LOGIroute

Martin Prouvot - Michel Mendi

HISTOIRE DES PROBLEMES DE TRANSPORT ET DE PROBLEMATIQUES DE FLOT MAXIMUM
ALGORITHME DE STEPPING STONE

CNAM de LILLE

mai 2020

Introduction :

- **Quelques figures marquantes**
- **Histoire de l'algorithmie dans les problématiques de transports.**

Etude de l'algo Stepping-stone dans une problématique de transport.

Methode

- **La méthode du coin Nord-Ouest**
- **La méthode Balas Hamer**
- **L'algorithme Stepping-Stone**

Modélisation informatique

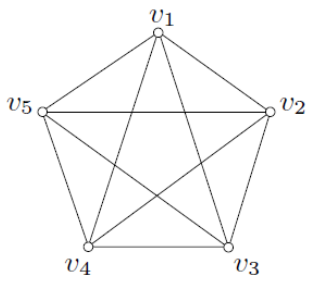
Perspectives et Conclusion

Préambule

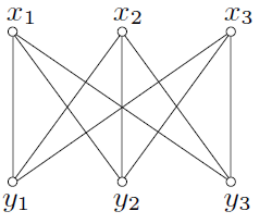
Partant de problématiques de logistiques rencontrées dans nos précédents métiers, notamment, la préparation de marchandises dans un entrepôt, nous avons redécouvert l'algorithme nommé stepping-Stone. Recherchant l'histoire de cette algorithmie, nous avons pu parcourir une littérature scientifique riche et passionnante. Cette littérature retrace l'histoire de l'analyse combinatoire et restitue de nombreux algorithmes dans leur contexte historique. Dans cette littérature et cette histoire, on retrouve de façon quasi permanente des problématiques de transports.

Notre problématique de départ est simple :

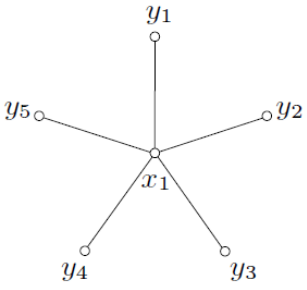
La répartition de marchandises issus de trois zones de départs vers trois emplacements d'arrivée. Qu'allons-nous résoudre ? : le problème de la tournée d'un postier, le problème de la répartition des marchandises en entrepôt et enfin la distribution d'une même marchandise entre 3 usines et 3 magasins.



Un graphe complet



un graphe biparti complet



une étoile

L'histoire de l'algorithmie dans les problématiques de transport et de distribution.

Après avoir décrit quelques figures qui nous interpelle, nous vous proposons un rapide aperçu d'un enchainement d'études en partant du 18ème siècle pour arriver jusqu'aux années 1960.
Commençons par nous arrêter sur quelques personnalités fortes qui ont retenu notre attention.

Quelques figures marquantes

Avant de rentrer dans l'historique des problématiques de transports résolues par l'algorithmie, arrêtez-nous sur quelques personnages de l'histoire de la Théorie des graphs.

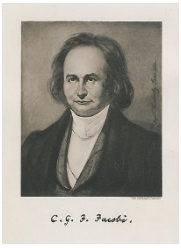
lithographie de François Delpech d'après Henri-Joseph Hesse.

Gaspard Monge, comte de Péluse, né le 9 mai 1746 à Beaune[1] et mort le 28 juillet 1818 à Paris (ancien 10e arrondissement), est un mathématicien et homme politique français.

Son œuvre mêle géométrie descriptive, analyse infinitésimale et géométrie analytique.

Il concourt également avec Berthollet, Chaptal et Laplace à la **création de l'École d'arts et métiers. (vive le cnam !)**

Charles Gustave Jacob Jacobi, (10 décembre 1804 - 18 février 1851), est un mathématicien allemand surtout connu pour ses travaux sur les intégrales elliptiques, les équations aux dérivées partielles et leur application à la mécanique analytique. Il est l'un des fondateurs de la théorie des déterminants. En particulier, il invente le déterminant de la matrice (dite jacobienne)



Jacobi a, probablement le premier, décrit l'algorithme hongrois pour le problème d'affectation.

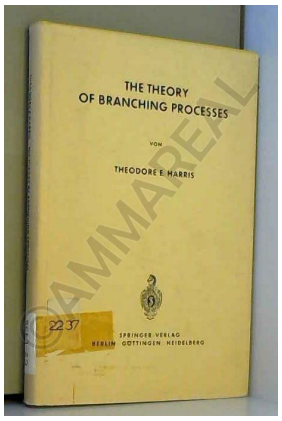
Andrei Markov 1856-1922 - Mathématicien russe
Il est considéré comme le fondateur de la théorie des processus stochastiques. C'est lui qui va à l'origine de l'idée des « maillons augmentants » qui par la suite seront utilisées dans de nombreux algorithmes. Il a inspiré les différents auteurs sur les problématiques de transport.

Ferdinand Georg Frobenius 1849-1917
Mathématicien allemand, Il est l'un des premiers, avec Heinrich Weber, à s'intéresser à la théorie des groupes pour elle-même et non comme outil, et il redémontre dans ce cadre les théorèmes de Sylow. On lui doit l'introduction des caractères d'un groupe non commutatif (en). Il travaille aussi en algèbre linéaire et donne en 1878 la première démonstration générale du théorème de Cayley-Hamilton , les chaines hamiltoniennes. C'est lui donne les fondamentaux de la théorie de la correspondance dans les graphes bipartites.

Dénes Kőnig (September 21, 1884 – October 19, 1944)

Kőnig est né à Budapest, le fils du mathématicien Gyula Kőnig. En 1907, Il reçoit son doctorat et rejoint la faculté de Royal Joseph University in Budapest . Son cours est fréquenté par Paul Erdős, qui est le premier étudiant à résoudre l'un de ses problèmes. Les activités et études de Kőnig ont joué un rôle important dans les travaux de : László Egyed, Paul Erdős, Tibor Gallai, György Hajós, József Kraus, Tibor Szele, Pál Turán, Endre Vázsonyi, . Il écrivit le premier livre sur la théorie des graphes : Theorie der endlichen und unendlichen Graphen in 1936. Cela marqua le début de la théorie des graphes comme discipline mathématique, Ensuite en 1958, Claude Berge écrivit un deuxième livre, Théorie des Graphes et ses applications, suivant son exemple.

Alexis (Alekseï) Nikolaïevitch Tolstoï 1882 – 1945.
Mathématicien Russe. Petit cousin des comtes et de l'écrivain Léon TolstoÏ par son père. Il a travaillé sur l'ébauche des problèmes initiaux sur la base très concrètes des transports de marchandise en U.R.S.S.



Theodore .E. Harris 1919 – 2005

Mathématicien américain connu pour ses travaux sur les processus stochastiques et les chaînes de **Markov**, la théorie des processus de branchement et les modèles stochastiques d'interaction de particules comme les processus de contact. Il a donné son nom à l'inégalité de Harris utilisée en physique statistique et en théorie de la percolation. Il est à l'origine de nombreux ouvrages :

"The theory of branching processes", "Optimal inventory policy" et
" Contact interactions on a lattice" dans *The Annals of Probability*,

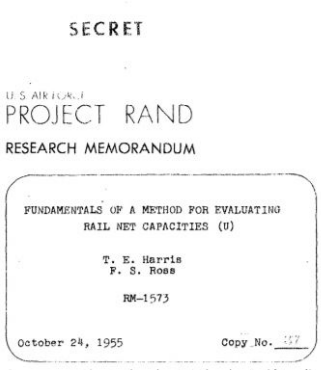
Frank S. Ross, nommé brigadier général américain en 1945.
Il nous faut parler du **projet RAND**

Arretons nous quelques instants sur cette société qui a regroupé de nombreux chercheurs en algorithmie.

F. S. Ross Now Brigadier General

LONDON, July 5 (Reuter)—Colonel Frank S. Ross, organizer of transportation facilities for the United States Army in Great Britain and North Africa, has been promoted to brigadier General, it was officially announced tonight. General Ross will continue as chief of transportation, Services of Supply, in the European Theatre of Operations, United States Army.

