École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
3e année
2010 - 2011

Rapport Projet Algorithme-C

Tournée d'un véhicule multicritéres

Encadrant
Emmanuel Neron
emmanuel.neron@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Cyrille PICARD
cyrille.picard@etu.univ-tours.fr
Michael PURET
michael.puret@etu.univ-tours.fr

DI3 2010 - 2011

Version du 9 juin 2011

Table des matières

Table des figures

Liste des tableaux

1. Introduction

Pour ce projet nous intéressons à réaliser un programme qui permet en fonction d'une liste de lieux de ressortir la liste des différents parcours possible pour réaliser cette tournée entre les différents lieux. Ce problème est un peu similaire à celui du voyageur de commerce. Le principe de l'application qu'on essaye de développer est de permettre à partir d'une Base de données représentant la configuration d'une ville. C'est à dire un ensemble de lieux relié entre par des arcs, de calculer un parcours pour passer par tout les lieux en fonction de leur intérêt pour l'utilisateur. La difficulté majeur est de retourner des solutions dans une période très courte.

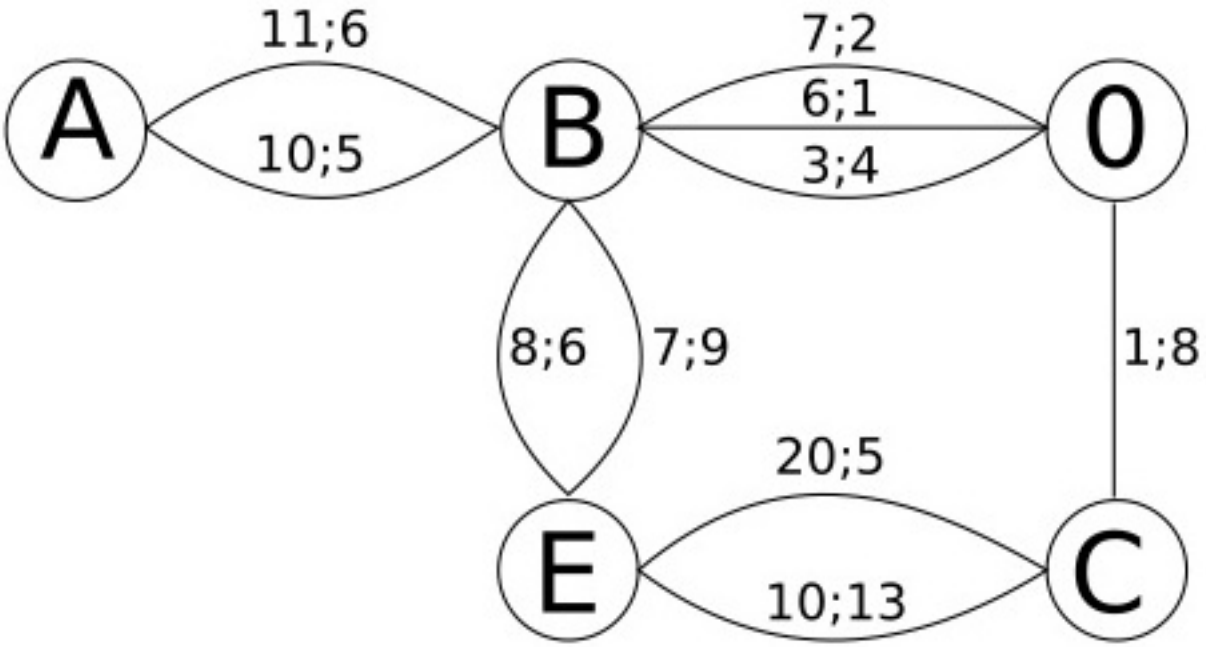

Figure 1.1 – Représentation simplifiée de la configuration

Première partie

Spécification

2. Cahier des charges

Suite à la première rencontre avec notre encadrant un premier cahier des charges a été évoquer. Le logiciel doit retourner la liste des différents parcours possible pour visiter une série de lieux dans l'ordre désiré par l'utilisateur. Le but est de calculer un parcours qui minimise l'insécurité et la distance à parcourir tout en maximisant l'intérêt des lieux.

2.1 Besoins

- Le programme devra retourner une solution de parcours viable rapidement,
- Le logiciel permettra de retourner une nouvelle solution si on interverti des lieux ou/et si on change l'ordre des lieux à visiter.
- Le programme doit être simple à utiliser
- Le programme peut retourner un certain nombre de parcours calculé dans le temps d'exécution impartie.

3. Modélisation du problème

3.1 Variables

Il y a des paramètres sur la configuration de la ville qui vont être importants à prendre en compte car ils vont influencer les résultats ce sont les variables. Un variable qui dépend de l'utilisateur c'est l'intérêt des différends lieux de la ville. Par exemple pour un même lieu deux utilisateurs vont pas lui attribuer obligatoirement le même intérêt. Les autres variables dont il faut tenir compte sont les caractéristiques des arcs qui sont la distance et l'insécurité.

3.2 Contraintes

La principale contrainte de ce problème est la liaisons entre les différents lieux, c'est à dire si il existe un ou plusieurs arc entre deux lieux pour pouvoir aller d'un lieu dit source à un lieu dit destination. Si on reprend la configuration présenter dans la figure on voit que pour aller du lieu de départ (0) au lieu A ou E il faut passer par le lieu B. En d'autre termes pour réaliser le Parcours on peut se déplacer d'un lieu à un autre si il existe au moins un arc reliant les deux lieux en question.

3.3 Fonctions Objectifs

Pour notre problème on peu considérer plusieurs fonctions objectifs qui sont les suivantes :
1re fonction : Il faut que l'ensemble des lieux soit visités en essayant de maximiser l'intérêt
2e fonction : Le Parcours doit minimiser la distance
3e fonction : Le Parcours doit aussi minimiser l'insécurité

4. Spécification

Maintenant que nous avons réaliser la définition et la modélisation du problème a été effectué, on va chercher à spécifier le programme qui permettra de répondre au problème et à définir de manière concise la structure général de ce programme.

4.1 Délimitation système/environnement

On peu représenter cette délimitation à l'aide du schéma ci-dessous

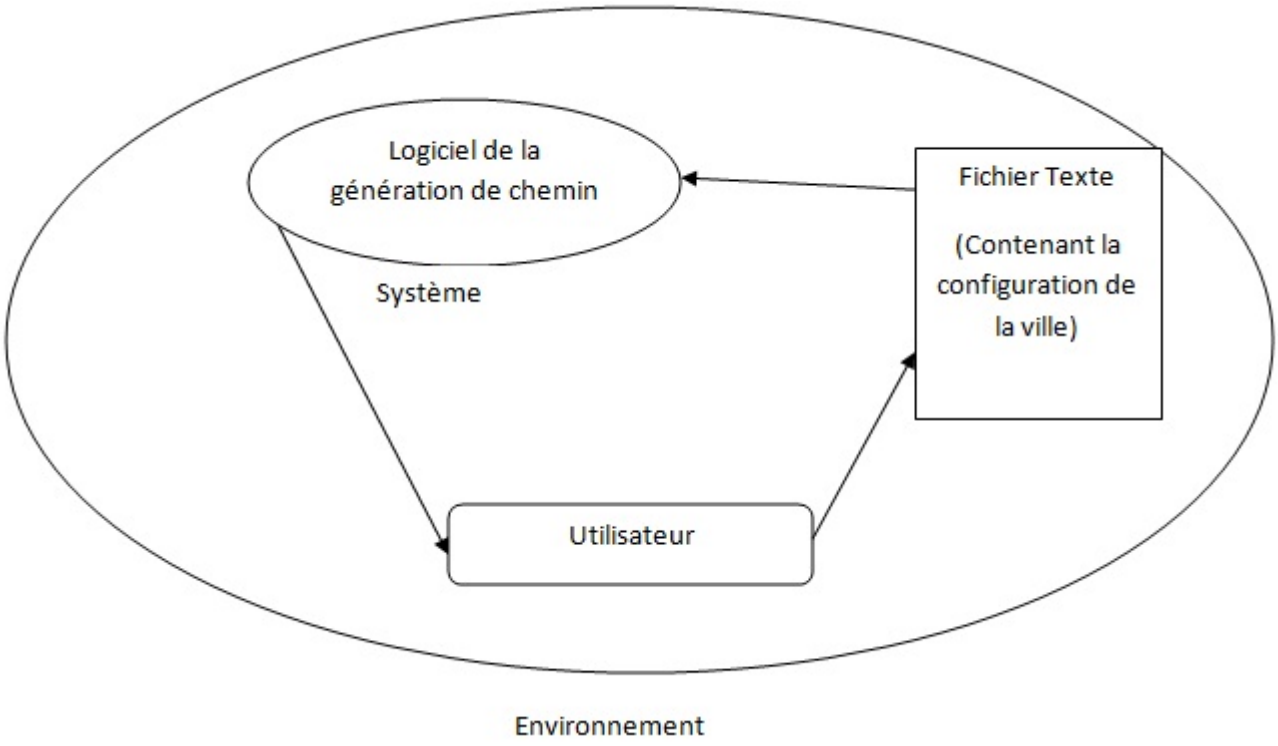

FIGURE 4.1 – Schéma représentant la délimitation système/environnement

En plus de distinguer le système de l'environnement le schéma précèdent nous permet de voir en même temps les flux d'informations entre les différentes identités de l'environnement.

- La flèche entre l'utilisateur et le fichier texte représente le processus par lequel l'utilisateur rentre la configuration de la ville, c'est à dire la position des lieux avec leur intérêt pour l'utilisateur et les arcs avec leurs caractéristiques.
- Le fichier texte une fois remplis par l'utilisateur sert d'entré au programme pour créer la structure de données

– Une fois que le programme a finis de tourner il retourne à l'utilisateur la liste des parcours possible par rapport à la ville qui a été rentré dans la fichier texte.

4.2 Définition du programme à réaliser

Pour réaliser un programme qui permet de répondre au mieux au besoins nous avons séparer le problème en 4-5 parties :

1. Pour gérer la configuration des lieux qui provient du fichier texte prévu comme entrée, il faut un ensemble de structure de données qui permet de stocker et d'utiliser les différents paramètre lier à la configuration, c'est à dire : l'intérêt des lieux, l'intercommunication entre les différents lieux, les arcs qui permette cette communication et les caractéristiques de ces arc (Distance, Insécurité).
2. Générer un chemin de base
3. Générer de nouveaux chemin en réalisant des permutations dans le chemin de base en fonction des permutations possible entre les lieux.
4. Avoir un structure permettant de retourner les chemins générer

On peut représenter de manière simplifier la structure générale du programme qui répondra au problème posé :

Deuxième partie

Algorithimique

5. Principe de fonctionnement du programme

Une fois tout l'étape de définition du problème et du contexte, ainsi que de modélisation du programme qui permetterai de répondre au problème. Nous avons pu commencer la partie réalisation d'algorithme et structure de données qui permette de répondre au problème. Cette partie a pour but de présenter le principe de fonctionnement du programme. Dans un premier temps, elle traite de la gestion des données fournies par l'utilisateur. Et dans une seconde phase, il sera question de la recherche de résultats. Le but étant toujours de tenir compte des fonctions objectifs :

- L'intérêt des lieux visité doit être maximisé.
- L'insécurité et la distance du trajet sont à minimiser.

