt plus vite, il faut créer les 3 tables « point », « categorie » et « description » pour récupérer les points d'intérêts. Ces tables contiennent les coordonnées, la catégorie et la description d'un point d'intérêts. On peut aussi utiliser la commande SQL pour récupérer toutes les informations mais ça prend beaucoup de temps.

Il faut créer autre table « solution ». Pendant le prétraitement, toutes les paires de noeuds sont calculées par l'algorithme de Dijkstra existant. On peut obtenir les solutions non dominées après le calcul. Ces solutions doivent être stockées dans la table « solution ». Donc chaque fois, quand on lance le serveur, il a seulement besoin de charger toutes les solutions dans la structure de données qui a créée. Il ne faut pas calculer toutes les solutions encore une fois. Cela permet d'économiser beaucoup de temps.

### 3.2 Les outils Postgresql et Postgis

Pour la base de données, on utilise les outils Postgresql et Postgis. Par rapport aux autres outils concernant la base de données, il existe plusieurs avantages pour l'outil Postgresql. Notamment pour la grande base de données, sa performance est excellente. Postgis ajoute la gestion de données spatiales et les fonctions sur l'outil Postgresql. Les données spatiales contiennent des informations géographiques tels que des points, des polylinéaires ou des polygones.

Dans la base de données de Postgresql, il faut charger le module de Postgis pour traiter les données géographiques. On utilise la commande ci-dessous (toto est le nom de la base de données):

```
$ su postgres
$ cd /usr/share/postgresql-8.2-postgis
$ createdb toto
$ createlang plpgsql toto
// charger le module Postgis
$ psql -d toto -f lwpostgis.sql
// charger les différents systèmes de projections spatiales
$ psql -d toto -f spatial_ref_sys.sql
```

« Geometry » est la plus importante caractéristique de Postgis. C'est-à-dire Postgis peut traiter plusieurs types de données, notamment le type de géométrie. Les types contiennent POINT, MULTIPONT, LINESTRING, POLYGON, MULTIPOLYGON, MULTILINESTRING. Dans mon PFE Il faut utiliser le type POINT(x,y) pour sauvegarder les coordonnées des points et le type LINESTRING((x1, y1), (x2, y2), (x3, y3), ...) pour sauvegarder toutes les informations concernant le tronçon. Un tronçon est composé d'un ensemble de points.

Postgis fournit beaucoup de fonctions très importantes. Je les ai utilisé pour consulter les enregistrements dans la base de données pour les données géographiques. Par la suite, nous allons présenter plusieurs fonctions très utiles.

```
// Ajouter une colonne géométrique sur une table
AddGeometryColumn(<schema_name>, <table_name>, <column_name>,
<srid>, <type>, <dimension>)
schema_name indique le pattern de la table. srid associe à la table de
« SPATIAL_REF_SYS ». type décrit le type de géométrie.

// Supprimer une colonne géométrique existante
DropGeometryColumn(<schema_name>, <table_name>, <column_name>)

// Affecter un SRID (système de projection) à un objet géométrie. 4326 associe
au domaine international, 27572 associe au domaine français. -1 est par défaut.
ST_SetSRID(geometry, integer)

// calculer le nombre de points sur ce tronçon
npoint()
// obtenir une section du tronçon de « début » à « fin »
line_substring(linestring, début, fin)
// obtenir le point d'après delta [0,1] qui présente la proportion
```

```

line_interpolate_point(linestring, delta)

// obtenir la proportion d'après ce point
Line_locate_point(linestring, point)

// transformer les objets géométriques dans le domaine indiqué (par exemple
pour passer du système de projection français au système international
transform(geometry,integer)

// étendre la borne d'après le deuxième paramètre
expand(geometry, float)

// Le constructeur des objets géométriques
GeomFromText(text, [])

```

### 3.3 Le modèle conceptuel des données (MCD)

Pour réaliser mon PFE, il faut créer le graphe sur la carte. Donc la table « node » et « edge » qui proviennent de la table « departement37\_v2 » servent à construire les arcs et les noeuds du graphe. La table « node » abstrait les points de la table « departement37\_v2 ». Elle stocke l'identifiant et l'objet géométrique concernant le type POINT pour chaque point. La table « edge » abstrait certains attributs de « departement37\_v2 » et crée les nouveaux attributs. Je vais présenter les principaux attributs de ces trois tables.

Il existe trois attributs dont la signification est la même dans la table « departement37\_v2 » et la table « edge ».

L'attribut catégorie : permet d'indiquer le type d'aménagement vélo

- 1 : non renseigne
- 2 : piste cyclable
- 3 : bande cyclable
- 4 : mixte (piéton/pas piéton)
- 5 : rien

L'attribut tranquilité : indiquer si le tronçon est conseillé pour le vélo ou non

- 0 : conseillé
- 1 : non conseillé

L'attribut desserte : indiquer si le tronçon est un tronçon de desserte ou non

- 0 : pas desserte
- 1 : desserte

Il y a aussi un lien concernant la direction entre elles.

L'attribut sens : indiquer le sens pour chaque tronçon

Direct : partir du sens de création du tronçon

Indirect : partir du sens inverse de création du tronçon

Double : deux sens

L'attribut `sens_creation` : permet d'indiquer si la direction de l'arc est identique au sens de création du tronçon

1 : identique

0 : différent

Les attributs longueur et categorie sont calculés comme le poids quand on utilise l'algorithme pour chercher le plus court chemin. Les attributs categorie et tranquilité sont utilisés pour distinguer les différents tronçons.

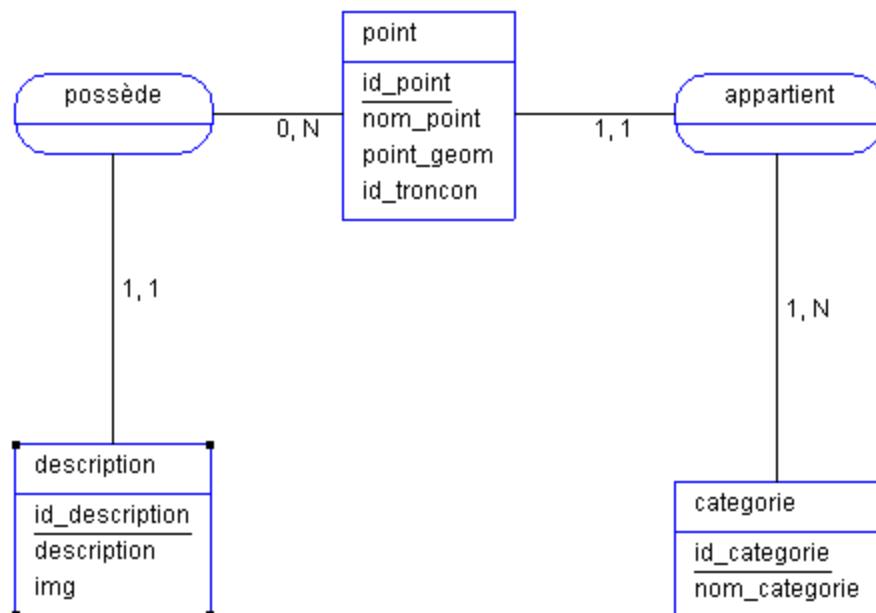


Figure 3.1 MCD

### 3.4 Le modèle logique des données (MLD)

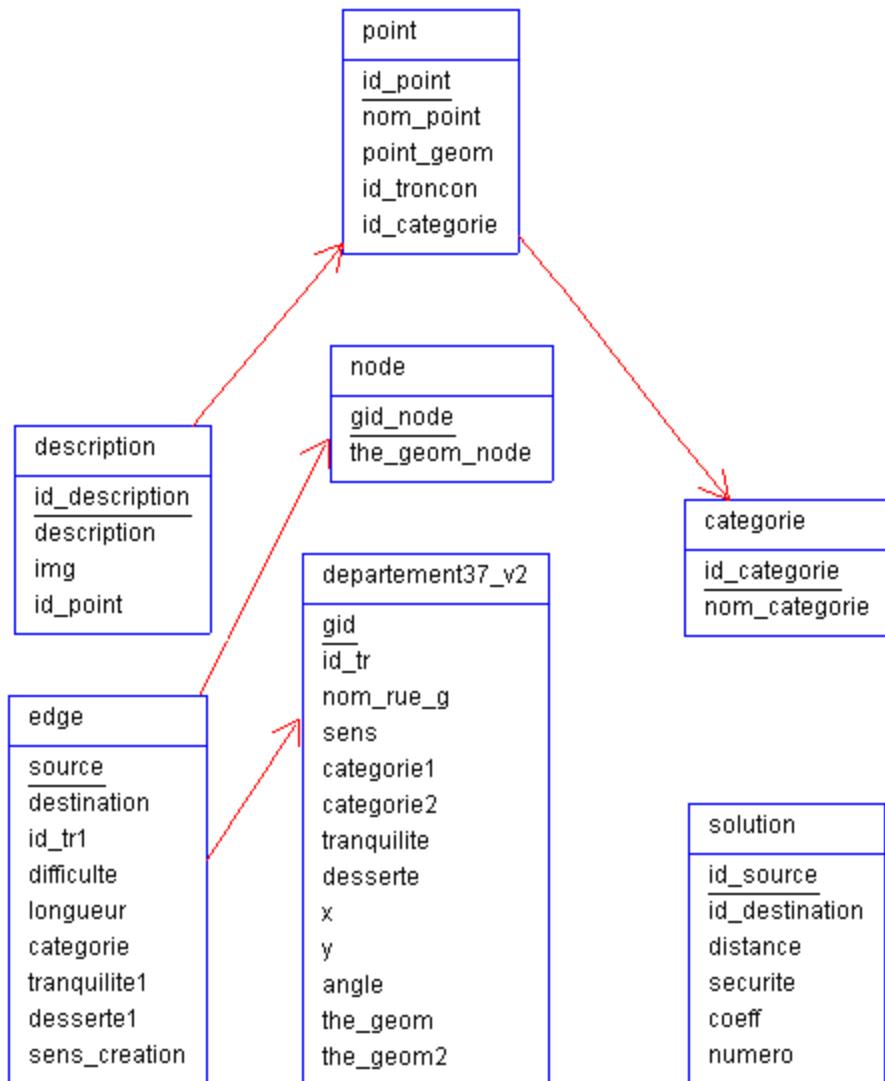


Figure 3.2 MLD

### 3.5 La réalisation de la base de données

Pour mon PFE, j'ai créé trois tables dépendantes « point », « categorie » et « description » et une autre table « solution ».

La table « categorie » contient seulement les attributs id\_categorie et nom\_categorie. Elle stocke les différents types. La table « description » qui contient les attributs id\_description, description et img pour décrire chaque

point d'intérêt.

Il existe quelques étapes pour récupérer un point d'intérêt. D'abord, on cherche un point d'intérêt sur l'Internet, pour le récupérer, il faut le positionner sur le centre dans la carte de Google Maps. J'ai tapé la commande :

`javascript:void(prompt('',gApplication.getMap().getCenter()));`  
dans l'URL de page web. On peut obtenir la latitude et la longitude de ce point (le point vert dans la figure 3.3).

J'utilise la requête SQL ci-dessous d'après les coordonnées de ce point. Cette commande nous permet de trouver le point le plus proche d'un point d'intérêt. On peut trouver le point rouge sur la figure 3.3.

```
SELECT line_locate_point(linemerge(T.the_geom2),
                           GeomFromText('POINT(lng lat)',4326)) as delta,
          E.source as source, E.destination as destination,
          E.id_tr,
          E.sens_creation,
          POINT '(lng, lat)' <-> (line_interpolate_point(linemerge(T.the_geom2),
line_locate_point(linemerge(T.the_geom2),           GeomFromText('POINT(lng
lat)',4326))),) as distance
FROM departement37_v2 T, edge E, node N
WHERE T.the_geom2 && Expand(GeomFromText('POINT(lng
lat)',4326),0.001) AND E.id_tr = T.id_tr AND E.source = N.gid
ORDER BY distance LIMIT 1;
```

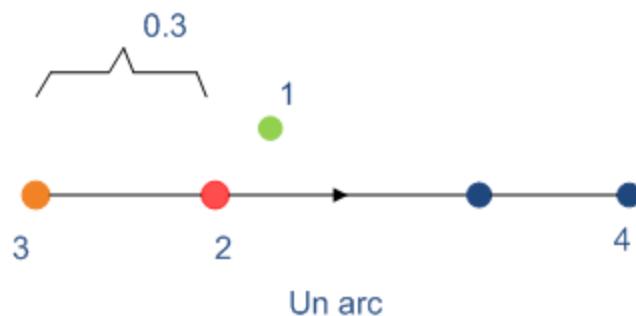


Figure 3.3 Récupérer un point d'intérêt

Maintenant on doit trouver une extrémité pour remplacer le point d'intérêt réel. La fonction `line_locate_point` nous retourne la proportion comme 0.3. Par rapport aux deux extrémités, on peut connaître le point orange (figure 3.3) le plus proche de l'extrémité 4. D'après la requête, on peut aussi savoir le `sens_creation`. S'il est égal à 1, c'est-à-dire le sens de l'arc est identique au sens de création du tronçon, donc on prend l'extrémité 3. Sinon, on prend l'autre extrémité. C'est pareil comme la proportion supérieure ou égale à 0.5.