ARTICLE NEW YORK TIMES
TUESDAY, JULY 6, 1943, P. 3



SECRET

U.S. AIR FORCE
PROJECT RAND
RESEARCH MEMORANDUM

FUNDAMENTALS OF A METHOD FOR EVALUATING RAIL NET CAPACITIES (U)

T. E. Harris
F. S. Ross

RM-1573

October 24, 1955 Copy No. 117

« La RAND Corporation (« Research ANd Development »), fondée en 1948 par la Douglas Aircraft Company pour conseiller l'armée américaine, est une institution américaine de conseil et de recherche qui se donne pour objectif d'améliorer la politique et le processus décisionnel par la recherche appliquée et l'analyse stratégique. Elle a ensuite progressivement élargi son champ d'action en travaillant pour d'autres gouvernements, pour des fondations privées, pour des organisations internationales, et pour des entreprises privées, sur des questions de défense et de sécurité mais aussi sur l'économie industrielle en général. La RAND Corporation est basée en Californie, en Belgique, en Louisiane, en Angleterre et en Australie. Rand Corporation est considérée comme un laboratoire d'idées (think-tank) américain au service de la décision politique et économique. Elle est financée par le gouvernement américain, par des dotations privées, par des entreprises, des universités et des dons de particuliers. La RAND se caractérise depuis sa fondation par une approche interdisciplinaire et quantitative de la résolution de problèmes, en traduisant des concepts théoriques de l'économie formelle et des sciences en applications nouvelles dans d'autres sphères, c'est-à-dire via les sciences appliquées et la recherche opérationnelle. » WikiPedia

BUSINESS
SINESS OPPORTUNITIES
ORIDA REAL ESTATE &
USINESS PROPOSITIONS

The New York Times.

60, by The New York Times Company. SUNDAY, MAY 22, 1960.

Air Force Think Factory Is One of the Least-Known Reservoirs of Brain Power

RAND Corporation, operating on annual $13,500,000 Air Force budget

RAND Corporation Furnishes Brain Power for the Air Force

By BILL BECKER

AD 93458

Armed Services Technical Information Agency

Reproduced by
DOCUMENT SERVICE CENTER
KNOTT BUILDING, DAYTON, 2, OHIO

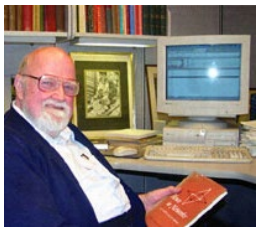
This document is the property of the Government. It is furnished for the duration of the contract and shall be returned when no longer required, or upon recall by ASTIA to the following address: Armed Services Technical Information Agency, Document Service Center, Knott Building, Dayton 2, Ohio.

NOTICE: WHEN GOVERNMENT OR OTHER DRAWINGS, SPECIFICATIONS OR OTHER DATA ARE USED FOR ANY PURPOSE OTHER THAN IN CONNECTION WITH A DEFINITELY RELATED GOVERNMENT PROCUREMENT OPERATION, THE U. S. GOVERNMENT THEREBY INCURS NO RESPONSIBILITY, NOR ANY OBLIGATION WHATSOEVER; AND THE FACT THAT THE GOVERNMENT MAY HAVE FORMULATED, FURNISHED, OR IN ANY WAY SUPPLIED THE SAID DRAWINGS, SPECIFICATIONS, OR OTHER DATA IS NOT TO BE REGARDED BY IMPLICATION OR OTHERWISE AS IN ANY MANNER LICENSING THE HOLDER OR ANY OTHER PERSON OR CORPORATION, OR CONVEYING ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE OR SELL ANY PATENTED INVENTION THAT MAY IN ANY WAY BE RELATED THERETO.

Au sein de ce projet RAND, on retrouve pleins de personnalités comme Jon Hal Folkman 1938-1969), Lester Randolph Ford, fils du mathématicien Lester R. Ford senior. Ford est connu pour sa contribution au problème de flot maximum : le théorème flot-max/coupe-min sur le problème de flot maximum et l'algorithme de Ford-Fulkerson pour le résoudre paraissent dans des rapports techniques en 1954 resp. 1955 et dans un périodique public en 1956.

Lester Randolph Ford, Sr.1886-1967
mathématicien américain, rédacteur en chef de l'American Mathematical Monthly de 1942 à 1946, et président de la Math.Association of America de 1947 à 1948. Il a décrit en 1938 une famille de cercles, qui portent désormais son nom : les cercles de Ford.

Delbert Ray **Fulkerson 1924-1976**
mathématicien américain, coauteur de l'algorithme de Ford-Fulkerson. On a donné son nom au prix Fulkerson, qui récompense tous les trois ans des articles dans le domaine des mathématiques discrètes.

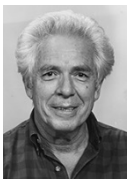


Lester Randolph Ford, Jr .1927-2017
mathématicien americain spécialiste des problèmes des réseaux de transport. Il est le fils du mathématicien Lester R. Ford senior. Il est connu pour sa contribution au problème de flot maximum et sa co-écriture de l'algorithme de Ford-Fulkerson.

Julia Hall Robinson, née Bowman, (8 décembre 1919 à Saint-Louis, Missouri, États-Unis – 30 juillet 1985 à Oakland, Californie) était une mathématicienne américaine. Elle est maintenant surtout connue pour ses travaux sur la résolution du dixième problème de Hilbert.

Anecdote étonnante : à neuf ans, son résultat à un premier test de QI s'avéra être légèrement inférieur à la moyenne (98), selon elle parce qu'elle lisait lentement et n'était pas habituée à passer des tests. C'est à cette époque qu'elle commença à s'intéresser aux mathématiques. Elle fut la seule fille à choisir l'orientation mathématiques-physique, et reçut les félicitations en fin d'année.
Elle va travailler pendant un an en 1940 sur le Rand Project.

George Bernard Dantzig (8 novembre 1914 à Portland (Oregon) - 13 mai 2005 à Palo Alto, en Californie) est un mathématicien américain, promoteur des méthodes cycliques, avec une projection de résolution par les méthodes du simplexe en optimisation et programmation linéaire.



Harold William Kuhn (July 29, 1925 – July 2, 2014)

Mathématicien américain qui a étudié la théorie des jeux. Il va populariser les méthodes stepping stone en les appelant Méthode hongroise dans les années
1955-1960 et redécouvrir l'origine des travaux de Jacobi.

Tjalling Charles Koopmans (28 août 1910 - 26 février 1985)

Économiste néerlandais, spécialiste de l'économie mathématique. Il met en pratique l'algorithme de transport sur le transport maritime. C'est aux États-Unis qu'il publie ses principaux travaux au **Combined Shipping Adjustment Board,**

Egon Balas (June 7, 1922 à Cluj, Roumanie – March 18, 2019)
Mathématicien appliqué et professeur en mathématique appliqué à l'industrie.
Il fit un travail de recherche sur le développement d'intégrale et de programmation disjonctive.

Peter L. Hammer (23 décembre 1936 à Timisoara, Roumanie - 27 décembre 2006 à Princeton, New Jersey) est un mathématicien américain d'origine roumaine. Ses travaux s'inscrivent dans le champ de la recherche opérationnelle et des mathématiques discrètes Appliquées et ont porté essentiellement sur l'étude des fonctions pseudo-booléennes, avec des connexions en théorie des graphes et en analyse de données.

Alexander Schrijver, né le 4 mai 1948 à Amsterdam est un mathématicien et informaticien hollandais, professeur de mathématiques discrètes et d'optimisation à l'université d'Amsterdam et membre du Centrum voor Wiskunde en Informatica à Amsterdam. Depuis 1993, il est coéditeur en chef du journal Combinatorica.

Son article complet (en Annexe) est formidablement structurant pour l'ensemble de l'histoire de la théorie des graphes. Il y explique qu'à partir des années 1950, la programmation linéaire forme la charnière de l'optimisation combinatoire et que l'ensemble des recherches algorithmiques se structurent comme une discipline cohérente des mathématiques.

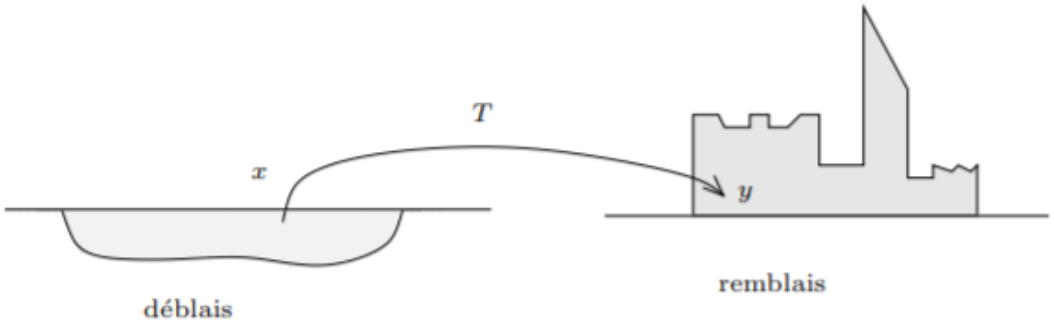
Son article sur l'histoire des optimisations combinatoires et sur l'utilité dans les transports se résume ainsi.