Chaque lieu est relié par un ou plusieurs chemins et le retour sur des lieux déjà visités est autorisé. Comme le programme duplique les chemins en mémoire de manière à ce qu'il devienne unidirectionnel, nous ne parlerons plus de chemin pour relier les lieux, mais d'arc. Par abus de langage, nous utiliserons le terme ă"chemin" comme équivalant à "trajet". Ils désignent tous deux un résultat.

Exécution du programme étape par étapeă :

1. L ?utilisateur fournit l ?ensemble des données au programme par l ?intermédiaire d ?un fichier texte. Celui-ci est passé dans les paramètres lors lancement du solveur.
2. Le programme mémorise l ?ensemble des informations.
3. Il y a un tri et une suppression des informations erronées.
4. Le programme recherche le chemin de référence en ajoutant un lieu de plus au parcours déjà existant.
5. Le chemin de référence est utilisé pour produire des solutions optimisées, placées dans la pille en vue d ?un traitement final.
6. Les solutions sont dépilées, l ?ensemble des possibilités est calculé et enregistré un tableau de résultats classé par nombre de lieux.
7. On remonte au pointă4 tant que tous les lieux ne sont pas dans la solution et qu ?il reste du temps de recherche.
8. On affiche les résultats sur la sortie standard.

Pour récupérer les résultats dans un fichier texte, il suffit de rediriger la sortie standard vers le fichier de résultats. Sous Window comme sous Linux, la redirection se fait avec le symbole supérieur. *Exemple :*

```
\$ : nom_du_programe [parametre(s)] > chemin_du_fichier
```

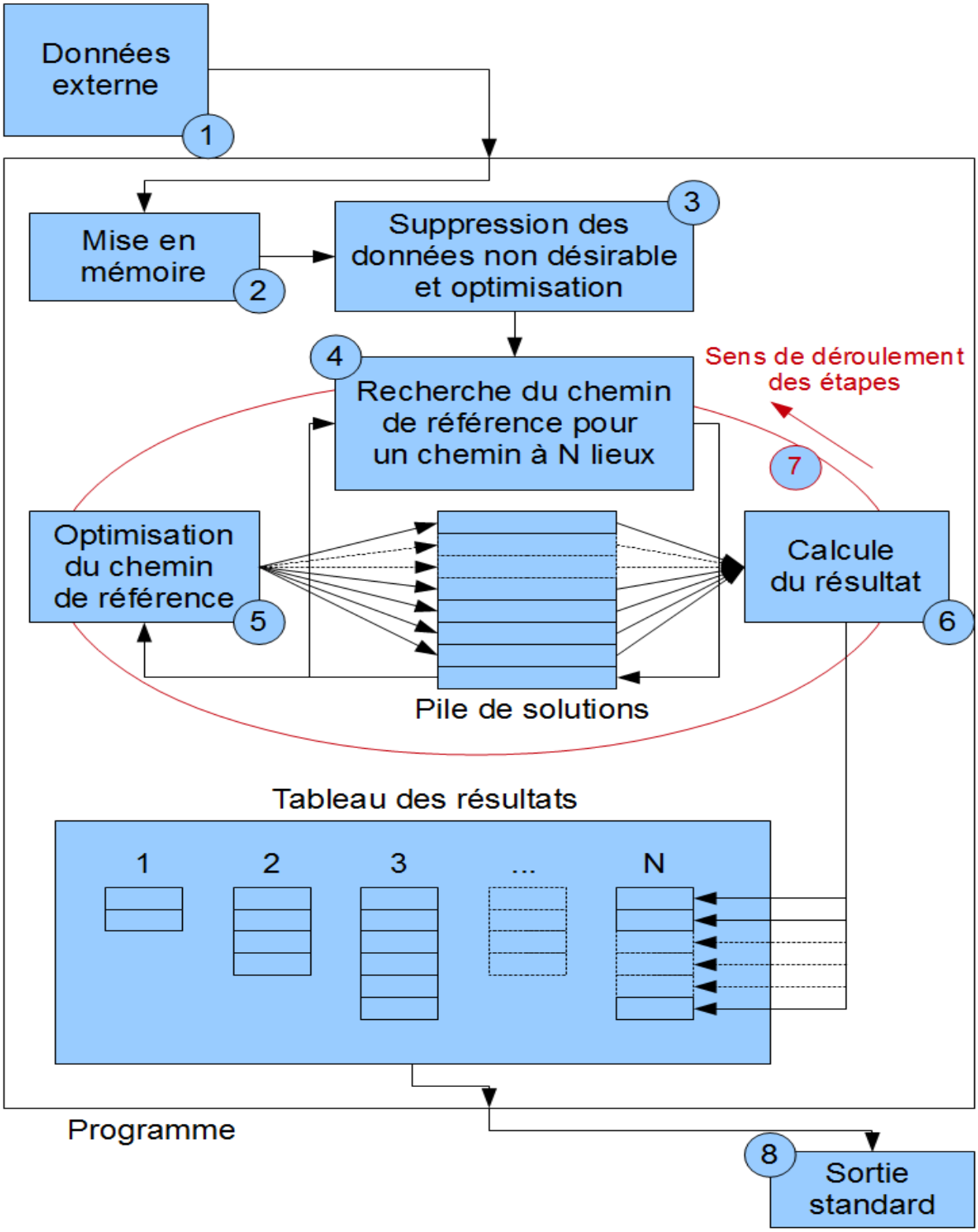

FIGURE 5.1 – Schéma représentant le principe de fonctionnement du programme

6. Gestion des données en mémoire

6.1 Données à mémoriser

L'utilisateur fournit les informations aux programmes par l'intermédiaire d'un fichier texte. Ce document répond à une structure particulière :

- Il est constitué de trois parties : Paramètres, Lieux, Arcs.
- Le nom des parties commence par un dièse et se termine par un saut de ligne.
- Les commentaires sont précédés du symbole pourcent et sont placés avant que commence l'une des trois parties.

6.1.1 Les paramètres

Toutes les informations sont contenues sur une seule ligne et séparées par des points virgule. On retrouve :

- Temps de recherche en seconds (entier)
- Nombres de lieux totaux (entier)
- Nombres d'arcs totaux (entier)
- {d|c} caractère qui indique si, lors de la réalisation du chemin de référence, les lieux doivent suivre un intérêt croissant ou décroissant (caractère)

Exemple :

```
#Parametres
60;5;11;d
```

Indique un temps de recherche de 60s dans un graphe constitué de 5 lieux, 11 chemins. Le chemin de base doit être constitué suivant un intérêt décroissant.

6.1.2 Les lieux

Les lieux sont caractérisés par trois valeurs et espérés par un retour à la ligne. Le premier lieu est celui de départ de la recherche, il est numéroté zéro et possède un intérêt nul.
Paramètres des lieux :

- Numéro : commence à partir de zéro et s'incrémente d'un à chaque lieu. (entier)
- Intérêt : indique la valeur d'intérêt du lieu. (entier)
- Nom du lieu : le nom du lieu (chaine de caractères)

Exemple :

```
# Lieux
0;0;Départ
1;6;L1
2;2;L2
3;12;L3
4;1;L4
```

Il y a 5 lieux numérotés de zéro à quatre possédant chacun un intérêt est un nom.

6.1.3 Les Arcs

Les arcs sont séparés par des retours à la lignes, ils sont constitués de quatre paramètres :
- Le numéro de lieu de départ (entier)
- Le numéro du lieu d'arrivée (entier)
- Sa distance (entier)
- Son insécurité (entier)

Exemple :

```
# Arcs
0;3;10;1
0;3;7;2
0;3;8;3
0;3;1;2
0;2;7;9
0;1;6;3
0;1;2;5
2;3;2;2
2;1;7;1
2;1;6;2
2;1;5;3
2;4;8;8
2;4;9;9
4;1;1;9
```

On retrouve ici 11 arcs, tous reliant des lieux et possédant une distance et une insécurité.

Les exemples précédents permettent de créer une ville de cette forme :

6.2 Mémorisation des informations

Toutes les informations utiles au programme sont contenues dans une structure principales nommée "Donnee", cela permet d'avoir un programme ordonné, avec une seule variable à passer en paramètre aux fonctions