Après avoir trouvé l'extrémité du tronçon, on remplace le point d'intérêt

qu'on a trouvé sur le site web par cette extrémité. On peut récupérer les coordonnées et l'identifiant de cette extrémité dans la table de « node ». Donc on peut récupérer soit les coordonnées, soit l'identifiant dans la table de « point ».

Si on veut stocker les coordonnées, il faut créer un attribut géométrique. J'ai utilisé la commande ci-dessous pour ajouter une colonne géométrique dans la table « point » qui contient les attributs id\_point, nom\_point, id\_troncon et id\_description. L'attribut géométrique point\_geom est créé automatiquement.

```
zhangjuan =# select AddGeometryColumn('', 'point', 'point_geom', '4326', 'POINT', 2);
```

J'ai utilisé la commande insert pour ajouter un nouvel enregistrement.

```
Zhangjuan=# insert into point(id_point, nom_point, point_geom, id_troncon, id_destination) values(1,'Musée Saint Martin',GeomFromText('POINT (latitude longitude)'),4326) ;
```

Si on stocke l'identifiant de cette extrémité, c'est plus simple que l'autre façon.

La table « solution » stocke l'identifiant de source, l'identifiant de destination, la distance, la sécurité et le coefficient pour chaque solution. Il nous permet de charger les solutions et mettre en ordre de points à visiter.

# Chapitre 4 La réalisation

Dans ce chapitre, je vais présenter les travaux qui ont été réalisés pendant mon PFE. Je vais expliquer les détails de traitement pour ce projet, la réalisation de l'aspect de l'algorithme, la construction du site web et quelques fonctionnalités qui sont offertes à l'utilisateur.

## 4.1 L'architecture

Dans cette section, je vais présenter l'architecture concernant deux parties de mon PFE et la structure de données qui est utilisée pour la création du graphe et pour stocker les solutions.

### 4.1.1 Fonctionnement général

Il existe deux parties sur mon PFE. Une partie est la création du site web et l'autre partie, c'est la réalisation de l'algorithme en C++. Le XMLRPC est un lien entre ces deux parties. Sur le côté du site web, on reçoit un ensemble de points correspondant au départ, aux points d'intérêts et à la destination que l'utilisateur a choisie et on les transmet au serveur pour calculer l'itinéraire touristique. Après avoir calculé l'itinéraire, le serveur retourne les solutions sélectionnées au site web en utilisant XMLRPC. L'utilisateur obtient le résultat.

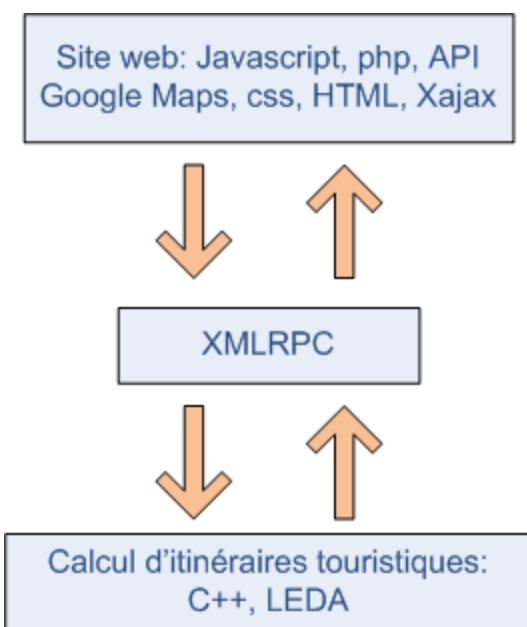


Figure 4.1 La structure de projet

Pour le site web, j'utilise HTML avec CSS pour ordonner la page web. Je transmets les formulaires entre HTML et Javascript, et j'utilise Xajax pour réaliser la mise à jour asynchrone de la page web. L'API Google Maps est utilisé dans les fichiers Javascript et PHP offre beaucoup de fonctions. Ils sont utilisés pour réaliser les fonctionnalités liées à la gestion de la carte.

Pour le calcul d'itinéraire, j'ai créé trois projets : *libServer*, *pretraitement* et *itineraire*. Le projet *libServer* est une librairie dynamique créée à partir des travaux du doctorant Gaël Sauvanet. Ce projet contient principalement la construction du graphe, la fonction concernant l'algorithme de Dijkstra bicritère. Par rapport à la librairie statique, la librairie dynamique est appelée seulement lorsqu'elle est nécessaire. Le projet *pretraitement* sert à prétraiter les données. Il est utilisé pour sauvegarder toutes les solutions obtenues entre toutes les paires de noeud en fonction de l'algorithme de Dijkstra bicritère dans la base de données. Le travail *itineraire* sert à calculer des itinéraires touristiques. Il correspond au traitement du problème multi-objectif, la combinaison des solutions, la méthode de la distance de crowding et les autres algorithmes.

#### 4.1.2 La structure de données

L'algorithme de Dijkstra est un algorithme qu'on utilise principalement. Il se base sur un graphe connexe. Donc il faut créer un graphe pour utiliser cet algorithme.

On imagine que les chemins reliant les points touristiques et les points touristiques sur la carte sont stockés dans la base de données et peuvent être considérés comme les arcs et les noeuds de graphe. LEDA offre certaines fonctions pour nous aider à construire le graphe.

Je vais représenter la structure de données existante et la structure de données que j'ai utilisée.

La structure de données existante :

```
struct structDynaNode {  
    int dist;  
    int dist2;  
    int securite;  
    edge edgeBack;  
    _sortseq<two_tuple<int,int> ,three_tuple<int,int,double>,skiplist >  
    solutions3;
```

```
};
```

Cette structure sert à sauvegarder toutes les informations dynamiques spécifiques à un noeud. Elle est utilisée sur l'algorithme de Dijkstra pour chercher le plus court chemin. Elle stocke les informations temporaires pendant le calcul d'un itinéraire tels que la distance, la sécurité, les arcs qui arrivent à ce noeud et les solutions non dominées.

```
struct structStatNode {
    unsigned int idNode;
    double x;
    double y;
    double combL;
    LEDA_MEMORY(structStatNode)
};
```

La structure structStatNode est utilisée pour sauvegarder les informations statiques à un noeud. Cette structure peut stocker l'identifiant d'un noeud et les coordonnées comme la longitude et la latitude d'un noeud.

```
struct structStatEdge {
    int idTroncon;
    int length;
    int securite;
    int categorie;
    int tranquilité;
    list <two_tuple<double,double> > listeTroncons;
    LEDA_MEMORY(structStatEdge)
};
```

Cette structure sauvegarde les informations spécifiques concernant un arc. Elle stocke l'identifiant, la longueur, la sécurité, la catégorie, la tranquillité d'un tronçon, et toutes les paires de coordonnées comme {longitude, latitude} pour les points qui composent le tronçon. La catégorie et la tranquillité sont utilisées pour distinguer les différents types de tronçon.

La structure de données que j'utilise :

```
structurePoidArc{
    int idSource;
    int idDestination;
    int distance;
    int securite;
    double coeff;
    int numero;
    LEDA_MEMORY(structurePoidArc)
};
```

Cette structure que j'ai utilisée sert à faire le prétraitement. Elle est

utilisée pour sauvegarder toutes les solutions obtenues entre toutes les paires de noeud en fonction de l'algorithme de Dijkstra. Elle permet de nous aider à faire la mise en ordre pour tous les points à visiter.

## 4.2 Le prétraitement

Le prétraitement est un projet C++, on peut l'exécuter indépendamment. Ce projet nous permet de calculer l'itinéraire entre toutes les paires de points d'intérêts (sauf les paires entre le départ et les autres points d'intérêts et les paires entre les autres points d'intérêts et la destination). On sauvegarde toutes les solutions dans la base de données. Ces solutions sont calculées une fois. Quand on relance le serveur, il faut seulement charger ces solutions dans la structure de données que j'ai créée. On n'a pas besoin de recalculer l'itinéraire pour obtenir les solutions. D'après cette raison, ce projet peut nous aider à économiser le temps.

## 4.3 XMLRPC

J'utilise XMLRPC pour faire appel à distance entre les pages web PHP et le serveur. J'écris le code concernant XMLRPC dans le fichier PHP. Il faut ajouter le fichier « xmlrpc.inc » pour configurer le client et ajouter les fichiers « xmlrpcs.inc » et « xmlrpcs.inc » pour configurer le serveur.

D'abord, je crée un objet contenant le répertoire, le nom du serveur et le port pour le client.

```
$cli = new xmlrpc_client('/ServerXmlRpc', 'localhost', 8082);
```

Il faut utiliser un tableau pour transmettre les données, parce que le deuxième paramètre de la fonction xmlrpclmsg est un tableau. Dans ce tableau, il y a quatre éléments à transmettre. Ils sont les points à visiter, le nombre de points, le coefficient de solution, et la limitation du temps.

```
$myParams = array(new xmlrpcval($listPoints, 'string'), new
xmlrpcval($nbPoints, 'int'), new xmlrpcval($coeff, 'int'), new
xmlrpcval($distanceLimite, 'int'));
```

Je crée un message de XMLRPC contenant le nom de fonction à distance et le tableau de paramètres. 'CalculItineraireTouristique' est le nom de la fonction dans le serveur. On fait appel la fonction à distance, et transmet les données à serveurs.

```
$mess = new xmlrpclmsg('CalculItineraireTouristique', $myParams, 'array');
```

Le message de XMLRPC est transmis, on peut obtenir la réponse par le

serveur. S'il existe des erreurs ou le serveur ne répond pas, on peut les indiquer sur la fenêtre. Sinon, ça veut dire qu'on a déjà connecté le serveur. Ensuite on va traiter les données retournées.

```
$res = $cli->send($mess);
```

## 4.4 La réalisation de calcul d'itinéraire touristique

Je vais présenter les étapes de calcul d'itinéraire touristique. Dans cette partie, je prends un exemple pour expliquer comment faire pour la réalisation.

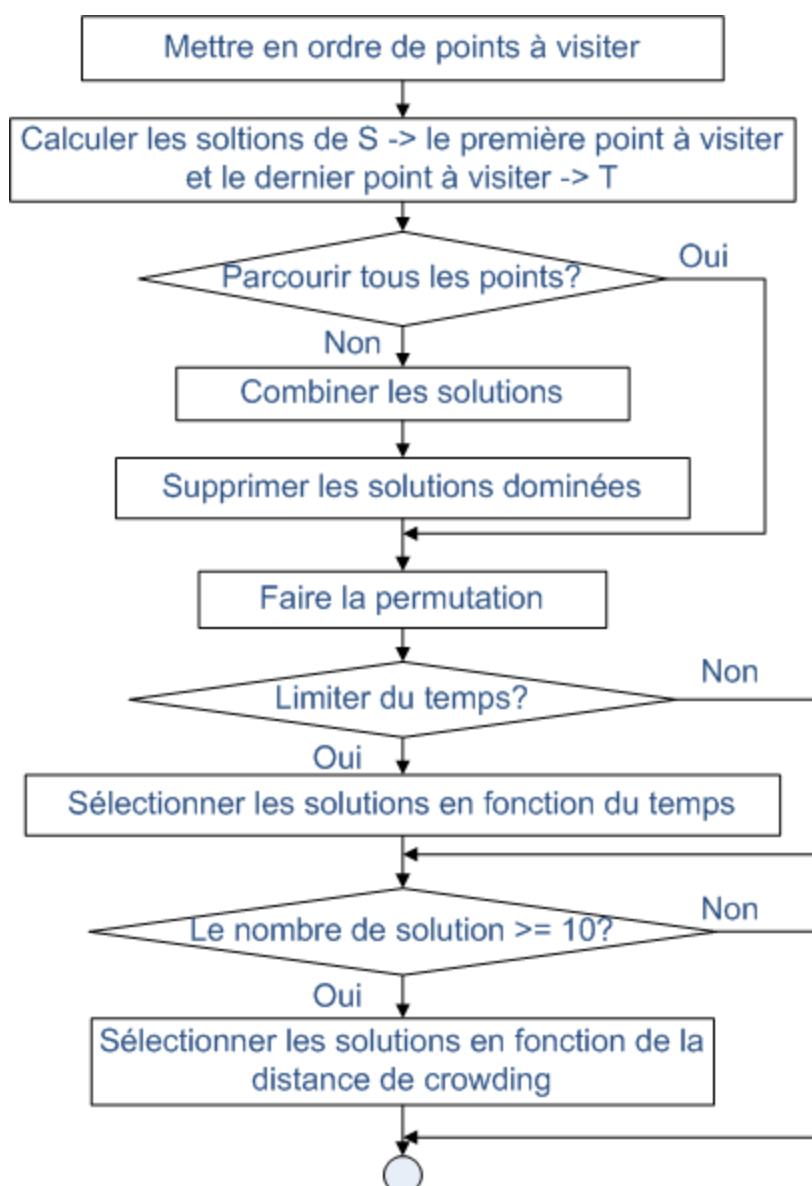


Figure 4.2 Les étapes de calcul d'itinéraires touristiques

#### 4.4.1 La mise en ordre de points à visite

L'utilisateur peut choisir le départ, les points d'intérêts n'importe lesquels et la destination. Pour renvoyer les solutions à l'utilisateur, il faut choisir l'ordre de visite par défaut. Il est impossible de calculer tous les ordres de visite parce que cela prend trop de temps. Donc l'ordre de visite par défaut est très important. Pour résoudre ce problème, je choisis le point prochain en fonction de la distance la plus petite.

