

# Rapport de Projet de Recherche Opérationnelle

---

Modèle stratégique d'allocation de sillons  
ferroviaires  
Groupe ilpr511

Moez BEN MBARKA    Francois CARRIE    Florian JALOUX  
Julie QUAGLIOZZI    Mortada ZNIBER

1 juin 2005

# Table des matières

<b>1</b>	<b>Recherche Opérationnelle</b>	<b>4</b>
1.1	Présentation . . . . .	4
1.1.1	Les données d'entrée . . . . .	4
1.1.2	Les contraintes à respecter . . . . .	4
1.1.3	La fonction économique et le problème à résoudre . . . . .	4
1.2	Exemple . . . . .	5
1.2.1	Présentation de l'exemple . . . . .	5
1.2.2	Equations liées aux contraintes . . . . .	5
1.2.3	Scénarios : . . . . .	10
<b>2</b>	<b>Programmation</b>	<b>19</b>
2.1	Types de données . . . . .	19
2.1.1	Le type <i>gare</i> . . . . .	19
2.1.2	Le type <i>tronçon</i> . . . . .	20
2.1.3	Le type <i>train</i> . . . . .	22
2.1.4	Les fonctions accédant à la structure de la gare . . . . .	24
2.1.5	Les fonctions accédant à la structure du tronçon . . . . .	27
2.1.6	Les fonctions accédant à la structure du train . . . . .	28
2.2	Utilisation : . . . . .	29

# Introduction

Lors de ce projet de recherche opérationnelle, nous travaillons sur un réseau ferroviaire. Le but de ce projet est de trouver les vitesses et les heures de départ de chaque circulation, afin que la somme de toutes les pénalités (ralentissement, désheutage et suppression) soient le minimum possible.

Pour chaque tronçon du réseau, nous devons respecter certaines contraintes. De ces contraintes, vont ressortir certaines équations. Le logiciel `lp_solve` nous permet ensuite de connaître les vitesses et les heures de départ optimales.

En ce qui concerne la partie programmation, nous devons réaliser un programme en langage C. Ce programme doit générer des équations liées aux contraintes du réseau. Ces contraintes générées seront ensuite utilisées avec `lp_solve` afin d'optimiser le réseau ferroviaire étudié.

Ce rapport est divisé en 2 grandes parties :

- une première partie qui contient la présentation du problème et un exemple
- une seconde partie qui contient les explications du programme réalisé

# Chapitre 1

## Recherche Opérationnelle

### 1.1 Présentation

#### 1.1.1 Les données d'entrée

Pour cette partie, nous disposons d'un tronçon rectiligne (voie unique), sans possibilité de croisement des trains. La longueur du tronçon est  $LT$ , la vitesse maximale autorisée sur le tronçon est  $V_{maxT}$  et l'intervalle minimum entre les trains est égal à  $i$ .

Pour chaque train, nous disposons de certaines données :

- le caractère imposé ou variable (I/V) de l'heure de départ et d'arrivée du train
- le caractère obligatoire ou supprimable (O/S) du train
- la vitesse maximale autorisée du train ( $V_{maxC}$ )
- l'heure idéale de départ ou l'heure idéale d'arrivée ( $H_{id}/H_{ia}$ )
- la pénalité liée à un ralentissement du train par rapport à sa vitesse maximale per minute de ralentissement ( $PénRalC$ )
- la pénalité liée au désheutage du départ du train par minute de désheutage ( $PénDésC$ )
- la pénalité liée à la suppression éventuelle du train ( $PénSuppC$ ) pour des trains supprimables.

#### 1.1.2 Les contraintes à respecter

Chaque train doit respecter des contraintes :

- la vitesse du train (homogène sur tout le tronçon)  $VC$  doit être inférieure à  $V_{maxT}$  et à  $V_{maxC}$
- l'intervalle minimum entre les trains doit être égal à  $i$
- impossibilité de dépassement d'un train par un autre
- respect des blancs des trains sur le tronçon
- les trains à horaires imposés circulent obligatoirement aux heures spécifiées
- les trains non supprimables ne peuvent être supprimés.

#### 1.1.3 La fonction économique et le problème à résoudre

La fonction économique de ce problème est la somme de toutes les pénalités pour tous les tronçons et tous les trains.

Le problème à résoudre consiste à trouver les vitesses et les heures de départ de chaque train afin de minimiser :

$PenRalC + PenDesC + PenSuppC$  appliqué au réseau entier (pour tous les trains et les tronçons).

## 1.2 Exemple

### 1.2.1 Présentation de l'exemple

Nous avons utilisé un petit exemple afin de nous familiariser avec `lp_solve`. Cet exemple comporte 4 gares, 3 tronçons et 3 trains.

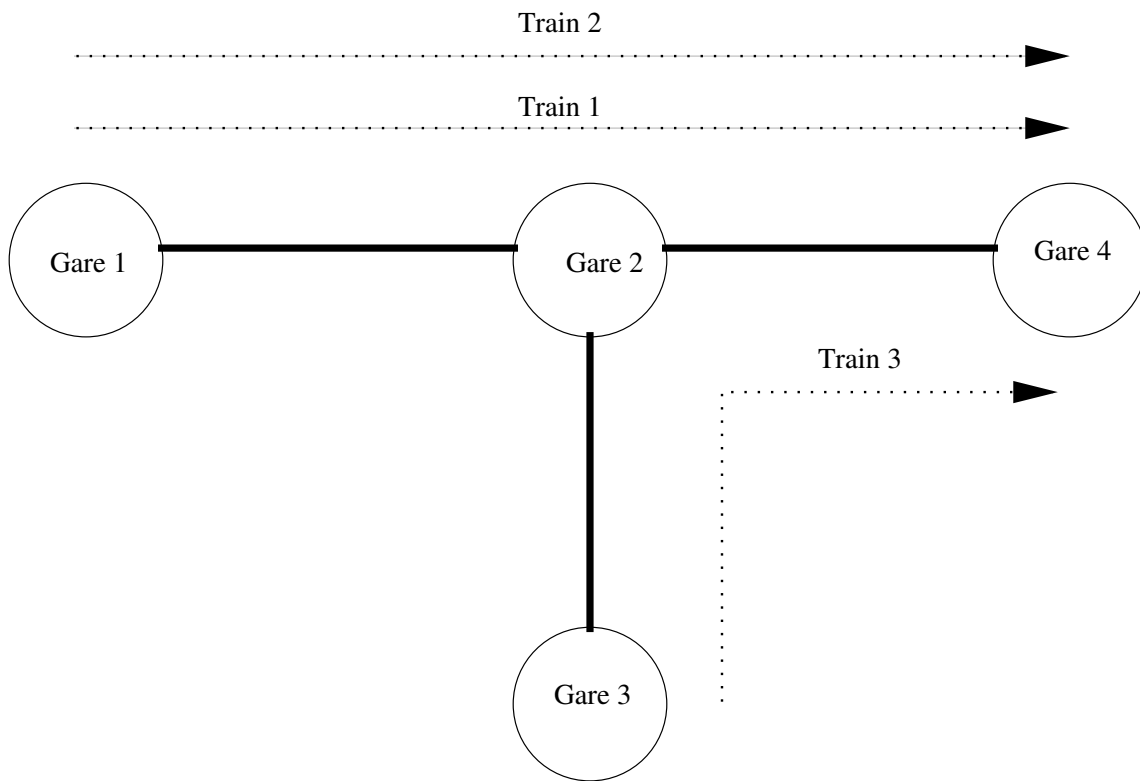


FIG. 1.1 – Le reseau ferroviaire utilisé pour notre exemple

Le temps de sécurité entre 2 trains est  $ts = 10min = 0.17$ .