« Après la formulation de la programmation linéaire comme une problématique générique, et son développement en 1947 par la méthode simplex comme un outil, on a essayé de soumettre toutes les problématiques d'optimisation combinatoires avec des techniques de programmation linéaire, souvent avec

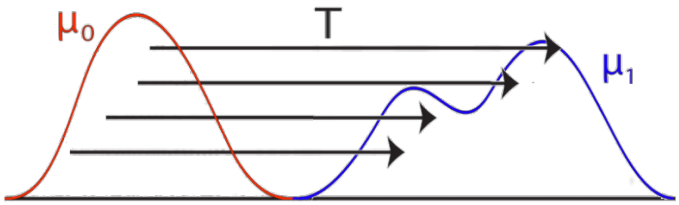
succès «
la cause de la diversité importante des sources de problématiques vient du fait qu'elles viennent directement de la pratique et des exemples de ces sources sont encore aujourd'hui l'objet d'étude algorithmique. On peut imaginer que même dans les sociétés primitives, il était essentiel de résoudre de trouver le chemin le plus court. Le voyageur de commerce ou un médecin qui fait sa tournée, assigner des tâches ou répartir des marchandises pour un trajet, sont des problèmes élémentaires qui ne s'adressent pas qu'aux mathématiciens. C'est pourquoi on retrouve des traces nombreuses dans l'histoire. »

Histoire de l'algorithmie dans les problématiques de transports.

Il démarre son étude avec le travail de Gaspard Monge. Il va étudier en 1784 la problématique du déplacement de la terre d'un fossé vers un remblais :



« Lorsqu'on doit transporter des terres d'un lieu dans un autre, on a coutume de donner le nom de Déblai au volume des terres que l'on doit transporter, & le nom de Remblai à l'espace qu'elles doivent occuper après le transport. Le prix du transport d'une molécule étant, toutes choses d'ailleurs égales, proportionnel à son poids et à l'espace qu'on lui fait parcourir, et par conséquent le prix du transport total devant être proportionnel à la somme des produits des molécules multipliées chacune par l'espace parcouru, il s'ensuit que le déblai et le remblai étant donnés de figure et de position, il n'est pas indifférent que telle molécule du déblai soit transportée dans tel ou tel autre endroit du remblai, mais qu'il y a une certaine distribution à faire des molécules du premier dans le second, d'après laquelle la somme de ces produits sera la moindre possible, et le prix du transport total sera un minimum »



L'article de Monge permet de comprendre qu'il y a une recherche de translation de matière au mieux. Typiquement, c'est une problématique d'affectation qui s'annonce dans le cas pratique exposé par Gaspard Monge.

L’analyse très complète d’Alexandre Schrijver explique les différentes études qui ont suivies et leurs applications.

En résumé, même si je vous invite à parcourir l’article complet :

Frobenius, en 1912, analyse le problème dans une matrice en terme de matrices et déterminants et résume le problème à une technique d’assignement ou d’affectation.

En recherche à son exposé, König va proposer de résoudre le problème sous la forme d’un graphe Bipartite.

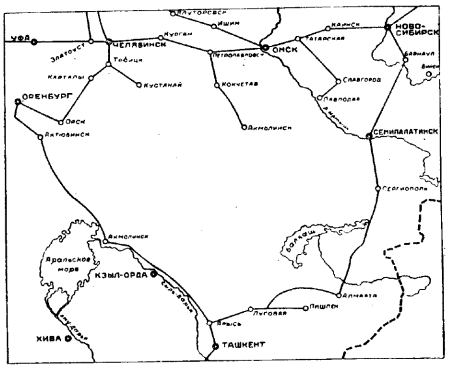
En 1914, ils vont proposer au Congrès de Philosophie mathématique à Paris la théorie que la résolution de leur théorème est une solution optimale.

En 1927, Menger étudie également l’algorithme mais c’est Egerváry en 1931 qui va proposer d’analyser le problème avec une assignation tenant compte du poids de chaque élément déplacé.
C’est lui inspirera Kuhn et la méthode hongroise.

Easterfield en 1946 publiera le premier problème d’affectation. Il proposera de travailler sur l’affectation des soldats de la Royal Aire Force.

Julia Robinson en 1949 proposera une réduction par itération et par cycle.
Elle dira que c’est une tentative échouée de résoudre la problématique du voyageur de commerce qui la poussera à essayer de résoudre le problème de l’affectation par cycle.

Une avancée remarquable dans la résolution du problème d’affectation viendra quand Dantzig en 1951 montra qu’elle peut être formulée en termes de programmation linéaire et qu’elle intègre automatiquement une solution optimale. La raison est que le théorème de Birkhoff en 1946 Indique que la coque convexe des matrices de permutation est égale à l'ensemble de double des matrices stochastiques - matrices non négatives dans lesquelles chaque somme de ligne et de colonne est égale à 1.
Par la suite, Thorndike en 1950 proposera, pour résoudre une problématique d’assignation de personnel, trois résolutions qui touchent plus à l’heuristique. « the Method of Divine Intuition”, “the Method of Daily Quotas”, et “the Method of Predicted Yield”.
Dans des cas complexes, Von Neumann va s’intéresser à la complexité des problématiques d’affectations et à leurs résolutions. En réfléchissant aux méthodes de simplifications initiales, on va aller vers la méthode hongroise de Kuhn 1955-1956 et Munkres 1957 qui va résumer ce qu’on pourrait appeler la première méthode stepping-stone.



Le cas particulier du problème du transport.

C'est à travers un article de A .N. Tolstoï, qui a étudié les problèmes de transport en 1930, pour le train soviétique que l'on rentre vraiment dans une problématique complexe qui inclut affectation initiale de marchandises, distances et optimisation par les coûts à grande échelle. L'article de A .N. Tolstoï, qui a étudié les problèmes de transport en 1930, pour le train soviétique décrit par un critère de cycle négatif qui peut optimiser les problèmes de transport de large échelle. Il applique dès 1930 sur une problématique étendue pour l'époque (10 origines et 68 destinations).

Au cœur de ces problématiques, on retrouve la logique de « chainons augmentant » du mathématicien russe Andrei Markov. Par la suite, les américains souhaiteront étudier le train soviétique, principalement pour des raisons de stratégie militaire et ainsi va se constituer toute une approche autour du « RAND report », projet dévoilé récemment. Ce projet sera la réunion de nombreux mathématicien qui vont prendre conscience de l'intérêt économique et stratégique de leur connaissance et qui vont principalement œuvrer dans le sens du transport de marchandises.

On retient les écrits de T.E. Harris et F.S Ross en 1955, notamment un article qui a inspiré Ford et Fulkerson, le mentionnant comme important pour étudier le flot maximum. Les deux articles ont en commun de s'être appliqués à résoudre initialement la problématique du train soviétique.

On peut citer Kantorovitch en 1940 qui a travaillé simultanément sans connaître le travail de Tolstoï sur un modèle similaire de transport des marchandises en bateau et Hitchcock en 1941 qui a donné une définition mathématique précise du problème. Enfin Koopmans à partir de 1942 trouvera des exemples très concrets sur le transport maritime.

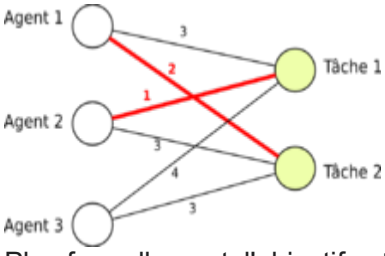
Je vous invite à lire l'article passionnant d'Alexandre Schrijver qui présente l'histoire passionnante de nombreux autres théoriciens comme Menger, Shimbel, Rado, Bellman, Dantzig, Dijkstra, Moore, et j'en oublie…

Il résume les problèmes de transports à six grandes pistes de recherche :

- L'affection ou l'assignement
- Le choix optimisé de routes de transport de marchandises.
- Le flux maximum,
- L'arbre le plus court,
- Le chemin le plus court,
- La problématique du voyageur de commerce

Etude de l'algo Stepping stone dans une problématique de transport.

Dans démarrer notre étude, nous étudierons d'abord le problème de l'affectation dans le cadre d'un graphe bipartite. Pour bien comprendre l'affectation, relisons ce qu'en dit Wikipedia :



En informatique, plus précisément en recherche opérationnelle et d'optimisation combinatoire, le **problème d'affectation** consiste à attribuer au mieux des tâches à des agents. Chaque agent peut réaliser une unique tâche pour un coût donné et chaque tâche doit être réalisée par un unique agent. Les affectations (c'est-à-dire les couples agent-tâche) ont toutes un coût défini. Le but est de minimiser le coût total des affectations afin de réaliser toutes les tâches.

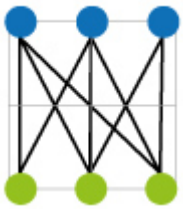
Plus formellement, l'objectif est de déterminer un couplage parfait de poids minimum (ou de poids maximum) dans un graphe biparti valué. Le problème d'affectation peut être résolu en temps polynomial par l'algorithme hongrois, il appartient par conséquent à la classe de complexité P.

(La classe **P**, est une classe très importante de la théorie de la complexité, un domaine de l'informatique théorique et des mathématiques. Par définition, un problème de décision est dans **P** s'il est décidé par une machine de Turing déterministe en temps polynomial par rapport à la taille de l'entrée. On dit que le problème est décidé en **temps polynomial**. Les problèmes dans **P** sont considérés comme « faisables » (*feasible* en anglais), faciles à résoudre (dans le sens où on peut le faire relativement rapidement).)

