M1 IWOCS

Année universitaire 2020-2021

Implémentation en Java et Simulation via Anylogic, d'algorithmes de re-routage de camions pour aller chercher des conteneurs frigorifiques vides

---

Auteurs :
Noel Camara
Mamadou Saliou Dile Diallo

Encadrant :
M. Nakechbandi

Résumé

Ce document décrit l'implémentation et la simulation d'un algorithme qui a été réfléchit et conçu, dans le but de rendre plus efficace le transport dans un réseau logistique, cela, en proposant aux transporteurs, au lieu de retourner à vide, une fois qu'ils ont livré un conteneur, de prendre sur leurs chemins d'autres conteneurs (qui ont la particularité d'êtres rares et chères) pour les déposer à un autre endroit toujours sur leurs chemins, quitte à dévier un peu de leurs chemins habituels. Ainsi, ils gagneront non seulement un petit profit, mais permettront surtout de rééquilibrer ces ressources rares qui se retrouvent souvent en surnombre à un endroit, alors qu'on a besoin d'elles à un autre endroit.

Abstract

This document describes the implementation and simulation of an algorithm that has been thought out and designed, with the aim of making transport more efficient in a logistics network, by offering carriers, instead of returning empty, once that they have delivered a container, to take on their way, other containers (which have the particularity of being rare and expensive) to deposit them in another place always on their way, even if it means deviating a little from their usual way. Thus, they will gain a small profit, And above all, they will make it possible to rebalance these rare resources which are often found in excess in one place, when they are needed in another place.

Table des matières

1 Introduction

Les recherches continues dans le domaine de la logistique ont donné naissance à un concept qui a permis d'apporter au fonctionnement du réseau logistique, beaucoup de propositions innovantes et optimisantes, dont cet algorithme mise en œuvre par des enseignants chercheurs du laboratoire LITIS de l'Université Le Havre Normandie. Ce concept est celui d'"Internet Physique". L'idée, est d'appliquer au réseau logistique, les mêmes principes que le réseau d'internet. C'est-à-dire, d'en faire un réseau mondial, ouvert, utilisant un ensemble de protocoles collaboratifs et standardisées, pour transporter, non pas des « paquets » d'information comme le fait Internet, mais des biens physiques contenus dans des modules standards [1]. Et de la même manière qu'il existe des modèles de référence pour les réseaux de données, tels que le modèle OSI, il existe des modèles pour l'encapsulation et le transport d'objets physiques tels que le modèle NOLI.

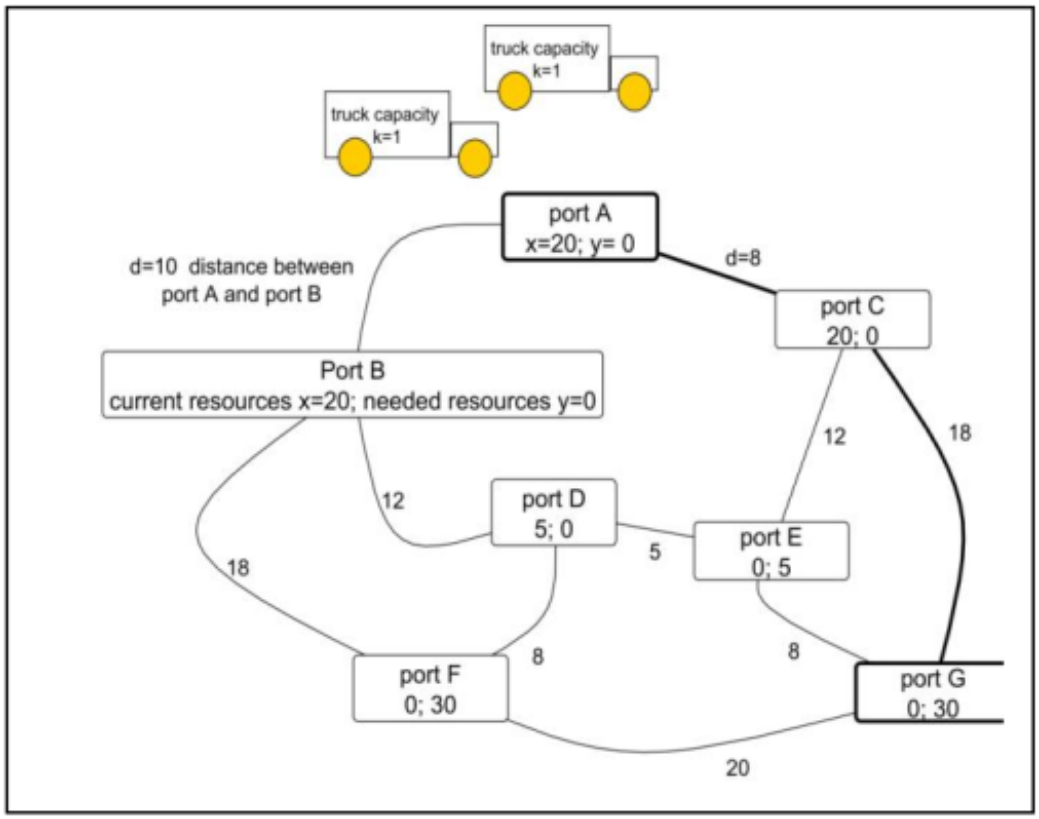
Position in the NOLI model	**Layer Name**	**Role of the Layer**
7	Product Layer	Defines the possible products or goods that can be transported inside π-containers. It fills the π-containers with the products and establishes the related contracts.
6	Container Layer	Defines the physical characteristics of the π-containers allowed on the Logistics Network. It will check the physical integrity of the π-containers and combine them into "sets" according to their characteristics.

