Sommaire

LISTE DES FIGURES

Partie 1: Présentation

I. Introduction

Toute les entreprises qu'elle que soit sa taille, son domaine d'activité est amenée à faire face à des problèmes de gestion au quotidien.

Parmi ces problèmes, on cite les problèmes d'affectation et de transport qui nécessitent la mise en œuvre d'un procédé de prise de décision rationnel, à cause de leur niveau de complexité particulièrement élevé et à cause des coûts supplémentaires qu'ils génèrent s'ils sont mal gérés.

Ce qui souligne l'importance qu'occupe ce type de problème dans la gestion quotidienne de l'entreprise.

Dans cette mémoire on va donner quelque méthode théorique et pratique pour la résolution de ces deux problèmes.

II. Objectif

Ce projet a pour but de crée une application permettant de résoudre le problème de transport et le problème d'affectation. L'utilisateur qui va bénéficier de ce service disponible dans notre application va pouvoir trouver la solution optimale de ces deux problèmes.

Donc notre premier objectif et de trouver la quantité transporté de chaque origines (ou usine) vers les destinations (ou clients), de telle manière que le coût totale de transport soit minimale.

Notre deuxième objectif et d'affecter un ensemble de personne à un ensemble des tâches, telle que le coût total d'affectation soit optimal (minimal ou maximal).

III. Les Technologies utilisées

- **Eclipse**

Eclipse est un projet de la fondation Eclipse visant à développer tout un environnement de développement libre, extensible, universel et polyvalent.

Son objectif est de produit et fournir divers outils gravitant autour de la réalisation de logiciel, englobant les activités de codage logiciel proprement dites (avec notamment un environnement de développement intégré) mais aussi de modélisation, de conception, de test, etc. Son environnement de développement notamment vise à la généricité pour lui permettre de supporter n'importe quel langage de programmation.

- **NetBeans**

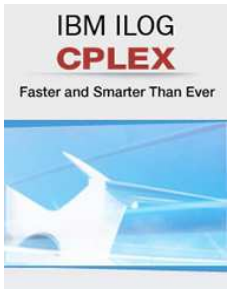
NetBeans est un environnement de développement intégré (EDI), placé. NetBeans permet également de supporter différents autres langages, comme Python, C, C++, JavaScript, XML, Ruby, PHP et HTML. Il comprend

toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web).

Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X ou sous une version indépendante des systèmes d'exploitation (requérant une machine virtuelle Java). Un environnement Java Development Kit JDK est requis pour les développements en Java.

NetBeans constitue par ailleurs une plate forme qui permet le développement d'applications spécifiques (bibliothèque Swing (Java)). L'IDE NetBeans s'appuie sur cette plate forme.

- **Cplex :**



ILOG CPLEX (plus communément appelé "CPLEX") est un outil informatique d'optimisation. Son nom fait référence au langage C et à l'algorithme du simplexe.

Il est composé d'un exécutable (CPLEX Interactif) et d'une bibliothèque de fonctions pouvant s'interfacer avec différents langages de programmation (CPLEX Callable Library) : C, C++, C#, Java et Python.

IV. Langage utilisé

Java est à la fois un langage de programmation informatique orienté objet et un environnement d'exécution informatique portable créé par James Gosling et Patrick Naughton employés de Sun Microsystems avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld.

Java est à la fois un langage de programmation et un environnement d'exécution. Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels qu'Unix, Microsoft Windows, Mac OS ou Linux avec peu ou pas de modifications... C'est la plate-forme qui garantit la portabilité des applications développées en Java.

Le langage reprend en grande partie la syntaxe du langage C++, très utilisé par les informaticiens. Néanmoins, Java a été épurée des concepts les plus subtils du C++ et à la fois les plus déroutants, tels que l'héritage multiple remplacé par l'implémentation des interfaces. Les concepteurs ont privilégié l'approche orientée objet de sorte qu'en Java, tout est objet à l'exception des types primitifs (nombres entiers, nombres à virgule flottante, etc.).