Le problème d'affectation est classiquement résolu par un algorithme de Kuhn (la méthode hongroise). Cette méthode consiste à trouver sur un tableau N x m, une seule case par ligne et colonne de sorte que la somme des cases retenues soit la plus petite. La méthode consiste à réduire chaque case du minima de la ligne, puis de la colonne, puis à choisir sur les valeurs 0, les cases qui vont être conservés. (voir annexe)
On peut considérer que la méthode Hongroise résoud une problématique de transport avec une quantité de marchandise à UN par ligne et UN par colonne et des coûts de transports qui correspondent aux distances entre les points origines et les points destinations.

Passons maintenant pas à pas d'un problème d'affectation à une problématique de transport.
Voici quelques exemples simples et concrets pour comprendre notre projet :

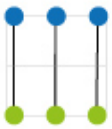
1/ Premier exemple : trois postiers et trois colis à trois clients.



Exemple simple où les points bleus sont les postiers et les points verts les clients à livrer.
Les distances à parcourir sont les lignes.

	Client 1 Attend 1 colis	Client 2 Attend 1 colis	Client 3 Attend 1 colis
Postier 1 livre 1 colis	2 km (Distance)	2.23 km	2.82 km
Postier 2 livre 1 colis	2.23 km ($\sqrt{5}$ ~)	2 km	2.23 km
Postier 3 livre 1 colis	2.82 km ($\sqrt{8}$ ~)	2.23 km	2 km

La question qui nous est posée : Quelles affectations de colis à quel postier dans l'objectif de minimiser les kilomètres. Dans ce cas simple, la réponse apparait intuitivement. Soit 6 km pour les trois postiers. Chaque source cherchant sa destination la plus directe et la plus proche.



		C1	C2	C3
		1	1	1
P1	1	2 km		
P2	1		2 km	
P3	1			2 km

C'est la seule solution optimale. On a une logique d'affectation complète dans une graphe bipartite isomorphe.

On verra par la suite les méthodes qui peuvent être appliqués pour trouver ce résultat.

2/ Deuxième exemples, plusieurs colis ou palettes dans un entrepôt vers trois zones de stockage

Prenons le cas de l'équidistance de tous les points origines (en bleu) et les point de destination en vert. On a une distance mais cette fois ci, on peut proposer des quantités de marchandises différentes.

		D1	D2	D3
	>> Quantités demandées >>	3	3	4
O1	2	1	1	1
O2	3	1	1	1
O3	4	1	1	1
	^^ Quantités en origine ^^			

On voit dans cette exemple qu'il y a 9 colis à transporter. Il y a plusieurs solutions d'affectations pour un coût identique. Le coût distance sera toujours 9 x 1, quelle que soit l'affectation.

		D1	D2	D3
	Quantités	3	3	4
O1	2	2	-	-
O2	3	-	3	-
O3	4	-	-	4
O4		1	-	-

Ce premier tableau propose une solution avec seulement trois déplacements différents. (par exemple sur un chariot). On a créé une origine virtuelle qui indique qu'un colis virtuel n'a pas rempli la destination. Cette technique d'origine ou de destination virtuelle permet de résoudre des cas où le nombre de colis en entré et en sortie en sont pas les mêmes.

D'autres solutions sont possibles car les coûts ou distances sont identiques partout.
Il n'y a pas une solution optimale.

		D1	D2	D3
	Quantités	3	3	4
O1	2	2	-	-
O2	3	1	2	-
O3	4	-	1	3
O4		-	-	1

3 / Troisième exemple, maintenant imaginez que dans un entrepôt, il y ait seulement deux quais d'approvisionnements et trois entrepôts de stockage qui sont à des distances différentes (en général c'est lié à une faible, moyenne ou forte rotation selon le type de produit).

Ou encore imaginez trois usines de productions et 3 magasins à livrer.

On considère que chaque colis produit à un coût de déplacement qui est proportionnel à la distance entre les Usines Origines et les Magasins de Destinations.

Voici un exemple avec des quantités et coûts unitaires au hasard.

		D1	D2	D3
	Quantités	7	8	12
O1	6	1	2	3
O2	11	3	1	2
O3	9	2	3	1

Le magasin D1 demande 7 colis, D2 8 colis, etc. L'usine O1 est capable de livrer 6 colis, etc
Les coûts de déplacement de O1 à D1 sont de 1 par colis livré.

On trouve une première solution simple quelconque

		D1	D2	D3
	Quantités	7	8	13
O1	6	6		
O2	11			11
O3	9		8	1
O4		1		1

Un magasin D1 et un magasin D3 ne seront pas fournis d'un colis.
on calcule un coût global de :
6x1 + 11 x 2 + 8x3 + 1x1, soit au total un coût de 53

La seule solution optimale est :

		D1	D2	D3
	Quantités	7	8	13
O1	6	5		1
O2	11		8	3
O3	9			9
O4		2		

Le coût minimum optimal est de 5 x 1 + 8 x 1 + 9 x 1 + 3 x 2 + 1 x 3, soit 31. Bien moins que 53 au départ. C'est le coût minimum. A noter dans cet exemple que 2 colis attendus par le magasin D1 ne pourront pas être fournis.

Découverte des méthodes.

La méthode d'affectation va être réalisée en deux temps.
1 / Trouver une méthode empirique ou approchante

2/ Trouver une solution optimale par un jeu de cycles.

Nous allons vous présente la méthode du coin Nord-Ouest simple et empirique, puis la méthode Balas Hammer, approchante mais non optimale, puis la méthode Stepping Stone.

1 Méthode du Coin Nord-Ouest.

C'est une méthode empirique très simple qui consiste à remplir les cases de haut en bas et de gauche à droite en remplissant au maximum les cases. Voici un exemple

	Dest. 1	**Dest. 1**	**Dest. 1**	**Dest. 1**	**Dest. 1**	**Offre**
Origine 1	**5**	**3**	**5**	**4**	**5**	12
Origine 2	**6**	**5**	**5**	**3**	**3**	11
Origine 3	**2**	**2**	**8**	**6**	**7**	14
Origine 4	**7**	**6**	**5**	**5**	**5**	8
Demande	10	11	15	5	4	

En bleu, les quantités de la demande des magasins en destination et des entrepôts de production.
en rouge, le coût unitaire de déplacement d'une marchandise.

1.1 remplissage en partant du haut à gauche en essayant de remplir au maximum

En vert, case, ligne ou colonne saturée ; on a rempli au maximum, en orange, il y a un reste.

	Dest. 1	**Dest. 1**	**Dest. 1**	**Dest. 1**	**Dest. 1**	**Offre**
Origine 1	**10**					12
Origine 2						11
Origine 3						14
Origine 4						8
Demande	10	11	15	5	4	

Entre O1 et D1, on peut affecter 10, il reste 2 dans Origine.

	Dest. 1	**Dest. 1**	**Dest. 1**	**Dest. 1**	**Dest. 1**	**Offre**
Origine 1	**10**	**2**				12
Origine 2		?				11
Origine 3						14
Origine 4						8
Demande	10	11	15	5	4	

Puis on approvisionne depuis l'origine 2

	Dest. 1	**Dest. 1**	**Dest. 1**	**Dest. 1**	**Dest. 1**	**Offre**
Origine 1	**10**	**2**				12
Origine 2		**9**	?			11
Origine 3						14
Origine 4						8
Demande	10	11	15	5	4	

	Dest. 1	Dest. 1	Dest. 1	Dest. 1	Dest. 1	Offre
Origine 1	**10**	**2**				12
Origine 2		**9**	**2**			11
Origine 3			**13**	**1**		14
Origine 4				**4**	**4**	8
Demande	10	11	15	5	4	

Voici donc une première solution possible déplacement.

Maintenant calculons le coût.

	Dest. 1	Dest. 2	Dest. 3	Dest. 4	Dest. 5	Offre
Origine 1	**5*10=50**	**3*2=6**	**5**	**4**	**5**	12
Origine 2	**6**	**5*9=45**	**5*2=10**	**3**	**3**	11
Origine 3	**2**	**2**	**8*13=104**	**6*1=6**	**7**	14
Origine 4	**7**	**6**	**5**	**5*4=20**	**5*4=20**	8
Demande	10	11	15	5	4	

Le coût total est de 281.

comparons avec le résultat optimal calculé avec stepping Sone.

	Dest. 1	Dest. 2	Dest. 3	Dest. 4	Dest. 5	Offre
Origine 1	**5**	**3*7=21**	**5*5=50**	**4**	**5**	12
Origine 2	**6**	**5**	**5*2=10**	**3*5=15**	**3*4=12**	11
Origine 3	**2*10=20**	**2*4=8**	**8**	**6**	**7**	14
Origine 4	**7**	**6**	**5*8=40**	**5**	**5**	8
Demande	10	11	15	5	4	

Le coût total est de 151. Soir une optimisation de presque la moitié.