[2]

Dans le modèle NOLI, la 6^e couche (la couche container) est chargée de définir les types d'objets physiques transportables et les conteneurs standards pouvant être utilisés pour les transporter. Elle est également celle qui s'occupe d'acheminer aux points de départ les conteneurs vides pour qu'ils

y soient remplis, et de les récupérer aux point d'arrivée une fois qu'ils sont vidés. Un des problèmes que doit résoudre cette couche 6, est le transport de certains conteneurs sophistiqués (qui sont plus rares et plus chers que les conteneurs basiques) tels que les reefers (conteneurs à température dirigée), servant à transporter des aliments surgelés ou des médicaments. Il arrive souvent, en raison des déséquilibres permanents ou saisonniers des échanges, que ces conteneurs rares et chers, se retrouvent en surnombre aux points de livraison une fois vidés, et en quantités insuffisantes aux points de départ, pour être remplis de nouveau et affecter à d'autres trajets.

2 Modélisation du problème à l'aide d'un graphe



[3]

Le problème est modélisé à l'aide d'un graphe correspondant à un réseau routier. Nous avons un ensemble d'emplacements (port A, port B, ...), représentant des unités de production ou des unités de consommation, et un ensemble d'arcs représentant des chemins physiques entre les emplacements. Les arcs sont valués par un coût $d_{i,j}$ représentant la distance entre i et j. Les emplacements sont valués par une paire (x, y) représentant respectivement l'offre et la demande de conteneurs à cet emplacement.

3 Description de l'algorithme

L'algorithme proposé a pour but de permettre de réduire ces déséquilibres mentionnés dans l'introduction, en incitant les transporteurs de camions à utiliser leurs trajets retour qui sont souvent fait à vide, pour prendre des conteneurs frigorifique à un endroit où ils sont en surnombre et les déposés à un endroit où ils en manquent, contre une rémunération qui sera le plus rentable possible, au moins pas plus coûteux que le retour à vide des camions. Voici les principales étapes de l'algorithme :

Étape 1 :

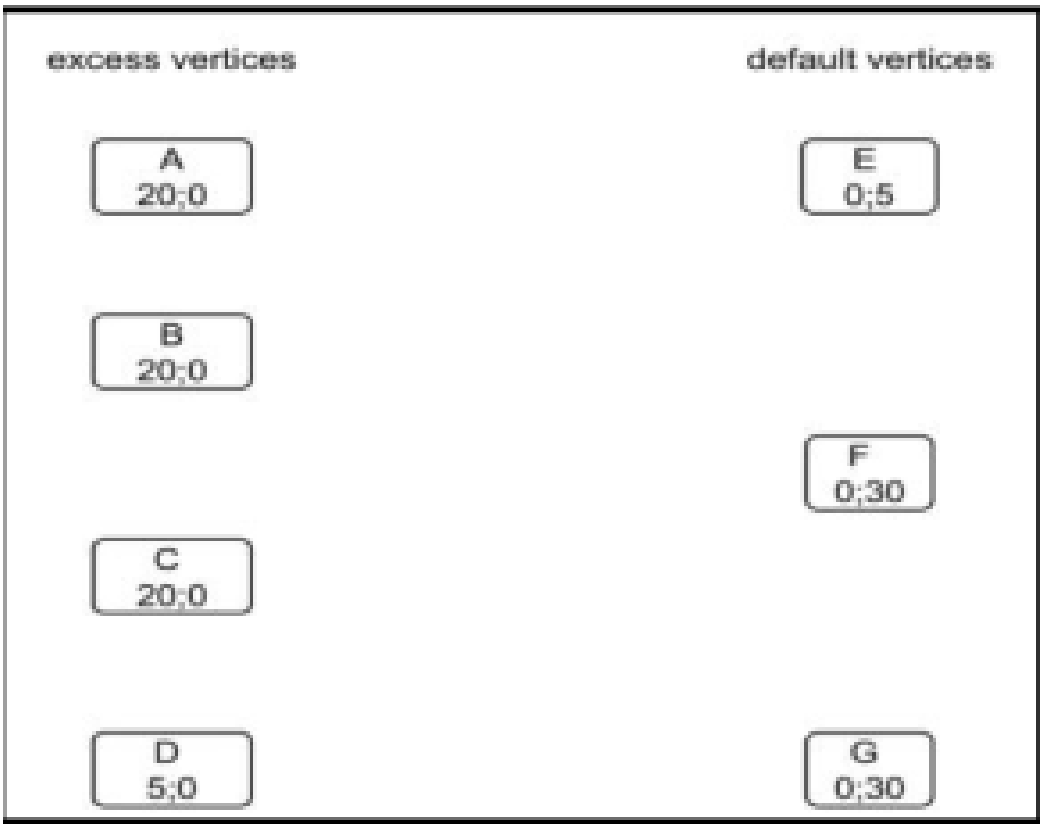
La première étape construit une table des plus courts chemins entre toutes les paires de nœuds du graphe en utilisant l'algorithme de Warshall-Roy. L'algorithme de Floyd-Warshall, également connu sous le nom d'algorithme de Floyd, d'algorithme de Roy-Floyd, d'algorithme de Roy-Warshall ou encore d'algorithme WFI, est un algorithme permettant de trouver efficacement et simultanément les chemins les plus courts entre chaque paire de nœuds d'un graphe pondéré. Le plus grand avantage de l'utilisation de cet algorithme est que toutes les plus courts chemins entre deux nœuds donnés du graphe, peuvent être calculées en O ($|V|^3$) , où V est le nombre de sommets du graphe [4].