Après avoir fait le prétraitement, les solutions entre toutes les paires de points d'intérêts sont stockées dans la base de données. Ensuite il faut utiliser la fonction de l'algorithme de Dijkstra bicritère pour calculer les solutions entre le départ et tous les autres points d'intérêts et les solutions entre tous les points d'intérêts et la destination. Donc on peut connaître la distance entre le point actuel et tous les autres points. On choisit le point prochain qui est le plus proche. On prend un exemple (voir Figure 4.3 et Figure 4.4).

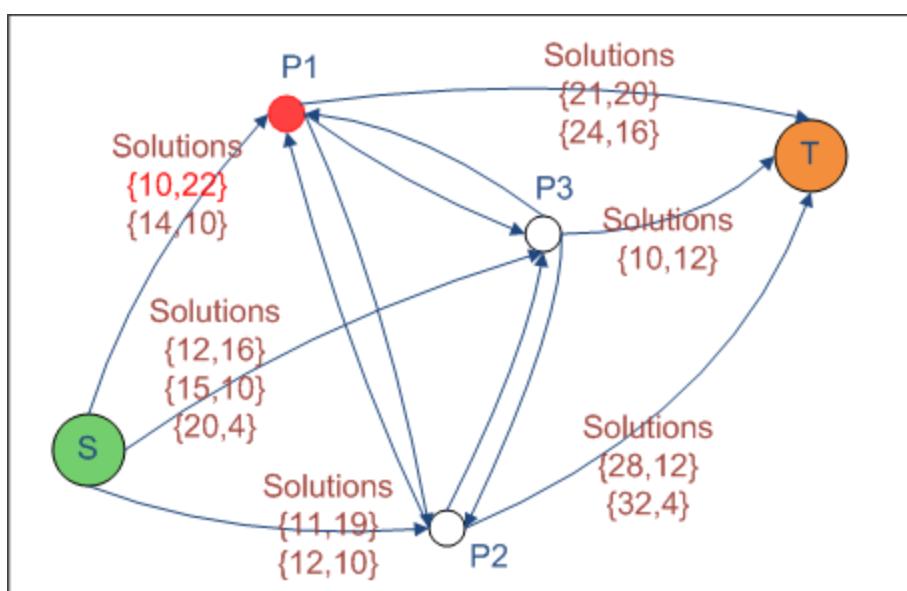


Figure 4.3 Le choix de l'ordre de visite

La figure 4.3 montre qu'il existe plusieurs solutions comme {distance, sécurité} entre le départ S et les autres points d'intérêts. La solution {10, 22} nous intéresse beaucoup parce que la distance est la plus petite. Donc on prend P1 comme premier point à visiter.

|        | Solution: {distance, sécurité} |         |         |
|--------|--------------------------------|---------|---------|
| P1->P2 | {16,23}                        | {20,15} | {24,5}  |
| P1->P3 | {11,20}                        | {14,18} | {16,14} |
| P2->P1 | {8,29}                         |         |         |
| P2->P3 | {26,15}                        | {20,14} | {16,10} |
| P3->P1 | {10,40}                        | {15,32} |         |
| P3->P2 | {12,24}                        |         |         |

Figure 4.4 Les solutions stockées dans la base de données

La figure 4.4 montre toutes les solutions stockées dans la base de données pendant le prétraitement. On a déjà pris le point P1. Ensuite P3 est choisi comme le deuxième point à visiter d'après la solution {9, 20}. On continue de parcourir les solutions suivantes qui commencent du point P3. Par rapport à la solution {12, 24}, la distance de la solution {10, 40} est plus petite. Mais P2 a déjà été parcouru, donc on ne considère pas le point P1. On prend le point P2. A la fin, l'ordre de visite est S – P1 – P3 – P2 – T (voir Figure 4.5).

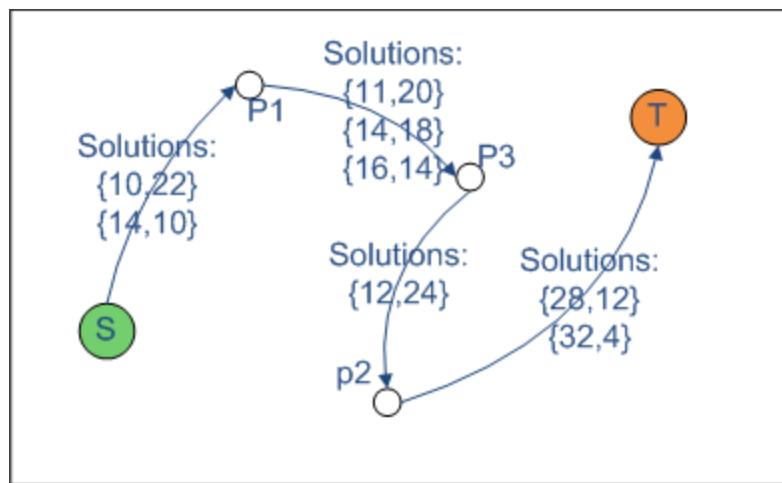


Figure 4.5 L'ordre de visite S – P1 – P3 – P2 – T

#### 4.4.2 La combinaison

On prend l'exemple précédent. Après avoir choisi l'ordre de visite et obtenu l'ensemble des solutions non dominées entre S – P1, P1 – P3, P3 – P2 et P2 – T, il faut faire la combinaison pour trouver les solutions entre le départ S et la destination T. D'abord, on prend les solutions de S à P1 et les solutions de P1 à P3. On combine les deux ensembles de solutions en un ensemble de solutions (voir Figure 4.6).

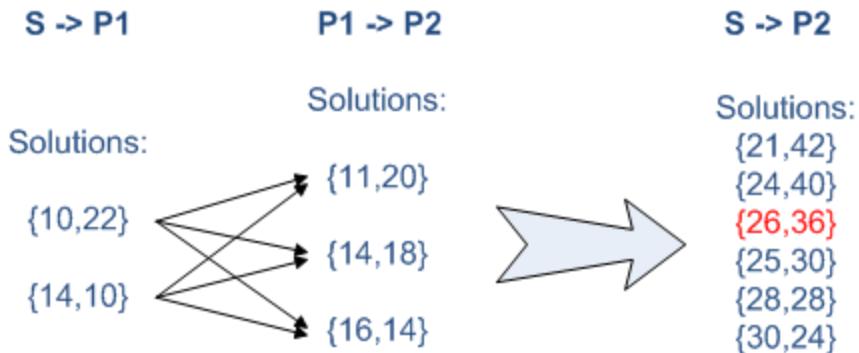


Figure 4.6 La combinaison

Le nombre de solutions obtenues égale le nombre de premier ensemble de solutions multiplié par le nombre de deuxième ensemble de solutions.

Ensuite il faut faire la mise en ordre croissant des solutions en fonction de la distance pour supprimer les solutions dominées. On compare la sécurité de chaque solution. Quand il existe des solutions dont leur distance est identique, on cherche la sécurité la plus petite et supprime les autres parmi elles. Sinon on supprime la solution dont sa sécurité est plus grande que la sécurité de la solution précédente. Les solutions supprimées sont dominées.

Dans la figure 4.6, on peut voir qu'il existe une solution dominée  $\{26, 36\}$ . Cette solution est dominée par la solution  $\{25, 30\}$  (voir Figure 4.7). Donc il faut supprimer cette solution, parce qu'elle peut engendrer plus de solutions dominées dans l'itération de la prochaine fois.

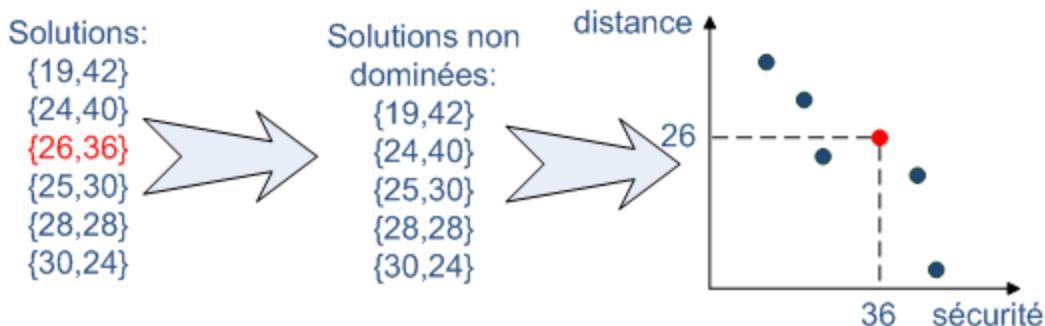


Figure 4.7 La suppression de la solution dominée

#### 4.4.3 La permutation

J'ai échangé la position de points d'intérêts adjacents sur l'ordre de visite par défaut. Le nombre d'échange égale le nombre de points de visite moins trois. Pour chaque échange, je fais la combinaison entre les solutions à

l'égard de nouvel ordre et les solutions concernant l'ordre de visite précédent. On peut savoir si ces deux ensembles de solutions sont non dominées, parce qu'on les a déjà traité. Ensuite on peut obtenir un ensemble de solutions combinées. Je supprime les solutions dominées comme l'étape précédente. On s'arrête jusqu'à ce que tous les ordres soient traités.

On prend un exemple comme la figure 4.8, après avoir obtenu un nouveau ensemble de solutions non dominées en fonction de nouveau ordre de visite, on combine cet ensemble de solutions (solutions rouges) et un ensemble de solutions précédentes (solutions bleues). Il faut supprimer les solutions dominées. A la fin, on peut obtenir les solutions non dominées (voir Figure 9) (solutions vertes).