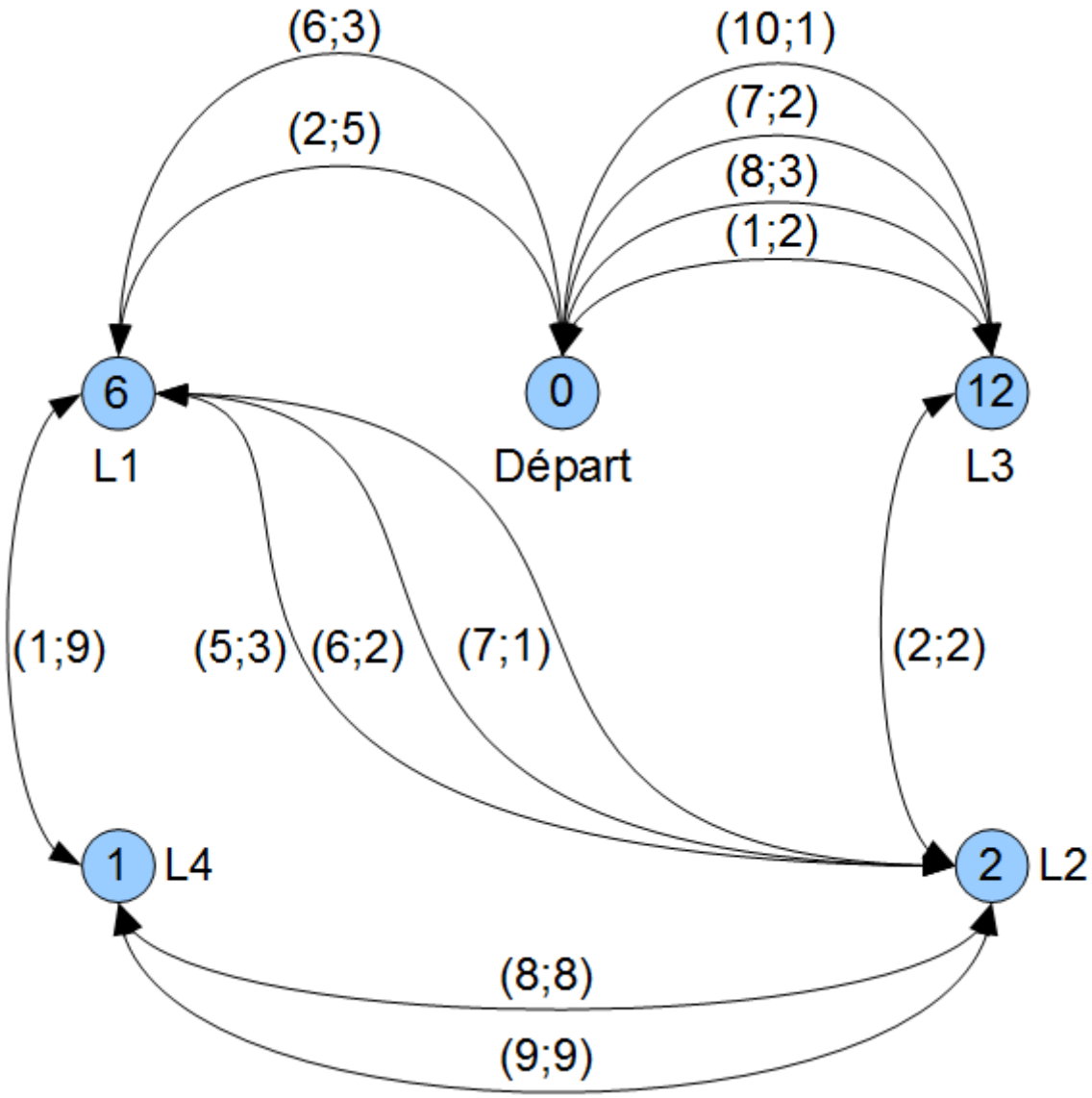

FIGURE 6.1 – Schéma représentant la configuration

6.2.1 Structure de "Donnee"

- parametres (Parametres)
- liste_lieu (Coef_lieu)
- lieu (Lieu)
- index_lieu (Index_arc)
- map (Arc)
- resultat (Resultats)
- solution (Solution)

On va détailler les informations contenues dans cette structure dans les parties suivant.

6.2.2 Les paramètres de la recherche

La structure "Parametre" contient les informations sur la recherche, voir la partie correspondante pour plus d'information.

Structure Parametres :

- temps_execution (entier)
- nb_lieux (entier)
- nb_arcs (entier)
- ordre_lieu (caractère)

6.2.3 La gestion des lieux dans la mémoire

Les lieux sont contenus dans un tableau "lieux" de type "lieu" qui est accessible de puis la structure générale. Il y a sont stockés en fonction de leur apparition dans le fichier de données. Le tableau est donc ordonné en fonction de leur numéro.
La structure "lieu" est utilisée pour contenir toutes les informations relatives à un lieu, ref section correspodante.

Structure Lieu :

- id (entier)
- interet (entier)
- nom (chaine de caractères)
- nb_arc (entier)

On détaille cette structure de cette manière :

- id : fait référence au numéro du lieu du fichier de donnée.
- interet : valeur
- nom : celui du lieu
- nb_arc : nombre d'arc sortant de ce lieu

6.2.4 La gestion des lieux pour le chemin de base

Comme le solver doit fournir des solutions comportant les chemins intermédiaires, il est nécessaire de pouvoir rajouter rapidement un lieu au chemin de référence dès que la recherche pour n lieux est terminée.
Pour éviter d'avoir à parcourir la liste des lieux à chaque ajout, le tableau liste_lieu de paramètre ordre_lieu renseigné par l'utilisateur.

6.2.5 La gestion des arcs

Tous les chemins décrits dans le fichier sont dupliqués de manière à devenir unidirectionnelle, c'est pourquoi l'on parle d'arc.
Ils sont contenus dans un tableau à trois dimensions nommé "map", de type "Arc" et est accessible depuis la structure générale. La première dimensions permet de pointer les arcs stockés en mémoire. La troisième dimension est l'arc lui-même. Cette table est ainsi construite afin de permettre les tris et la suppression des arcs dominés.
La structure "Arc" est utilisé pour contenir toutes les informations définissant un arc. Voir partie sur

arc pour plus d'information.

Structure Arc :

- distance (entier)
- insecurite (entier)
- depart (entier)
- destination (pointeur lieu)

Le départ et la destination sont des pointeurs sur les structure de type "lieu" détenues par lieux de la structure data. Faire cette référence évite de surcharger la mémoire d'informations redondantes.

Voici a quoi doit ressembler la table "map" après suppression des arcs dominés dans l'exemple du chapitre précédent.

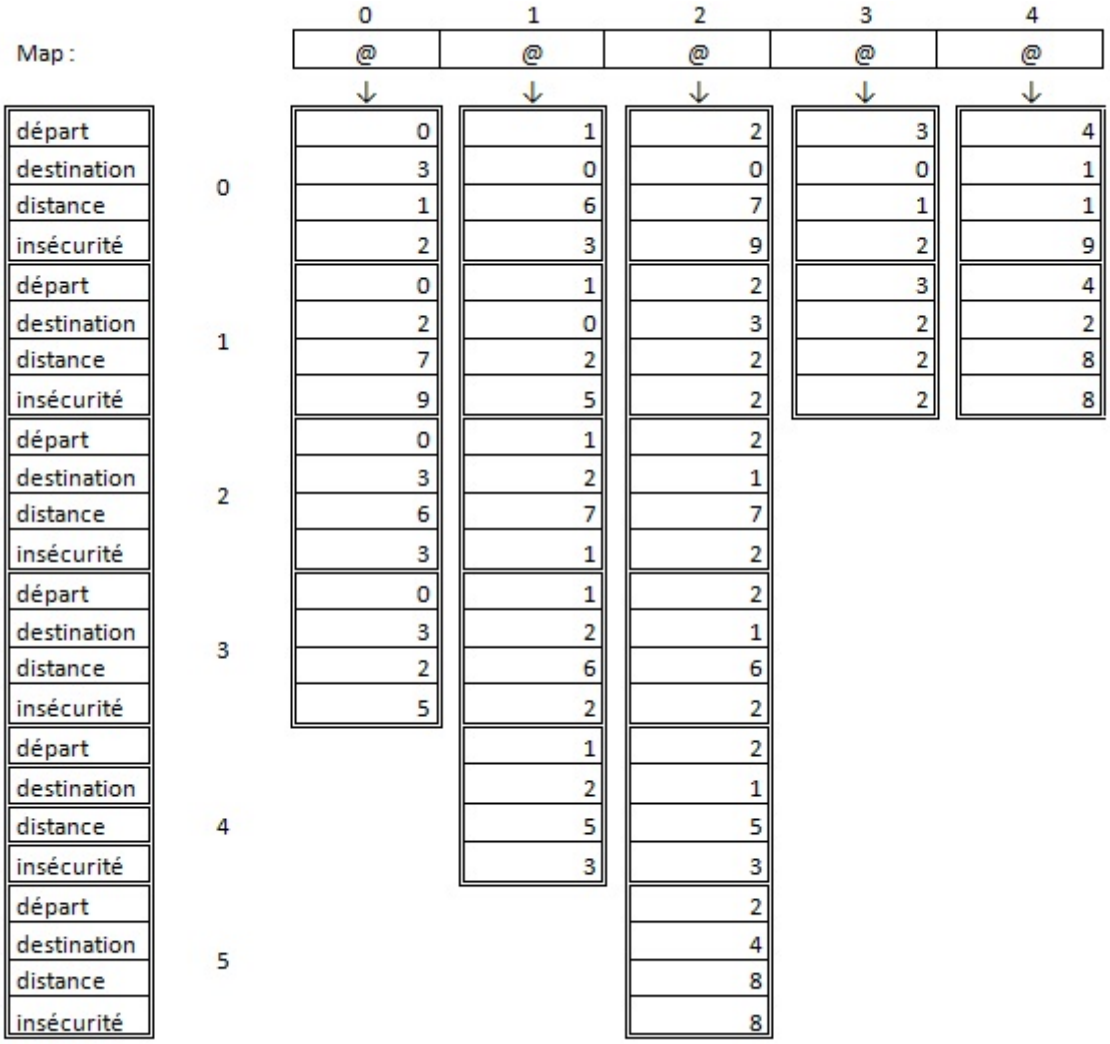

FIGURE 6.2 – Schéma représentant la structure "map"

6.2.6 Relation entre le lieu de départ et le lieu d'arrivée

Le tableau "map" indique déjà le lieu de départ, mais comme le nombre d'arcs entre les lieux est variables, seul le Parcours des dimensions deux et trois permet de connaitre le lieu de destination. C'est pour éviter cette recherche que l'on créé une table d'index nommé "index_lieu" de type "index_arc" et accessible depuis la structure "Donnee". Il s'agit d'un tableau carré, utilisé comme une tableau à deux entrées (le lieu de départ et le lieu d'arrivée). La structure "index_arc" permet de savoir combien d'arcs ces lieux ont en commun et contiens l'identifiant du premier arc utile dans le tableau map[lieu_depart].

Structure index_arc :

- itemize id_arc (entier)
- itemize nb_arc (entier)

Voici une représentation possible de la table index_lieu et des structures Inde_arc. Les adresses en mémoire sont fictives.