2 Méthode de Balas Hammer.

L'algorithme de Balas-Hammer, aussi appelé méthode des différences maximales ou méthode des regrets, est un heuristique permettant d'optimiser un programme de transport (cas particulier d'un problème d'optimisation linéaire). Le but de cet algorithme est d'assurer les transports à moindre coût.

3 l'algorithmie de Stepping Stone

Dans cet algorithme, on va tenir compte des coûts.

Etape 1 : le départ - Trouver une solution initiale plausible utilisant une des méthodes ci-dessus (coin Nord-Ouest, Balas Hammer, ou d'autres décrites ultérieurement.

	D_1	D_2	D_3	D_4	D_5
O_1	7	12	1	5	9
O_2	15	3	12	6	14
O_3	8	16	10	12	7
O_4	18	8	17	11	16

	D_1	D_2	D_3	D_4	D_5
O_1	10	2			
O_2		9	2		
O_3			13	1	
O_4				4	4

	D_1	D_2	D_3	D_4	D_5
O_1	7	12			
O_2		3	12		
O_3			10	12	
O_4				11	16

Etape 2 : le cycle
2.1 Dessiner un chemin de potentiels qui correspondent à des réductions de coût quand on ajoute 1 unité à cette case.

1	D_1	D_2	D_3	D_4	D_5	Pot.
O_1	7	12				0
O_2		3	12			
O_3			10	12		
O_4				11	16	
Pot.	7					

3	D_1	D_2	D_3	D_4	D_5	Pot.
O_1	7	12				0
O_2		3	12			9
O_3			10	12		
O_4				11	16	
Pot.	7	12				

5	D_1	D_2	D_3	D_4	D_5	Pot.
O_1	7	12				0
O_2		3	12			9
O_3			10	12		11
O_4				11	16	
Pot.	7	12	21	23		

2	D_1	D_2	D_3	D_4	D_5	Pot.
O_1	7	12				0
O_2		3	12			
O_3			10	12		
O_4				11	16	
Pot.	7	12				

4	D_1	D_2	D_3	D_4	D_5	Pot.
O_1	7	12				0
O_2		3	12			9
O_3			10	12		
O_4				11	16	
Pot.	7	12	21			

6	D_1	D_2	D_3	D_4	D_5	Pot.
O_1	7	12				0
O_2		3	12			9
O_3			10	12		11
O_4				11	16	12
Pot.	7	12	21	23	28	

On commence par une destination avec une valeur quelconque, ici 7, puis une origine, puis une destination…..en soustrayant de D à O et en ajoutant de O à D.
On fixera le premier potentiel arbitrairement et…suivre les flèches (on soustrait des destinations aux origines et on ajoute des origines aux destinations).

2.2 Identifier une variation de cout de transport « case vide rapportée au potentiel »

Coûts	D_1	D_2	D_3	D_4	D_5	Pot.
O_1			1	5	9	0
O_2	15			6	14	9
O_3	8	16			7	11
O_4	18	8	17			12
Pot.	7	12	21	23	28	

δ	D_1	D_2	D_3	D_4	D_5	Pot.
O_1			-20	-18	-19	0
O_2	17			-8	-5	9
O_3	12	15			-10	11
O_4	23	8	8			12
Pot.	7	12	21	23	28	

On va pouvoir en utilisant les potentiels déterminer la variation de coût.
Pour chaque case nulle, on calculera d en ajoutant au coût unitaire de la case le potentiel de l'origine associée et en retranchant le potentiel de la destination correspondante :

Répéter cette opération pour toutes cellules inoccupées.

2.4 identifier la variation de coût négative la plus forte
On va pouvoir identifier la case o1-d3, soit -20, pour laquelle la variation de coût est la plus forte.

2.5 Identifier le nombre de marchandises déplaçables sur cette case en suivant un chemin de case

	D_1	D_2	D_3	D_4	D_5	Offre
O_1	10	2				12
O_2		9	2			11
O_3			13	1		14
O_4				4	4	8
Demande	10	11	15	5	4	

→

	D_1	D_2	D_3	D_4	D_5	Offre
O_1	10		2			12
O_2		11				11
O_3			13	1		14
O_4				4	4	8
Demande	10	11	15	5	4	

On définit une nouvelle répartition grâce à ce déplacement.

Etape 3 : Cycle 2 : on refait plusieurs fois l'algorithme Cycle2 pour trouver un autre déplacement intéressant.
Dans l'exemple ci-dessus, il faudra 5 étapes pour atteindre cet optimum :

	D_1	D_2	D_3	D_4	D_5
O_1			12		
O_2		11			
O_3	10		3	1	
O_4				4	4

	D_1	D_2	D_3	D_4	D_5
O_1			12		
O_2		11			
O_3	10		3		1
O_4				5	3

	D_1	D_2	D_3	D_4	D_5
O_1			12		
O_2		11			
O_3	10				4
O_4			3	5	

Etape 4 : la condition de fin : Si toutes les variations de coût sont positives, arrêter car la solution optimale est trouvée.

note : il peut y avoir plusieurs solutions optimales.

Les autres méthodes :

Il existe bien d'autres méthodes pour l'affectation initiale .
La méthode du moindre coût,
La méthode du minima de colonne, ou de ligne,
Des approches heuristiques basés sur les coûts totaux et sur les pénalités.

Une méthode de résolution optimale : l'approche MODI

On peut noter également L'**algorithme d'Edmonds pour les couplages** est un algorithme permettant de déterminer un couplage maximum dans un graphe. L'algorithme **des fleurs et des pétales** et ***Blossom algorithm***. Je vous invite à aller sur le site https://cbom.atozmath.com/ qui présente un grand nombre de méthodes autour de l'affectation. Voici deux méthodes VOGT et RUSSEL très proche de BALAS HAMMER

L'approximation de
Heinrich Vogt 1890 –1968, astronome,

L'approximation de Henry Norris Russel 1877 - 1957,
considéré comme *Dean of American astronomers*.

Ils ont tous deux travaillés sur des analyses de masses dans l'espace, probablement dans la même logique que Monge avec son analyse des mouvements de motte de terre !

PROGRAMME INFORMATIQUE

Choix du langage :

Notre choix s'est orienté vers Java car c'est le langage que nous avons le plus étudié en cours, de plus il répond parfaitement aux attentes permettant de développer l'application.
Le langage C ou python, parait plus adapté, essentiellement pour sa vitesse d'exécution mais cela peut concerner des algorithmes de transport beaucoup plus complexes. La programmation orienté objet eut été également intéressante dans notre projet.

Difficultés :

Difficulté n°1 : les imbrications (boucles) et tests. Il a été souvent difficile de s'y retrouver quand il y a des contrôles imbriqués. Exemple : pour le calcul de la différence des 2 plus petits chiffres d'une ligne ou colonne, à la 1ère itération tout va bien, mais à la 2eme l'algorithme ne fonctionne plus suite à une ligne ou colonne remise à zéro.

Difficulté n°2 : Manque de connaissance des différents langages, qui a restreint les solutions techniques. Peu de maitrise de l'objet ou des différentes façons de gérer des matrices.

Difficulté n° 3: Pour Michel, de bien comprendre la mécanique des algorithmes de transport. Il a y peu de documentations pour un néophyte (obligé de « picorer » sur de nombreux sites pour comprendre).

Difficulté n°4 : D'avoir un diagramme d'état assez fin pour ne pas se perdre dans les différentes boucles et tests.

Difficulté n°5 : De trouver la méthode la plus propre afin d'éviter au maximum l'imbrication qui génère une plus grande probabilité d'erreurs.

Surmonter les difficultés :

1- L'utilisation du déboggeur a été indispensable pour situer les endroits générateurs d'erreurs.
Par exemple, nous avons utilisé un tableau à une dimension pour stocker les valeurs d'une ligne, les trier et récupérer les 2 plus petites valeurs. Facile ! il suffit de prendre la valeur de l'indice 0 et 1 du tableau.
Mais comment fait-on quand il y a des 0 dans le tableau ?