Les longueurs des tronçons sont :

- tronçon(1,2) 100km
- tronçon(2,3) 60km
- tronçon(2,4) 120km

### 1.2.2 Equations liées aux contraintes

Il est important de noter que toutes les variables suivantes sont entières et valent 0 ou 1 :  $yd1\_2\_0, yd1\_2\_1, ya1\_2\_1, yb1\_2\_1, yc1\_2\_1, ya1\_2\_3, yd1\_3\_1, ya1\_3\_1, yb1\_3\_1, yc1\_3\_1, ya1\_3\_3, ded1\_2, ded1\_3$

#### La fonction objective

La fonction objective à minimiser est :

$$150 * tra10\_1 + 120 * tdes0\_1 + 100 * sup0\_1 + 150 * tra10\_2 + 120 * tdes0\_2 + 100 * sup0\_2 + 150 *$$

$$tral0_3 + 120 * tdes0_3 + 100 * sup0_3$$

Elle consiste à multiplier les temps de desheutage, de ralentissement par leurs coûts respectifs et de les additioner si il y a une suppression avec le coût de suppression. Pour cet exemple et pour le train 1 :

- $tral0_1$  correspond au temps de ralentissement et son coût de ralentissement est de 150
- $tdes0_1$  correspond au temps de desheutage et son coût de desheutage est de 120
- $sup0_1$  est un booléen indiquant si ce train a été supprimé ou pas ;

Les 3 équations qui suivent permettent de rendre le temps de désheutage et de ralentissement si un train est supprimé :

- $150 * tral0_1 + 200 * tdes0_1 < 1 - 1 * sup0_1$
- $10 * tral0_2 + 10 * tdes0_2 < 1 - 1 * sup0_2$
- $150 * tral0_3 + 120 * tdes0_3 < 100 - 100 * sup0_3$

Pour chaque train  $i$ ,  $sup0_i$  est un binaire (0 ou 1). Il vaut 1 si le train est supprimé et 0 sinon. Si le train est supprimé, alors le membre de droite dans les 3 équations est nul ; or le membre de gauche ne peut pas être négatif, donc  $tral0_i$  et  $tdes0_i$  sont nuls. On utilise cette méthode afin de ne pas prendre en compte un temps de désheutage ou de ralentissement dans la fonction objective si un train est supprimé. A noter que dans le membre de droites, on utilise le cout de la suppression (exemple : dans  $100 - 100 * sup0_3$ , 100 représente le cout de la suppression du train 3. En effet, le choix de supprimer un train n'est pris que si le cout correspondant est inférieur aux couts de ralentissements et désheutage. En d'autres termes, si la variable de suppression est nulle (pas de suppression) on immédiatement :  $coutRal + coutDes < outSupp$ .

### Les temps de sécurité entre les trains

Les inéquations ci-dessous permettent tout d'abord d'exprimer la valeur absolue  $|Hd1_1 - Hd2_1| > 0.17$ . Elles imposent un temps de sécurité entre tous les trains : tous les trains seront séparés d'une distance de sécurité.

- $Hd1_1 - Hd2_1 = upd1_2_0 - umd1_2_0$
- $upd1_2_0 + umd1_2_0 > 0.17$

On oblige aux 2 variables intermédiaires  $umd1_2_0$  et  $upd1_2_0$  de ne pas s'annuler si une des deux est différente de zéro par les inéquations :

- $upd1_2_0 < 1000 * yd1_2_0$
- $umd1_2_0 < 1000 - 1000 * yd1_2_0$
- $yd1_2_0 < 1$

La variable  $yd1_2_0$  se comporte comme un booléen. Si elle vaut 1 alors  $umd1_2_0 < 0$  donc  $umd1_2_0 = 0$  et  $upd1_2_0 < 1000$ . Le phénomène inverse se produit si  $yd1_2_0 = 0$ .

Maintenant, il faut prendre en compte la suppression du train. Si ce dernier est supprimé, les contraintes imposées doivent être annulées. Pour cela on ajoute  $1000 * sup0_1 + 1000 * sup0_2$  à l'inéquation  $upd1_2_0 + umd1_2_0 > 0.17$  où  $sup0_1$  et  $sup0_2$  sont des booléens indiquant respectivement par 1 si les trains 1 et 2 sont supprimés. Ainsi, si l'une des deux variables valent 1 alors l'inéquation  $upd1_2_0 + umd1_2_0 + 1000 > 0.17$  n'a plus de sens et n'impose alors plus rien.

- $Hd1_1 - Hd2_1 = upd1_2_0 - umd1_2_0$
- $1000 * sup0_1 + 1000 * sup0_2 + upd1_2_0 + umd1_2_0 > 0.17$

- $upd1\_2\_0 < 1000 * yd1\_2\_0$
- $umd1\_2\_0 < 1000 - 1000 * yd1\_2\_0$
- $yd1\_2\_0 < 1$

Temps de sécurité concernant le train 1 et 2 dans la gare 2 (pour l'arrivée et le départ) :

- $Hd1\_2 - Hd2\_2 = upd1\_2\_1 - umd1\_2\_1$
- $1000 * sup0\_1 + 1000 * sup0\_2 + upd1\_2\_0 + umd1\_2\_0 > 0.17$
- $upd1\_2\_1 < 1000 * yd1\_2\_1$
- $umd1\_2\_1 < 1000 - 1000 * yd1\_2\_1$

- $Ha1\_2 - Ha2\_2 = upa1\_2\_1 - uma1\_2\_1$
- $1000 * sup0\_1 + 1000 * sup0\_2 + upa1\_2\_1 + uma1\_2\_1 > 0.17$
- $upa1\_2\_1 < 1000 * ya1\_2\_1$
- $uma1\_2\_1 < 1000 - 1000 * ya1\_2\_1$

- $Hd1\_2 - Ha2\_2 = upb1\_2\_1 - umb1\_2\_1$
- $1000 * sup0\_1 + 1000 * sup0\_2 + upb1\_2\_1 + umb1\_2\_1 > 0.17$
- $upb1\_2\_1 < 1000 * yb1\_2\_1$
- $umb1\_2\_1 < 1000 - 1000 * yb1\_2\_1$

- $Ha1\_2 - Hd2\_2 = upc1\_2\_1 - umc1\_2\_1$
- $1000 * sup0\_1 + 1000 * sup0\_2 + upc1\_2\_1 + umc1\_2\_1 > 0.17$
- $upc1\_2\_1 < 1000 * yc1\_2\_1$
- $umc1\_2\_1 < 1000 - 1000 * yc1\_2\_1$