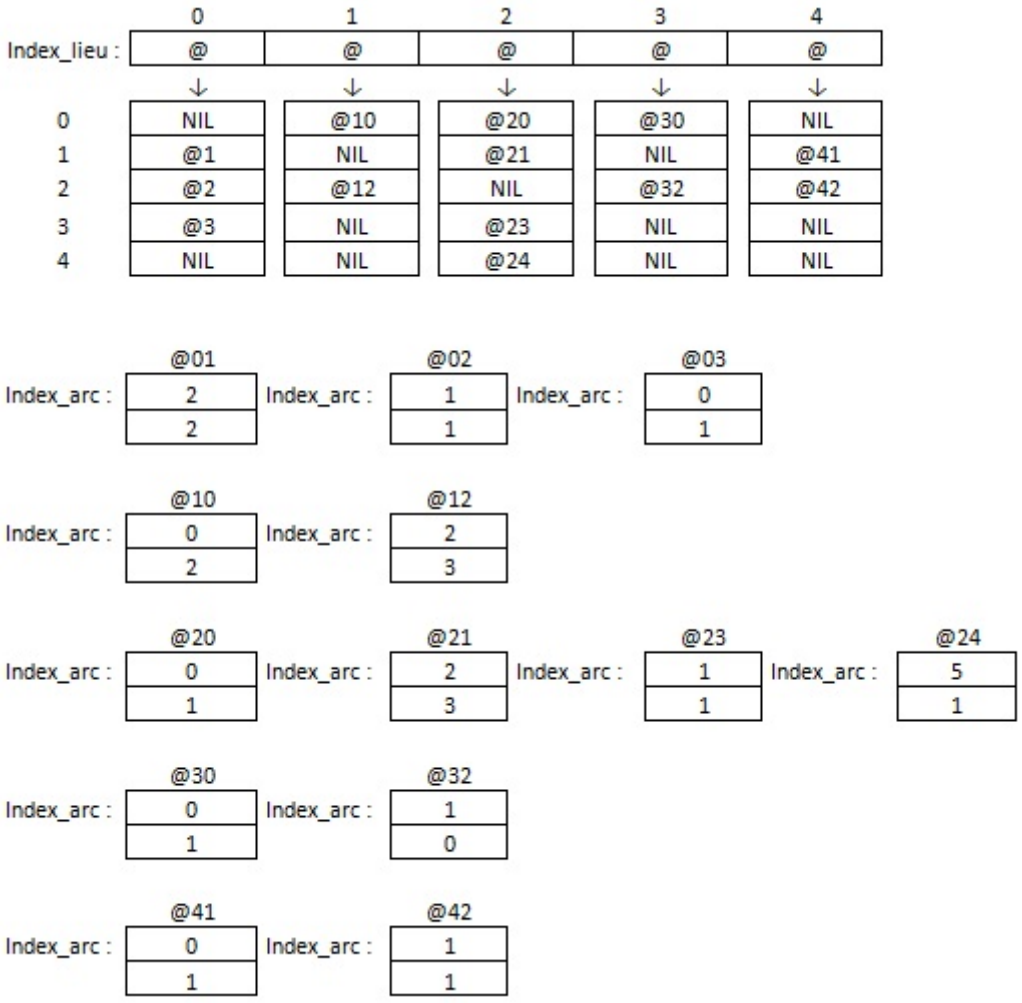

FIGURE 6.3 – Schéma représentant la structure

6.2.7 La gestion des solutions

Le tableau des solutions était prévu pour contenir les solutions obtenus après la recherche. Mais cela posait des problèmes avec l'algorithme permettant de générer tous les chemins avec les différents arcs.
C'est pour cela que cette table est maintenant utilisée comme une pile de résultats intermédiaires, nommés "solution", pour qui tous les chemins non pas encore étaient généré.
Cette table nommée "solution", de type "solutions" et accesible dans la structure "Donnee"
La structure "solution" permet de contenir tous les chemins obtenus grâce aux algorithmes de recherche. Elle contient le nombres de solutions disponibles dans cette table est un tableau nommé "solutions" de type "Parcours".
La structure "Parcours" définit un résultat, une partie "carac" de type "caractéristique" contient les informations générales sur la solution, un tableau "trajet" contient tous les arcs utilisés dans l'ordre de Parcours. "Itiniaire" est l'équivalent de trajet, mais pour les lieux, quant à la table "visite", elle n'est utilisée que par l'algorithme de génération du chemin de base pour savoir quel lieu est déjà présent sur le chemin.
La structure "caractéristique" contient les informations globales de la solution. C'est à dire, l'intérêt, la distance, l'insécurité, le nombre de lieux total, le nombre de lieux utile et le nombre d'arcs du chemin.

Structure Solutions :

- nb_solution (entier)
- solution (pointeur Parcours)

Structure Parcours :

- Carac (Caracteristique)
- Trajet (pointeur Arc)
- Itineraire (pointeur Lieu)
- visite (entier)

Structure Caracteristique :

- interet (entier)
- distance (entier)
- insecurite (entier)
- nb_lieux_utile (entier)
- nb_lieux_total (entier)
- nb_arc (entier)

6.2.8 La gestion des résultats

L'entrer "resultat" de type "resultat" est disponible dans la structure "Donnee" elle contient, dans le tableau "résultats", les chemins non dominés et dont toutes les possibilités au niveau de la permutation des arcs ont été testées.
Afin de disposer de résultat ordonnés et pour faciliter l'algorithme de suppression des chemins dominés et identiques, le tableau "résultats" de type "Parcours" est à trois dimensions. Une première dimension permet de classer les chemins en fonction du nombre de lieux total, la deuxième dimension est un pointeur sur le résultat.
Comme cette structure est entièrement dynamique, la structure "résultats" dispose d'un entier nommé

"nb_lieu" qui indique le nombre de lieux constituant un chemin qui est géré par la table "resultat" mais aussi un tableau nommé "nb_resultat" qui indique le nombre de chemins alloués et le nombre de chemins utilisés en fonction du nombre de lieux.

Structure resultat :

- nb_lieux (entier)
- nb_resultats (entier)
- resultats (pointeur Parcours)

7. Tris et suppression des arcs dominés

7.1 But de la suppression et du tri

Dans les arcs renseignés par l'utilisateur rien n'indique qu'ils soient non dominés entre eux. Un arc dominé est, selon le critère de recherche, forcement mauvais. Par exemple, l'arc d'intérêt 6, de distance 5 et d'insécurité 3 et dominé par un arc don les valeurs sont respectivement 7,5,3. Dans ce cas, bien qu'il y ai une distance et une insécurité identique, le deuxième arc l'emporte, car il possède un intérêt plus fort.

Il est donc nécessaire de supprimer, avant le début de la recherche, tous ces arcs qui sont ignorés dans la solution finale, mais qui feraient perdre du temps.

La relation de dominance entre les arcs ne peut s'établir que s'ils ont la même destination et la même source. Comme le tableau "map" qui détient l'ensemble des arcs est dans un premier temps rempli dans l'ordre où le fichier de donnée a été écrit, rien ne garantis que les arcs homologues se suivent.

Afin de simplifier l'algorithme de suppression des arcs dominés, il est donc nécessaire de trier la table "map" afin de regrouper les destinations entre elles. La structure de ce tableau ayant déjà effectué un regroupant des arcs en fonction du lieu de départ.

De même, comme on le verra dans la partie XX, la création du chemin de référence nécessite de connaitre l'arc de distance la plus courte entre deux points donnés. Le fait de trier les arcs en fonction de leur distance puis de leur insécurité en cas d'égalité, permet d'avoir en sommet de liste l'arc de distance minimal, ce qui évite de faire une recherche a chaque ajout d'un arc au chemin de base.
Il est donc intéressant de trier cette table d'arcs, et ce en fonction de plusieurs critères :

- les arcs possédant une destination identique doivent être regroupés. Comme on cherche à maximiser l'intérêt du Parcours, les intérêts les plus forts sont placés au début. Si deux lieux on un intérêt identique, la destination possédant l'identifiant le plus faible est placé avant.
- Dans chaque groupe, les arcs sont triés a fin de minimiser la distance puis l'insécurité des premiers arcs.

7.2 Algorithme de tris multi-critère

L'algorithme de tris utilisé est celui du "Quicksort", bien que récursif, il permet d'avoir une complexité en $\Theta(n \log(n))$ dans le cas moyen. C'est à dire quand le pivot n'est pas sur une des extrémités du tableau à trier.
Comme les données sont rentrées manuellement, rien ne permet de près-dire où le pivot à des chances de ce trouver. On aurait pu choisir un pivot aléatoirement, mais nous avons décidé de le prendre au milieu de la table à trier.

Pour ce tri il y a plusieurs éléments à prendre en compte afin de décider si un élément doit se trouver avant ou après le pivot, les testes sont effectués par une fonction externe à celle de tris.

Algorithme de tri :
Il s'agit une implémentation standard du ńăQuicksortăż, deux boucles positionnent les éléments dans un tableau qui est ensuite partagé en deux pour subir le même algorithme.

Algorithme 1 Quicksort_map

Précondition:

Entrée :

- data pointeur sur la structure Donner
- id_Lieu entier, identifiant du lieu a trier
- m entier borne droit du tri
- n entier borne gauche du tri

Toutes les données sont valides

Postcondition:

Sortie : $\emptyset$

Postcondition : data → map[id_lieu] est trier par :

- Intérêt décroissant
- Distance croissant
- Insécurité croissant

```
1:  Arc**map ← data->map[id_lieu];
2:  int i, j, k;
3:  si (m < n) alors
4:          /*determination et sauvgarde du pivot*/
5:          k ← (m+n)/2
6:          swap (map[m],map[k])
7:          /*placement des marqueurs*/
8:          i ← m + 1
9:          j ← n
10:         /*recherche des elements a permuter*/
11:         tantque i ≤ j faire
12:                 /*element a gauche*/
13:                 tantque ((i ≤ n) & position(data, id_lieu, i, m) faire
14:                         i ← i + 1
15:                 fin tantque
16:                 /*element a droit*/
17:                 tantque ((j ≤ n) & position(data, id_lieu, j, m) faire
18:                         j ← j + 1
19:                 fin tantque
20:                 si (i < j) alors
21:                         /*permutation*/
22:                         swap (map[i],map[j])
23:                 fin si
24:         fin tantque
25:         /* remise en place du pivot*/
26:         swap(map[m],map[j])
27:         /* appel récursif sur les deux demi-éléments droit et gauche*/
28:         quicksort_map(data,id_lieu,m,j-1)
29:         quicksort_map(data,id_lieu,j+1,n)
30: fin si
```

Algorithme de la fonction position :
Son rôle est d'indiquer si un arc se trouve avant ou après un autre. Le second arc, celui de référence, est nommé clef. Cette fonction prend en compte l'intérêt, la distance, la destination et l'insécurité. Si l'arc doit se trouver avant celui pointé par la clef, la fonction renvoie 0. Ce cas est vrai si :

1. L'intérêt de l'arc est supérieur à celui de la clef.
2. En cas d'égalité de l'intérêt, et de différence entre les identifiants des destinations, il faut que le numéro de destination de l'arc soit inférieur à celui de la clef.
3. En cas d'égalité de l'intérêt, il faut que la distance de l'arc soit inférieure à celui de la clef.
4. En cas d'égalité de l'intérêt et de la distance, il faut que l'insécurité de l'arc soit inférieure ou égale à celui de la clef.

Pour s'affranchir de changement éventuel dans la structure de donnée, de nombreuses fonctions ont été écrites afin de récupérer des valeurs précises. Cela présente également l'avantage de clarifier les algorithmes.