Étape 2 :

La deuxième étape déduit en partant de la table construite durant la première étape, les coûts de déviation supplémentaires de tous les trajets entres tous les nœuds du graphe. Si on suppose que A est le point de départ des camions et B le point d'arrivée. Alors : Le coût pour envoyer un camion d'un sommet S à un sommet R = le coût pour aller de A à S + le coût pour aller de S à R + le coût pour aller de R à G - le coût du plus court chemin pour aller de A à G.

Étape 3 :

La troisième étape se sert de ces résultats pour formuler un problème de transport classique entre nœuds sources et nœuds destinations.



Un problème du transport est un type particulier de problème de programmation linéaire, où l'objectif consiste à minimiser le coût de transport d'une marchandise donnée à partir d'un certain nombre de sources ou d'origines (ex. Usine) vers un certain nombre de destinations (ex. Entrepôt, magasin). Chaque source a un approvisionnement limité (c'est-à-dire le nombre maximum de produits qui peuvent en être expédiés) tandis que chaque destination a une demande à satisfaire (c'est-à-dire le nombre minimum de produits qui doivent lui être expédiés). Lorsque les approvisionnements et la demande sont égaux, le problème est considéré comme un problème de transport équilibré. Lorsque l'offre et la demande ne sont pas égales, on dit qu'il s'agit d'un problème de transport déséquilibré. Dans notre cas-ci, l'algorithme considère que nous avons à faire avec un problème de transport équilibré [5].
Le problème de transport ainsi formulé, sera résolu via l'algorithme de Stepping-Stone, qui est un algorithme itératif (c'est-à-dire par étapes successives) visant à faire baisser le coût global du transport. Le principe de cette méthode ou de cet algorithme est de partir d'une solution de base et de progresser par itération, pour enfin arriver à une solution optimale qui minimise les coûts de transport.

Étape 4 :

La dernière étape, qui est optionnelle, impose un ordre sur les trajets à accomplir, en les triant par ordre croissant suivant le coût de la déviation. Cette quatrième étape de l'algorithme est utile dans le cas où le nombre de camions est limité. Ainsi, les camions seront affectés aux trajets les moins coûteux en premier.

4 Implémentation en Java

4.1 Outils utilisés

GraphStream

GraphStream est une bibliothèque Java de gestion de graphes qui se concentre sur les aspects dynamiques des graphes. Son objectif principal est la modélisation de réseaux d'interactions dynamiques de différentes tailles. Le but de la bibliothèque est de fournir un moyen de représenter des graphes et de travailler dessus. Pour cela, GraphStream propose plusieurs classes de graphes qui permettent de modéliser des graphes orientés et non orientés. GraphStream permet de stocker tout type d'attribut de données sur les éléments du graphe : nombres, chaînes ou tout objet. De plus, GraphStream fournit un moyen de gérer l'évolution du graphe dans le temps. Cela signifie gérer la façon dont les nœuds et les arêtes sont ajoutés et supprimés, et la façon dont les attributs de données peuvent apparaître, disparaître et évoluer [6].

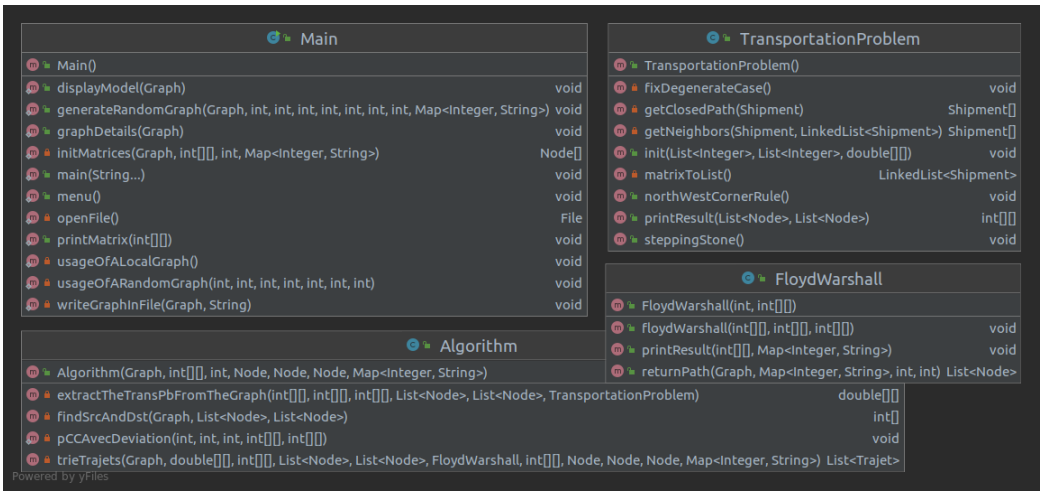
Matrix Toolkits for Java (MTJ)

MTJ est une collection complète de structures de données matricielles et de solveurs linéaires. C'est une bibliothèque open-source et contient un ensemble complet d'opérations d'algèbre linéaire [7].

Java Text Table Formatter

Java Text Table Formatter est une simple librairie proposant un ensemble de classes pour aider à afficher du texte dans la console sous forme de tableau. La bibliothèque Java Text Table Formatter dispose ainsi de la possibilité d'organiser les données en lignes et en colonnes. Chaque cellule de donnée peut contenir plusieurs lignes de texte et peut spécifier un alignement (gauche, centre, droite) et un alignement vertical (haut, centre, bas) [8].