Temps de sécurité entre les trains 1 et 2 dans la gare 4 (pour l'arrivée) :

- $Ha1\_4 - Ha2\_4 = upa1\_2\_3 - uma1\_2\_3$
- $1000 * sup0\_1 + 1000 * sup0\_2 + upa1\_2\_3 + uma1\_2\_3 > 0.17$
- $upa1\_2\_3 < 1000 * ya1\_2\_3$
- $uma1\_2\_3 < 1000 - 1000 * ya1\_2\_3$

Temps de sécurité entre les trains 1 et 3 dans la gare 2 (pour l'arrivée et le départ) :

- $Hd1\_2 - Hd3\_2 = upd1\_3\_1 - umd1\_3\_1$
- $1000 * sup0\_1 + 1000 * sup0\_3 + upd1\_3\_1 + umd1\_3\_1 > 0.17$
- $upd1\_3\_1 < 1000 * yd1\_3\_1$
- $umd1\_3\_1 < 1000 - 1000 * yd1\_3\_1$

- $Ha1\_2 - Ha3\_2 = upa1\_3\_1 - uma1\_3\_1$
- $1000 * sup0\_1 + 1000 * sup0\_3 + upa1\_3\_1 + uma1\_3\_1 > 0.17$
- $upa1\_3\_1 < 1000 * ya1\_3\_1$
- $uma1\_3\_1 < 1000 - 1000 * ya1\_3\_1$

- $Hd1\_2 - Ha3\_2 = upb1\_3\_1 - umb1\_3\_1$
- $1000 * sup0\_1 + 1000 * sup0\_3 + upb1\_3\_1 + umb1\_3\_1 > 0.17$
- $upb1\_3\_1 < 1000 * yb1\_3\_1$
- $umb1\_3\_1 < 1000 - 1000 * yb1\_3\_1$

- $Ha1\_2 - Hd3\_2 = upc1\_3\_1 - umc1\_3\_1$
- $1000 * sup0\_1 + 1000 * sup0\_3 + upc1\_3\_1 + umc1\_3\_1 > 0.17$
- $upc1\_3\_1 < 1000 * yc1\_3\_1$
- $umc1\_3\_1 < 1000 - 1000 * yc1\_3\_1$

Temps de sécurité entre les trains 1 et 3 dans la gare 4 (pour l'arrivée) :

- $Ha1_4 - Ha3_4 = upa1_3_3 - uma1_3_3$
- $1000 * sup0_1 + 1000 * sup0_3 + upa1_3_3 + uma1_3_3 > 0.17$
- $upa1_3_3 < 1000 * ya1_3_3$
- $uma1_3_3 < 1000 - 1000 * ya1_3_3$

### Equations sur les horaires

On obtient les heures de départ et d'arrivée de chaque train en ajoutant aux heures idéales les temps de ralentissement et de désheutage.

Les équations sont :

- $Ha1_2 = tral1_1_2 + 9.17 + tdes1_1_2$
- $Ha1_4 = tral1_2_4 + 10.54 + tdes1_2_4$
- $Ha2_2 = tral2_1_2 + 9.1733 + tdes2_1_2$
- $Ha2_4 = tral2_2_4 + 10.34 + tdes2_2_4$
- $Ha3_2 = tral3_3_2 + 9.17 + tdes3_3_2$
- $Ha3_4 = tral3_2_4 + 10.34 + tdes3_2_4$

### calcul des désheutages

Ces équations permettent de calculer les temps de desheutage en faisant la différence des heures véritables avec les heures idéales.

- $tdes1_1_2 = Hd1_1 - 8$
- $tdes1_2_4 = Hd1_2 - 9.17$
- $tdes2_1_2 = Hd2_1 - 8.17$
- $tdes2_2_4 = Hd2_2 - 9.17$
- $tdes3_3_2 = Hd3_3 - 8.5$
- $tdes3_2_4 = Hd3_2 - 9.17$

### Ralentissement et désheutage global pour chaque train

On additionne pour chaque train le temps de ralentissement et de désheutage pour tous les tronçons que ce train parcourt.

Les équations sont :

- $tdes0_1 = tdes1_1_2 + tdes1_2_4$
- $tral0_1 = tral1_1_2 + tral1_2_4$
  
- $tdes0_2 = tdes2_1_2 + tdes2_2_4$
- $tral0_2 = tral2_1_2 + tral2_2_4$
  
- $tdes0_3 = tdes3_3_2 + tdes3_2_4$
- $tral0_3 = tral3_3_2 + tral3_2_4$

### Inéquations afin d'éviter le dédoublement

Ces inéquations permettent d'éviter le dédoublement. Elles traduisent le fait que si un train 1 part avant un train 2 d'une même gare alors il arrive avant ce dernier à la gare suivante.

- $Hd1_1 - Hd2_1 < 1000 * deda1_2$
- $Ha1_2 - Ha2_2 < 1000 * deda1_2$
- $Hd2_1 - Hd1_1 < 1000 * dedb1_2$
- $Ha2_2 - Ha1_2 < 1000 * dedb1_2$

- $deda1.2 < 1000 * x1.2_0$
- $dedb1.2 < 1000 - 1000 * x1.2_0$
- $deda1.2 > 1001 - 1000 * sup0.1$
- $dedb1.2 > 1001 - 1001 * sup0.2$

Si l'heure de départ du train 1 est supérieur à l'heure de départ du train 2 alors la variable  $dedb1.2 = 0$  et l'inéquation  $Ha2.2 - Ha1.2 < 1000 * dedb1.2$  impose  $Ha2.2 < Ha1.2$  (l'heure d'arrivée du train 1 doit être supérieur à l'heure d'arrivée du train 2). D'autre part, les deux dernières inéquations  $deda1.2 > 1001 - 1000 * sup0.1$  et  $dedb1.2 > 1001 - 1001 * sup0.2$  obligent aux 2 variables intermédiaires  $deda1.2$  et  $dedb1.2$  de ne pas s'annuler si une des deux est différente (même principe qu'avec la valeur absolue). Par conséquent,  $deda1.2 = 1$  et les inéquations  $Hd1.1 - Hd2.1 < 1000 * deda1.2$  et  $Ha1.2 - Ha2.2 < 1000 * deda1.2$  deviennent insignifiantes  $Hd1.1 - Hd2.1 < 1000$   $Ha1.2 - Ha2.2 < 1000$ .

Le procédé inverse se procuit si l'heure de départ du train 1 est inférieure à l'heure de départ du train 2.