Si on prend les indices 0 et 1, le calcul est faux

0	0	2	5	12

Ici, on ne peut pas faire la différence entre les 2 plus petits chiffres car il n'y en a qu'un seul

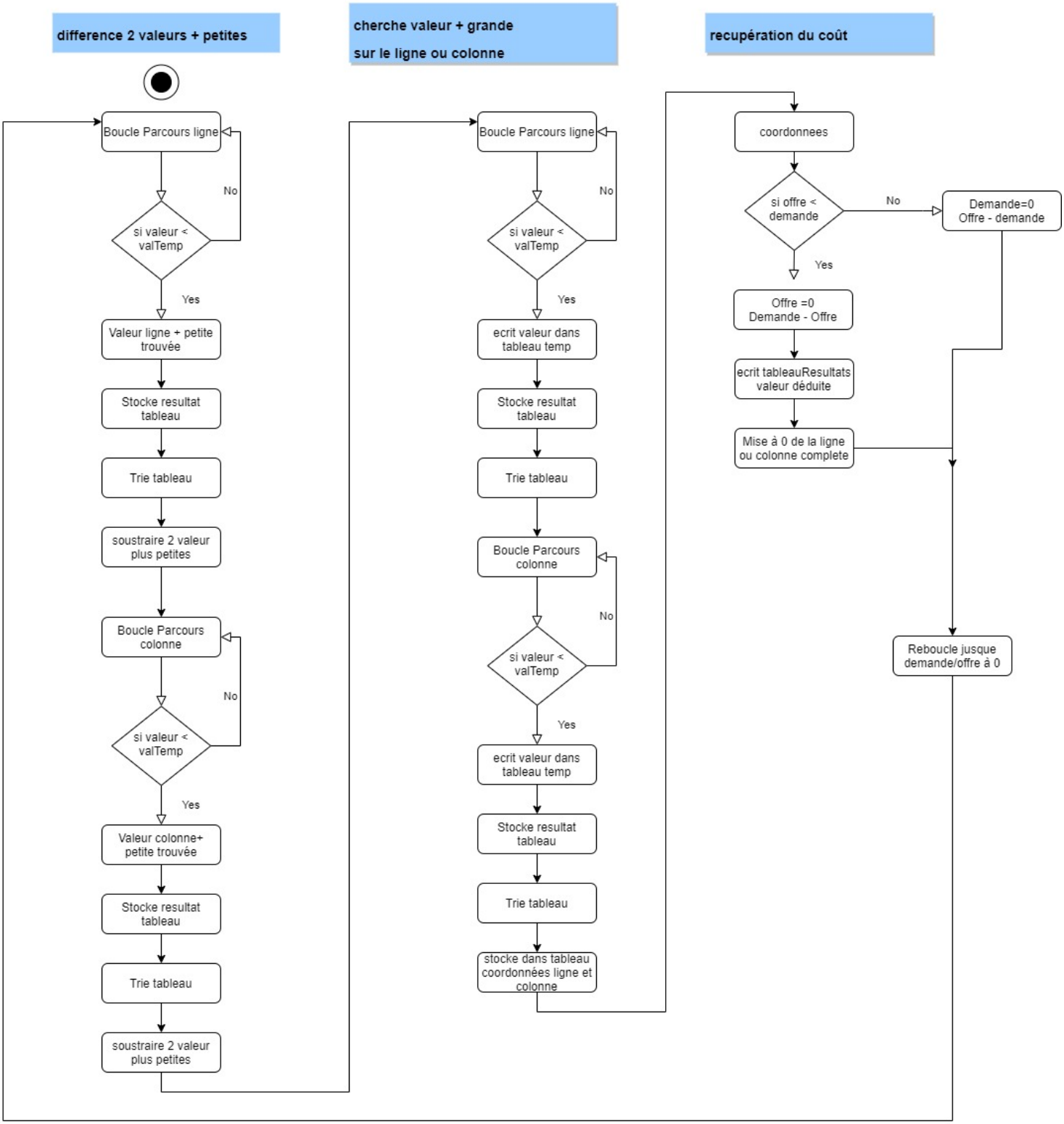
0	0	0	0	12

La solution a été de faire une boucle while qui boucle tant que la valeur est égale à 0 et que l'on ne dépasse pas la taille du tableau.
Ensuite, si le programme détecte qu'il n'y a qu'une seule valeur, il retourne la dernière valeur du tableau, sinon, il retourne la différence entre les deux plus petites valeurs après les 0.

```
int indice=0;
while (plusPetit[indice]==0 && indice<plusPetit.length-1){
        indice++;
}
if (indice==(plusPetit.length-1)){
        return plusPetit[indice];
}
else {
        return plusPetit[indice+1]- plusPetit[indice];    // on retourne la différence entre les 2 plus petites valeurs
}
```

2- Le manque de pratique du codage rend difficile l'appréhension du déroulement du code. De plus, par ce manque, les méthodes pour traiter une problématique particulière sont trop restreintes. Ici, la méthode utilisée (trouver les valeurs les plus petites, trouver des coordonnées dans un tableau) sont assez laborieuses. Il y a certainement des méthodes plus performantes.

Diagramme d'état



Explication du code Méthode du Coin Nord Ouest

Le code est basé sur une simple fonction pivot xStep et yStep,

```
public static int[][] cycleMCNO(int[][] tab){
	System.out.println("test");
	boolean sortie = false;
	int xStep = 1;
	int yStep = 1;
	int x = tab[0].length;
	int y = tab.length;

	while (!sortie) {
// On compare la valeur OrigineY et DestinationX
	if ( tab[yStep][0] > tab[0][xStep] ){
// si Y > X, on affecte une valeur Y-X à la case
		tab[yStep][xStep]= tab[0][xStep];
// on décremente les valeurs origines et destinations
		tab[0][xStep] -= tab[yStep][xStep];
		tab[yStep][0] -= tab[yStep][xStep];
// on s'oriente vers un nouveau xStep + 1, vers la droite
		xStep ++;                                  }
	else if ( tab[yStep][0] < tab[0][xStep] ){
// si X > Y, on affecte une valeur X-Y à la case
		tab[yStep][xStep]=  tab[yStep][0];
// on décremente les valeurs
		tab[0][xStep] -= tab[yStep][xStep];
		tab[yStep][0] -= tab[yStep][xStep];
// on s'oriente vers un nouveau yStep + 1, vers le bas
		yStep ++;

	} else {
// Y=X, on affecte une valeur Y-X à la case
		tab[yStep][xStep]= tab[0][xStep];
// on décremente les valeurs
		tab[0][xStep] -= tab[yStep][xStep];
		tab[yStep][0] -= tab[yStep][xStep];
// on s'oriente vers la droite et vers le bas
		xStep ++;          yStep ++;
	}
// sortie de Cycle
		if ( ( xStep == x ) && ( yStep == y ) )  {
		sortie = true;            }
		methodeCNO.afficheTableau2d(tab);
	}	return tab;         }
```

Explication du code Méthode Balas-Hammer

Le programme commence par l'initialisation des variables et tableaux :

'tableauOrigine' : qui stocke les données suivantes :

Origines ; Destinations ; Différentiels (*différence entre les deux plus petites valeurs d'une ligne ou d'une colonne*)

'coordonnee' : qui permet de connaitre la position du coût le plus faible.
'tableauResultat' : stocke les résultats des mouvements soit des origines, soit des demandes

```
// TODO Auto-generated method stub
        int nbLigneOrigin=7, nbColOrigin=8;  // Nombre de ligne et de colonnes du tableau des demandes et Stock

        int calculCout =0;                   // Variable qui stocke le calcul de coût

                                             // initialisation du tableau à 2 dimension
                                             // on utilise las lignes pour les destinations et le colonnes pour les Stocks
        int [][] tableauOrigine = new int [nbLigneOrigin][nbColOrigin];
                                             // tableau qui sert à stocker les coordonnée d'une case
                                             // qui permettra de chercher la ligne ou la colonne avec le coût le plus faible
        String [] coordonnee =new String[2];

                                             // tableau qui sert à stocker les coordonnée du resultat final de la methode Ballas hammer
        int [][] tableauResultat = new int [nbLigneOrigin-2][nbColOrigin-2];


        int OriginX=1,OriginY=3;             // coordonnées des origines (Stocks)
        int DestX=3,DestY=1;                 // coordonnées des destinations (Magasins)

        // remplissage des demandes
        tableauOrigine[1][3]=10;
        tableauOrigine[1][4]=11;
        tableauOrigine[1][5]=15;
        tableauOrigine[1][6]=5;
        tableauOrigine[1][7]=4;
        // remplissage des Stocks
        tableauOrigine[3][1]=12;
        tableauOrigine[4][1]=11;
        ……
        ……..
```

```
***************      TABLEAU PRINCIPALE      ***************************************
    ,      ,      ,      ,      ,      ,      ,      ,
    ,      ,      ,   10  ,  11  ,  15  ,   5  ,   4  ,        Demandes
    ,      ,      ,   -1  -  -5  -  -9  -  -1  -  -2  -        ≠ entre 2 valeurs les + petites
    ,   12 ,   -4-     7  ,  12  ,   1  ,   5  ,   9  ,
    ,   11 ,   -3-    15  ,   3  ,  12  ,   6  ,  14  ,        Coûts
    ,   14 ,   -1-     8  ,  16  ,  10  ,  12  ,   7  ,
    ,    8 ,   -3-    18  ,   8  ,  17  ,  11  ,  16  ,


***************      TABLEAU RESULTAT      ***************************************

 0  ,   0  ,  12  ,   0  ,   0  ,
 0  ,   0  ,   0  ,   0  ,   0  ,
 0  ,   0  ,   0  ,   0  ,   0  ,
       0  ,   0  ,   0  ,   0  ,   0  ,
```

Objectif : trouver le delta entre les deux plus petites valeurs en ligne ou en colonne.