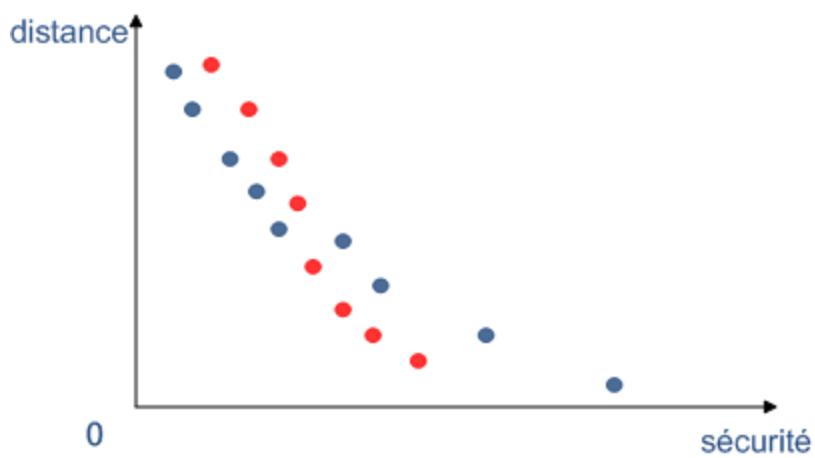


Figure 4.8 Combiner deux ensembles de solutions d'après l'ordre de visite

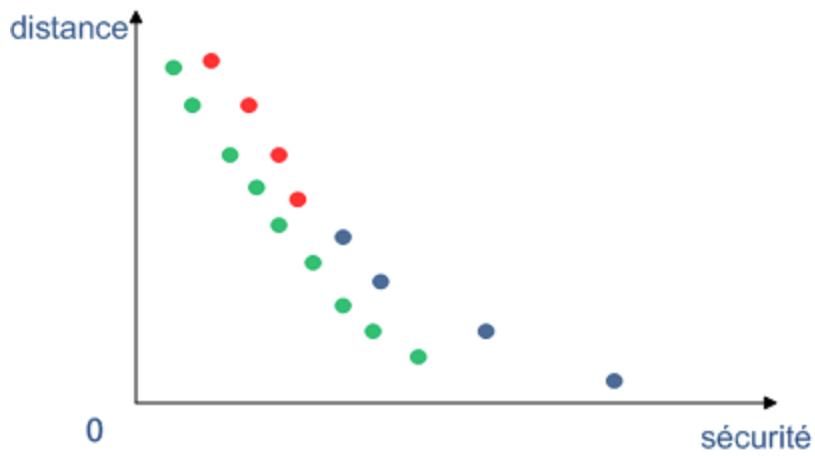


Figure 4.9 Les solutions vertes sont non dominées

#### 4.4.4 La limitation du temps

La limitation du temps permet à l'utilisateur de limiter de transport en vélo. L'utilisateur peut choisir le temps en heure et minute. Ces informations sont converties en distance. On suppose que la vitesse du vélo est entre 17km/h et 12km/h. On peut donc calculer la limitation maximale pour tous les points à visiter. Pour chaque visite différente, on peut obtenir la solution dont sa distance est la plus grande parmi un ensemble de solution. On convertit cette distance en heure et en minute d'après la vitesse de 17km/h, ça nous permet d'obtenir la limitation maximale. L'utilisateur peut choisir une limitation de temps dans un intervalle de limitation du temps. On prend un exemple ci-dessous (voir Figure 4.10). On suppose que l'utilisateur choisit 2 heures et 30 minutes comme la limitation du temps. On supprime les solutions dont leurs distances sont supérieures à la distance limite.

$$\begin{aligned}\text{La distance limite} &= 17 * (2 * 60 + 30) / 60 \\ &= 4250\text{m}\end{aligned}$$

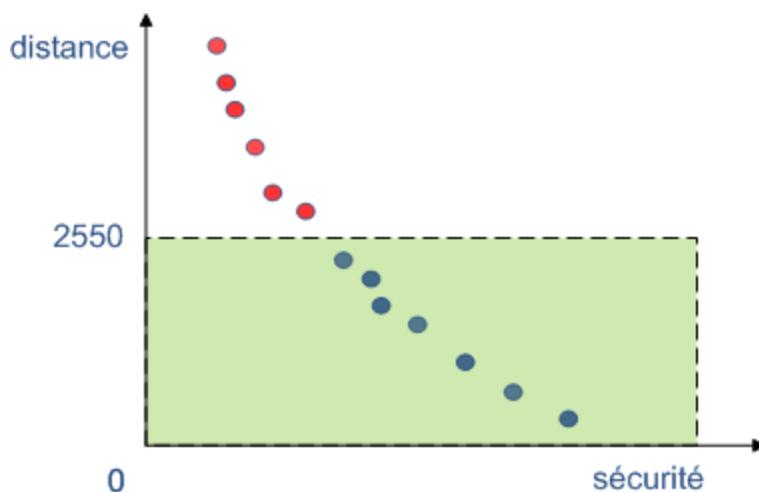


Figure 4.10 La limitation du temps

La figure 4.10 indique qu'il faut supprimer toutes les solutions (solutions rouges) dont la distance est supérieure à 4250. Il faut garder toutes les solutions qui sont dans la région verte. S'il n'y a pas de limitation, on ne considère pas la distance limite.

#### 4.4.5 La distance de crowding

La distance de crowding nous permet de sélectionner les solutions les plus représentatives. Cette méthode sert à estimer la densité de solutions. J'utilise l'algorithme de coupure qui se base sur la distance de crowding. Il faut donc choisir le facteur D concernant à la distance de crowding. S'il existe une

solution dont sa distance de crowding est inférieure ou égale au facteur D, je la supprime, parce qu'elle est très proche d'autres solutions. Donc le choix du facteur D est très important.

On prend un exemple et on suppose que le nombre maximal de solutions est au plus 5. D'abord, il faut faire la mise en ordre croissant de toutes les solutions par la distance (voir Figure 4.11). Ensuite je calcule la distance de crowding pour toutes les solutions sauf les extrémités. Les deux extrémités sont affectées à l'infini. Etant donné le nombre final de solutions inférieur à 10, on a donc besoin de sélectionner au plus de 9 bonnes solutions comme le résultat final à l'utilisateur. Pour supprimer les solutions mauvaises, il faut trouver le facteur D. Chaque fois, on calcule la distance de crowding maximale Dmax et la distance de crowding minimale Dmin. On prend 10 % de l'intervalle entre Dmax et Dmin plus Dmin comme valeur du facteur D. On peut assurer que chaque fois, il existe des solutions qui seront supprimées quand le nombre de solutions est supérieur ou égale à 10 (voir Figure 10).

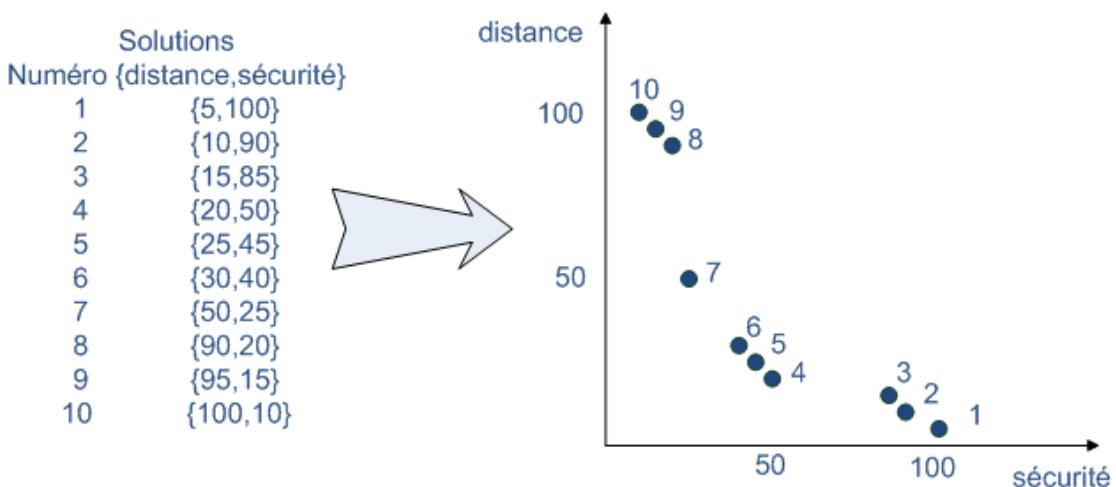
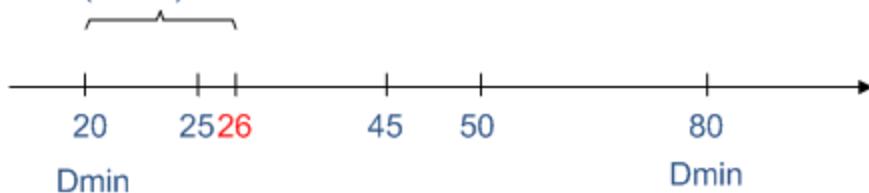


Figure 4.11 La mise en ordre croissant des solutions d'après la distance

$$D : 20 + \underbrace{(80-20)}_{20} * 10\% = 26$$



$$D = D_{\text{min}} + (D_{\text{max}} - D_{\text{min}}) * 10\%$$

Figure 4.12 Le choix du facteur D

Il faut supprimer les solutions dont leur distance est inférieure ou égale à D. Comme la figure 4.13. On supprime les solutions en rouges.

Mais il existe des problèmes comme j'ai écrit dans le chapitre 3. S'il y a au moins deux solutions adjacentes dont leur distance de crowding est inférieure ou égale à D, dans ce cas là, la suppression de ces solution influence le résultat final.

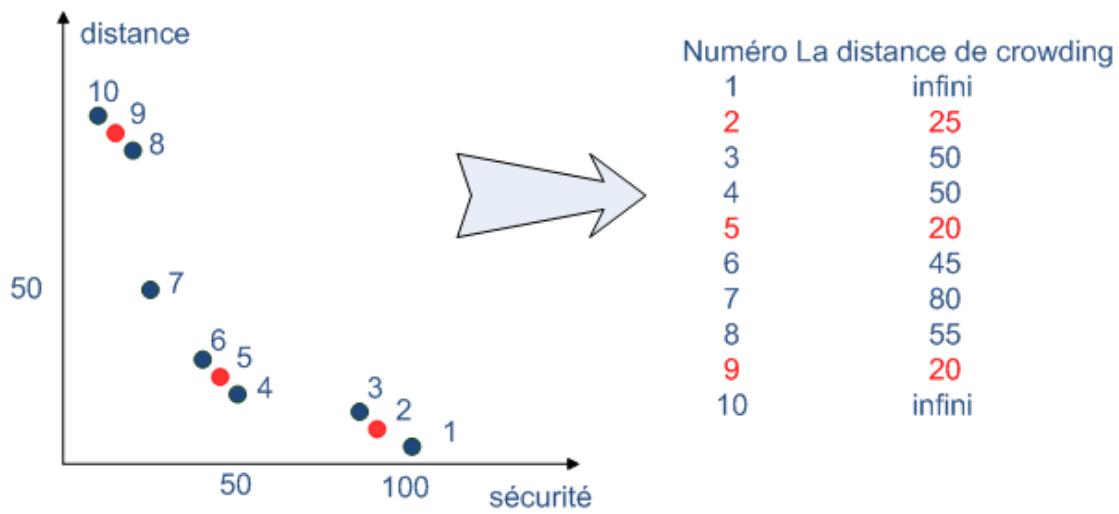


Figure 4.13 Calcul de la distance de crowding

## 4.5 La réalisation de site web

J'ai utilisé beaucoup de technologies pour créer le site web tels que HTML, CSS, Javascript, PHP, XAJAX, API Google Maps. Le design du site web que j'utilise se base sur un design existant. Il assure la cohérence pour toutes les pages web. Ils ont donc le même type de design. Dans ce secteur, je vais présenter les fonctionnalités du site web.

### 4.5.1 Le choix de points

L'utilisateur peut choisir le point de départ et le point de destination sur la carte. Et il peut aussi choisir les points d'intérêts comme les musées, les jardins, les monuments et les bâtiments (voir Figure 4.14)...

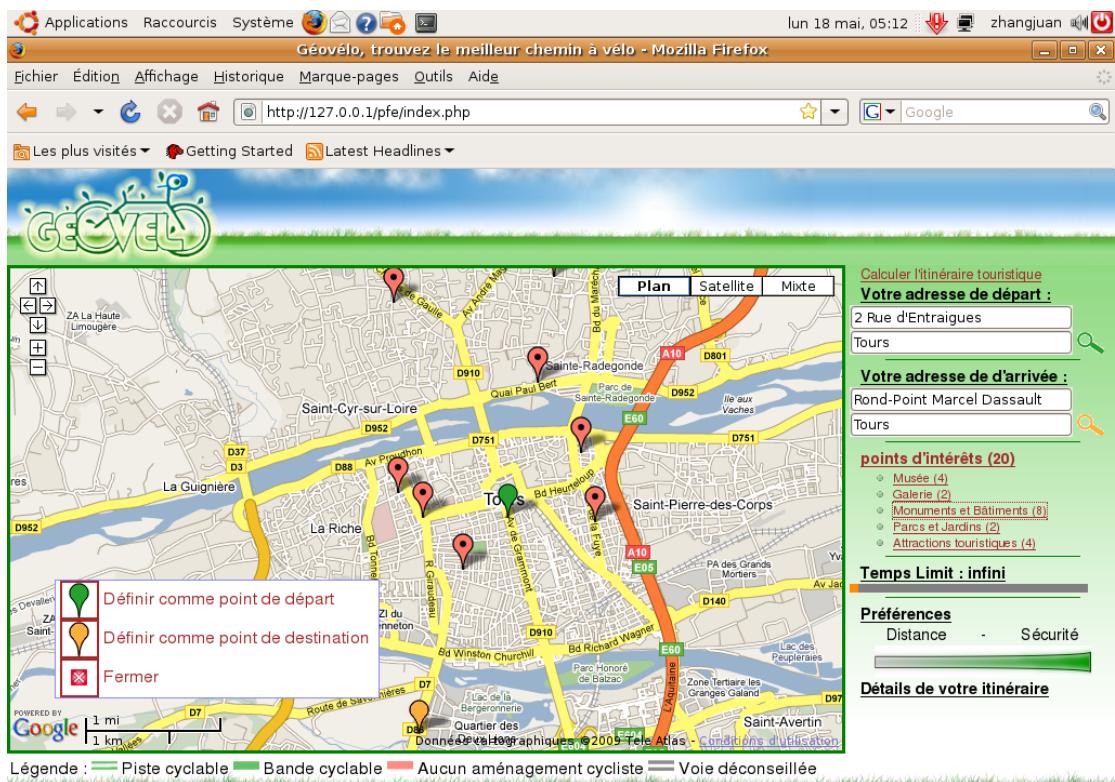


Figure 4.14 Le choix de points

Quand l'utilisateur clique sur le lien « musée » sur la carte, on fait appel à la fonction qui est dans le fichier .js. Dans cette fonction, on fait appel à la fonction de Xajax pour chercher les points dans la base de données. Cette fonction retourne les points qui correspondent aux conditions. Donc tous les points d'intérêts de type de musée sont affichés sur la carte. Pour chaque point affiché, l'utilisateur peut cliquer sur le bouton de gauche de la souris pour afficher toutes les informations contenant le nom, le type, l'image et un lien « ajouter » ou un lien « supprimer ». Quand l'utilisateur veut visiter un endroit, il peut cliquer sur le lien « ajouter ». Cet endroit est considéré comme le point à visiter. La couleur de ce point change alors de rouge à bleu. Quand l'utilisateur ne veut pas visiter l'endroit qu'il a déjà choisi, il peut cliquer sur le lien « supprimer ». La couleur est alors changée de bleu à rouge. Et on ne calcule pas d'itinéraire lié à ce point supprimé. Quand l'utilisateur choisit un autre type d'endroits, les points de l'autre type et les points déjà sélectionnés seront affichés sur la carte (voir Figure 4.15).