Algorithme 2 Position

Précondition:

Entrée :
- data pointeur sur la structure donnée
- id_lieu entier, identifiant du lieu a trier
- id_arc entier, identifiant de l'arc
- id_key entier, identifiant de la clef

Toutes les données doivent être valides

Postcondition:

Sortie : posi est un booléen
- posi = 0 : id_arc avant id_key
- posi = 1 : id_arc après id_key

```
1:  /*Initialisation : récuperation des valeurs*/
2:  key_interet ← interet_map_destination (data, id_lieu, id_key) ;
3:  key_distance ← distance_map_arc (data, id_lieu, id_key) ;
4:  key_insecurite ← insecurite_map_arc (data, id_lieu, id_key) ;
5:  key_destination ← destination_map_arc (data, id_lieu, id_key) ;
6:  arc_interet ← interet_map_destination (data, id_lieu, id_arc) ;
7:  arc_distance ← distance_map_arc (data, id_lieu, id_arc) ;
8:  arc_insecurite ← insecurite_map_arc (data, id_lieu, id_arc) ;
9:  arc_destination ← destination_map_arc (data, id_lieu, id_arc) ;
10: /*comparaison*/
11: si (arc_interet > key_interet) alors
12:         Retourner (posi = 0) ;
13: fin si
14: si (arc_interet = key_interet) alors
15:         si (arc_destination < key_destination) alors
16:                 Retourner (posi = 0) ;
17:         fin si
18:         si (arc_distance < key_distance) alors
19:                 Retourner (posi = 0) ;
20:         fin si
21:         si (arc_distance = key_distance) alors
22:                 si (arc_insecurite ≤ key_insecurite) alors
23:                         Retourner (posi = 0) ;
24:                 fin si
25:         fin si
26: fin si
27: Retourner(posi = 1)
```

7.3 Algorithme de suppression des arcs dominés

Le rôle de cet algorithme de comparer tous les arcs entre eux afin de supprimer les dominés et les identiques. Un arc A est dominé pour un arc B si :

- Sa distance est supérieure ou égale à celle de B
- Son insécurité est supérieure ou égale à celle de B

Cette Algorithme possède deux curseurs de lecture. Un premier qui sert de référence, et un deuxième qui sert de test. Le teste se déplace d'arc en arc jusqu'à ce que le lieu de destination entre teste et référence diffère. Lorsque c'est le cas, la référence ce décale d'un arc et le teste devient égale à la position de référence plus un. Ainsi, on ne compare que des arcs homologuesă ; ils sont tous testés et la fonction s'arrête quand la référence a atteint la fin du tableau.

Comme la suppression d'arc engendre des trous, nous avons du mètre en place un mécanisme qui décale le curseur de teste ou de référence s'ils viennent à designer un arc inexistant. Cette méthode est accompagnée d'une technique de copie qui déplace les arcs testés dès qu'un trou apparait. Pour ce fait il existe nu curseur nommé ă"id_cpy" qui indique la position vide.

Algorithme d'épuration :

```
int epure_map(Donnee *data,int id_lieu){
    int nbre_arc = nb_arc(data, id_lieu);
    int id_arc, id_key = 0, id_cpy = 1;

    int arc_distance, arc_insecurite, arc_destination;

    /*récupération des valeurs de la clef*/
    int key_distance = distance_map_arc(data, id_lieu, id_key);
    int key_insecurite = insecurite_map_arc(data, id_lieu, id_key);
    int key_destination = destination_map_arc(data, id_lieu, id_key);

    /*lecture de tous le tableau*/
    for(id_arc = 1 ; id_arc < nbre_arc ; ++id_arc){
        /*si l'arc n'existe pas, on prends le suivant*/
        while((id_arc != nbre_arc)&&(existe_map_arc(data, id_lieu, id_arc) == 0)) id_arc++;

        /*récupération des valeurs de l'arc*/
        arc_distance = distance_map_arc(data, id_lieu, id_arc);
        arc_insecurite = insecurite_map_arc(data, id_lieu, id_arc);
        arc_destination = destination_map_arc(data, id_lieu, id_arc);

        /*si l'arc est dominé, on le supprime*/
        /*à destination égale, les intérêt sont égaux*/
        if((arc_destination == key_destination)&&(arc_distance >= key_distance)&&(arc_insecuri
            spr_str_map_arc(data, id_lieu, id_arc);
            maj_str_map_arc(data, id_lieu, id_arc, NULL);
        }
        else{
```

```
            /*si les lieux de destination sont différant*/
            if(arc_destination != key_destination){
                /*on déplace la clef d'un*/
                ++id_key;
                /*si l'arc n'existe pas, on prends le suivant*/
                while((id_key != nbre_arc)&&(str_map_arc(data, id_lieu, id_key) == NULL)) ++id

                /*récupération des valeurs de la clef*/
                key_distance = distance_map_arc(data, id_lieu, id_key);
                key_insecurite = insecurite_map_arc(data, id_lieu, id_key);
                key_destination = destination_map_arc(data, id_lieu, id_key);
            }

            /* s'il y a des arcs de supprimes, on comble les trous*/
            if(id_arc != id_cpy){
                if(str_map_arc(data, id_lieu, id_cpy) != NULL) spr_str_map_arc(data, id_lieu,
                maj_str_map_arc(data, id_lieu, id_cpy, str_map_arc(data, id_lieu, id_arc));
                maj_str_map_arc(data, id_lieu, id_arc, NULL);
            }
            ++id_cpy;
        }
    }
    return id_cpy;
}
```

7.4 Utilisation des solutions

La table solution est utilisée pour stocker le chemin de référence et ceux qui sont en attente pour générer l ?ensemble des solutions qu ?ils renfermement. Comme ces résultats partiel son issues du chemin de base, Il est nécessaire de définir des méthodes pour pouvoir créer, copier, modifier et supprimer une solution afin que les algorithmes de recherche puissent opérer.

Attention, comme pour toutes les fonctions définit pour ce projet, il n ?y a aucuns teste pour savoir si l ?opération est permise ou non. C ?est à l ?utilisateur de savoir ce qu ?il fait et ce qu ?il désir.

7.4.1 Les méthodes de créationăet de suppression de solutions

La fonction all_solutions, est conçue pour créer et initialiser un certain nombre de nouvelles solutions, voir, si la table n ?existe pas, c ?est elle qui la créer. Ces nouveaux espaces sont obligatoirement ajoutés à la fin de la table. La structure de donnée et le nombre de solutions que l ?on désire créer sont passés en paramètre.

Après son utilisation on dispose du nombre de solutions désiré. Le nombre de solutions allouées est défini dans la structure ńăsolutionsż, cette fonction maintien cette valeur à jour. Toutes caractéristiques sont initialisées à zéro et les tables itinéraire, trajet et visite à NIL.

Algorithme de all_solutions :

```
void all_solutions(Donnee *data, int nb_ajout){
    int nb_solution_totale = nb_solution(data);
    int nb_reallocation = nb_solution_totale + nb_ajout;
    int i;
    Parcours **temp_doublep;

    /*on redimentionne la table du nombre specifie*/
    temp_doublep = (Parcours **)realloc(data->solution.solution, nb_reallocation*sizeof(Parcou
    if(temp_doublep == NULL) fatalerreur(data, "all_solutions : echeque de l'allocation");
    data->solution.solution = temp_doublep;

    /*definition des nouvelles tables*/
    for(i = data->solution.nb_solution; i < nb_reallocation; ++i){
        /*creation de la solution*/
        data->solution.solution[i] = (Parcours *)malloc(sizeof(Parcours));
        if(data->solution.solution[i] == NULL) fatalerreur(data, "all_solutions : echeque de l

        /*initialisation des caracteristique*/
        data->solution.solution[i]->carac.distance = 0;
        data->solution.solution[i]->carac.insecurite = 0;
        data->solution.solution[i]->carac.interet = 0;
        data->solution.solution[i]->carac.nb_lieux_total = 0;
        data->solution.solution[i]->carac.nb_lieux_utile = 0;
        data->solution.solution[i]->carac.nb_arc = 0;

        data->solution.solution[i]->itineraire = NULL;
        data->solution.solution[i]->trajet = NULL;
        data->solution.solution[i]->visite = NULL;
    }

    /* mise a jour du nombre de solution*/
    data->solution.nb_solution = nb_reallocation;
}
```

Pour supprimer des solutions, il existe plusieurs méthodes :

- unall_nb_solutions : elle est équivalente à all_solutionsă ; en plus de la supprimer, elle réadapte la taille du tableau "solutions". Les solutions qu ?elles suppriment sont donc nécessairement à la fin de cette table. Elle maintint à jour le nombre de solutions présent, mais ne désalloue pas totalement la table "solutions" même si elle devient vide.
- unall_table_solution : vide entièrement la table est la supprime de la mémoire.
- unall_solution : supprime une solution au milieu de la table, la case du tableau "solutions" sensé contenir l ?adresse de la solution supprimée obtiens la valeur NIL.

Pour désallouer correctement la mémoire, il est nécessaire de préalablement supprimer les tableaux "trajectoire", "itinéraire" et "visite" qui sont contenus dans la solution. Cependant, visite n ?existe pas forcement, c ?est pour quoi il est nécessaire de tester sont existence. Si cette table n ?existe pas, alors le pointeur "visite" présent dans la structure "solutions" est à NIL.

Algorithme de unall_solutions :

```
void unall_solution(Donnee *data, int id_solution){
```

```
    free(data->solution.solution[id_solution]->itineraire);
    free(data->solution.solution[id_solution]->trajet);

    if(data->solution.solution[id_solution]->visite != NULL){

        free(data->solution.solution[id_solution]->visite);
        data->solution.solution[id_solution]->visite = NULL;
    }

    free(data->solution.solution[id_solution]);

    data->solution.solution[id_solution] = NULL;
}
```

Algorithme unall_nb_solutions :

```
void unall_nb_solutions(Donnee *data, int nb_suppretion){
    int nb_solution_totale = nb_solution(data);
    int nb_solution_restant = nb_solution_totale - nb_suppretion;
    int i;
    Parcours **temp_doublep;

    for(i = nb_solution_restant; i < nb_solution_totale; ++i){
        unall_solution(data, i);
    }

    temp_doublep = (Parcours **)realloc(data->solution.solution, nb_solution_restant*sizeof(Pa
    if(temp_doublep == NULL) fatalerreur(data, "unall_table_solutions : echeque de la realloca
    data->solution.solution = temp_doublep;

    data->solution.nb_solution = nb_solution_restant;
}
```

Algorithme unall_table_solution :

```
void unall_table_solutions(Donnee *data){
    int i;

    for(i = 0; i < data->solution.nb_solution; ++i){
        unall_solution(data, i);
    }
    free(data->solution.solution);
}
```

7.4.2 La méthode pour copier une solution

Comme une solution est la modification du chemin de référence, il est plus commode de copier ce dernier avant toutes modifications. L ?existence de cette fonction de copie présente l ?avantage de

préserver le chemin de base et de ne pas avoir à la régénérer pour pouvoir le modifier.