Pour cela, 2 fonctions permettent de faire cette recherche :

calculMiniLigne : parcours chaque ligne du tableau en stockant ces valeur dans le tableau provisoire 'plusPetit' . Ensuite on tri ce tableau et on récupère les 2 plus petites valeurs (ou LA plus petite s'il n'y en a qu'une) qui sera retournée.

calculMiniColonne : idem que 'calculMiniLigne' mais pour les colonnes.

Le chiffre ainsi obtenu sera stocké dans le tableau principale (tableauOrigine).

```
        // Boucle qui va recupérer pour chaque ligne et fait la diffférence entre les 2 valeurs les plus petites
        for (int i=3; i<nbColOrigin-1;i++) {
                int differenceLigne = calculMiniLigne(tableauOrigine, nbColOrigin-3, i);          // appel de la fonction qui trouve
la différence entre les 2 plus petites valeurs
                tableauOrigine[i][2]=differenceLigne;
                        // on stocke dans le tableau cette valeur
        }

                // Boucle qui va recupérer pour chaque colonne et fait la diffférence entre les 2 valeurs les plus petites
        for (int p=3; p<nbLigneOrigin+1;p++) {
                int differenceCol = calculMiniColonne(tableauOrigine,p,nbLigneOrigin-3);          // appel de la fonction qui trouve
la différence entre les 2 plus petites valeurs
                tableauOrigine[2][p]=differenceCol;
                // on stocke dans le tableau cette valeur
        }
```

```
*
 *      Méthode calculMiniLigne
 * On créer un tableau temporaire 'plusPetit' qui permet de récupérer une ligne complete
 * du tableau principal 'tableauOrigine'. Ensuite on fait un trie du tableau 'plusPetit'
 * et on soustrait les 2 plus petites valeurs.
 */
public static  int  calculMiniLigne(int [][]tableauOrigine , int colonne, int ligne) {

        int [] plusPetit = new int [colonne];                           // initialisation du tableau provisoire
        for (int i=0; i<colonne;i++) {
                if (tableauOrigine[ligne][i+3]!=0) {
                        plusPetit[i]=tableauOrigine[ligne][i+3];        // on ajoute les valeurs d'une ligne du tableau dans
'plusPetit'
                }
        }
        Arrays.sort(plusPetit);                                         // on trie le tableau 'plusPetit'

        /*
         * Ici on fait une boucle while pour trouver la premiere valeur non nul du tableau 'plusPetit' qui a été trier
         * donc on recupère les 2 valeur les plus petites. Il suffit ensuite de les soustraires.
         * dans le cas où il ne reste plus qu'une seule valeur on gardera uniquement cette valeur.
         */
        int indice=0;                                                   // variable qui permet de se positionner dans la recherche des
valeurs les + faible dans
                                                                        // le tableau 'plusPetit'

        /*
         *   Le but est de trouver la ou les 2 valeurs les plus petites.
         *   Soit il n'y a qu'une seule valeur on la retourne tel quel, soit il y a 2 valeurs
         *   et dans ce cas on les soustraits avant de les retournées
         */
        while (plusPetit[indice]==0 && indice<plusPetit.length-1){
                indice++;
        }
        if (indice==(plusPetit.length-1)){
                return plusPetit[indice];
        }
        else {
                return plusPetit[indice+1]- plusPetit[indice];          // on retourne la différence entre les 2 plus petites valeurs
        }

}
```

```java
/*
 *       Méthode calculMiniColonne
 * On créer un tableau temporaire 'plusPetit' qui permet de récupérer une colonne complete
 * du tableau principal 'tableauOrigine'. Ensuite on fait un trie du tableau 'plusPetit'
 * et on soustrait les 2 plus petites valeurs.
 */
public static  int  calculMiniColonne(int [][]tableauOrigine , int colonne, int ligne) {

        int [] plusPetit = new int [ligne];                                     // initialisation du tableau provisoire
        for (int i=0; i<ligne;i++) {
                if (tableauOrigine[i+3][colonne]!=0) {
                        plusPetit[i]=tableauOrigine[i+3][colonne];              // on ajoute les valeurs d'une colonne du tableau dans
'plusPetit'
                }
        }
        Arrays.sort(plusPetit);

        /*
         * Ici on fait une boucle while pour trouver la premiere valeur non nul du tableau 'plusPetit' qui a été trier
         * donc on recupère les 2 valeur les plus petites. Il suffit ensuite de les soustraires.
         * dans le cas où il ne reste plus qu'une seule valeur on gardera uniquement cette valeur.
         */
        int indice=0;
        while (plusPetit[indice]==0 && indice<plusPetit.length-1){
                indice++;
        }
        if (indice==(plusPetit.length-1)){
                return plusPetit[indice];
        }
        else {
                return plusPetit[indice+1]- plusPetit[indice];                                  // on retourne la
différence entre les 2 plus petites valeurs
        }

}
```

Objectif : Trouver la valeur la plus grande en ligne ou colonne

L'étape suivante consiste à trouver la valeur la plus grande calculée précédemment, soit en colonne, soit en ligne.
Pour cela on parcours la ligne 2 et la colonne 2 du tableau 'tableauOrigine', puis on compare la plus grande valeur de la ligne avec celle de la colonne.
On retourne la plus grande valeur avec la mention « C » si c'est sur une colonne et « L » si c'est une ligne.

```java
        // appel de la fonction qui va retourner la valeur la plus grande
        coordonnee = valeurPlusGrande(tableauOrigine,nbColOrigin-3,nbLigneOrigin-3 );
```

```java
/*
 * Ici on recherche la valeur la plus grande parmis les valeurs calculer dans 'calculMiniColonne' et 'calculMiniLigne'
 */
public static String[] valeurPlusGrande (int [][]tableauOrigine, int colonne, int ligne) {

        String [] point = new String [2];                                               // initialisation du tableau des coordonnées
        int [] plusGrandCol = new int [colonne];                                // initialisation du tableau provisoire
        for (int i=0; i<colonne;i++) {
                plusGrandCol[i]=tableauOrigine[2][i+3];                                 // on ajoute les valeurs d'une colonne du
tableau dans 'plusGrandCol'

        }
        Arrays.sort(plusGrandCol);                                                                      // on trie le tableau 'plusGrandCol'

        int [] plusGrandLigne = new int [ligne];                                // initialisation du tableau provisoire
        for (int z=0; z<ligne;z++) {
                plusGrandLigne[z]=tableauOrigine[z+3][2];                       // on ajoute les valeurs d'une colonne du tableau dans
'plusGrand'

        }
        Arrays.sort(plusGrandLigne);                                                                    // on trie le tableau 'plusGrandLigne'


        /*
         * On compare la valeur la plus grande de la ligne et de la colonne
         * et on renvoi la valeur la plus grande des 2 plus l'indication "C" pour colonne et "l" pour ligne
         */
        if (plusGrandLigne[ligne-1]< plusGrandCol[colonne-1]) {
                point[0]=String.valueOf(plusGrandCol[ligne]);
                point[1]="C";
                return point;
        }
        else {
                point[0]=String.valueOf(plusGrandLigne[colonne-2]);
                point[1]="L";
                return point;
        }
}
```

Objectif : Trouver le coût le plus faible

Ici, le but est de trouver le coût le plus faible dans le tableau 'pointCoutPlusFaible '.
Pour cela, on parcours la ligne ou la colonne trouvée précédemment avec la méthode 'valeurPlusGrande '
Lorsque l'on a trouvé cette valeur la plus petite qui correspond au coût le plus faible, le programme retourne les coordonnées (ligne, colonne) où se trouve cette valeur dans le tableau 'coordonnee' .

```
// appel de la fonction qui va retourner les coordonnees de la valeur la + petite
        // soit sur la ligne soit sur la colonne
        int coord = Integer.parseInt(coordonnee[0]);                    // Valeur la plus grande
        coordonnee= pointCoutPlusFaible(tableauOrigine, coordonnee,coord, nbColOrigin-3,nbLigneOrigin-3);
```

```
/*
 * pointCoutPlusFaible : Cette méthode recherche le coût le plus faible
 * soit sur la ligne soit sur la colonne
 */

public static String[] pointCoutPlusFaible(int [][]tableauOrigine, String []coordonnee, int plusGrand,  int nbColonne, int nbLigne) {
        String [] point = new String [2];
        int ligneTemp=9999999;                                  // Valeur temporaire de la position de la ligne. 999999 pour etre sûr
que cette valeur soit la plus grande
        int posLigne=0;                                         // Stocke la positiopn de la ligne de la valeur trouvée
        int posCol=0;                                                                   // Stocke la positiopn de la colonne de
la valeur trouvée
        if (coordonnee[1] =="C") {                              //Si la valeur est trouvée dans une colonne
                for (int i=3;i<nbColonne+3;i++) {                       // boucle de parcours de la colonne
                        if (tableauOrigine[2][i]==plusGrand) {          // si on trouve la leur la plus grande
                                posCol=i;                               // on stocke la valeur trouvée dans posCol
                                for (int t=3;t<nbLigne+3;t++) {         // boucle pour poarcourir les lignes de la colonne

                                        if (tableauOrigine[t][i] <ligneTemp && tableauOrigine[t][i]!=0) {          // verifie si
on trouve une valeur plus petite mais differente de 0
                                                posLigne=t;
                                                 // si trouvé on stocke la valeur dans posLigne
                                                ligneTemp =tableauOrigine[t][i];
                        // et on affecte la valeur trouvée dans ligneTemp
                                        }
                                }
                                point[0]=String.valueOf(posLigne);   // on stocke le resultat de la ligne dans point[0]
                                point[1]=String.valueOf(posCol);     // on stocke le resultat de la colonne dans point[0]

                                }
                        }
                }
        else {
                for (int i=3;i<nbLigne+3;i++) {
                        if (tableauOrigine[i][2]==plusGrand) {
                                posLigne=i;
                                for (int t=3;t<nbColonne+3;t++) {
                                        if (tableauOrigine[i][t]<ligneTemp && tableauOrigine[i][t]!=0) {
                                                posCol=t;
                                                ligneTemp =tableauOrigine[i][t];
                                        }
                                }
                                point[0]=String.valueOf(posLigne);
                                point[1]=String.valueOf(posCol);

                        }
                }
        }

        return point;
}
```