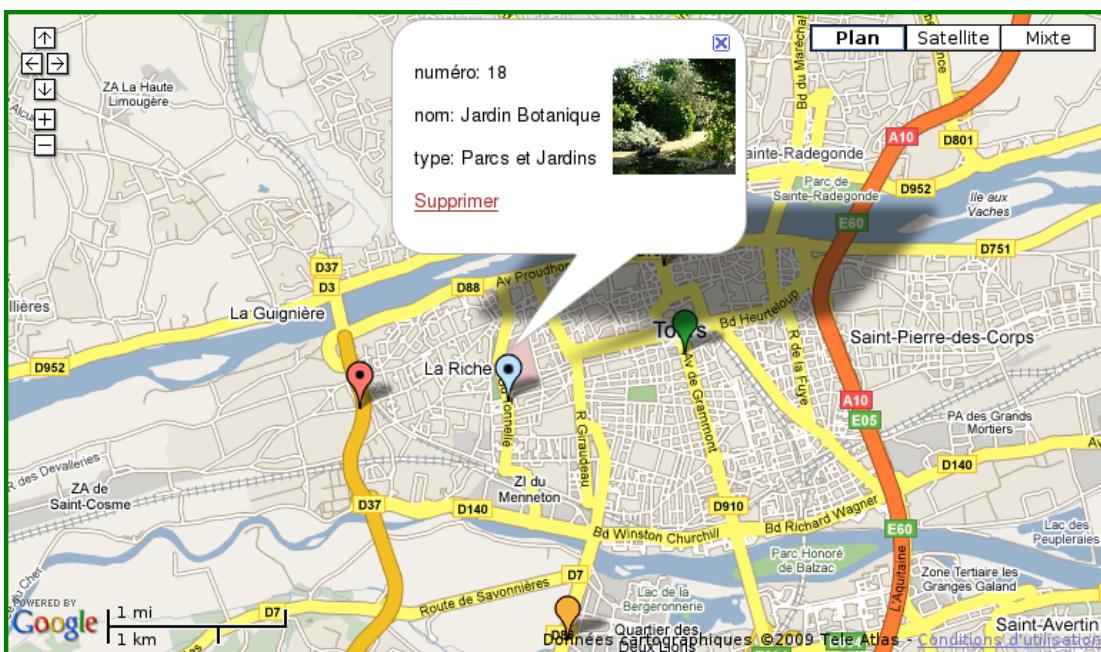


Figure 4.15 Ajout ou suppression d'un point d'intérêt

Après avoir consulté les points dans la base de données, on peut obtenir l'identifiant de chaque point. J'utilise un tableau pour sauvegarder l'identifiant de points que l'utilisateur veut visiter. Il est considéré comme un paramètre pour le calcul d'itinéraire.

#### 4.5.2 Le choix de solution

Pour obtenir les solutions, il faut transmettre les informations contenant le point de départ, les points d'intérêts, le point de destination, la limitation du temps ou la solution par défaut à la fonction dans le serveur. On peut cliquer sur le lien « Calculer l'itinéraire touristique » en haut à droite de la page web pour effectuer le calcul. Après avoir calculé, le serveur renvoi les valeurs du côté du client. D'après ces valeurs, on peut afficher les informations sur la carte.

D'abord, toutes les solutions calculées sont affichées sur la carte. Elles sont distribuées en moyenne dans la barre en fonction de la distance. La solution qui a la plus petite distance est une solution par défaut. Elle est marquée en rouge. Ensuite tous les tronçons liés à cette solution sont affichés sur la carte. Il existe quatre types de tronçons en différentes couleurs. Le type est défini en fonction de la catégorie et de la tranquillité de chaque tronçon dans la base de données. A la fin, la distance de la solution sélectionnée et l'intervalle de temps sont affichés sur la carte. L'intervalle de temps est obtenu à partir de la distance et de la vitesse à vélo (17km/h-12km/h) (voir Figure 4.16).

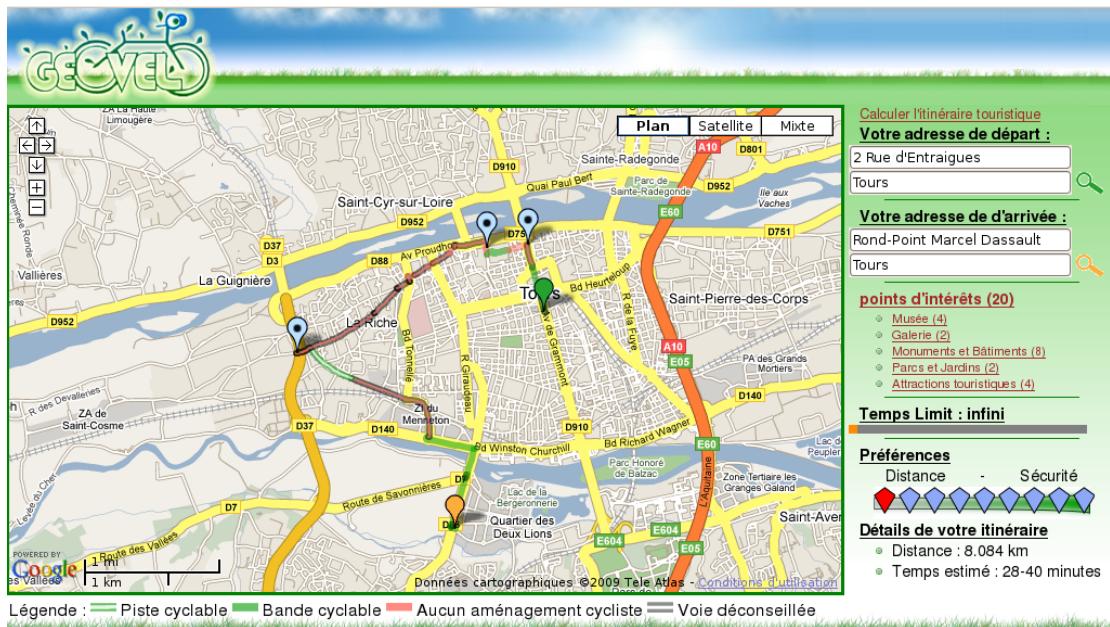


Figure 4.16 La solution par défaut

En plus, l'utilisateur peut choisir n'importe quelles solutions sur la barre. Quand il préfère la sécurité, il peut choisir la solution plus proche de la partie « Sécurité ». Donc l'autre itinéraire plus sécurisé sera affiché sur la carte (voir Figure 4.17).

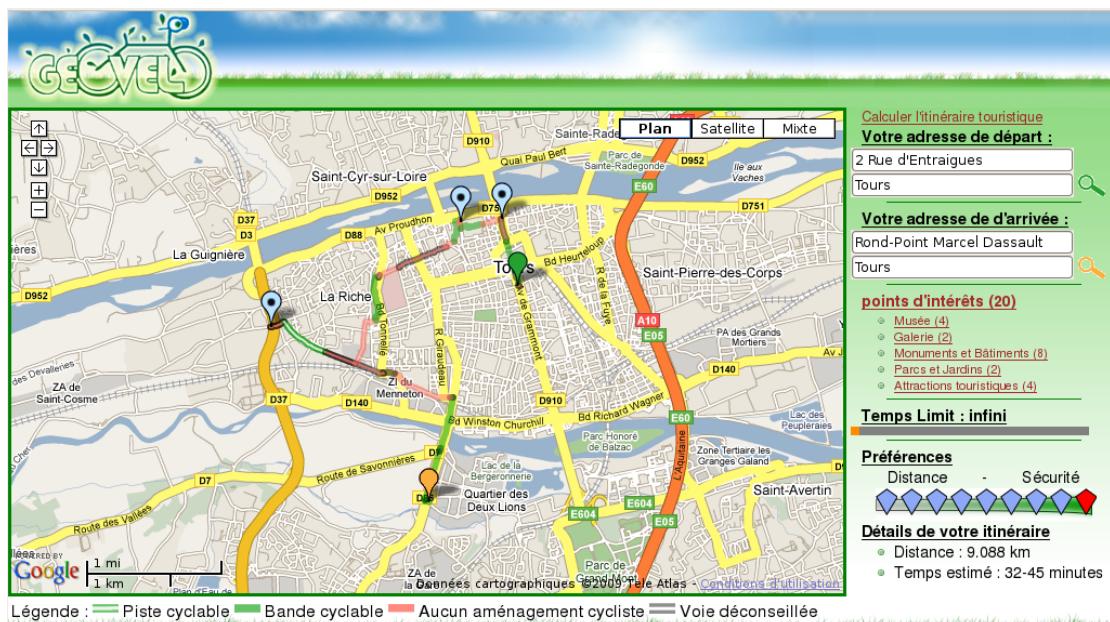


Figure 4.17 La solution la plus sécurisée

### 4.5.3 Le choix de limitation du temps

La limitation du temps sert à limiter le temps que l'utilisateur passe sur

son vélo. Par défaut, la limitation du temps est infini. Si l'utilisateur ne choisit pas de limite, je ne considère pas cette contrainte.

Pour assurer que la limitation du temps est valide, il faut calculer un intervalle entre la limitation minimale et la limitation maximale. Cet intervalle est changé à chaque que l'utilisateur clique sur le lien « Calculer l'itinéraire touristique ». Cette fonctionnalité est réalisée par un objet type « slider ». Le temps limite minimal est calculé en fonction de la solution par défaut (voir Figure 4.18). On considère le temps estimé maximal comme le plus petit temps limite. Quand l'utilisateur choisit le temps limite le plus petit, il existe au moins une solution pouvant être sélectionnée. L'extrémité gauche de ce « slider » est infinie et l'extrémité droite de ce « slider » est le temps maximal parmi toutes les solutions. Ici c'est la solution la plus à droite, parce que la solution la plus à droite a la distance la plus grande (voir Figure 4.19). Le calcul de temps est effectué en fonction de la distance. Si la distance est plus grande, le temps est plus long.