4.2 Résumé des différentes classes obtenues

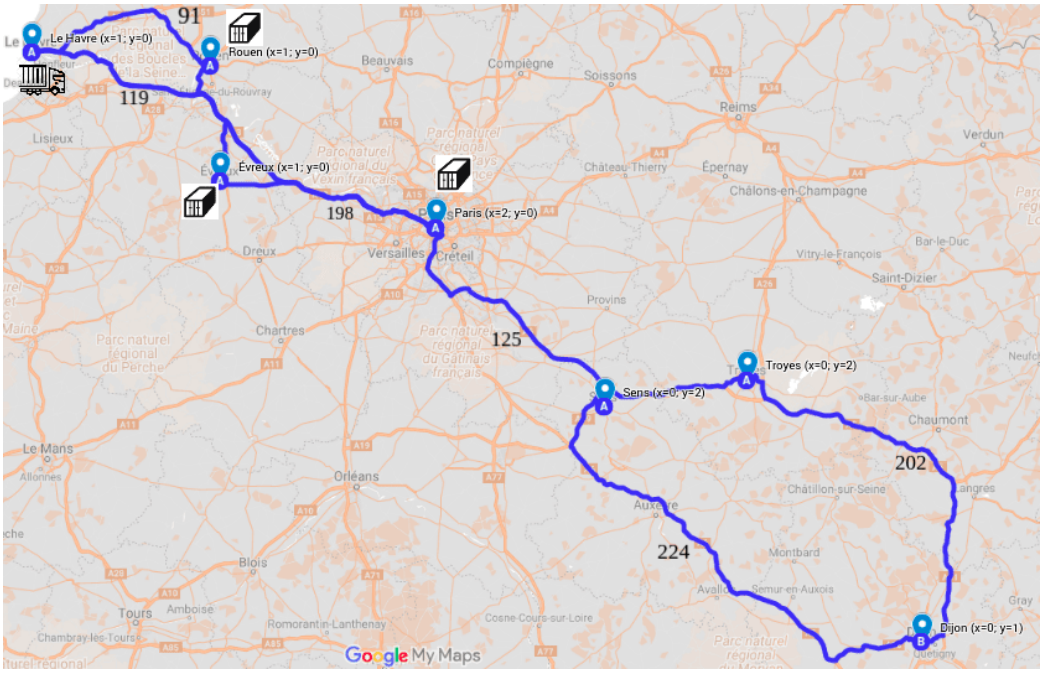


Nous avons une classe principale **Main** : qui se chargera de lire le graphe, d'initialiser tous les éléments nécessaires au démarrage de l'algorithme, une classe **Alogirthm** : qui est celle qui s'occupe d'exécuter l'algorithme étape par étape, en faisant appel à des méthodes définies en son sein, et en utilisant les autres classes, une classe **FloydWarshall** : pour le calcul du plus chemin entre les nœuds du graphe et une classe **TransportationProlem** : qui se charge de résoudre le problème de transport.

4.3 Exemple d'utilisation

Le programme prend en entrée uniquement le graphe (au format dgs) modélisant le problème. Et à partir de ce fichier (s'il est correcte), le programme déduira tous les paramètres nécessaires à l'algorithme et effectuera ensuite son traitement, en affichant dans une fenêtre, une représentation graphstream du graphe (il représentera en rouge le point de départ des camions, et en vert le point d'arrivée), puis en affichant dans la console le résultat de l'exécution de l'algorithme. Le tableau affiché à la fin (celle de la 4ème étape) étant le plus important.

Répresentation sur une carte du graphe que nous allons fournir en entrée



Répresentation au format dgs de ce graphe

```
DGS004
g 0 0

an LeHavre offre=1 demande=0 nodeOrigin
an Rouen offre=1 demande=0
an Evreux offre=1 demande=0
an Paris offre=2 demande=0
an Sens offre=0 demande=2
an Troyes offre=0 demande=2
an Dijon offre=0 demande=1 nodeDestination

ae e1 LeHavre Rouen dist=91
```

```
ae e2 LeHavre Evreux dist=119
ae e3 LeHavre Paris dist=198
ae e4 Rouen Evreux dist=57
ae e5 Rouen Paris dist=136
ae e6 Evreux Paris dist=98
ae e7 Paris Sens dist=125
ae e8 Sens Dijon dist=224
ae e9 Sens Troyes dist=81
ae e10 Troyes Dijon dist=202
```

Compilation et éxecution :

- Vérifier que vous avez bien java installer sur votre machine,
- ouvrir un terminal si vous êtes sous Linux,
- ou ouvrir l'invite de commandes si vous êtes sous Windows,
- se positionner dans le répertoire du projet,
- et puis taper les commandes suivantes :