Java permet de développer des applications autonomes mais aussi, et surtout, des applications client-serveur. Côté client, les applets sont à l'origine de la notoriété du langage. C'est surtout côté serveur que Java s'est imposé dans le milieu de l'entreprise grâce aux servlets, le pendant serveur des applets, et plus récemment les JSP (Java Server Pages) qui peuvent se substituer à PHP, ASP et ASP.NET.

Les applications Java peuvent être exécutées sur tous les systèmes d'exploitation pour lesquels a été développée une plate-forme Java, dont le nom technique est JRE (Java Runtime Environment - Environnement d'exécution Java). Cette dernière est constituée d'une JVM (Java Virtual Machine - Machine Virtuelle Java), le programme qui interprète le code Java et le convertit en code natif. Mais le JRE est surtout constitué d'une bibliothèque standard à partir de laquelle doivent être développés tous les programmes en Java. C'est la garantie de portabilité qui a fait la réussite de Java dans les architectures client-serveur en facilitant la migration entre serveurs, très difficile pour les gros systèmes.

Partie 2: Problème de Transport et Affectation

Chapitre 1 : Notions de Base

I. Introduction

Dans ce chapitre on va parler de quelles notions de base dans la théorie des graphes qu'on va utiliser par la suite soit dans la résolution du problème de transport ou bien on traitant le problème d'affectation, et des points générale sur l'optimisation combinatoire et la programmation linéaire.

II. L'optimisation combinatoire

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire. Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données.

Un problème **d'optimisation combinatoire** est défini par un ensemble d'instances. A chaque instance du problème est associé un ensemble discret de solutions S, un sous-ensemble X de S représentant les solutions admissibles (réalisables) et une fonction de coût f (ou fonction objectif) qui assigne à chaque solution $s \in X$ le nombre réel (ou entier) $f(s)$. Résoudre un tel problème (plus précisément une telle instance du problème) consiste à trouver une solution $s^* \in X$ optimisant la valeur de la fonction de coût f. Une telle solution s^* s'appelle une *solution optimale* ou un *optimum global*. Nous avons donc la définition suivante :

Définition 1-II-1:
Une *instance* I d'un problème de minimisation est un couple (X, f) où $X \subseteq S$ est un ensemble fini de solutions admissibles, et f une fonction de coût (ou objectif) à minimiser(ou maximisé) $f : X \rightarrow R$. Le problème est de trouver $s^* \in X$ tel que $f(s^*) = f(s)$ pour tout élément $s \in X$.

Notons que d'une manière similaire, on peut également définir les problèmes de maximisation en remplaçant simplement $\leq$ par $\geq$. L'optimisation combinatoire trouve des applications dans des domaines aussi variés que la gestion, l'ingénierie, la conception, la production, les télécommunications, les transports, l'énergie, les sciences sociales et l'informatique elle-même.

III. Théorie des Graphes

1. Un Graphe

Un graphe [1] est un ensemble de sommets, dont certaines paires sont directement reliées par un (ou plusieurs) lien(s)appelé arc (ou arête).

L'ensemble de sommet noté on générale par X, tandis que E désigne l'ensemble des arêtes. Dans le cas général, un graphe peut avoir des *arêtes multiples*, c'est-à-dire que plusieurs arêtes différentes relient la même paire de points.

Pour définir un graphe général, il faut une fonction d'incidence γ qui associe à chaque arête une paire de sommets (ou un couple en cas orienté). Ainsi, un graphe est un triplet $G=(X, E, \gamma)$ avec

$\gamma : E \rightarrow X \times X$.

Toutefois l'usage veut que l'on note simplement, sachant que $G=(X, E)$ ce n'est parfaitement rigoureux que pour les graphes simples.