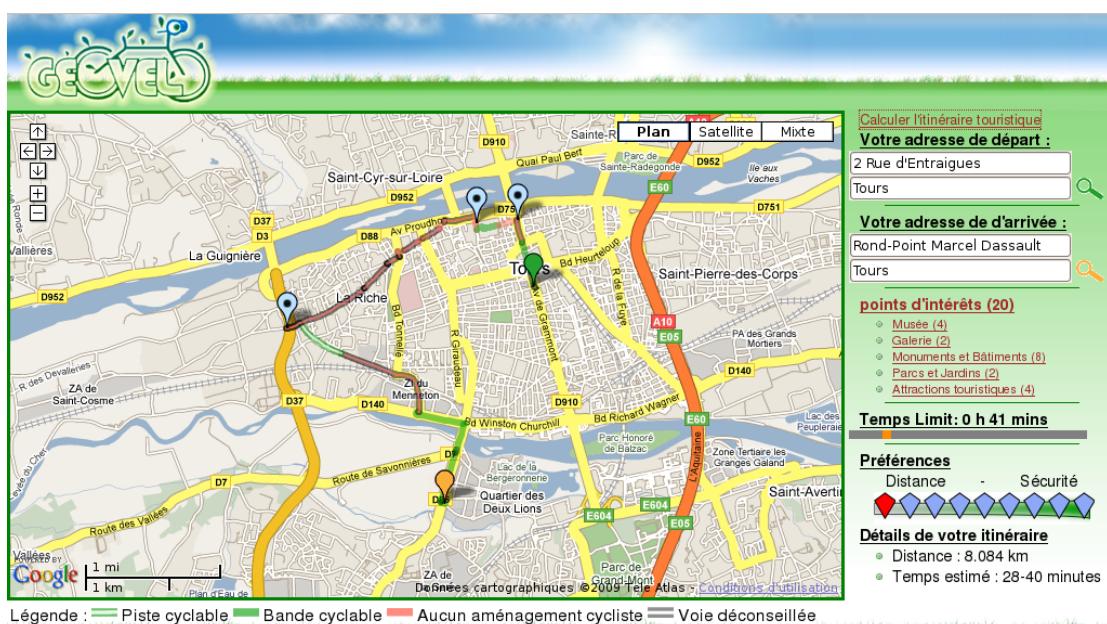


Figure 4.18 La limitation minimale du temps

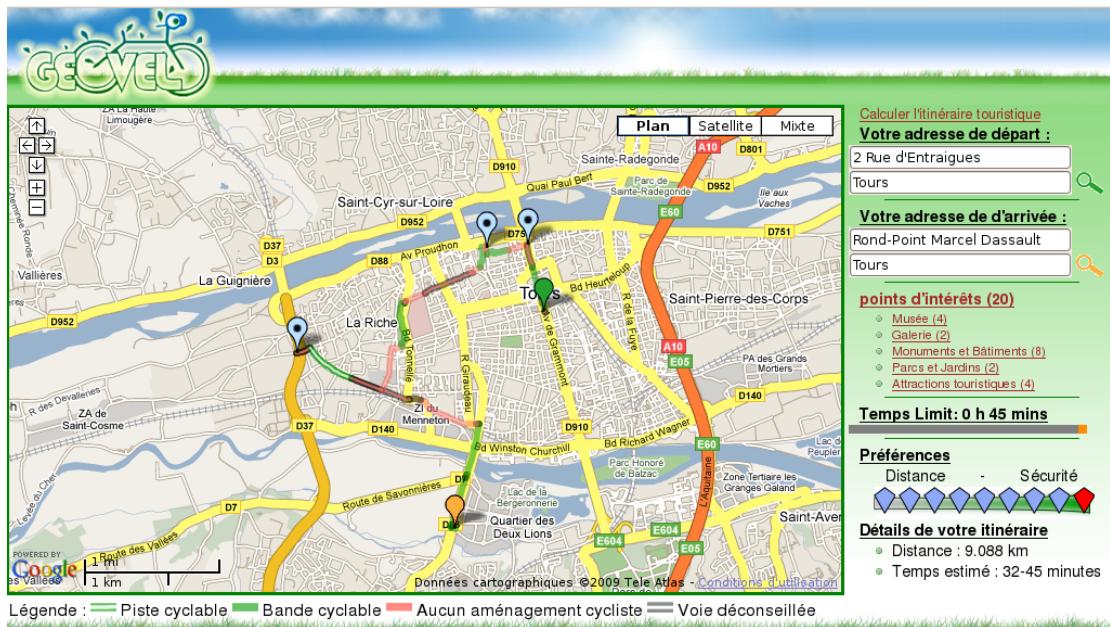


Figure 4.19 La limitation maximale du temps

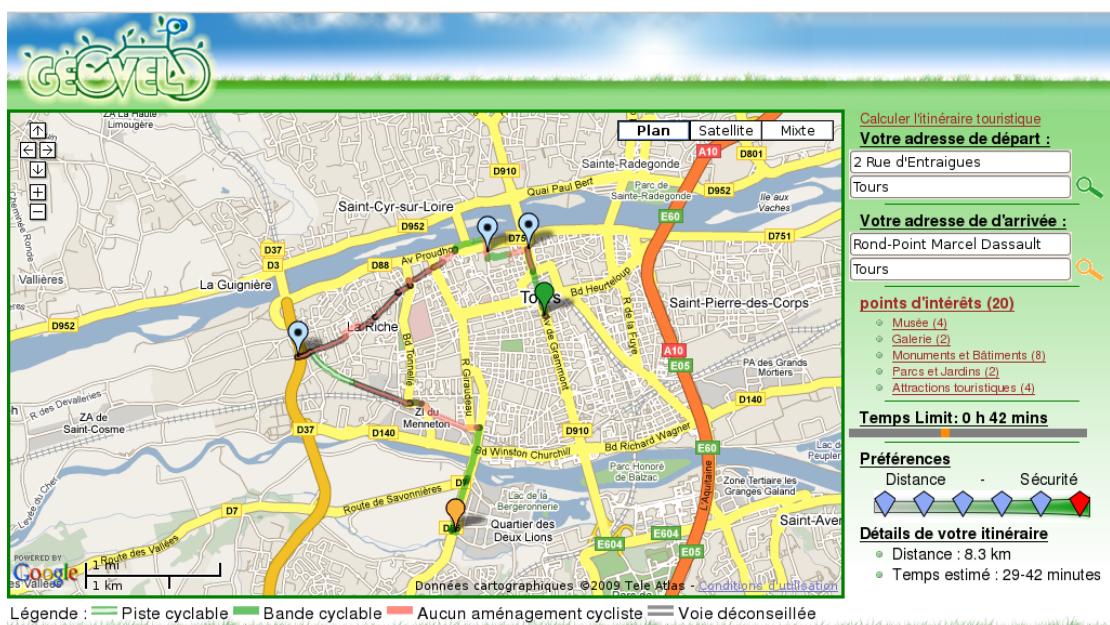


Figure 4.20 Les solutions sélectionnées en fonction du temps limite

Comme la figure 4.18 et 4.19, on peut voir l'intervalle de la limitation du temps. Quand l'utilisateur choisit un temps limite entre cet intervalle, les solutions satisfaisant cette contrainte sont affichées dans la barre (voir Figure 4.20).

On peut savoir que sur la barre toutes les solutions sont meilleures. Quand on choisit la solution la plus à gauche, la distance est la plus petite, mais l'itinéraire n'est pas sécurisé par rapport aux autres. Quand on choisit la solution la plus à droite, alors la solution est la plus sécurisée, mais la distance est la plus grande, le temps estimé est plus long.

## Chapitre 5 Les problèmes rencontrés

Au niveau du positionnement des points d'intérêt, il existe un problème mineur. Pour récupérer les points d'intérêts, on utilise une extrémité d'un tronçon qui est le plus proche de ce point d'intérêt. Les coordonnées de ce point ne sont pas réelles. Alors qu'il faudrait utiliser la position exacte.

Au niveau du calcul d'itinéraire, j'ai seulement calculé plusieurs ordre de visite en fonction de l'ordre par défaut (heuristique : choix du point suivant le plus proche). Ce qui veut dire que les solutions calculées sont optimales pour les ordres de visites calculés, mais pas pour tous les ordres possibles. Donc peut-être qu'il existe des autres solutions qui sont meilleures.

Quand on calcule la distance de crowding, peut-être qu'il existe des solutions adjacentes pour lesquelles leur distance de crowding est tous inférieures au facteur D. Quand on les supprime, cette méthode ne représente pas bien la densité des solutions. Quand on supprime la solution qui a sa distance de crowding la plus petite, il faut d'abord mettre en ordre la distance de crowding. Quand on la supprime, certaines distances sont changées. Il faut recalculer à chaque fois. Mais pour la mise en ordre, il faut prendre du temps. C'est un conflit. (Les détails sont dans le chapitre 3.)

Quand l'utilisateur choisit les points d'intérêts, tous les points sont parcourus une et une seule fois. Mais peut-être qu'il existe un point qui est parcouru au moins de 2 fois. Par exemple, on part de S, ensuite on prend le point 1 et le point 2 en fonction de la distance. Mais la fonction de l'algorithme est seulement effectuée d'après le départ et la destination. Donc quand on prend l'ordre S - 1 - 2 - 3 - T. Mais quand on prend S - 2 - 1 - 3 - T. Le point 1 est parcouru deux fois.

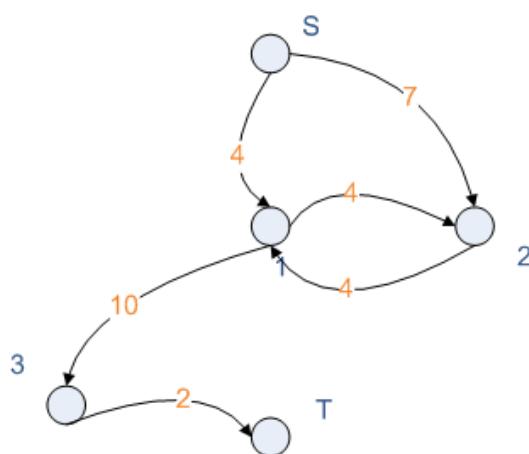


Figure 5.1 Le point 1 est parcouru deux fois

## Chapitre 6 Conclusion

Au cours de ce projet de fin d'étude, j'ai appris plusieurs technologies pour le site web et j'ai compris certaines méthodes et algorithmes liés par exemple au problème multi-objectif.

Ce sujet est intéressant, parce qu'il n'existe pas une unique solution et que calculer toutes les solutions peut prendre beaucoup de temps. Donc il est toujours possible de chercher des méthodes pour améliorer la performance du système et diminuer le temps de calcul. Donc le temps d'exécution et de réponse sont importants et ont toujours été considérés pendant mon PFE. Au long de mon PFE, j'ai bénéficié de connaissances de mon encadrant Gaël Sauvanet. Il m'a donné beaucoup de conseils et il est très responsable.

Pour mon PFE, le principal objectif a été atteint, j'ai créé les tables dans la base de données géographique pour réaliser le prétraitement et récupérer les points d'intérêts. J'ai aussi utilisé plusieurs méthodes pour trouver un ensemble de solutions en fonction des deux critères de la distance et de la sécurité.

Ce projet de fin d'étude m'a permis d'accumuler beaucoup d'expériences. La recherche de l'algorithme m'a permis de comprendre l'importance de l'optimisation. J'ai appris comment réfléchir sur des problématiques et résoudre des problèmes tout seul. Ce qui est très important pour moi.

## Références bibliographiques

1. <http://www.techno-science.net>
2. Cours de Théorie de graphe  
Polytech'Tours, Jean Charles Billaut
3. « A fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II »  
Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan  
Kanpur Genetic Algorithms Laboratory (KanGAL)  
Indian Institute of Technology Kanpur  
Kanpur, PIN 208 016, India
4. « LES ALGORITHMES GÉNÉTIQUES »  
La thèse de doctorat de Mme DRDI Le\_Ia (INRS-ETE, 2005)
5. Principes de l'algorithme de Dijkstra  
<http://www.nimbustier.net/publications/djikstra/djikstra.html>
6. « An Approach of Constructing Multi-Objective Pareto Optimal Solutions Using Arena's Principle » Journal of Software  
ZHENG Jin-Hua, JIANG Hao, KUANG Da (Institute of Information Engineering, Xiangtan University, Xiangtan 411105, China)  
SHI Zhong-Zhi (Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100080, China)
7. « Algorithme de Dijkstra »  
<http://www.xm1math.net/textes/termES/dijkstra.pdf>  
P.Brachet - Lycée B.Palissy - Agen  
2 janvier 2008
8. <http://www.geovelo.fr/proto.php>

## Annexe A Installation et configuration

Il faut d'abord installer les paquets ci-dessous sous le système d'exploitation de Ubuntu 8:

| Nom de paquet          | Utilisation                          |
|------------------------|--------------------------------------|
| apache2                | serveur web                          |
| apache2.2-common       |                                      |
| apache2-mpm-prefork    |                                      |
| apache2-utils          |                                      |
| libapache2-mod-php5    | module php pour apache               |
| php5                   | php                                  |
| php5-common            |                                      |
| php5-psql              | module postgresql pour php           |
| postgresql-8.2-postgis | serveur postgresql et module postgis |

Ensuite il existe les trois outils à installer :

| Nom d'outil   | Utilisation                                  |
|---|--|
| pgadmin3  | Gestion de postgres                          |
| codeblocks<br>( <a href="http://doc.ubuntu-fr.org/code_blocks">http://doc.ubuntu-fr.org/code_blocks</a> )   | IDE (integrated Development Environment)     |
| qgis<br>( <a href="http://download.qgis.org/downloads.rhtml">http://download.qgis.org/downloads.rhtml</a> ) | Visualisation pour les données géographiques |

Il faut ajouter les nouveaux dépôts pour la mise à jour et ajouter les clés par les sites ci-dessus. L'outil pgadmin3 est directement installé sous Ubuntu. Pour l'outil codeblocks, il faut installer 4 paquets : codeblocks, codeblocks-contrib, libcodeblocks0 et libwxsmithlib0. Et les trois paquets sont installés pour l'outil qgis : qgis, libqgis-gui0.11 et libqgis-core0.11.