- $Hd1.1 - Hd2.1 < 1000 * deda1.2$
- $Ha1.2 - Ha2.2 < 1000 * deda1.2$
- $Hd2.1 - Hd1.1 < 1000 * dedb1.2$
- $Ha2.2 - Ha1.2 < 1000 * dedb1.2$
- $deda1.2 < 1000 * x1.2_0$
- $dedb1.2 < 1000 - 1000 * x1.2_0$
- $deda1.2 > 1001 - 1000 * sup0.1$
- $dedb1.2 > 1001 - 1001 * sup0.2$
  
- $Hd1.2 - Hd2.2 < 1000 * deda1.2$
- $Ha1.4 - Ha2.4 < 1000 * deda1.2$
- $Hd2.2 - Hd1.2 < 1000 * dedb1.2$
- $Ha2.4 - Ha1.4 < 1000 * dedb1.2$
- $deda1.2 < 1000 * x1.2_1$
- $dedb1.2 < 1000 - 1000 * x1.2_1$
- $deda1.2 > 1001 - 1000 * sup0.1$
- $dedb1.2 > 1001 - 1001 * sup0.2$
  
- $Hd1.2 - Hd3.2 < 1000 * deda1.3$
- $Ha1.4 - Ha3.4 < 1000 * deda1.3$
- $Hd3.2 - Hd1.2 < 1000 * dedb1.3$
- $Ha3.4 - Ha1.4 < 1000 * dedb1.3$
- $deda1.3 < 1000 * x1.3_1$
- $dedb1.3 < 1000 - 1000 * x1.3_1$
- $deda1.3 > 1001 - 1000 * sup0.1$
- $dedb1.3 > 1001 - 1001 * sup0.3$

### Equations pour générer le temps d'attente

Ces inéquations permettent d'imposer un temps d'arrêt (10 min (0,17)) dans la gare intermédiaire traversée (ici gare 2). La différence entre l'heure de départ et l'heure d'arrivée d'un train dans une même gare doit être supérieur à 0.17.

Les équations sont :

- $1000 * sup0.1 + Hd1.2 - Ha1.2 > 0.17$
- $1000 * sup0.2 + Hd2.2 - Ha2.2 > 0.17$
- $1000 * sup0.3 + Hd3.2 - Ha3.2 > 0.17$

### 1.2.3 Scénarios :

En utilisant le schémas des gares défini précédemment on va essayer en modifiant les horaires et les couts d'illustrer plusieurs scénarios.

Par rapport à différents scénarios, nous allons soit désheurer, soit ralentir ou soit supprimer un ou plusieurs trains.

Dans tous ce qui suit on considérer les caractéristiques suivantes comme constantes :

- Les vitesses maximales des trains : train1, train2 et train3 sont successivement 100Km/h, 120Km/h et 120Km/h.
- Aucun de ces trains n'est non supprimable.

Cependant les données suivantes sont susceptibles d'être modifiées pour créer les différents scénarios :

- Les cout de suppressions pour les trois trains sont succesivement : 20, 30 et 100.
- Les pénalités liées au ralentissement sont successivement : 15, 10 et 15.
- Les pénalités liées au ralentissement sont successivement : 20, 10 et 12.

#### Scénario 1 : Horaires parfaits

Les horaires des trains sont définie dans les tableaux suivant :

	gare1	gare2	gare4
train 1	8	9 - 9.17	10,36

	gare1	gare2	gare4
train2	9	9.83 - 10	11

	gare3	gare2	gare4
train3	11.5	12 - 12.17	13.17

On obtient les valeurs numériques suivantes :

```
-----
Valeur total des couts: 0.000000
-----
```

```
-----
TRAIN n?: 1
-----
```

```
***** Trajet entre la gare 1 et 2: *****
```

```
Heure idéale de départ: 8.00
Heure idéale d'arrivée: 9.00
```

```
Heure effective de départ: 8.00
Heure effective d'arrivée: 9.00
```

```
Desheurage: 0.00
Temps de ralentissement: 0.00
```

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 9.17  
Heure idéale d'arrivée: 10.36

Heure effective de départ: 9.17  
Heure effective d'arrivée: 10.40

Desheurage: 0.00  
Temps de ralentissement: 0.00

-----  
TRAIN n?: 2  
-----

\*\*\*\*\* Trajet entre la gare 1 et 2: \*\*\*\*\*

Heure idéale de départ: 9.00  
Heure idéale d'arrivée: 9.83

Heure effective de départ: 9.00  
Heure effective d'arrivée: 9.83

Desheurage: 0.00  
Temps de ralentissement: 0.00

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 10.00  
Heure idéale d'arrivée: 11.00

Heure effective de départ: 10.00  
Heure effective d'arrivée: 11.00

Desheurage: 0.00  
Temps de ralentissement: 0.00

-----  
TRAIN n?: 3  
-----

\*\*\*\*\* Trajet entre la gare 3 et 2: \*\*\*\*\*

Heure idéale de départ: 11.50  
Heure idéale d'arrivée: 12.00

Heure effective de départ: 11.50  
Heure effective d'arrivée: 12.00

Desheurage: 0.00  
Temps de ralentissement: 0.00

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 12.70  
 Heure idéale d'arrivée: 13.17

Heure effective de départ: 12.70  
 Heure effective d'arrivée: 13.70

Desheutage: 0.00  
 Temps de ralentissement: 0.00

### Interprétation :

Les horaires étant bien séparés pour tous les trains et pour tous les tronçons, Tous les trains ont pu circuler selon leurs horaires idéaux.

### Scénario 2 : Imposer un désheutage

Pour cela on va prendre les horaires suivants :

	gare1	gare2	gare4
train 1	8	9 - 9.17	10,36

	gare1	gare2	gare4
train2	10	10.83 - 11	12

	gare3	gare2	gare4
train3	8.5	9 - 9.17	10.17

Le problème imposé par ces valeurs, est le conflit entre les deux trains : train1 et train3 : Il arrivent idéalement à la gare 2 au même moment.

On obtient les valeurs numériques suivantes :

Valeur total des couts: 8.52

-----  
 TRAIN n?: 1  
 -----

\*\*\*\*\* Trajet entre la gare 1 et 2: \*\*\*\*\*

Heure idéale de départ: 8.00  
 Heure idéale d'arrivée: 9.00

Heure effective de départ: 8.00  
 Heure effective d'arrivée: 9.00

Desheutage: 0.00  
 Temps de ralentissement: 0.00

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 9.17

Heure idéale d'arrivée: 10.36

Heure effective de départ: 9.17  
Heure effective d'arrivée: 10.40

Desheurage: 0.00  
Temps de ralentissement: 0.00

-----  
TRAIN n?: 2  
-----

\*\*\*\*\* Trajet entre la gare 1 et 2: \*\*\*\*\*

Heure idéale de départ: 9.17  
Heure idéale d'arrivée: 10.00

Heure effective de départ: 9.17  
Heure effective d'arrivée: 10.00

Desheurage: 0.00  
Temps de ralentissement: 0.00

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 10.17  
Heure idéale d'arrivée: 11.17

Heure effective de départ: 10.20  
Heure effective d'arrivée: 11.20

Desheurage: 0.00  
Temps de ralentissement: 0.00

-----  
TRAIN n?: 3  
-----

\*\*\*\*\* Trajet entre la gare 3 et 2: \*\*\*\*\*

Heure idéale de départ: 8.50  
Heure idéale d'arrivée: 9.00

Heure effective de départ: 8.84  
Heure effective d'arrivée: 9.34

Desheurage: 0.71  
Temps de ralentissement: 0.00

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 9.17  
Heure idéale d'arrivée: 10.17

Heure effective de départ: 9.54  
 Heure effective d'arrivée: 10.57

Desheurage: 0.71  
 Temps de ralentissement: 0.00

### Interprétation :

Comme prévu un des trains a été obligé de faire du désheurage. Le train selectionné est le train3. Sa pénalité associée au désheurage est la pénalité la plus petites parmi toutes les pénalités de tous les trains. A noter que le choix de la suppression n'est pas plus économique. la seule pénalité liée a la suppression inférieure à la pénalité de désheurage du train3, est celle du train2. Or la suppression de ce train ne résout pas le conflit des horaires entre les trains 1 et 3.