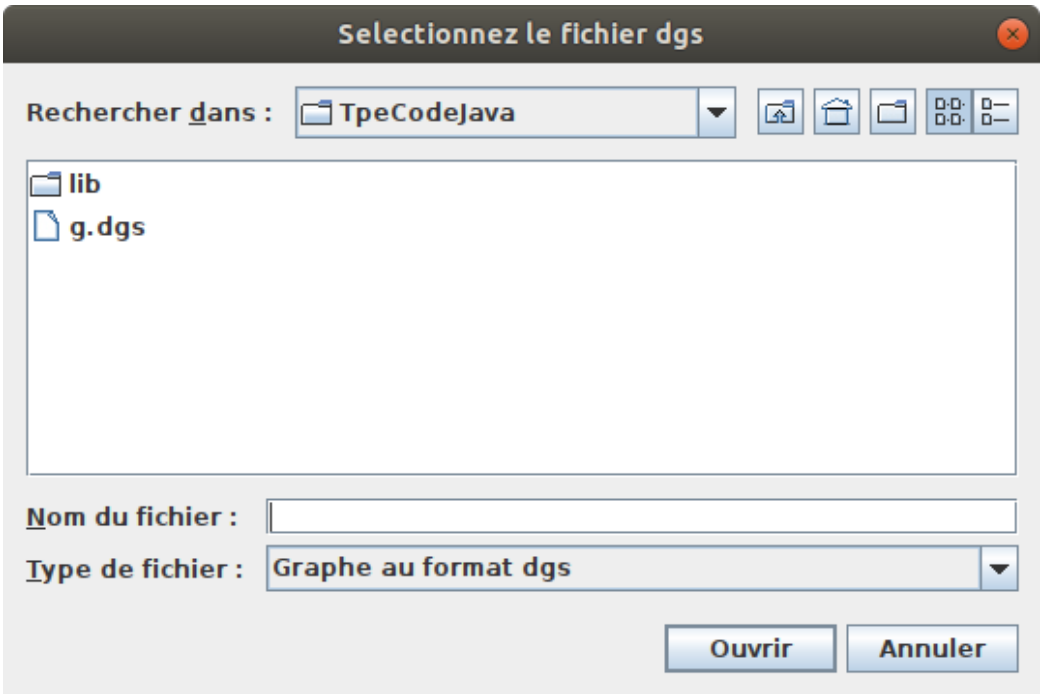
Sur Linux :

```
javac -cp .:./lib/* *.java
java -cp .:./lib/* Main
```

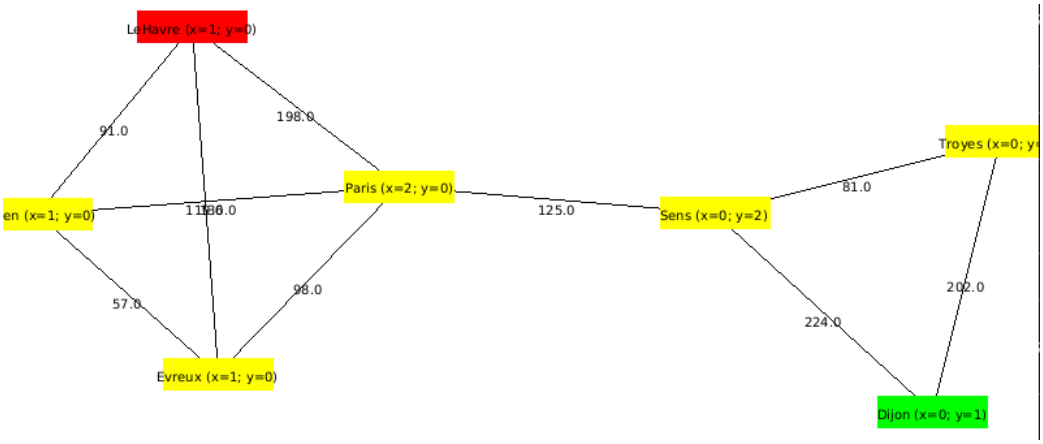
Sur Windows :

```
javac -cp .;./lib/* *.java
java -cp .;./lib/* Main
```

Sortie :



Resultat affiché dans la fenêtre :



Resultat affiché dans la console :

```
******* Algo de gestion des ressources mobiles rares dans un
↪  Internet Physique *******
(--- Version avec 1 seul point de départ ---)

Etape 1: Calcul du plus court chemin entre tous les pairs de
↪  noeuds :

+---------------------+----+------------------------+
|Origine -> Destination|dist|Plus court chemin       |
+---------------------+----+------------------------+
|LeHavre -> Rouen      |91  |LeHavre -> Rouen        |
|LeHavre -> Evreux     |119 |LeHavre -> Evreux       |
|LeHavre -> Paris      |198 |LeHavre -> Paris        |
|LeHavre -> Sens       |323 |LeHavre -> Paris -> Sens|
| etc ...                                           |
+---------------------+----+------------------------+

Etape 2: Calcul du Cout du plus court chemin avec deviation
↪  entre
toutes les paires de sommets :

+---------------------+----+------------------------+
|Origine -> Destination|dist|Plus court chemin       |
+---------------------+----+------------------------+
|LeHavre -> Rouen      |29  |LeHavre -> Rouen        |
|LeHavre -> Evreux     |19  |LeHavre -> Evreux       |
|LeHavre -> Paris      |0   |LeHavre -> Paris        |
|LeHavre -> Sens       |0   |LeHavre -> Paris -> Sens|
| etc ...                                           |
+---------------------+----+------------------------+

Etape 3.1 : Extraction du probleme de transport a partir des
↪  informations fournies avec le graphe :
```

Excess vertices	Quantity
LeHavre	1
Rouen	1
Evreux	1
Paris	2
Total	5

Default vertices	Quantity
Sens	2
Troyes	2
Dijon	1
Total	5

Cout du plus court chemin avec deviation entre les Excess vertices et les Default vertices :