La fonction cpy_solution, recopie un à un tous les éléments de la source vers la destination. Les caractéristiques sont dupliquées et les tables "trajet" et "itineraire" sont créés et affecté avec les valeurs correspondantes, seules la table "visite" est ignorée, car la probabilité pour qu ?elle soit utile sur ce chemin est presque nulle. Si cela est nécessaire, la fonction générant le chemin de base, seule fonction utilisatrice de cette table, est capable de la recréer. Faire cette omission présente l ?avantage de limiter l ?utilisation superflue de la mémoire, de minimiser les instructions de copie pour un risque quasi inexistant.

Algorithme cpy_solution :

```
void cpy_solution(Donnee *data, int id_solution_destination, int id_solution_source){
    int i, nb_lieux_total;
    Arc **temp_arc;
    Lieu **temp_lieu;
    Parcours *destination_solution = data->solution.solution[id_solution_destination];
    Parcours *source_solution = data->solution.solution[id_solution_source];

    /*copie des caracteristiques*/
    destination_solution->carac.distance = source_solution->carac.distance;
    destination_solution->carac.insecurite = source_solution->carac.insecurite;
    destination_solution->carac.interet = source_solution->carac.interet;
    destination_solution->carac.nb_arc = source_solution->carac.nb_arc;
    destination_solution->carac.nb_lieux_total = source_solution->carac.nb_lieux_total;
    destination_solution->carac.nb_lieux_utile = source_solution->carac.nb_lieux_utile;

    /*copie de l'itineraire*/
    nb_lieux_total = destination_solution->carac.nb_lieux_total;

    /*on augmente l'itineraire de nb_lieux_total*/
    temp_lieu = (Lieu **)realloc(destination_solution->itineraire, (nb_lieux_total)*sizeof(Lie
    if(temp_lieu == NULL) fatalerreur(data, "cpy_solution : echeque de la realocation de itine
    destination_solution->itineraire = temp_lieu;

    for(i = 0; i < nb_lieux_total; ++i){
        destination_solution->itineraire[i] = source_solution->itineraire[i];
    }

    /*copy du trajet*/
    /*on augmante le trajet de nb_lieux_total*/
    temp_arc = (Arc **)realloc(destination_solution->trajet, (nb_lieux_total -1)*sizeof(Arc *)
    if(temp_arc == NULL) fatalerreur(data, "cpy_solution : echeque de la realocation de trajet
    destination_solution->trajet = temp_arc;

    /*on affecte l'adresse de l'arc*/
    for(i = 0; i < nb_lieux_total -1; ++i){
        destination_solution->trajet[i] = source_solution->trajet[i];
    }

    /*on ne copy pas la table des visites car elle n'est utilisé que pour la generation de che
```

```
}
```

7.4.3 Les méthodes pour modifier une solution

La modification d ?une solution ne comprend pas uniquement la simple possibilité de modifier des valeurs contenues dans les caractéristiques ou un lieu ou un arc. Il faut aussi pouvoir rajouter des éléments à la solution ciblée. C ?est pour cela que de nombreuses de fonctions modification ont été crééesă :

- add_arc_solution : ajout un arc à la fin de la solution
- add_lieu_solution : ajout un lieu à la fin de la solution
- maj_lieu_total_soltuion : affecte un nouveau nombre de lieux total
- maj_nb_lieu_utile_solution : affecte un nouveau nombre de lieux utiles
- maj_distance_solution : change la distance de la solution
- maj_insecurite_solution : change l ?insécurité de la solution
- maj_interet_solution : change l ?intérêt de la solution
- maj_arc_solution : change l ?arc désigné dans la table trajet
- maj_lieu_solution : change le lieu désigné dans la table trajet

Seuls les algorithmes de add_arc_solution et add_lieu_solution sont présentés, les autres fonctions sont de simples modifications de valeurs dans une structure et ne présente que peu d ?intérêt.

```
void add_lieu_solution(Donnee *data, int id_solution, int id_lieu){
    int nb_lieu = data->solution.solution[id_solution]->carac.nb_lieux_total++;
    Lieu **temp;

    /*on augmente l'itineraire de 1*/
    temp = (Lieu **)realloc(data->solution.solution[id_solution]->itineraire, (nb_lieu +1)*siz
    if(temp == NULL) fatalerreur(data, "add_li; maj_lieu_totaleu_solution : echeque de la real
    data->solution.solution[id_solution]->itineraire = temp;

    /*on affecte l'adresse du lieu*/
    data->solution.solution[id_solution]->itineraire[nb_lieu] = str_lieu(data, id_lieu);
}

void add_arc_solution(Donnee *data, int id_solution, int id_lieu_depart, int id_lieu_arrive, i
    int nb_arc = data->solution.solution[id_solution]->carac.nb_arc++;
    Arc **temp;
    int id_arc;
    Arc *arc;

    /*recupere la position du premier arc disponible a la quelle on ajout l'offset*/
    id_arc = index_id_arc(data, id_lieu_depart, id_lieu_arrive);
    arc = str_map_arc(data, id_lieu_depart, id_arc) + offset;

    /*on augmante le trajet de 1*/
    temp = (Arc **)realloc(data->solution.solution[id_solution]->trajet, (nb_arc +1)*sizeof(Ar
    if(temp == NULL) fatalerreur(data, "add_lieu_solution : echeque de la realocation");
    data->solution.solution[id_solution]->trajet = temp;

    /*on affecte l'adresse de l'arc*/
```

```
    data->solution.solution[id_solution]->trajet[nb_arc] = arc;
}
```

Afin que l ?utilisateur soit le moins dépendant possible de la forme des structures des données, nous avons créé des fonctions intermédiaires capables de récupérer la bonne valeur, le bon pointeur. En contrepartie, aucuns testes n ?est effectué pour déterminer si l ?action est permise ou non. Parmi ces fonctions intermédiaires, se trouve :

- Distance_arc_solution : renvoie la distance d ?un arc en fonction de son identifiant dans la table "trajet" et de celui de la solution,
- Distance_solution : renvoie la distance totale de la solution
- Id_depart_trajet_solution : renvoie l ?identifiant du lieu de départ de l ?arc désigné
- Id_destination_trajet_solution : renvoie l ?identifiant du lieu de destination de l ?arc désigné
- Id_laste_lieu_solution : renvoie l ?identifiant du dernier lieu du chemin, utile pour connaitre le point d ?arrivée
- Id_lieu_solution : renvoie l ?identifiant du lieu désigné
- Insecurite_arc_solution : renvoie l ?insécurité de l ?arc désigné
- distance_arc_solution : renvoie la distance de l ?arc désigné
- interet_lieu_solution : renvoie l ?intérêt du lieu
- intérêt_solution : renvoie l ?intérêt total de la solution
- nb_arc_solution : renvoie le nombre d ?arcs dans la solution
- nb_lieu_solution : renvoie le nombre de lieu utile
- nb_lieu_totale_solution : renvoie le nombre de lieux total
- nb_solution : renvoie le nombre de solutions existantes dans la table

Cette liste non exhaustive présente les fonctions les plus utilisées, comme pour une grande partie d ?entre elles, elles ne sont constituées que d ?une seule ligne, nous ne présenterons pas leurs algorithmes.

Grace à ces fonctions nous disposons présent d ?outils pour créer modifier et supprimer une solution. Cela nous serra très utile pour la réalisation du chemin de référence et des optimisations.

7.5 Passage de la table des solutions à celle des résultats

7.5.1 Utilisation de l ?espace mémoire

Avant de commencer à transférer des données d ?un tableau à un autre, il est nécessaire de réserver de l ?espace mémoire pour les accueillir. C ?est pour cela que la fonction all_resultats a été créée. Tous comme son homologue all_solutions, sont rôle est d ?agrandir le tableau qui stockera les résultats. Elle possède cependant quelques nuances :

- contrairement à la structure "solutions" qui renferme la variable "nb_solution" indiquant le nombre d ?éléments du tableau, ici la structure "resultats" contient un tableau a deux dimensions nommée "nb_resultats"ă(voir la partie consacrée à la structure "résultats" pour plus de détaille).
- Elle est capable de désallouer les résultats lorsqu ?on lui fournit un nombre d ?allocations négatifs. Mais elle ne supprime pas la table pour autant.
- Elle met à jour à la fois le nombre de résultats alloués, mais aussi le nombre de résultats utilisés si nécessaire.

Bien que lors de la copie des solutions, la table "visite" n ?est pas traitée, il se peut qu ?elle soit présent suit a l ?utilisation de la fonction couper/déplacée présenté plus loin dans ce chapitre. Il est

donc toujours nécessaire de vérifier l ?existence de "visite" lors de la désallocation. Cela est vrai pour all_resultats et unall_resultat.