### Scénario 3 : Possibilité de dédoublement

Pour apparaitre la possibilité du dédoublement sur le troncon gare1-gare2, on utilisera les horaires suivants :

	gare1	gare2	gare4
--	-------	-------	-------

train 1	8	9 - 9.17	10,36
	gare1	gare2	gare4

train2	8.17	9 - 9.17	10.17
	gare3	gare2	gare4

train3	11.5	12 - 12.17	13.17
--------	------	------------	-------

Ainsi le train2 partira idéalement de la gare1 juste après le départ idéal du train1 (en ajoutant le temps de sécurité).

On obtient les valeurs numériques suivantes :

```

-----
Valeur total des couts: 7.10
-----
-----
TRAIN n?: 1
-----
***** Trajet entre la gare 1 et 2: *****

Heure idéale de départ: 8.00
Heure idéale d'arrivée: 9.00

Heure effective de départ: 8.00
Heure effective d'arrivée: 9.00

Desheurage: 0.00
Temps de ralentissement: 0.00

```

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 9.17  
Heure idéale d'arrivée: 10.36

Heure effective de départ: 9.17  
Heure effective d'arrivée: 10.40

Desheurage: 0.00  
Temps de ralentissement: 0.00

-----  
TRAIN n?: 2  
-----

\*\*\*\*\* Trajet entre la gare 1 et 2: \*\*\*\*\*

Heure idéale de départ: 8.17  
Heure idéale d'arrivée: 9.00

Heure effective de départ: 8.17  
Heure effective d'arrivée: 9.34

Desheurage: 0.37  
Temps de ralentissement: 0.34

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 9.17  
Heure idéale d'arrivée: 10.17

Heure effective de départ: 9.54  
Heure effective d'arrivée: 10.57

Desheurage: 0.37  
Temps de ralentissement: 0.34

-----  
TRAIN n?: 3  
-----

\*\*\*\*\* Trajet entre la gare 3 et 2: \*\*\*\*\*

Heure idéale de départ: 11.50  
Heure idéale d'arrivée: 12.00

Heure effective de départ: 11.50  
Heure effective d'arrivée: 12.00

Desheurage: 0.00  
Temps de ralentissement: 0.00

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 12.70  
 Heure idéale d'arrivée: 13.17

Heure effective de départ: 12.70  
 Heure effective d'arrivée: 13.70

Desheurage: 0.00  
 Temps de ralentissement: 0.00

### Interprétation :

Le problème créé ici est le fait que le train2 part juste après le train1 de la gare1, mais avec une vitesse plus importante. Si le dédoublement était possible, le train2 arriverait à la gare2 bien avant le train1. On voit bien sur les résultats numériques que le dédoublement a été bien interdit. Le choix fait était de ralentir et désheurer le train2 pour qu'il arrive à la gare2 après le train1 tout en respectant le temps de sécurité.

Heure idéale de départ: 8.17  
 Heure idéale d'arrivée: 9.00

Heure effective de départ: 8.17  
 Heure effective d'arrivée: 9.34  
 Desheurage: 0.37  
 Temps de ralentissement: 0.34

D'autre part, les coûts sur le ralentissement et le désheurage étant égaux, la charge du retard a été partagée entre les deux pénalités. A noter aussi que, le choix de la suppression n'a pas été pris, vu que son coût est plus élevé que celui du ralentissement et du désheurage, et ceci pour tous les trains.

### Scénario 4 : Imposer la suppression d'un train

On va continuer avec les mêmes horaires précédents pour les trains, mais en modifiant les pénalités liées au train2 pour favoriser sa suppression à la place du ralentissement et le désheurage. pour cela on prend les pénalités suivantes pour le train2 :

- Pénalité du ralentissement : 10
  - Pénalité du désheurage : 10
  - Pénalité de la suppression : 5 llllllll 1.12
- On obtient les valeurs numériques suivantes :

-----  
 Valeur total des coûts: 5.00  
 -----  
 -----

TRAIN n?: 1  
 -----

\*\*\*\*\* Trajet entre la gare 1 et 2: \*\*\*\*\*

Heure idéale de départ: 8.00  
 Heure idéale d'arrivée: 9.00

Heure effective de départ: 8.00  
Heure effective d'arrivée: 9.00

Desheutage: 0.00  
Temps de ralentissement: 0.00

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 9.17  
Heure idéale d'arrivée: 10.36

Heure effective de départ: 9.17  
Heure effective d'arrivée: 10.40

Desheutage: 0.00  
Temps de ralentissement: 0.00

-----  
TRAIN n?: 2  
-----

CE TRAIN A ETE SUPPRIME !

-----  
TRAIN n?: 3  
-----

\*\*\*\*\* Trajet entre la gare 3 et 2: \*\*\*\*\*

Heure idéale de départ: 11.50  
Heure idéale d'arrivée: 12.00

Heure effective de départ: 11.50  
Heure effective d'arrivée: 12.00

Desheutage: 0.00  
Temps de ralentissement: 0.00

\*\*\*\*\* Trajet entre la gare 2 et 4: \*\*\*\*\*

Heure idéale de départ: 12.70  
Heure idéale d'arrivée: 13.17

Heure effective de départ: 12.70  
Heure effective d'arrivée: 13.70

Desheutage: 0.00  
Temps de ralentissement: 0.00

**Interprétation :**

Comme prévu le train2 a été supprimé pour avoir un cout total égal ici à 5 inférieur au cout total dans le scénario précédent(7). Il faut remarquer aussi qu'en gardant le cout de la suppression du train1 assez élevé(20), le choix de supprimer ce train est loin d'être le plus économique.

# Chapitre 2

## Programmation

### 2.1 Types de données

Il est nécessaire d'introduire de nouveaux types pour permettre le stockage des données relatives au réseau ferroviaire et aux trains y circulant

#### 2.1.1 Le type *gare*

##### Le type *gare*

Nous avons tout d'abord défini le type *gare* qui est une structure permettant de stocker les différentes données d'une gare, à savoir son nom, le nombre de trains qu'elle peut accueillir, les trains qui y sont en garage et les temps de sécurité entre deux départs et entre deux arrivées.

```
typedef struct gare_{
    char* nomGare;
    unsigned int nn;
    int* traingarage;
    float tsgrdep;
    float tsgrar;
}gare;
```

##### Le type *tabGares*

Nous avons également défini un type *tabGares* qui permet de stocker un ensemble de gares. Pour cela on a créé une structure dont les champs sont un tableau de gares et le nombre de gares contenues dans le tableau.

```
typedef struct tabGares_{
    gare* gares;
    int nbGr;
}tabGares;
```