Objectif : construite le tableau d'affectation d'optimisation de déplacement

La méthode 'affectation' récupère les coordonnées précédemment trouvées dans la méthode 'pointCoutPlusFaible' et l'affecte dans le tableau 'tableauResultat'

```
                // On stocke la quantite soit de la destination soit de l'origine dans le tableau 'tableauResultat'
                affectation(tableauOrigine, coordonnee,tableauResultat, nbColOrigin-3,nbLigneOrigin-3);
```

```
public static void affectation(int [][]tableauOrigine, String []coordonnee, int [][]tableauResultat, int nbColonne, int nbLigne) {

        int ligne = Integer.parseInt(coordonnee[0]);                    // on recupere la coordonnée ligne
        int colonne = Integer.parseInt(coordonnee[1]);                  // on recupere la coordonnée colonne
        if (tableauOrigine[ligne][1]<tableauOrigine[1][colonne]) {      // si l'offre est inférieure à la demande
                tableauResultat[ligne-3][colonne-3]=tableauOrigine[ligne][1];                   // on stocke
                tableauOrigine[1][colonne]=tableauOrigine[1][colonne]-tableauOrigine[ligne][1];    // on soustrait l'offre à la demande
                for (int x=0;x<nbColonne+3;x++) {

                        tableauOrigine[ligne][x]=0;
                                // l'offre de la ligne etant à 0 on supprime la ligne en mettant tout à 0
                }
        }
        else {
                tableauResultat[ligne-3][colonne-3]=tableauOrigine[ligne][1];
                tableauOrigine[ligne][1]=tableauOrigine[ligne][1]-tableauOrigine[1][colonne];
                for (int x=0;x<nbLigne+3;x++) {
                        tableauOrigine[x][colonne]=0;
                }
        }
}
```

Explication du code Méthode Stepping Stone

Dans Balas Hammer, on a ajouté une ligne et une colonne reliquat de quantités.
Dans Stepping Stone, il va falloir ajouter une ligne L+1 et une colonne C+1 pour le CheminPotentiel.
puis on crée une fonction CheminDePotentiel qui affecte à la colonne c, l+1 une valeur de départ théorique tèl qu'on obtient toujours des valeurs de potentiels positifs.

Un modèle avec un tableau simplifié 3x3

Public static int[][] cheminDePotentiel(int[][] tabPotentiel, init[][] tab){

int valinit = 100 ; xStep=1; yStep=1 ;

while (!sortie) {
// On part de la valeur du bas
tabPotentiel[yStep][3] = valinit;
tabPotentiel [3][xStep] = valinit - tab[yStep][xStep]
FunctionEvolutionDeXstepEtYStep();
// sortie de Cycle
if ((xStep == 4) && (yStep == 4)) {
sortie = true; }
methodeCNO.afficheTableau2d(tab);
} return tab; }
}

On crée un deuxième tableau coût tabPotentiel qui va calculer chaque case avec la notion de potentiel.

Dans ce tableau, on tague les cases vides.
On calcule la case vide qui obtient le potentiel de réduction de coût le plus fort.
On applique une fonction qui permet de savoir quelle quantité sera déplaçable.
On conserve le principe d'un xStep et d'un yStep qui permet d'analyser une cellule suivante vers la droite ou vers le bas.

Perspective et v2.0

Etudier la meilleure technique de gestions des matrices.

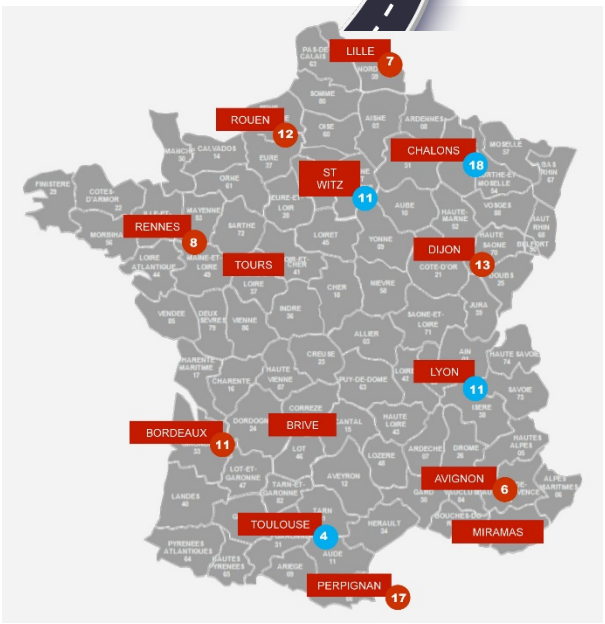
Utilisation de la POO

Créer une interface utilisateur (graphique) afin de faciliter l'utilisation. On peut imaginer une carte de France où l'on pourrait visualiser les endroits des demandes (ex : magasins) et les origines (ex : entrepôts)

	Chalons	St Witz	Lyon	Toulouse
Qte demandée	12	11	18	22

Stock Entrepôt	
Lille	50
Rouen	32
Rennes	28
Bordeaux	66
Perpignan	42
Avignon	7
Dijon	12

Répartition			
5			8
	3		
3	6	4	9
4		3	
		2	4
	2		1
		9	



L'utilisateur pourrait saisir directement les demandes (ronds bleus) et les origines (ronds rouges) sur la carte de France et avoir l'affichage de la répartition sous forme d'un tableau comme la figure ci-dessus.

On pourrait également configurer différentes possibilités, comme la gestion de stockage d'un entrepôt.

Conclusion

Comme indiqué dans la présentation initiale de l'historique, la méthode a connu à la fois de multiples facettes car elle a été créée pour de multiples applications par de nombreuses personnes différentes :
« Comment déplacer de la terre, comment organiser un mouvement d'armée, comment organiser un transport, jusqu'à comment analyser le mouvement des masses de matières en astronomie. »
Le XXème siècle a vraiment pu structurer au sein de la théorie des graphes l'approche de l'affectation en permettant d'en faire une base de travail pour d'autres algorithmes.

La question est posée de savoir si on peut trouver une correspondance avec les autres algorithmes.
Le voyageur de commerce ou le train soviétique pourrait maintenant être analysés ou organisés comme un assemblage de graph bipartite dans lequel on peut opérer des cycles par ensemble de points (villes ou gares).

C'est ce qu'ont démontré les mathématiciens progressivement, pas à pas avec petite expérience pratique de terrain qui a apporté son lot d'idées pour moderniser l'algorithme.
Que nous réserve la suite ? Les grecques nous parlaient du Logistikon (origine du mot logistique) comme la prévision du 'raisonnable'. (par rapport à epistemologikon, étude de l'existant)
les algorithmes de logistique appliqués à l'intelligence artificielle pourra être peut-être un prochain sujet d'étude ?

Remerciements

Merci à Daniel Lambillotte Alexandre Schrijver, Frédéric Havet, et Marc Beveraggi qui par leurs ouvrages nous ont aidés à avancer vite dans la compréhension des problématiques :

Daniel Lambillotte « la fonction logistique dans l'entreprise » de Daniel Lambillotte »
Alexandre Schrijver « Histoire des mathématiques combinatoires »
Frédéric Havet « Traduction de la Théorie des graph de J.A. Bondy et U.S.R. Murty »
Marc Beveraggi « http://www.beveraggi.fr/ »

Notre excellent professeur de Théorie des graphes au Cnam de Lille – Nicolas Mesureur.

Nos anciens collègues Christophe DAUTEL – motoBlouz, Martin Querleu – Effisys, Christophe et Ludo Fandi - Fandi Emballage, Jean-Christophe Koral DIALOG, Benoit Duhamel – SupplyGroup, Marc Tierny - Daforib, Laurent Desprez – Euralogistic, et bien d'autres.