Vocabulaire :

- Un **graphe orienté** est un graphe dont les arêtes sont orientées (fléchées). On distingue alors le sommet **origine** de l'arête et son **extrémité**.
- Deux sommets reliés par au moins une arête sont dits **adjacents**.
- Une arête partant et arrivant au même sommet est appelée **boucle**.
- Une chaine est une suite d'arcs telle que chaque arc de la suite a une extrémité en commun avec l'arc suivant. La direction n'a pas d'importance.
- Un circuit est chemin fini est fermé dont l'extrémité finale du dernier arc coïncide avec l'extrémité initiale du premier arc.
- Un cycle est une chaine dont les sommets initial est final coïncide et qui n'emprunte pas le même arc constitue un cycle.
- Un **graphe** est dit **simple** s'il n'a pas de liens doubles ni de boucles.

2. Graphe bipartie

Un graphe est dit **biparti** [2] s'il existe une partition de son ensemble de sommets en deux sous-ensembles u et v telle que $u \cap v = \emptyset$. En d'autres termes, chaque arête ait une extrémité dans u et l'autre dans v. En note on générale un graphe bipartie $G = (U \cup V, E)$.

Exemple :

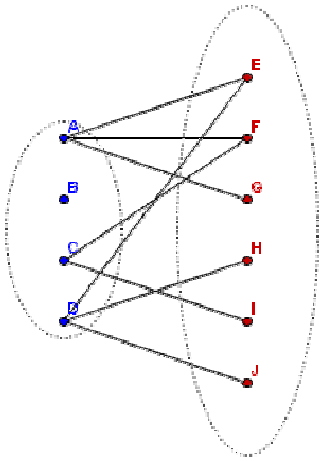


Figure 1 : la représentation d'un graphe biparti.

3. Couplage

Soit un graphe G=(X, E).
Un couplage M est un ensemble d'arêtes deux à deux non adjacentes.

$$\forall (a, a') \in M^2, \qquad alors a \neq a' \rightarrow a \cap a' = \emptyset$$

Vocabulaire :

- Un sommet x est saturé par un couplage M si x est l'extrémité d'une arête de M. Dans le cas contraire, x est insaturé.
- Le couplage M **sature** U, si U est un ensemble des sommets couplés par M.

Définition 1-III-1:

- Un couplage **maximal** est un couplage M ayant la propriété que si une arête e est ajoutée, alors M ∪ {e} n'en est pas un couplage.
- Un couplage **maximum** est un couplage contenant le plus grand nombre possible d'arêtes.

Exemple :