##### Les accesseurs

Pour une meilleure utilisation de ces structures, on a créé des accesseurs permettant de retourner ou de modifier la valeur d'un champ de la structure *tabGares*, ainsi que des accesseurs permettant de modifier ou de retourner la valeur d'un champ de l'une des structures *gares* contenues dans la structure *tabGares*.

accesseurs retournant la valeur d'un champ :

```
gare* getTabGr_gares(tabGares* tabGr);
int getTabGr_nbGr(tabGares* tabGr) ;
gare* getGare(tabGares* tabGr, unsigned int indice);

char* getGr_nomGare(tabGares* tabGr, unsigned int indice);
unsigned int getGr_nn(tabGares* tabGr, unsigned int indice);
float getGr_tsgrdep(tabGares* tabGr, unsigned int indice);
float getGr_tsgrar(tabGares* tabGr, unsigned int indice);
```

accesseurs modifiant la valeur d'un champ :

```
void setTabGr_gares(tabGares* tabGr, gare* gares) ;
void setTabGr_nbGr(tabGares* tabGr, int nbGr) ;

void setGr_nomGare(tabGares* tabGr, unsigned int indice, char* nom) ;
void setGr_nn(tabGares* tabGr, unsigned int indice, unsigned int nn) ;
void setGr_tsgrdep(tabGares* tabGr, unsigned int indice, float dep) ;
void setGr_tsgrar(tabGares* tabGr, unsigned int indice, float ar) ;
```

## Les fonctions auxiliaires

Il est aussi nécessaire de créer d'autres fonctions relatives à ces structures.

La fonction *creerGares* qui permet l'allocation de la mémoire pour un nombre de structure *gare* passé en paramètre. La fonction *indiceGare* renvoie l'indice dans un tableau de gares, de la gare dont le nom est passé en paramètre. Si le nom passé en paramètre ne correspond à aucune gare, celle ci retourne -1.

La fonction *lectureGares* crée un tableau de gares à partir des données contenues dans un fichier que l'on passe en paramètre. On prend comme conventions que le fichier commence par un entier indiquant le nombre de gares, que les données pour chaque gare sont écrites dans le même ordre que les champs de la structure auxquelles elles se rapportent et qu'aucune donnée n'est omise.

### 2.1.2 Le type *tronçon*

#### Le type *troncon*

Nous avons ensuite défini une structure *troncon* qui va permettre de stocker les données relatives à chaque tronçon du réseau ferroviaire, à savoir la vitesse maximale autorisée sur le tronçon, la longueur du tronçon, le temps de sécurité à respecté entre deux trains, un tableau des intervalles de temps pendant lesquels aucun train ne doit circuler sur le tronçon (appelé blancs) et le nombre de blancs.

```
typedef struct toncon_{
    float vmaxt;
    float lt;
    float tstr;
    intervalle* blanctrancon;
    unsigned int nbBlanc;
```

#### Le type *intervalle*

Nous avons donc choisi de définir également un type *intervalle*, qui correspond à un intervalle de temps et est donc représentée par une structure composée de réel qui sont le temps de début et le temps de fin.

```
typedef struct intervalle_{
    float debut;
    float fin;
}intervalle;
```

### Le type *matTroncons*

Pour pouvoir stocker les données de chaque tronçon du réseau, créer un type *matTroncons* composée d'une matrice de *troncon* et d'un entier qui est la taille de la matrice. La matrice est une matrice triangulaire privée de sa diagonale, elle est indiquée par les indices des gares, c'est-à-dire que l'intersection de la ligne *i* et de la colonne *j* dans la matrice correspond au tronçon compris entre les gares *i* et *j*. Si il n'existe pas de tronçon entre ces deux gares, un pointeur nul est à l'intersection.

```
typedef struct matTroncons_{
    troncon** trons;
    unsigned int taille;
}matTroncons;
```

### Les accesseurs

Pour une meilleure utilisation de ces structures, nous définissons également des accesseurs permettant de modifier ou de retourner les valeurs des champs des structures. On peut distinguer ceux accédant aux champs de la structure *matTroncons*, ceux accédant aux champs de la structure *tronconet* ceux accédant aux champs de la structure *intervalle*.

accesseurs retournant la valeur d'un champ :

```
troncon** getMatTroncons_tron(matTroncons* matTron) ;
unsigned int getMatTroncons_nbTron(matTroncons* matTron) ;
```

```
float get_vmaxt(troncon* troncon);
float get_lt(troncon* troncon);
intervalle* get_blanctroncon(troncon* troncon);
float get_tstr(troncon* troncon);
unsigned int get_nbBlanc(troncon* troncon);
```

```
float get_debut(intervalle* intervalle);
float get_fin(intervalle* intervalle);
```

accesseurs modifiant la valeur d'un champ :

```
void setMatTroncons_tron(matTroncons* matTron, troncon** tron);
void setMatTroncons_nbTron(matTroncons* matTron, unsigned int nbTron);
```

```
void set_nbBlanc(troncon*, unsigned int nb) ;
void set_vmaxt(troncon* troncon, float x);
void set_lt(troncon* troncon, float x);
void set_blanctroncon(troncon* troncon, intervalle* intervalle);
void set_tstr(troncon* troncon, float x);
```

```
void set_debut(intervalle* intervalle, float x);
void set_fin(intervalle* intervalle, float x);
```

## Les fonctions auxiliaires

Il est nécessaire de créer des fonctions permettant d'allouer la mémoire pour un certain nombre de ces structures, ce sont les fonctions *creerIntervalle*, *creerTroncon* et *creerMatrice*. On crée également une fonction *libereMatrice* qui permet de libérer la mémoire allouée pour une structure *matTroncons*.

Nous avons également besoin de définir les fonctions *setTroncon* et *getTroncon* qui permettent respectivement de modifier ou de retourner un *troncon* à partir de la matrice de *troncons* et des indices des gares qu'il relie.

La fonction *lectureTroncons* gère la création et l'initialisation de ces structures à partir des données contenu dans le fichier passé en paramètre. On considère les même convention d'écriture des données que pour les gares, de plus on considère que les gares sont représenté par leur nom. Pour lire la liste des blancs, on utilise la fonction *lireListeBlanc*, en considérant que celle ci commence également par le nombre de blancs qu'elle contient.