Algorithme all_resultats une fois implementé en C :

```
void all_resultats(Donnee *data, int nb_lieux, int nb_ajout){

  int **tmp_doubptr_int;
  int i, nb_l;
  Parcours ***tmp_tripptr_Parcours;
  Parcours **tmp_doubptr_Parcours;

/*ajout de parcours*/

  if(data->resultat.nb_lieux < nb_lieux){
  /*reallocation de la table des Parcourss enxistant en fonction des leux*/
    tmp_doubptr_int = (int**)realloc(data->resultat.nb_resultats, nb_lieux*sizeof(int*));
    if(tmp_doubptr_int == NULL)
      fatalerreur(data, "all_resultats : echeque de la reallocation de la table des resultats
    data->resultat.nb_resultats = tmp_doubptr_int;

    for(i = data->resultat.nb_lieux; i < nb_lieux; ++i){
      data->resultat.nb_resultats[i] = (int*)malloc(2*sizeof(int));
      if(data->resultat.nb_resultats[i] == NULL)
        fatalerreur(data, "all_resultats : echeque de la reallocation de la table des resultat
      data->resultat.nb_resultats[i][0] = 0;
      data->resultat.nb_resultats[i][1] = 0;
        }

     /*realocation des parcours*/
      tmp_tripptr_Parcours = (Parcours ***)realloc(data->resultat.resultats, nb_lieux*sizeof(P
      if(tmp_tripptr_Parcours == NULL)
       fatalerreur(data, "all_resultats : echeque de la reallocation des Parcourss lv1");
      data->resultat.resultats = tmp_tripptr_Parcours;

      for( i = data->resultat.nb_lieux; i < nb_lieux; ++i)
            data->resultat.resultats[i] = NULL;

        /* mise à jour du maximum de lieux utilise*/
        data->resultat.nb_lieux = nb_lieux;
    }


    if(nb_ajout > 0){

        /*allocation des nouveaux resultat*/
        nb_l = data->resultat.nb_resultats[nb_lieux -1][1] + nb_ajout;
        tmp_doubptr_Parcours = (Parcours **)realloc(data->resultat.resultats[nb_lieux -1], (nb
        if(tmp_doubptr_Parcours == NULL) fatalerreur(data, "all_resultats : echec de la reallo
```

```
        data->resultat.resultats[nb_lieux -1] = tmp_doubptr_Parcours;

        /*initialisation de la partie rajoute*/
        for(i = data->resultat.nb_resultats[nb_lieux -1][1]; i < data->resultat.nb_resultats[n
            /*creation des desultat*/
            data->resultat.resultats[nb_lieux -1][i] = (Parcours *)malloc(sizeof(Parcours));
            if(data->resultat.resultats[nb_lieux -1][i] == NULL) fatalerreur(data, "all_result

            /*initialisation des caracteristique*/
            data->resultat.resultats[nb_lieux -1][i]->carac.distance = 0;
            data->resultat.resultats[nb_lieux -1][i]->carac.insecurite = 0;
            data->resultat.resultats[nb_lieux -1][i]->carac.interet = 0;
            data->resultat.resultats[nb_lieux -1][i]->carac.nb_lieux_total = 0;
            data->resultat.resultats[nb_lieux -1][i]->carac.nb_lieux_utile = 0;
            data->resultat.resultats[nb_lieux -1][i]->carac.nb_arc = 0;

            data->resultat.resultats[nb_lieux -1][i]->itineraire  = NULL;
            data->resultat.resultats[nb_lieux -1][i]->trajet = NULL;
            data->resultat.resultats[nb_lieux -1][i]->visite = NULL;
        }
    }
    else{
        for(i = data->resultat.nb_resultats[nb_lieux -1][1] + nb_ajout; i < data->resultat.nb_
            free(data->resultat.resultats[nb_lieux -1][i]->trajet);
            free(data->resultat.resultats[nb_lieux -1][i]->itineraire);

            if(data->resultat.resultats[nb_lieux -1][i]->visite != NULL)
                free(data->resultat.resultats[nb_lieux -1][i]->visite);

            free(data->resultat.resultats[nb_lieux -1][i]);
        }

        /*desallocation des resultat*/
        nb_l = data->resultat.nb_resultats[nb_lieux -1][1] + nb_ajout;
        tmp_doubptr_Parcours = (Parcours **)realloc(data->resultat.resultats[nb_lieux -1], (nb
        if(tmp_doubptr_Parcours == NULL) fatalerreur(data, "all_resultats : echec de la reallo
        data->resultat.resultats[nb_lieux -1] = tmp_doubptr_Parcours;
    }

    /*mise a jour du nombre de résultats en fonction du nombre de lieux*/
    if(data->resultat.nb_resultats[nb_lieux -1][1] + nb_ajout < data->resultat.nb_resultats[nb
        data->resultat.nb_resultats[nb_lieux -1][0] = data->resultat.nb_resultats[nb_lieux -1]

    data->resultat.nb_resultats[nb_lieux -1][1] += nb_ajout;
}
```

Il existe aussi une fonction nommée unall_resultats qui est chargé d ?effacer totalement le tableau "resultats" de la mémoire

Alogorithme Unall_resultat :

Pratiquement identique à unall_solutions, il doit cependant travailler sur un tableau à trois dimensions, ce qui explique la présence d ?une boucle supplémentaire. Voici l'implémentation en C

```
void unall_resultats(Donnee *data){
    int i, j;

    for(i = 0; i < data->resultat.nb_lieux ; ++i){
        if(data->resultat.resultats[i] != NULL){
            for(j = 0; j < data->resultat.nb_resultats[i][1]; ++j){
                if(data->resultat.resultats[i][j] != NULL){
                    free(data->resultat.resultats[i][j]->itineraire);
                    free(data->resultat.resultats[i][j]->trajet);

       if(data->resultat.resultats[i][j]->visite != NULL)
free(data->resultat.resultats[i][j]->visite);
                }
                free(data->resultat.resultats[i][j]);
            }

            free(data->resultat.resultats[i]);
        }
    }
    free(data->resultat.resultats);

    for(i = 0; i < data->resultat.nb_lieux; ++i){
        free(data->resultat.nb_resultats[i]);
    }
    free(data->resultat.nb_resultats);
}
```

7.5.2 Copie de la solution vers les résultats

Le chemin résultant de l ?optimisation est un résultat en soit, est il doit être placé dans le tableau ńărésultatsăż, pour cela nous avons créé deux fonctions :

- cpy_solution_to_resultat : qui fonctionne de la même manière que cpy_solutions.
- cut_solution_to_resultat : copie et retire l ?adresse mémoire de la solution pour la placer dans les résultats. Elle présente l ?avantage d ?avoir une complexité nettement plus faible que cpy_solution_to_resultatăce qui lui garantit un temps d ?exécution plus faible. Mais comme elle retire le chemin des solutions, elle équivaut aussi à un dépilage. Ces deux raisons majeures nous permettent de n ?utiliser que cut_solution_to_resultatăpour obtenir un phénomène de pille sur le tableau "solutions".

C ?est deux fonctions place les éléments à copier à la fin du tableau "resultats" désigné parăle nombre de lui que possède le chemin. C ?est pour cela qu ?il n ?est pas nécessaire de préciser la position de destination. Par contre, l ?utilisateur de la fonction peut avoir besoin de savoir à quelle position le chemin a été copier, c ?est pour quoi, c ?est fonction renvoie cette information.

Algorithmeăcpy_solution_to_resultat :

```
int cpy_solution_to_resultat(Donnee *data, int nb_lieux, int id_solution){
```

```
int i, nb_lieux_total;
Arc **temp_arc;
Lieu **temp_lieu;
Parcours *destination_resultat;
Parcours *source_solution;
int id_resultat;

/*si la table de resultat n'existe pas encore, on la cree*/
if(data->resultat.nb_lieux < nb_lieux)
    all_resultats(data, nb_lieux, 1);

id_resultat = nb_resultats_use_by_lieu(data, nb_lieux);

if(id_resultat >= nb_resultats_all_by_lieu(data, nb_lieux))
    all_resultats(data, nb_lieux, 1);

destination_resultat = data->resultat.resultats[nb_lieux -1][id_resultat];
source_solution = data->solution.solution[id_solution];

/*copy des caracteristique*/
destination_resultat->carac.distance = source_solution->carac.distance;
destination_resultat->carac.insecurite = source_solution->carac.insecurite;
destination_resultat->carac.interet = source_solution->carac.interet;
destination_resultat->carac.nb_arc = source_solution->carac.nb_arc;
destination_resultat->carac.nb_lieux_total = source_solution->carac.nb_lieux_total;
destination_resultat->carac.nb_lieux_utile = source_solution->carac.nb_lieux_utile;

/*copy de l'itineraire*/
nb_lieux_total = destination_resultat->carac.nb_lieux_total;

/*on augmente l'itineraire de nb_lieux_total*/
temp_lieu = (Lieu **)realloc(destination_resultat->itineraire, (nb_lieux_total)*sizeof(Lie
if(temp_lieu == NULL) fatalerreur(data, "cpy_solution_to_resultat : echeque de la realocat
destination_resultat->itineraire = temp_lieu;

for(i = 0; i < nb_lieux_total; ++i){
    destination_resultat->itineraire[i] = source_solution->itineraire[i];
}

/*copy du trajet*/
/*on augmante le trajet de nb_lieux_total*/
temp_arc = (Arc **)realloc(destination_resultat->trajet, (nb_lieux_total -1)*sizeof(Arc *)
if(temp_arc == NULL) fatalerreur(data, "cpy_solution_to_resultat : echeque de la realocati
destination_resultat->trajet = temp_arc;

/*on affecte l'adresse de l'arc*/
for(i = 0; i < nb_lieux_total -1; ++i){
    destination_resultat->trajet[i] = source_solution->trajet[i];
}
```

```
    data->resultat.nb_resultats[nb_lieux -1][0]++; //augmante de 1 le nombre de resultat utili
    /*on ne copy pas la table des visites car elle n'est utilisé que pour la generation de che
    return id_resultat;// renvoi la position de la copie
}
```

Algorithme cut_solution_to_resultat : Contrairement à cpy_solution_to_resultat, cut_solution_to_resultat doit gère le vide occasionné par la suppression d ?un élément du tableau. Il se charge aussi de la désallocation et de la suppression définitive de cette table quand il n ?y a plus de solution dedans.

```
int cut_solution_to_resultat(Donnee *data, int nb_lieux, int id_solution){
    int id_resultat;
    Parcours **tmp;

    /*récupère le nombre de résultat utilisé. comme ils sont contenues dans un tableau (debut
    id_resultat = nb_resultats_use_by_lieu(data, nb_lieux);

    /*libere cette espace, en effet contrairement a copie, cette espace existe deja dans le ta
    free(data->resultat.resultats[nb_lieux -1][id_resultat]);

    /*cpoie de l'adresse*/
    data->resultat.resultats[nb_lieux -1][id_resultat] = data->solution.solution[id_solution];

    /*increùmantation du nombre de resultats utilisé*/
    data->resultat.nb_resultats[nb_lieux -1][0]++;

    /*comble le vide engendre dans le tableau des solutions*/
    while(id_solution < data->solution.nb_solution -1){
        data->solution.solution[id_solution] = data->solution.solution[id_solution +1];
        id_solution++;
    }

    /*desaloue la derniere solution du tableaux, si le tableau est vide, il est suprimé*/
    if(data->solution.nb_solution == 1){
        data->solution.nb_solution = 0;

         free(data->solution.solution);
    }
    else{
        data->solution.nb_solution -= 1;
        //unall_solution(data, data->solution.nb_solution);

        tmp = (Parcours **)realloc(data->solution.solution, (data->solution.nb_solution)*sizeo
        if(tmp == NULL) fatalerreur(data, "cut_solution_to_resultat : echeque de la reallocati
        data->solution.solution = tmp;
    }

    /*renvoie la position de destination*/
    return id_resultat;
}
```

7.5.3 Enumeration des résultants

La fonction genere_resultats est prévue pour créer tous les résultants possibles en permutant les arcs du chemin qui vient juste d ?être copiée. En effet, elle prend comme référence le dernier résultat de la table. Comme les algorithmes d ?optimisation ne se basent que sur des villes, les arcs restent ceux du chemin de référence. Et comme il est construit en utilisant les arcs de plus petites distances on connaît déjà leur position dans la table ă"map", il s ?agit des arcs au début du tableau désigné par "index_map".
Cette information est intéressante, car comme nous faisons un parcourt en largeur des arcs, cela évite d ?avoir à gérer l ?arc numéroté zéro dans le tableau.
Cependant, tous les chemins ne sont pas intéressants, si un résultat dominé est trouvé, il n ?est pas enregistré. De même, si le trajet servent de graine devient dominé, il est remplacé. En précédant ainsi, le tableau des résultats contient des chemins de lus en plus acceptable. Mais cela n ?empêche pas le tableau résultats de contenir des trajets non désirables, car un chemin non dominé à une énumération donnée peut le devenir à la suivante. Pourtant il à bien été enregistré. C ?est pour cela que nous avons créé la fonction epure_resultats, qui est comparable a epure_map.
epure_resultats à pour but de supprimer tous les chemins dominés ou identiques. Il travaille sur l ?ensemble de la table "resultats" qui est désigné par son nombre de lieux. Ainsi, s ?il y a eu plusieurs générations, on est sûrs que les potentiels doublons ont disparu tous comme les chemins dominés.