Nous entrons « <http://localhost/> » dans un navigateur, il affiche « It works ! ». C'est signifie que le serveur web fonctionne. Le fichier index.html est dans le répertoire /var/www/.

Après avoir installé tous les paquets, il est nécessaire de configurer l'environnement.

Nous nous référons au site web :

« <http://doc.ubuntu-fr.org/postgresql> »

pour créer les nouveaux utilisateurs et la nouvelle base de données et pour les configurer. PostgreSQL est un serveur qui permet de se connecter à différentes bases de données. Par défaut, seul l'utilisateur « postgres » peut se connecter.

```
$ sudo su postgres
```

Password:

Puis tapez simplement :

```
$ psql
postgres=# CREATE USER <nom_utilisateur>;
postgres=# ALTER ROLE <nom_utilisateur> WITH CREATEDB;
postgres=# CREATE DATABASE <nom_utilisateur>;
postgres=# ALTER USER <nom_utilisateur> WITH ENCRYPTED PASSWORD
'mon_mot_de_passe';
postgres=# \q
postgres@ubuntu:~$ exit
nom_utilisateur@ubuntu:~$ psql
La ligne de commande est devenue :
nom_utilisateur=>
```

Notez la transformation du # en > : vous n'êtes plus superutilisateur...

Modifier le fichier en remplaçant ident sameuser par md5 afin d'obtenir les lignes suivantes :

```
...
local    all            postgres                      md5
# TYPE  DATABASE      USER      CIDR-ADDRESS        METHOD
# "local" is for Unix domain socket connections only
local    all            all                           md5
# IPv4 local connections:
host     all            all      127.0.0.1/32      md5
# IPv6 local connections:
host     all            all      ::1/128          md5
```

On peut modifier le droit d'utilisateur et faire un superutilisateur.

```
postgres=# alter user <nom_utilisateur> superuser;
```

Il faut attribuer le mot de passe pour l'utilisateur par défaut « postgres ».

```
$ psql -d postgres -c "ALTER USER postgres WITH PASSWORD 'mot_de_passe'"
```

Nous ouvrons l'outil pgadmin3 et cliquons « Fichier -> Ajouter un server » pour créer le nouveau serveur. On tape les commandes ci-dessous pour ajouter les tables dans la base de données (ici le nom de base de données et le nom d'utilisateur sont « toto »):

« postgres » est l'utilisateur comme « root » de postgresql.

```
$ su postgres
$ cd /usr/share/postgresql-8.2-postgis
```

La command createlang permet d'ajouter la langage de procédures PL/pgSQL sur la base de données toto.

```
$ createlang plpgsql toto
```

Ajouter les fonctions postgis sur la base de données toto.

```
$ psql -d toto -f lwpostgis.sql
```

Ajouter les différents systèmes de projections spatiales.

```
$ psql -d toto -f spatial_ref_sys.sql
```

Ensuite nous chargeons toutes les tables utilisées dans la base de données. Il est nécessaire de changer les droits pour l'utilisateur. La commande « alter » peut nous aider à changer les droits. Il faut désigner que l'utilisateur a les droits de modification pour la base de données et toutes les tables. Si l'on actualise dans l'outil pgadmin3, toutes les tables sont affichées.

Ex :

```
<superutilisateur>=# alter table spatial_ref_sys owner to <nom_utilisateur>;
<superutilisateur>=# alter table geometry_columns owner to
                           <nom_utilisateur>;
```

L'outil qgis sert à visualiser la base des données géographiques. Il faut entrer « qgis & » en ligne de commande pour ouvrir cet outil. Nous cliquons « Couche -> Ajouter une ou plusieurs tables PostGIS » pour créer une nouvelle connexion PostGIS.

Ensuite il faut installer les paquets pour c++ :

| Nom de paquet   | Utilisation   |
|-----------------|---|
| libxmlrpc++0    | librairie de communication XMLRPC                             |
| libxmlrpc++-dev | pour c++  |
| libpq5          | librairie de communication avec la base de données postgresql |
| libpq-dev       |   |

Après avoir installés les paquets, nous devons copier le dossier « geovelo » dans le répertoire « /var/www ». Nous tapons :

```
$ cd /var/www
$ sudo mkdir geovelo
$ sudo chmod 777 geovelo
```

Copiez tous les fichiers dans ce répertoire.

```
$ cd geovelo/
$ sudo chmod 777 *
$ sudo chmod 777 -R geovelo
```

Ensuite on tape:

```
$ nano .bashrc
```

Ajoutez « export LEDAROOT=/home/<nom\_utilisateur>/LEDA-6 » après cette phrase « export HISTCONTROL=ignoreboth ». Nous le sauvegardons. LEDA est la librairie C++ de types de données et d'algorithmes(conteneurs divers, matrices, graphes, combinatoire...).

Ensuite nous lançons l'outil Code::Blocs et ouvrons le projet « ServerXmlRpc.cbp ». On clique sur le bouton droite de la souris droite sur « ServerXmlRpc », choisit « Build options... » et clique « ServerXmlRpc ». On modifie le répertoire dans le « Linker settings » et « Search directories ». A la fin nous exécutons le fichier et entrons « <http://localhost/geoveloprototype.php> » dans un navigateur et ce projet doit fonctionner.

# Annexe B Cahier des charges

## 1. Contexte

Le projet de fin d'étude est proposé pour les étudiants en cinquième année du cycle d'ingénieur de l'Ecole Polytechnique Département Informatique de l'Université de Tours. Son but est de familiariser les étudiants avec les connaissances apprises en cours et d'en acquérir de nouvelles.

Le sujet de mon projet de fin d'étude est : calcul d'itinéraires touristiques pour vélo. Ce projet est proposé par le doctorant Gaël Sauvanet. Ce PFE est basé sur les travaux en cours réalisés par Gaël Sauvanet.

Ce projet est utilisé par les personnes qui veulent chercher un itinéraire touristique pour vélo sur le site web. Par rapport aux autres moyens de transport, le vélo est de plus en plus populaire. Grâce à ce moyen de transport, les touristes peuvent traverser tous types de chemins et s'arrêter n'importe où pour voir les paysages pittoresques le long de la route. Ils ne s'inquiètent pas du problème de stationnement. Un itinéraire touristique est calculé d'après les souhaits des utilisateurs comme les lieux de départ et de destination bien sûr, mais également à partir des préférences en terme d'attrait touristiques (jardins, musées, etc.). Le calcul d'itinéraire doit également tenir compte des critères de distance, de difficulté et de sécurité. Enfin, il est important de proposer plusieurs itinéraires à l'utilisateur qui en choisira ensuite un.

## 2. Analyse de l'existant

Ce PFE est basé sur les travaux de mon encadrant Gaël SAUVANET. Il a déjà réalisé un site web pour calculer un itinéraire en fonction d'un départ, d'une destination et de plusieurs critères comme la distance, la difficulté et la sécurité. Dans cette partie nous allons donc détailler le service existant.

### 2.1 Fonctionnalités

Un touriste peut choisir le départ, la destination et ses préférences. La solution contenant la distance et le temps estimé est affichée sur la carte. L'itinéraire affiché permet de visualiser le type d'aménagements (piste cyclable, bande cyclable, etc.) grâce à un code couleur.

Un touriste peut également cliquer sur le bouton de droite de la souris pour définir le point de départ et de destination. Après avoir choisi de façon graphique les points, le marqueur est positionné automatiquement à l'endroit géographique le plus proche d'un tronçon de route présent dans la base de données.

L'algorithme utilisé est le plus court chemin (ici une version modifiée de l'algorithme de Dijkstra).



Figure 1 le site geovelo

## 2.2 Technologies

Ce projet utilise plusieurs langages et APIs :

Site web : API de Google Maps, HTML, CSS, JavaScript, PHP, Xajax.

Base de données : Postgresql, PostGIS.

Communication : XMLRPC.

Calcul d'itinéraires touristiques : C++, LEDA.

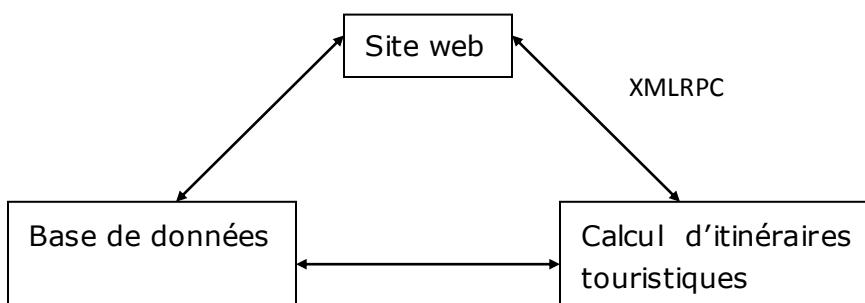


Figure 2 la structure générale de Géovélo

### API de Google Maps :

L'API Google Maps permet d'intégrer Google Maps dans son propre site Web avec JavaScript. L'API fournit des fonctionnalités concernant la manipulation de cartes et les services sur la carte (par exemple obtenir une

61 / 68

position géographique à partir d'une adresse). Il permet de créer des applications sur notre site Web.

**HTML :**

Le HTML (« HyperText Mark-Up Language ») sert à rédiger le contenu statique pour la page web.

**CSS :**

CSS(« Cascading StyleSheets ») nous permet de gérer les styles de présentation pour les pages Web.

**JavaScript :**

Le Javascript est un langage de script incorporé dans un document HTML. Il peut exécuté des commandes du côté client, sans passer par les procédures pour les applications web. Il est ici utilisé pour traiter les formulaires et appeler les fonctions xajax.

**PHP :**

PHP est un langage interprété exécuté du côté serveur et non du côté client. Il est conçu pour le Web. Le code PHP peut être incorporé dans un document HTML et il est exécuté par l'appel d'une page web. Le code est interprété par le serveur web et retourne le résultat.

**Xajax :**

Xajax est un framework dans PHP pour développer des applications Ajax. Il est utilisé pour développer des applications. Il fait appel de façon asynchrone à des fonctions de PHP et permet la mise à jour de contenu sans recharger la page.

**Postgresql :**

PostgreSQL est un système de gestion de base de données relationnelle (SGBDR) fonctionnant sur des systèmes de type UNIX (par exemple Linux, FreeBSD...).

**PostGIS :**

PostGis est module spatial pour Postgresql. Il permet de gérer des données telles que des points géographiques, des suites de points, des polygones, etc.

**XMLRPC :**

XML-RPC est un protocole RPC (Remote procedure call). Il est ici utilisé pour faire dialoguer les pages web PHP avec le serveur de calcul d'itinéraire qui est programmé en C++.

**C++ :**

C++ sert à réaliser l'algorithme de calcul d'itinéraire.

**LEDA :**

LEDA fournit une collection de types de données et algorithmes (matrices, graphes, combinatoire...). Il se présente sous la forme de classes C++.