### 2.1.3 Le type *train*

#### Le type *train*

Nous avons enfin défini le type *train* qui est une structure permettant de stocker les différentes données d'un train, à savoir son trajet, sa vitesse maximale, un booléen indiquant si le train peut être supprimé, un booléen indiquant si le train peut être désheuré, et les coûts relatifs au ralentissement, au désheurage ou à la suppression.

```
typedef struct train_{
    trajet traj;
    float vmaxc;
    int supp_on;
    int deplacable_on;
    float penral;
    float pendes;
    float pensup;
}train;
```

#### Le type *trajet*

Il a donc été nécessaire de créer un type *trajet* permettant de stocker le trajet d'un train. Pour cela on a créé une structure contenant un tableau des heures idéales de départ du train, un tableau de ses heures idéales d'arrivée, un tableau contenant les indices des gares par lesquelles il passe ainsi que les tailles de ses tableaux.

```
typedef struct trajet_{
    float* Hid;
    float* Hia;
    unsigned int* chemin;
    unsigned int tailleH; /* taille du tableau des horaires */
    unsigned int tailleCh; /* taille du tableau des chemins */
}trajet;
```

#### Le type *tabTrains*

Nous avons également défini un type *tabTrains* qui permet de stocker un ensemble de trains. Pour cela on a créé une structure dont les champs sont un tableaux de trains et le nombre de trains contenus dans le tableau.

```
typedef struct tabTrains_{
    train* trains;
    unsigned int nbTrains;
}tabTrains;
```

### Les accesseurs

Pour une meilleure utilisation de ces structures, on a créé des accesseurs permettant de retourner ou de modifier la valeur d'un champ de la structure *tabTrains*, des accesseurs permettant de modifier ou de retourner la valeur d'un champ de l'une des structures *trains* contenues dans la structure *tabTrains*, ainsi que des accesseurs permettant de retourner ou de modifier la valeur d'un champ de la structure *trajet* contenue dans le champ *traj* de l'une des structure *train* du tableau de trains.

accesseurs retournant la valeur d'un champ :

```
train* getTabTrain_trains(tabTrains* tab);
unsigned int getTabTrain_nbTrains(tabTrains* tab);

trajet* getTr_traj(tabTrains* tabTr, unsigned int indice);
float getTr_vmax(tabTrains* tabTr, unsigned int indice);
int getTr_supp(tabTrains* tabTr, unsigned int indice);
int getTr_deplacable(tabTrains* tabTr, unsigned int indice);
float getTr_penral(tabTrains* tabTr, unsigned int indice);
float getTr_pendes(tabTrains* tabTr, unsigned int indice);
float getTr_pensup(tabTrains* tabTr, unsigned int indice);

float getTr_hid(tabTrains* tabTr, unsigned int indiceTr, unsigned int indiceGr );
float getTr_hia(tabTrains* tabTr , unsigned int indiceTr, unsigned int indiceGr);
float getTr_chem(tabTrains* tabTr, unsigned int indiceTr, unsigned int indiceGr);
unsigned int getTr_tailleH(tabTrains* tabTr, unsigned int indice);
unsigned int getTr_tailleCh(tabTrains* tabTr, unsigned int indice);
```

accesseurs modifiant la valeur d'un champ :

```
void setTabTrains_trains(tabTrains* tab, train* tr);
void setTabTrains_nbTrains(tabTrains* tabTr, unsigned int nbTrains);

void setTr_traj(tabTrains* tabTr, unsigned int indice, trajet* traj);
void setTr_vmax(tabTrains* tabTr, unsigned int indice, float vmax);
void setTr_supp(tabTrains* tabTr, unsigned int indice, int bool);
void setTr_deplacable(tabTrains* tabTr, unsigned int indice, int bool);
void setTr_penral(tabTrains* tabTr, unsigned int indice, float pral);
void setTr_pendes(tabTrains* tabTr, unsigned int indice, float pdes);
void setTr_pensup(tabTrains* tabTr, unsigned int indice, float pral);

void setTr_tailleH(tabTrains* tabTr, unsigned int indice,
                  unsigned int tailleH);
void setTr_tailleCh(tabTrains* tabTr, unsigned int indice,
                  unsigned int tailleCh);
void setTr_hid(tabTrains* tabTr, unsigned int indiceTr,
              unsigned int indiceGr, float hid);
void setTr_hia(tabTrains* tabTr, unsigned int indiceTr, unsigned int indiceGr,
              float hia);
void setTr_chem(tabTrains* tabTr, unsigned int indiceTr,
```

```
unsigned int indiceGr, unsigned int che);
```

### Les fonctions auxiliaires

Comme pour les structures précédente, il est nécessaire de créer des fonctions qui gère l'allocation de la mémoire pour ces structures. La fonction *creerTr\_hid\_hia* permet d'allouer la mémoire pour les tableaux d'horaire d'un train (contenus dans le trajet), à partir d'un tableau de trains, de l'indice du train auquel correspond l'allocation et de la taille des tableaux.

La fonction *creerTr\_ch* permet d'allouer la mémoire pour le chemin que parcourt le train correspondant à l'indice passé en paramètre dans le tableau de trains passé en paramètre. Le chemin est un tableau d'indice des gares traversées par le train qui est contenu dans la structure *trajet*.

La fonction *creerTrajet* alloue la mémoire pour le trajet du train situé à l'indice passé en paramètre dans le tableau passé en paramètre. Elle inclue les deux fonctions précédentes.

La fonction *creerTrain* alloue la mémoire pour une liste de structure *train* dont le nombre est passé en paramètre. Les fonctions *liberer\_train* et *liberer\_trajet* libère la mémoire allouée par les fonctions précédentes.

De même que pour les autres types de données nous avons besoin d'une fonction permettant de créer et d'initialiser ces structures à partir d'un fichier contenant les données relatives aux trains. La fonction *lectureTrains* permet de lire les données du fichier passé en paramètre et de créer les structures en conséquence. Elle intègre la fonction *lectureTrajet* qui réalise la même chose pour des données relatives à un trajet. On considère les même convention que précédemment en ce qui concerne l'ordre des données et la position en tête de fichier du nombre de trains qu'il contient, de plus on considère que les données du trajet commence par le nombre de gares traversées, et que celles ci sont représentées par leur nom.

Enfin nous avons défini une fonction *verifieHiaHid* qui permet de vérifier que tous les horaires idéaux on bien été donnés en les calculant. Pour cela elle fait appel aux fonctions *calculeHdi* et *calculeHai*.

Dans cette partie, on présente tous les accesseurs relatifs au module gare.c.

En ce qui concerne les modules troncon.c et train.c, on ne présentera que les fonctions principales vu que le nombre d'accesseurs utilisés est très important.

### 2.1.4 Les fonctions accédant à la structure de la gare

#### ->La fonction getGare

**Entrée** : un pointeur sur un tableau de gares et un entier.

**précondition** : le pointeur ne doit pas être un pointeur NULL et l'entier doit être inférieur au nombres de gare dans le tableau.

**Sortie** : la gare se trouvant à la case numéro l'entier du tableau.

#### ->La fonction getGR\_nomGare

**Entrée** : un pointeur sur un tableau de gares et un entier.

**Précondition** : le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction renvoie le champ nom de la gare située à la case numéro l'entier en entrée du tableau.

**->La fonction getGR\_nn**

**Entrée** : un pointeur sur un tableau de gares et un entier.

**Précondition** : le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction renvoie le champ nombre de trains que la gare placée a la case numéro l'entier en entrée du tableau peut accueillir.

**->La fonction getGR\_tsgardep**

**Entrée** : un pointeur sur un tableau de gares et un entier.

**Précondition** : le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction renvoie le champ temps de sécurité entre deux départs de la gare placée dans le tableau des gares à la case numéro l'entier en entrée.

**->La fonction getGR\_tsgrar**

**Entrée** : un pointeur sur un tableau de gares et un entier.

**Précondition** : le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction renvoie le champ temps de sécurité entre deux arrivés de la gare placée dans le tableau des gares à la case numéro l'entier en entrée.