Implémentation en C de l'algorithme de genere_resultats :

```
void genere_resultats(Donnee *data, int nb_lieux){
    int id_resultat = data->resultat.nb_resultats[nb_lieux -1][0] -1;
    int nb_lieux_resultat = data->resultat.resultats[nb_lieux -1][id_resultat]->carac.nb_arc;
    int lieu, arc;
    Parcourt* table_resultat = data->resultat.resultats[nb_lieux -1][id_resultat];
    int id_destination, id_depart;
    int nb_arc_genere;
    int nb_resutats_vide;
    int id_write_resultat = id_resultat + 1;
    int distance_new, insecurite_new;
    int distance_ref, insecurite_ref;

    /*pour tous les lieux du chemin*/
    for(lieu = 0; lieu < nb_lieux_resultat; ++lieu){
        /* on recupere leur valeur*/
        id_depart = table_resultat->itineraire[lieu]->id;
        id_destination = table_resultat->itineraire[lieu + 1]->id;
        nb_arc_genere = index_nb_arc(data, id_depart, id_destination); //nombre d'arc entre le
        nb_resutats_vide =  data->resultat.nb_resultats[nb_lieux -1][1] - data->resultat.nb_re

        if(nb_resutats_vide < nb_arc_genere){ //on alloue si le nombre de resultat disponible
            all_resultats(data, nb_lieux, data->resultat.nb_resultats[nb_lieux -1][1] + nb_arc
        }

        distance_ref = distance_totale_resultat(data, nb_lieux, id_resultat);
        insecurite_ref = insecurite_totale_resultat(data, nb_lieux, id_resultat);

        /*pour tous les arcs disponible entre le lieu de depart et le lieu d'arrive*/
```

```
        for(arc = 1; arc < nb_arc_genere; ++arc){
            distance_new = distance_ref - distance_arc_resultat(data, nb_lieux, id_resultat, l
            insecurite_new =  insecurite_ref - insecurite_arc_resultat(data, nb_lieux, id_resu

            if((distance_ref < distance_new)&&(insecurite_ref < insecurite_new)){
                continue; //new est dominé
            }
            if((distance_ref > distance_new)&&(insecurite_ref > insecurite_new)){
                /*le chemin de referance, est dominé, on modifi sont arc pour ne pas garder un
                change_arc_resultat(data, nb_lieux, id_resultat, lieu, id_depart, id_destinati

                distance_ref = distance_totale_resultat(data, nb_lieux, id_resultat);
                insecurite_ref = insecurite_totale_resultat(data, nb_lieux, id_resultat);
            }
            else{
                cpy_resultat(data, nb_lieux, id_write_resultat, id_resultat); /* c'est un nouv
                change_arc_resultat(data, nb_lieux, id_write_resultat, lieu, id_depart, id_des
                id_write_resultat++;
            }
        }


    }

    /*on vide la table de tous les resultat non utilisé*/
    all_resultats(data, nb_lieux, data->resultat.nb_resultats[nb_lieux -1][0] - data->resultat

}
```

Implémentation en C de l'algortime de epure_resultats Le fonctionnement de cet algorithme est semblable à epure_map, il semble plus simple car la fonction suprime_resultat comble immédiatement les trous créer dans le tableau par la suppression. Cependant, il reste des erreurs de désallocation mémoire qui ne sont pas géré.

```
void epure_resultats(Donnee *data, int nb_lieux){
    int id_referance, id_test;
    int interet_referance, distance_referance, insecurite_referance;
    int interet_test, distance_test, insecurite_test;

    /* pour tous les resultats du tableau, meme si cete valeir change*/
    for(id_referance = 0; id_referance < nb_resultats_use_by_lieu(data, nb_lieux); ++id_refera

        /*valeurs de la referance*/
        interet_referance = data->resultat.resultats[nb_lieux -1][id_referance]->carac.interet
        distance_referance = data->resultat.resultats[nb_lieux -1][id_referance]->carac.distan
        insecurite_referance = data->resultat.resultats[nb_lieux -1][id_referance]->carac.inse

        /*pour de refance +1 a la fin du tableau*/
```

```
        for(id_test = id_referance +1; id_test < nb_resultats_use_by_lieu(data, nb_lieux); ++i
            /*valeur du teste*/
            interet_test = data->resultat.resultats[nb_lieux -1][id_test]->carac.interet;
            distance_test = data->resultat.resultats[nb_lieux -1][id_test]->carac.distance;
            insecurite_test = data->resultat.resultats[nb_lieux -1][id_test]->carac.insecurite

            /*si les caracteristique sont identique*/
            if((interet_referance == interet_test)&&(distance_referance == distance_test)&&(in

                if(compar_resultats(data, nb_lieux, id_referance, id_test) == 0){// teste si l
                    //suprimer test
                    suprime_resultat(data, nb_lieux, id_test);
                }
            }
            else {
                /*si il est dominé*/
                if((interet_referance >= interet_test)&&(distance_referance <= distance_test)&
                    suprime_resultat(data, nb_lieux, id_test);
                }
                else {
                    /*si la referance est dominé*/
                    if((interet_referance <= interet_test)&&(distance_referance >= distance_te
                        replace_resultat(data, nb_lieux, id_referance, id_test);
                        suprime_resultat(data, nb_lieux, id_test);

                        /*mise a jour des valeurs de la referance*/
                        interet_referance = interet_test;
                        distance_referance = distance_test;
                        insecurite_referance = insecurite_test;
                    }
                }
            }
        }
    }
}
```

Voici l'implémentation de l'algorithme suprime_resultat

```
static void suprime_resultat(Donnee *data, int nb_lieux, int id_resultat){
    int i;
    //Parcourt **tmp_doubptr_parcourt;

    free(data->resultat.resultats[nb_lieux -1][id_resultat]->itineraire);
    free(data->resultat.resultats[nb_lieux -1][id_resultat]->trajet);
    if(data->resultat.resultats[nb_lieux -1][id_resultat]->visite != NULL){
        free(data->resultat.resultats[nb_lieux -1][id_resultat]->visite);
    }
```

```
    for(i = id_resultat; i < nb_resultats_use_by_lieu(data, nb_lieux) -1; ++i){
        data->resultat.resultats[nb_lieux -1][i] = data->resultat.resultats[nb_lieux -1][i +1]
    }

    data->resultat.resultats[nb_lieux -1][i] = NULL;

    /*free(data->resultat.resultats[nb_lieux -1][i]->itineraire);
    free(data->resultat.resultats[nb_lieux -1][i]->trajet);
    if(data->resultat.resultats[nb_lieux -1][i]->visite != NULL){
        free(data->resultat.resultats[nb_lieux -1][i]->visite);
    }

    tmp_doubptr_parcourt = (Parcourt **)realloc(data->resultat.resultats[nb_lieux -1], i*sizeo
    if(tmp_doubptr_parcourt == NULL) fatalerreur(data, "suprime_resultat :  echeque de la real
    data->resultat.resultats[nb_lieux -1] = tmp_doubptr_parcourt;

    data->resultat.nb_resultats[nb_lieux -1][1] = i;
    data->resultat.nb_resultats[nb_lieux -1][0]--;
}
```

8. Création de chemin

8.1 Création d'un chemin de base

8.2 Création de nouveau chemin

Pour créer de nouveau chemin à partir du chemin de base, il y a deux méthodes qui sont la permutation et l'insertion.

8.2.1 Permutation

La permutation consiste à interchanger des lieux dans la solutions, pour cela il faut procéder de la manière suivant :

1. Récuperer le chemin de base dans un tableau vide pour pouvoir travailler dessus
2. Parcourt du tableau pour tester l'existence d'arcs entre les différents lieux
3. Si il existe un arc entre deux lieux on peu envisager de les permuter à condition qu'il existe un arc entre le lieux précédent le 1er lieu concerner par la permutation ainsi que le lieu successeur de deuxième lieu concerner par la permutation.
4. Si les deux conditions sont remplis on réalise la permutation.

Algorithme 3 Permutation

Précondition:
Postcondition:

```
1: Cpy_Solution(data,Id_Solution_base,Id_Solution_New) ;
2: pour i du 1er lieu au dernier lieu faire
3:         si Permutation_Possible(i + 2, i + 1) = 1 alors
4:                 tmp ← lieu[i + 2];
5:                 lieu[i + 2] ← lieu[i + 1];
6:                 lieu[i + 1] ← tmp;
7:                 Sortie de la boucle
8:         sinon
9:                 Retourner("Il n'y a pas de permutation possible")
10:        fin si
11: fin pour
12: Retourner(Le nouveau chemin)
```

9. Conclusion

Annexes

A. Fiche de suivi de projet

17/01/2011	1ére rencontre avec Emmanuel Néron pour prendre une explication approfondi du sujet ainsi que le premier objectif à réaliser qui est le choix d'une structure de donnée pour gérer les villes.

20/02/2011 au 04/03/2011	Réflexion sur les méthodes possible pour la structure de donnée à mettre en place, ainsi que les algorithmes à utiliser pour parcourir les différentes structures de données

04/03/2011 au 29/03/2011	Réalisation de la structure de données pour gérer la configuration de la ville, mise en place d'algorithme de tri pour les arcs.

07/04/2011	Finalisation de la structure de données, vérification des fonctions permettant d'interroger la structure de données pour les algorithmes permettant de créé les trajets, première réalisation d'un algorithme pour générer le trajet de référence.

11/04/2011	Réunion avec notre encadrant pour lui présenter la structure de données et l'esquisse de l'algorithme pour générer le trajets de référence.

11/04/2011 au 05/05/2011	Réalisation et implémentation de l'algorithme pour générer un chemin de base et résolution d'un problème par rapport au tri des arc entre les lieux.

05/05/2011 au 01/06/2011	

Tournée d'un véhicule multicritéres

Département Informatique
3e année
2010 - 2011

Rapport Projet Algorithme-C

Résumé : Description en français

Mots clefs : Mots clés français

Abstract: Description en anglais

Keywords: Mots clés en anglais

Encadrant
Emmanuel Neron
emmanuel.neron@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Cyrille PICARD
cyrille.picard@etu.univ-tours.fr
Michael PURET
michael.puret@etu.univ-tours.fr

DI3 2010 - 2011