Source -> Destination	dist
LeHavre -> Sens	0.0
LeHavre -> Troyes	59.0
LeHavre -> Dijon	0.0
Rouen -> Sens	29.0
Rouen -> Troyes	88.0

```
|Rouen -> Dijon       |29.0|
| etc ...                  |
+---------------------+----+
```

```
Etape 3.2 : Resolution du probleme de transport :
La methode de Stepping-Stone nous permet de trouver la quantite
↪  optimal a envoyer
en respectant l_offre et en satisfaisant la demande au moindre
↪  cout :
```

```
+---------------------+------------------+
|Source -> Destination|Quantite a envoyer|
+---------------------+------------------+
|LeHavre -> Sens      |        1         |
|LeHavre -> Troyes    |        0         |
|LeHavre -> Dijon     |        0         |
|Rouen -> Sens        |        1         |
|Evreux -> Sens       |        0         |
|  etc ...                               |
+---------------------+------------------+
|Cout total (en km)   |      166.0       |
+---------------------+------------------+
```

```
Etape 4 (la plus importante) : Liste des trajets triee par
↪  ordre croissant suivant le cout de la deviation de
↪  trajectoire :
```

```
+---------------+----+----+---------------------------------+
|Source -> Dest  |Qtte|Cout|Trajet complet par le PCC        |
+---------------+----+----+---------------------------------+
|LeHavre -> Sens |1   |0   |LeHavre -> Paris -> Sens -> Dijon |
|Paris -> Dijon  |1   |0   |LeHavre -> Paris -> Sens -> Dijon |
|Rouen -> Sens   |1   |29  |LeHavre -> Rouen -> Paris -> Sens
↪  -> Dijon |
```

```
|Paris -> Troyes |1   |59  |LeHavre -> Paris -> Sens -> Troyes
↪  -> Dijon |
+---------------+----+----+---------------------------------+
|Evreux -> Troyes|1   |78  |LeHavre -> Evreux -> Paris -> Sens
↪  -> Troyes -> Dijon |
+---------------+----+----+---------------------------------+

******** Fin du résultat ********
```

Ainsi dans l'exemple ci-dessus, si on regarde le tableau affiché à la fin (qui est le plus intéressant), on trouve que les premiers trajets seront 1 conteneur transporté directement de Le Havre à Sens (coût de déviation = 0). S'il reste des camions vides, les trajets suivants seront 1 conteneur pris à Paris, pour servir Dijon (coût de déviation = 0). S'il reste toujours des camions vides, les trajets suivants seront 1 conteneur pris à Rouen pour servir Sens (coût de déviation = 29). etc.

4.4 Implémentation d'une 2ème version de l'algorithme

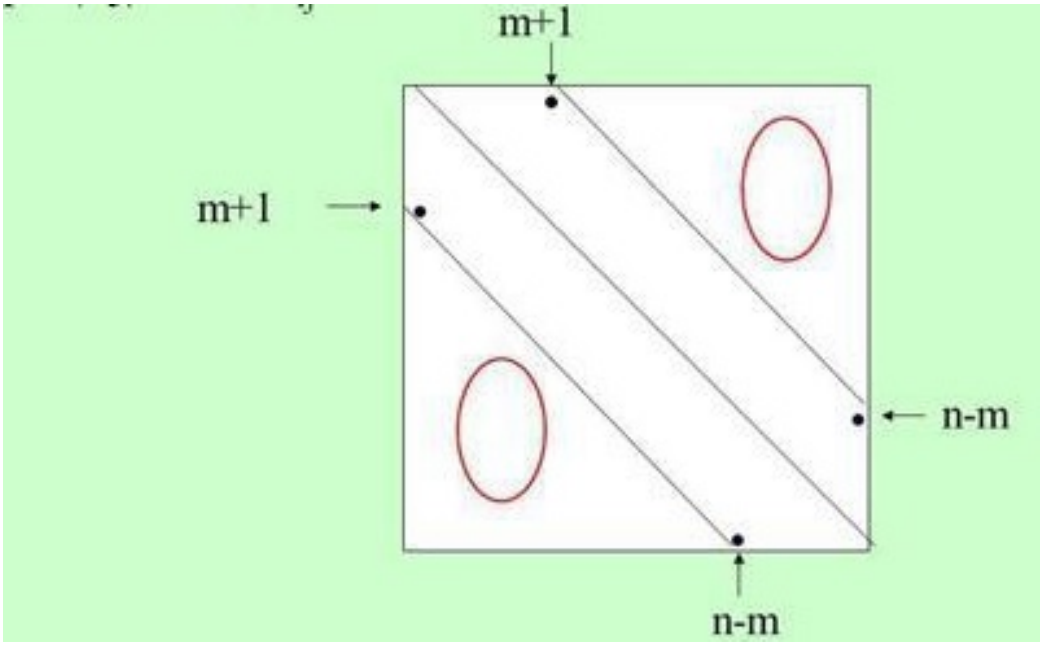
Dans cette deuxième version, il est supposé que les camions sont sur deux centres de départ et doivent retourner vers un seul centre de destination. La légère différence dans l'implémentation entre les deux versions, se trouve dans les étapes 2 et 3 de l'algorithme. Pour cette deuxième version, il faudra déduire, en partant de la table construite dans la première étape, les coûts de déviation supplémentaires de tous les trajets entres tous les nœuds du graphe, par rapport aux deux points de départ des camions. Nous aurons ainsi deux tables en sortie pour cette deuxième étape. Ensuite, il faudra construire une troisième table définie par : T3 = MIN (T1, T2) [9].

5 Simulation directe avec java

5.1 Graphe aléatoire utilisé pour la simulation

Pour pouvoir générer des graphes aléatoires correspondant le mieux à un réseau routier, nous avons fait recours à la notion de **matrice à structure**

bande, encore appelée **matrice bande**. On dit qu'une matrice A(n, n) est de structure bande avec une demi-largeur de bande m, si m est le plus petit entier tel que $|i - j| > m => a_{ij} = 0$. Dans le cas d'un réseau routier, les valeurs nuls sont remplacées par infini.