**->La fonction setGR\_nomGare**

**Entrée** : un pointeur sur un tableau de gares, un entier et un nom.

**Précondition** : le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction ne renvoie rien.

Cette fonction affecte au champ nom de la gare placée à la case numéro l'entier en entrée le nom donné en entrée.

**->La fonction setGR\_nn**

**Entrée** : un pointeur sur un tableau de gares, deux entiers.

**Précondition** : le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction ne renvoie rien.

Cette fonction affecte une valeur dans le champ nombre de train que la gare en question peut accueillir.

**->La fonction setGR\_tsrdep**

**Entrée** : un pointeur sur un tableau de gares, un entier et un réel.

**Précondition** : le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction ne renvoie rien.

Cette fonction affecte une valeur dans le champ temps de sécurité entre deux départs.

**->La fonction setGR\_tsrar**

**Entrée** : un pointeur sur un tableau de gares, un entier et un réel.

**Précondition** s : le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction ne renvoie rien.

Cette fonction affecte une valeur dans le champ temps de sécurité entre deux arrivés.

**->La fonction creerGares**

**Entrée** : un entier

**Précondition** : l'entier doit être positif.

**Sortie** : cette fonction alloue de la mémoire au nombre voulu de structures de gares.

**->La fonction getTabGr\_nbGr**

**Entrée** : un pointeur sur une structure de type tabgares

**Précondition** : Le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction retourne le champs gares.

**->La fonction getTabGr**

**Entrée** : un pointeur sur une structure de type tabgares

**Précondition** : Le pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction retourne le champs nombres de gares.

**->La fonction setTabGr\_gares**

**Entrée** : un pointeur sur une structure gare et un autre sur tabgares.

**Précondition** : le second pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction ne renvoie rien.

**->La fonction setTabGr\_nbGr**

**Entrée** : un pointeur sur une structure gare et un autre sur tabgares.

**Précondition** : le second pointeur ne doit pas être un pointeur NULL.

**Sortie** : cette fonction ne renvoie rien.

**->La fonction indiceGare**

**Entrée** : un pointeur sur une structure de type tabGare et un nom.

**Sortie** : cette fonction renvoie l'indice du nom de la gare passé en paramètre dans le tableau.

**->La fonction lectureGares**

**Entrée** : un pointeur sur une structure de type tabGare et un fichier.

**Sortie** : cette fonction lit le fichier et remplit la structure passée en paramètre. elle renvoie 1 si c'est le cas et 0 sinon.

**2.1.5 Les fonctions accédant à la structure du troncon****->La fonction setTroncon**

**Entrée** : deux entiers et deux pointeurs sur une structure troncon et matTroncon.

**Sortie** : un troncon.

**->La fonction getTroncon**

**Entrée** : un pointeur sur une structure matTroncon et deux entiers.

**Sortie** : un troncon

**->La fonction libereMatrice**

**Entrée** : un pointeur sur une structure `matTroncon` et un entier.

**Sortie** : cette fonction retourne 1 après avoir libéré le champ `troncon`.

**->La fonction lireListeBlanc**

**Entrée** : un fichier et un entier.

**Sortie** : cette fonction renvoie une structure type `intervalle`.

**->La fonction lectureTroncon**

**Entrée** : un fichier, un pointeur sur une structure `matTroncon` et un autre pointeur sur une structure `tabGares`.

**Sortie** : cette fonction remplit la matrice `troncon` à partir du fichier donné en entrée et renvoie 1 si c'est bon, 0 sinon.

**2.1.6 Les fonctions accédant à la structure du train****->La fonction lectureTrajet**

**Entrée** : un entier et trois pointeurs qui pointent sur une structure `tabTrain`, une structure `tabGares` puis un fichier.

**précondition** : le pointeur sur le fichier ne doit pas être `NULL`.

**Sortie** : cette fonction remplit les différents champs de la structure `trajet` à partir du fichier donné en entrée et renvoie 1 si c'est le cas et 0 sinon.

**->La fonction lectureTrains**

**Entrée** : un pointeur sur un fichier, un pointeur sur une structure `tabTrain` et un autre pointeur sur une structure `tabGare`

**Sortie** : cette fonction renvoie le nombre de trains dans le fichier.

**->La fonction calculeHdi**

**Entrée** : deux entiers et trois pointeurs qui pointent respectivement sur une structure `tabTrain`, `tabGares` et `matTroncon`.

**Sortie** : cette fonction renvoie l'heure idéale de départ d'un train à partir d'une gare, celle-ci est évidente si elle est donnée dans le fichier des trains sinon elle est calculée à partir de `Hai`.

**->La fonction calculeHdi**

**Entrée** : deux entiers et trois pointeurs qui pointent respectivement sur une structure `tabTrain`, `tabGares` et `matTroncon`.

**Sortie** : cette fonction renvoie l'heure idéale d'arrivée d'un train à partir d'une gare, celle-ci est évidente si elle est donnée dans le fichier des trains sinon elle est calculée à partir de `Hdi`.

**->La fonction verifieHdiHid**

**Entrée** : trois pointeurs qui pointent respectivement sur une structure `tabTrain`, `tabGares` et `matTroncon`.

**Sortie** : cette fonction retourne 1.

Cette fonction vérifie si toutes les heures idéales de départ et d'arrivée sont données, dans le cas contraire elle les calcule.

## 2.2 Utilisation :

Les sources peuvent être compilés à l'aide du script de compilation : `./compilation`.  
L'exécutable obtenu est `menu`.

Le programme crée un fichier texte exploitable par `lp_solve`. A partir des fichiers de données `gares.i1pr511`, `trains.i1pr511` et `troncons.i1pr511` il génère les équations relatives aux contraintes prises en compte et les écrit dans un fichier `Equations.i1pr511`, il exécute ensuite `lp_solve` sur ce fichier et retourne les résultats dans le fichier `Resultats.i1pr511`, en affichant en même temps les résultats de manière plus lisible sur l'écran.

Si l'on souhaite modifier les équations ou la forme dont elles sont écrites, pour par exemple utiliser un autre outil que `lp_solve`, il faut apporter les modifications sur les modules :

- `equations_lp_solve.c` qui génère les équations en déterminant leurs formats.
- `système.c` pour modifier ou ajouter des contraintes. `Equations.i1pr511`.

Les autres modules sont indépendants, ainsi que les différentes structures implémentées avec une bibliothèque d'accesseurs rendant leur utilisation assez transparente.

# Conclusion

Ce projet nous a permis de mettre en pratique les connaissances acquises pendant le cours de recherche opérationnelle. Ce cours nous a montré qu'une simple mise en équations permet de résoudre bien des problèmes, qu'il s'agisse de train dans une gare, de personne dans une file d'attente ou encore de la gestion de machine outil.

En ce qui concerne la partie programmation, on a utilisé le langage C, permettant d'automatiser la création des équations nécessaire à la résolution du problème sans pour autant penser à réduire la place nécessaire à stocker l'information.

La représentation schématique du problème nous a facilité la mise en équations de ce dernier qui une fois données à `lp_solve`, ce dernier génère automatiquement les résultats nécessaires à la minimisation de la perte occasionnée par les déplacements de train.