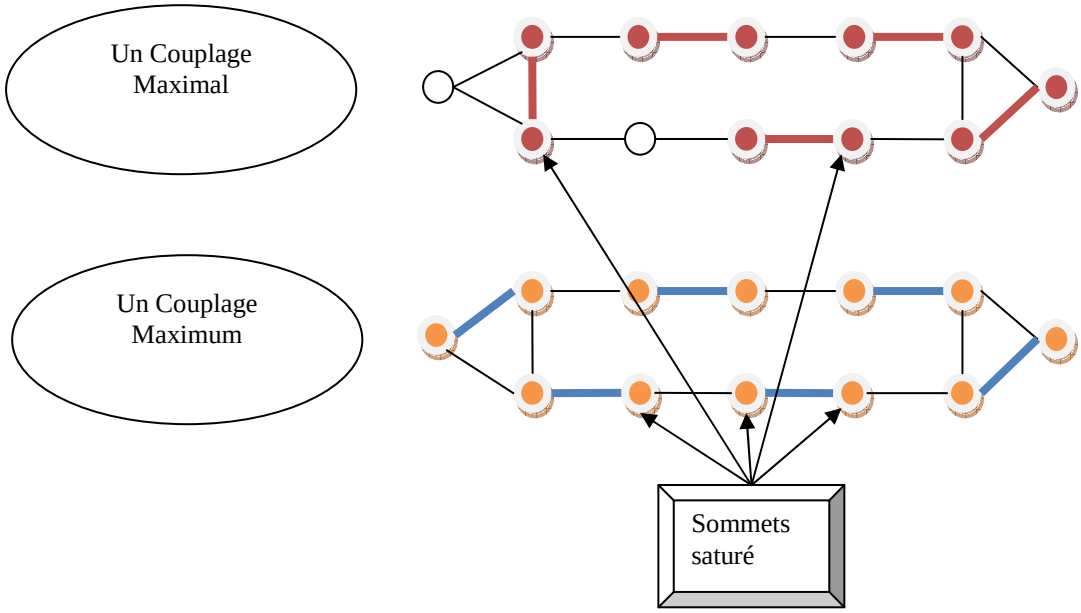


Figure 2 : Exemple d'un couplage maximal et maximum.

4. Couplage dans un graphe biparti

Soit G= ((A, B), E) un graphe biparti.

Définition 1-III-3:

Soit S un sous-ensemble de sommet on note par N(S) l'ensemble des sommets privé de S, voisins d'un sommet de S.

$$N(S) = \bigcup_{s \in S} N(S) \backslash S$$

Exemple :

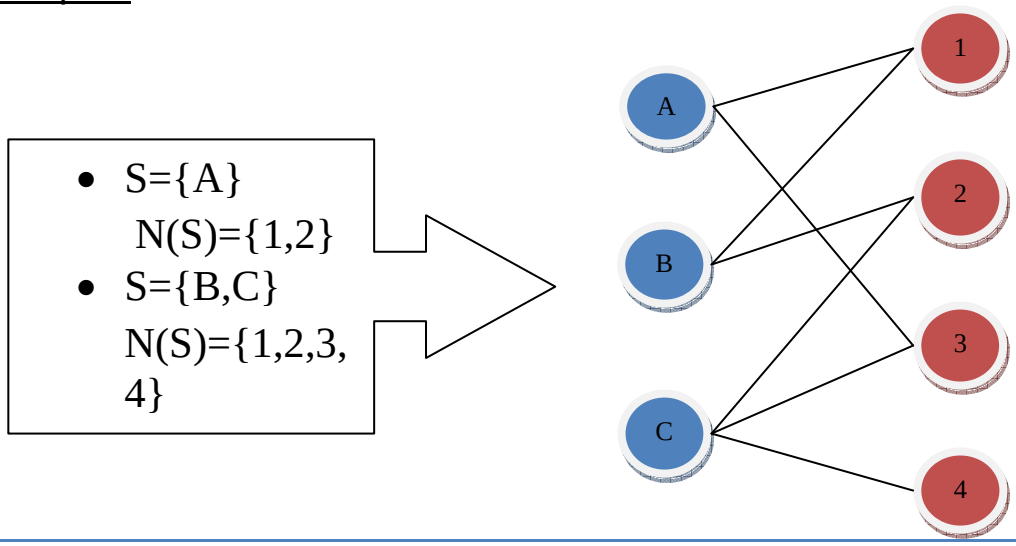


Figure 3: Exemple de voisinage d'un ensemble.

IV. Programmation linéaire :

Dans le contexte de la **programmation linéaire**, le terme programmation désigne l'organisation des calculs et non la réalisation d'un programme informatique. Du point de vue des applications, l'optimisation linéaire est d'une grande portée. Elle s'applique à des problèmes très variés qui sont issus de l'économie, de l'ingénierie, de la physique ou encore des modèles probabilistes. Dans ce cadre, on peut citer par exemple, les problèmes de type gestion de stock, gestion de production, transport de marchandise, affectation du personnel, systèmes industriels, réseaux de communication, etc. Pour les modèles de programmation linéaire, on est souvent amené à maximiser un gain ou minimiser un coût. Ceci explique d'ailleurs pourquoi la fonction à maximiser s'appelle fonction d'objectif ou économique.