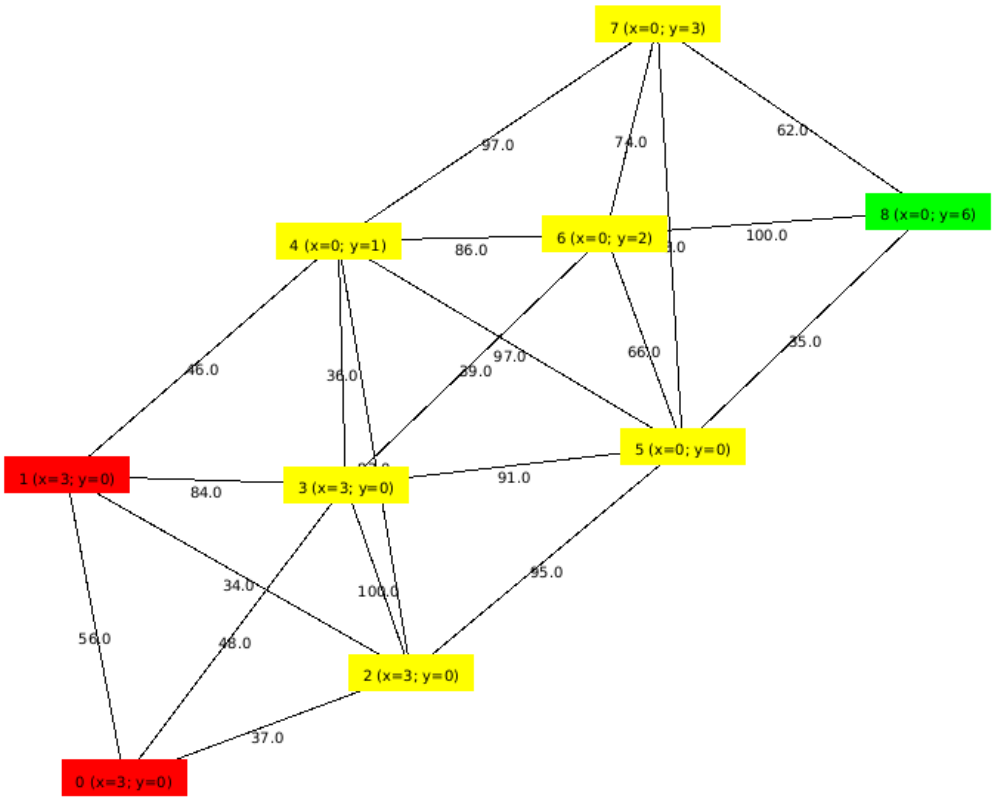


[10]

Exemple :

	0	1	2	3	4	5	6	7	8
0	0	56	37	48	0	0	0	0	0
1	56	0	34	84	46	0	0	0	0
2	37	34	0	100	83	95	0	0	0
3	48	84	100	0	36	91	39	0	0
4	0	46	83	36	0	97	86	97	0
5	0	0	95	91	97	0	66	43	35
6	0	0	0	39	86	66	0	74	100
7	0	0	0	0	97	43	74	0	62
8	0	0	0	0	0	35	100	62	0

Graphe correspondant



Algorithme de génération du graphe aléatoire

ENTRÉE :
- n = nombre de sommets, ça peut varier de 20 à 50
- d est une constante, c'est le nombre max des successeurs d'un sommet, dans l'exemple précédent, d=3. Ainsi, le nombre de voisins pouvait aller jusqu'à 2*d=6
- a(i, j) est la matrice de distance
- alea(p, q) est une fonction qui génère une valeur entière aléatoire v entre p $\leq$ v $\leq$ q
- c = 10 est le nombre maximum de conteneur à affecter à une ville

```
ALGORITHME :
Initialisation
    Pour i = 1 .. n faire
        Pour j = 1 .. n faire
            Si i = j alors
                a(i,j) = 0
            sinon
                a(i,j) = ∞
            fin si ;
        fin pour i
    fin four j
fin initialisation ;

Création de graphe
    pour i = 1 .. n faire
        pour j = i+1 .. i + d faire // d est la largeur de la
        ↪ bande pouvant être 6
            a(i,j) = alea(0,1) ;
        fin pour j
        pour k = i+1 .. i + d faire
            a(k,i) = a(i,k); // la matrice d'adjacence est
            ↪ symétrique
        fin pour j
    fin Pour i
fin Création de graphe ;

Création des conteneurs (x,y) // somme (x) doit être égal à
↪ somme (y)
    Pour k = 1.. n/2 faire //on suppose ici que n est un nombre
    ↪ pair
        Si alea (0,1) <> 0 alors
            q = alea(0, c) ; // c=10 est le nombre maximum de
            ↪ conteneurs à affecter à une ville
```

```
            z = 2 *k -1 ;
            x(z) = q ;
            y(n-z+1) = q // ainsi somme (x) = somme (y)
        Fin si ;
    Fin pour ;
Fin Création des conteneurs ;
FIN ALGORITHME
```

5.2 Résultats de quelques tests sur des graphes aléatoires

Nb nodes	Nb edges	Nb refeers	Exec Time (s)
10	24	14	0,433
18	87	26	0,364
30	182	72	0,644
50	322	63	1,017
70	399	127	1,754

On remarque en observant les temps d'exécution obtenue, que l'implémentation de l'algorithme est plutôt efficace et fournit le résultat en un temps raisonnable.