### 2.3 Architecture de Géovélo

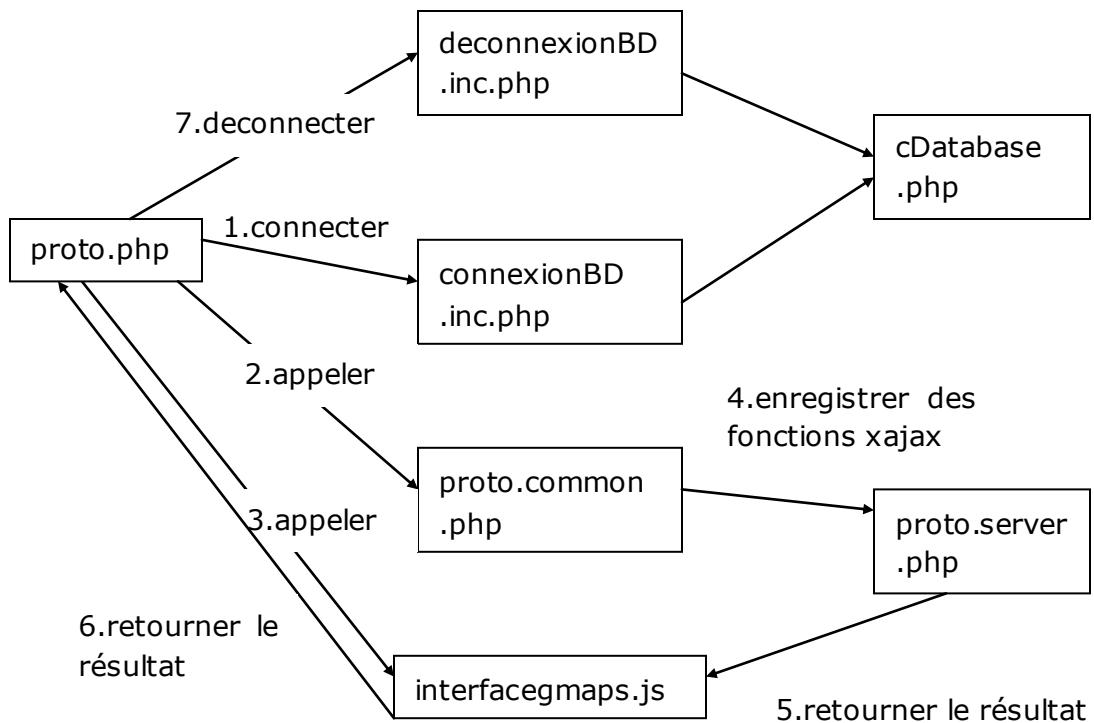


Figure 3 Architecture web de Géovélo

Les fichiers principaux :

**proto.php :**

Créer la page web et soumettre les formules.

**cDatabase.php :**

Créer la classe « Database » qui contient les méthodes de connexion et de déconnexion de la base de données.

**connexionBD.inc.php :**

Se connecter à la base de données.

**deconnexionBD.inc.php :**

Se déconnecter de la base de données.

**interfacegmaps.js :**

Recevoir le contenu de formules et appeler les fonctions xajax.

**proto.server.php :**

Créer les fonctions xajax faisant appel au calcul d'itinéraire, consulter la base de données, créer le client XMLRPC pour appeler à distance les fonctions et retourner les résultats à JavaScript.

**proto.common.php :**

Créer l'objet de Xajax, les enregistrements des fonctions à appeler à travers XAJAX, et le procède à l'envoi de la requête Ajax.

### 3. Description de la demande

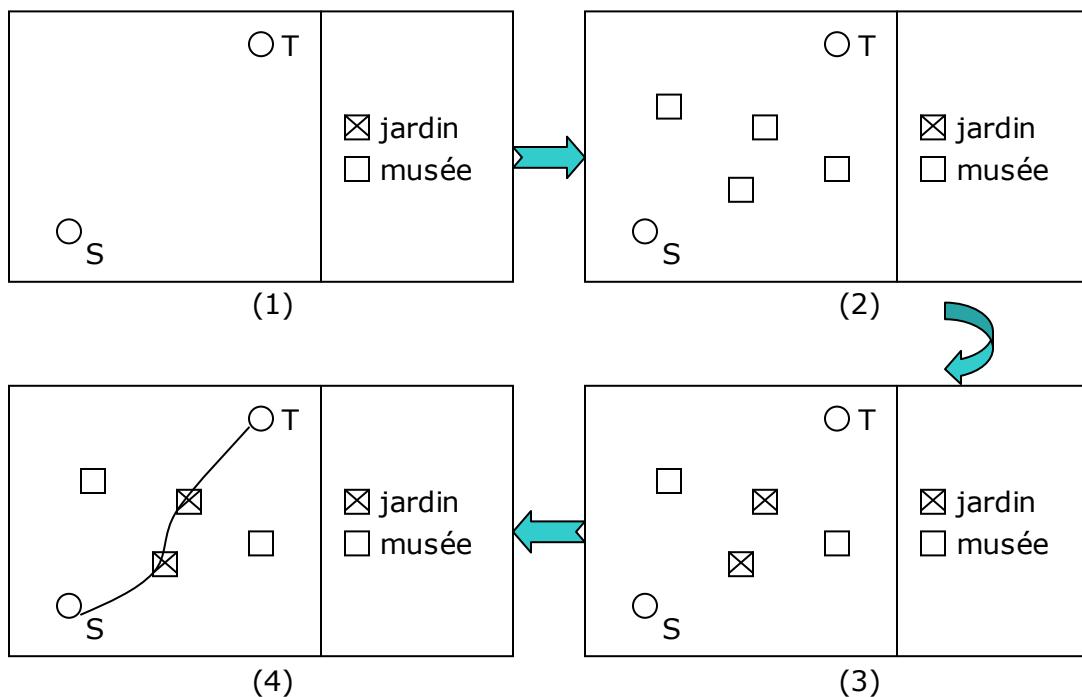
#### 3.1 Objectifs

Le sujet de mon PFE est calcul d'itinéraires touristiques pour vélo.

L'objectif de mon PFE est de mettre en place des méthodes de calcul d'itinéraires touristiques. Un touriste spécifie le départ, la destination sur la carte et choisit le type d'endroit à visiter. Tous les endroits sélectionnés sont affichés sur la carte. Un itinéraire touristique passe par un ensemble de points d'intérêts (patrimoines, paysages, etc.) préalablement choisi par l'utilisateur et doit optimiser plusieurs critères comme la distance mais aussi la difficulté et la sécurité. Le calcul d'itinéraire touristique sera basé sur une API permettant de calculer un itinéraire sécurisé d'un point à un autre qui sera fourni.

Mes missions sont :

1. installer les outils et configurer l'environnement
2. créer la base de données qui contiendra des informations sur les points d'intérêt (identifiant, nom, adresse, latitude, longitude, catégorie, pertinence/notes/intérêt, etc.)
3. trouver l'algorithme pour déterminer un ou plusieurs chemins touristiques en fonction des critères de distance, de difficulté et de sécurité.



(1) D'abord les clients choisissent le point de départ (S), le point de destination (T) et le type d'endroit.

(2) Tous les points sont affichés sur la carte.

(3) Les clients peuvent sélectionner les endroits qu'ils veulent visiter sur cette carte.

(4) Il faut construire le graphe, calculer les itinéraires et envoyer les solutions sur le site. A la fin les touristes peuvent choisir un itinéraire d'après la distance, la difficulté et la sécurité.

#### 4. implémenter cet algorithme en c++ sous la forme d'un web service

Je vais programmer en l'algorithme en C++ à partir de fonctions existantes qui permettent soit de calculer un chemin entre deux points selon un coefficient de compromis donné soit de calculer une ensemble de solutions intéressantes.

#### 5. créer une interface web permettant de configurer un itinéraire et d'afficher le chemin à suivre sur la carte.

Un travail de réflexion sera également fait sur l'intégration de l'aspect multimodal, c'est à dire que l'utilisateur peut emprunter plusieurs moyens de transport comme par exemple le train, le bus ou le vélo.

### **3.2 Expression des besoins**

#### **3.2.1 Besoins fonctionnels**

- Il faut réaliser une interface web pour choisir le départ, la destination, le type d'endroit (jardin, musée, ...).
- Une fonctionnalité doit permettre d'afficher les points d'intérêts sur la carte.
- Il faut mettre en place un calcul d'itinéraire touristique (algorithme + webservice).
- A partir de la carte, un utilisateur doit pouvoir obtenir le maximum d'informations sur un point d'intérêt (nom, position, description, commentaires, etc.) mais également en ajouter (notes, commentaire)
- L'utilisateur doit pouvoir effectuer un itinéraire train+vélo.

#### **3.2.2 Besoins non fonctionnels**

- L'interface doit être conviviale.
- Les différentes actions et opérations sur le site doivent être les plus simples possibles.
- Le calcul d'un itinéraire touristique doit prendre moins de 2 secondes.
- Une documentation d'installation et de configuration devra être rédigée.
- Il faudra également gérer différents messages d'erreur pour les utilisateurs et l'administrateur du site.

### 3.3 Produit du PFE

Les fichiers sources, fichiers exécutables, le rapport ainsi que le présent cahier des charges, seront remis aux encadrants à l'aide d'une clé USB, et seront également archivés sur l'intranet de Polytech'Tours.

### 3.4 Critères d'acceptabilité et de réception

Ce PFE doit être totalement achevé. Le code devra être lisible, les résultats exacts et la performance stable. Son exécution sur différentes plateformes devra avoir été testée, afin de s'assurer de sa portabilité et de son efficacité.

## 4. Contraintes

### 4.1 Date livraison

Il faut finir ce PFE en mai 2009.

### 4.2 Contraintes techniques

Ce PFE est exécuté sous Ubuntu8. Les méthodes retenues seront développées en C++ sous la forme d'un web service et mise en oeuvre sur un site web en php/xajax.

- plusieurs langages et APIs imposés:

API de Google Maps, HTML, CSS, JavaScript, PHP, Xajax, Postgresql, PostGIS, XMLRPC, C++, LEDA.

- plusieurs outils peuvent être utilisés :

CodeBlock : IDE(integrated Development Environment)

qgis : Visualisation des données géographiques

### 4.3 Autre contraintes

- Ce PFE utilise l'API de Google Maps. Il faut donc respecter les conditions d'utilisation de Google Maps.

- Il faut réutiliser l'architecture générale de Géovélo.

## 5. Déroulement du projet

### 5.1 Planification

Le projet se déroule de la manière suivante :

- premier contact avec l'encadrant
- remise d'un cahier des charges à l'encadrant
- étude et mise en oeuvre du projet
- prise de contact intermédiaire : premier bilan
- modification si nécessaire et finition
- point final, remise du projet ainsi que du rapport
- soutenance

## 5.2 Contraintes de délais

Pour la réalisation du PFE, je dois respecter plusieurs échéanciers.

Le stage dure deux semestres d'année 2008 - 2009 et commence le 08/10/2008.

### Points intermédiaires:

08/10/2008-14/10/2008 :

- Installer les logiciels et configurer l'environnement

15/10/2008-21/10/2008 :

- Connaître les fonctionnalités de l'existant

22/10/2008-28/10/2008 :

- Prendre en main HTML+CSS, PHP, JavaScript, C++

29/10/2008-11/11/2008 :

- Travailler avec l'API de Google Maps et programmer pour s'exercer
- Apprendre la gestion de la base de données et l'utilisation des fonctions dans Postgresql+postgis

- Se connecter et consulter la base de données

- Rédiger le cahier des charges

12/11/2008-19/11/2008 :

- Rédiger le document d'installation et de configuration

- Prendre en main Xajax

- Première étape de programmation pour afficher tous les points d'intérêts

18/11/2008-02/12/2008 :

- Prendre en main XMLRPC

- Etude de l'algorithme et discussion avec mon encadrant Gaël

03/12/2008-16/12/2008 :

- Prendre en main LEDA

- Déterminer une première version de l'algorithme avec mon encadrant

17/12/2008-18/03/2009 :

- Développement et discussions régulières avec mon encadrant

19/03/2009-15/04/2009 :

- Tests

- Corrections de bugs

- Amélioration des fonctionnalités

16/04/2009-mai 2009 :

- Augmenter la performance du système

- Rédiger le rapport

- Préparer la soutenance

**Résumé:**

Le projet de fin d'étude se déroule durant l'année 2008-2009. L'objectif de mon PFE est de mettre en place des méthodes de calcul d'itinéraires touristiques bicritère concernant la distance et la sécurité. Il faut trouver les bons compromis pour les deux critères et trouver meilleures solutions à proposer à l'utilisateur. Ce rapport comprend plusieurs phases et tâches du projet : le cahier des charges, la mise en oeuvre de la base de données géométrique, la recherche des méthodes et des algorithmes, les réalisations et les problèmes rencontrés.

**Mot-Clé :**

Dijkstra, multi-objectif, PHP, Xajax, Postgresql, Postgis, base de données géographique, front de Pareto, distance de crowding.

**Abstract :**

I carried out my final project during the years 2008-2009. The aim of my PFE is to implement methods for computing tourist itinerary on distance and safety. I found the right compromise for both criteria and found the solutions to the user according to the methods and algorithms. This report includes several phases and tasks of the project: the specification, the implementation of the geometric database, research of methods and algorithms, achievements and problems encountered.

**Keywords:**

Dijkstra, multi-objective, PHP, Xajax, Postgresql, Postgis, geometric database, Pareto front, crowding distance.

**Encadrants à l'école :****Gaël Sauvanet**[gael.sauvanet@univ-tours.fr](mailto:gael.sauvanet@univ-tours.fr)**Emmanuel Néron**[Emmanuel.neron@univ-tours.fr](mailto:Emmanuel.neron@univ-tours.fr)**Étudiant :****Juan ZHANG**[juan.zhang@etu.univ-tours.fr](mailto:juan.zhang@etu.univ-tours.fr)

Université François - Rabelais, Tours