Un programme linéaire est un problème dans lequel on est amené à maximiser (ou minimiser) une application linéaire, appelée fonction d'objectif ou fonction économique, sur un ensemble d'équations et/ou d'inéquations linéaires, dites contraintes. Autrement dit, la programmation linéaire est une branche des mathématiques qui a pour but de résoudre des problèmes d'optimisation linéaire de type :

$$\max(\, ou \,\min)\,[Z(x_1, \dots, x_p) = \sum_{j=1}^{p} c_j x_j]$$

$$\sum_{j=1}^{p} a_{ij}\, x_j \leq (et \backslash ou =) b_i, pour\ i = 1, \dots, m.$$

Les coefficients c_j, a_{ij} et b_i sont des réels fixés et les x_j sont des variables réelles. Les contraintes d'inégalités éventuelles sont toutes larges et non strictes. Il se peut qu'une contrainte d'inégalité soit de type " $\geq$ ". En multipliant chaque inégalité de type " $\geq$ " par (-1), on peut supposer que toutes les contraintes d'inégalité sont de type " $\leq$ ".

Définition 1- IV-1:

Pour un problème d'optimisation linéaire, tout point qui vérifie l'ensemble des contraintes s'appelle **point réalisable** ou admissible. L'ensemble de tous les points réalisables s'appelle **domaine réalisable**. Il est facile de vérifier que le domaine réalisable d'un programme linéaire est convexe. On rappelle qu'un ensemble est dit convexe si à chaque fois qu'il contient deux points *x* et *y*, il contient le segment joignant *x* et *y*. Ce segment est souvent noté par
[x, y] et il est défini par **:** $[x, y] = \{ \lambda.x + (1-\lambda).y / \lambda \in [0,1] \}$

Définition 1- IV-2:

Un point est dit **optimal** s'il est admissible et s'il réalise l'optimum de la fonction d'objectif sur le domaine réalisable.

Chapitre 2 : Problème de Transport

I. Présentation du problème:

C'est en 1941 que Frank Lauren Hitchcock [4] à formulé pour la première fois le problème de transport, et en suit par le mathématicien français Gaspard Monge en 1781[3]. D'importants développements ont été réalisés dans ce domaine pendant la Seconde Guerre mondiale par le mathématicien et économiste russe Leonid Kantorovich[5], ce problème consiste à minimiser le coût de transport total d'un plan d'expédition. Le fait de minimiser à la fois la distance totale et le coût de transport fait partie de la théorie des flux de réseaux. Le problème de transport « classique »est en fait un cas particulier d'un problème de flux de réseaux.

Le problème du transport est un problème linéaire que peut être représenté sous forme d'un graphe et qu'on peut le résoudre en utilisant les différentes méthodes de résolution des problèmes linéaire qu'on va présenter par la suite.

Un problème de transport peut être défini comme l'action de transporter des marchandises ou des produits fabriqué par m origines(ou usines) vers n destinations(ou clients), d'une manière que le coût total de transport soit minimale.

Donc, la résolution d'un problème de transport consiste à organiser le transport de façon à minimiser le coût total de transport.

II. Formulation du problème :

Avant de commencer la formulation du problème, considèrent la notation suivante :
a_i = la quantité disponible du produit à l'origine i.
b_j= la quantité requise à la destination j.
X_{ij} = quantité transportée des unités de l'origine i vers la destination j.
C_{ij}= le coût de transport d'une unité de l'origine i vers la destination j.

On peut d'écrire un problème de transport de la façon suivante. Une quantité donnée d'un produit uniforme est disponible à chacune des origines (par exemple des dépôts ou usines). Il s'agit d'envoyer des quantités spécifiées à chacune des destinations (par exemple des points de vente). On connait le coût de transport d'une unité de l'une des origines vers l'une des destinations. En supposant qu'il est possible d'expédier des produits depuis n'importe quelle origine vers n'importe quelle destination, il s'agit de déterminer le coût de transport minimum des origines et n destination.

Nous supposerons qu'il y a m origines et n destinations.