6 Simulation avec AnyLogic

6.1 Outil utilisé

AnyLogic

Anylogic est un outil de simulation développé par la compagnie du même nom. Il possède un langage de modélisation graphique et permet de créer des modèles de simulation avec du code java. Le logiciel est payant, mais depuis quelque temps, une édition gratuite est disponible pour un usage pédagogique. AnyLogic offre la capacité de pouvoir utiliser des cartes dans

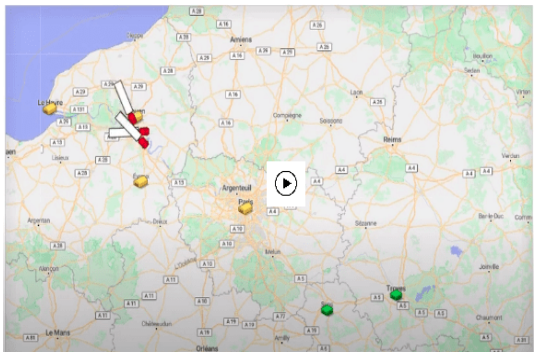
les modèles de simulation. Ces cartes peuvent être utilisées pour modéliser des systèmes tels que des réseaux logistiques ou tout autres situations où il est nécessaire de prendre en compte des emplacements et des routes [11].

6.2 Intégration du résultat dans AnyLogic

Pour simuler le résultat de l'algorithme sous AnyLogic, il a fallu fournir à Anylogic deux fichiers Excel, contenants respectivement les villes ayant un surnombre de conteneurs frigorifiques et les villes ayant un déficit de conteneurs frigorifiques. Ensuite, nous avons défini via l'outil graphique d'AnyLogic, et via du code Java, un modèle simple qui permet de voir un peu plus concrètement, l'usage de l'algorithme dans le monde réel.

6.3 Exemple de simulation du programme

(cliquer sur l'image)



7 Conclusion

Ce travail nous a permis de nous familiariser encore plus avec le très utilisé langage Java, et aussi de découvrir l'outil de Simulation qu'est Anylogic. Nous avons ainsi pu mettre en ouvre une implémentation et une simulation d'un algorithme, dont le but principal était d'essayé d'utiliser les trajets retour des transporteurs de camions, qui sont souvent faits à vide ou presque, pour rééquilibrer les ressources rares présentes dans un réseau logistique.

8 Références

[1] Alice. « Qu'est ce que le concept de l'Internet Physique ? » In : *mines-paris* (mai 2014). [En ligne ; consulté le : 29/05/2021]. URL : https://www.cip.minesparis.psl.eu/concept-de-linternet-physique/.

[2] Moustafa NAKECHBANDI Jean-Yves COLIN Hervé MATHIEU. « A Proposal for an Open Logistics Interconnection Reference Model for a Physical Internet ». In : *deepAI* (oct. 2019). [En ligne ; consulté le : 29/05/2021]. URL : https://deepai.org/publication/a-proposal-for-an-open-logistics-interconnection-reference-model-for-a-physical-internet.

[3] Moustafa NAKECHBANDI Jean-Yves COLIN Hervé MATHIEU. « La gestion des ressources mobiles rares dans un Internet Physique sans ressources dédiées ». In : *LOGISTIQUA 2019* (juin 2019). [En ligne ; consulté le : 29/05/2021]. URL : https://hal.archives-ouvertes.fr/hal-02310223/.

[4] Sridharan T. « Floyd Warshall Algorithm ». In : *DevGenius* (juin 2020). [En ligne ; consulté le : 29/05/2021]. URL : https://blog.devgenius.io/floyd-warshall-algorithm-f004a01ae40e.

[5] Roberto Salazar. « Operations Research with R — Transportation Problem ». In : *towards data science* (nov. 2019). [En ligne ; consulté le : 29/05/2021]. URL : https://towardsdatascience.com/operations-research-in-r-transportation-problem-1df59961b2ad.

[6] GraphStream Team. « Some words about GraphStream ». In : *graphstream-project* (sept. 2020). [En ligne ; consulté le : 29/05/2021]. URL : https://graphstream-project.org/doc/Tutorials/Getting-Started/.

[7] Contributors. « Summary overview of Matrix Toolkits for Java (MTJ) ». In : *GitHub* (mar. 2017). [En ligne ; consulté le : 29/05/2021]. URL : https://github.com/fommil/matrix-toolkits-java/wiki.

[8] I. Namik. « Java-Text-Table-Formatter ». In : *GitHub* (mai 2016). [En ligne ; consulté le : 29/05/2021]. URL : https://github.com/iNamik/Java-Text-Table-Formatter/blob/master/README-USAGE.txt.

[9] Moustafa NAKECHBANDI Jean-Yves COLIN. « Studying the Rerouting of Empty Carriers during their Return Trips to Manage Rare Mobile Resources in a Physical Internet ». In : *ResearchGate* (oct. 2020). [En ligne ; consulté le : 29/05/2021]. URL : https://www.researchgate.net/publication/349165199_Studying_the_Rerouting_of_Empty_Carriers_during_their_Return_Trips_to_Manage_Rare_Mobile_Resources_in_a_Physical_Internet.

[10] Morgaine Chauveau. « matrice à structure bande ». In : *SlidePlayer* (juin 2018). [En ligne ; consulté le : 29/05/2021]. URL : https://slideplayer.fr/slide/502784/2/images/46/III-2+Matrice+%C3%A0+structure+bande.jpg.

[11] Ilya Grigoryev. « AnyLogic in Three Days ». In : *AnyLogic* (juil. 2018). [En ligne ; consulté le : 29/05/2021]. URL : https://www.anylogic.com/resources/books/free-simulation-book-and-modeling-tutorials/.