La variable X_{ij} représentera le nombre d'unités expédiées de l'origine i vers la destination j. $X_{ij} \geq 0$ pour tout i, j.

Pour chaque origine i donnée, il y a n valeurs de j possibles ; cela implique qu'il y a $(m \times n)$ X_{ij} différents.

Donc on peut résumer le problème par le graphe suivant :

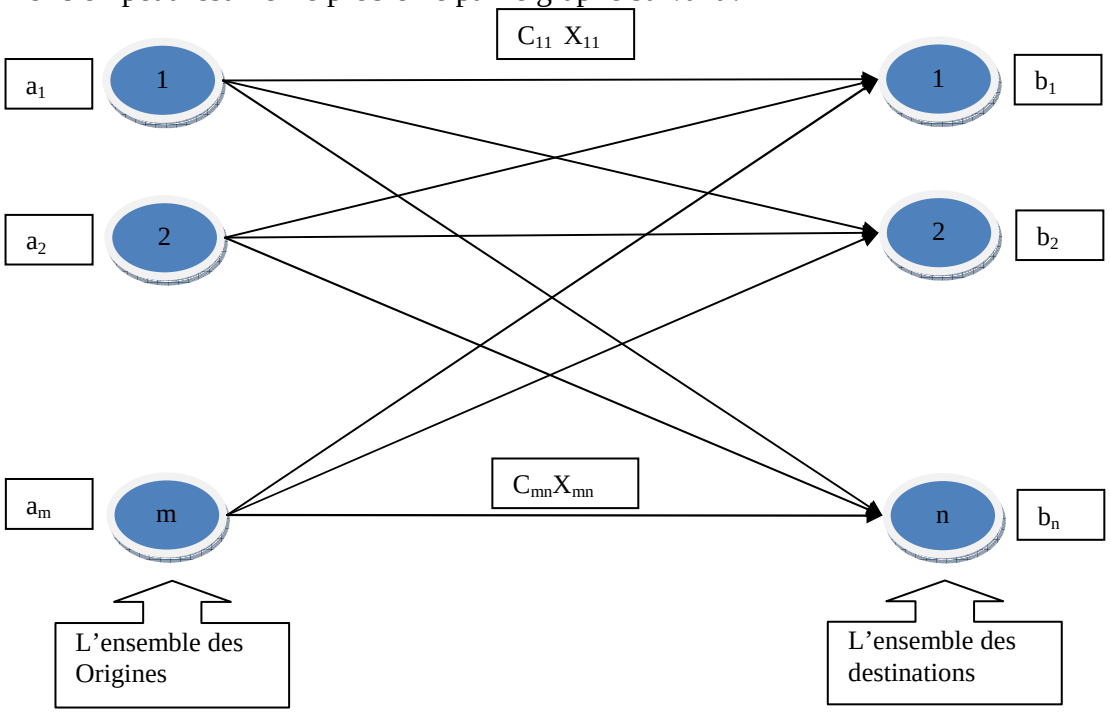


Figure 4: la représentation du problème sous forme d'un graphe

Donc on peut modéliser le problème de transport de la manière suivante :

$$\min z = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}X_{ij}$$

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j \quad \forall i \in [1,\dots,m];\ \forall j \in [1,\dots,n] \qquad (1)$$

$$\sum_{j=1}^{n} X_{ij} = a_i \quad \forall i \in [1,\dots,m] \qquad (2)$$

$$\sum_{i=1}^{m} X_{ij} = b_j \quad \forall j \in [1,\dots,n] \qquad (3)$$

$$a_i \in N;\ \forall i \in [1,\dots,m]$$

$$b_j \in N\ ;\ \forall j \in [1,\dots,n]$$

$$X_{ij} \in N \quad \forall i \in [1,\dots,m];\ \forall j \in [1,\dots,n]$$

Exemple :

Soit la société X possède trois abattoirs en trois villes différentes et fournit trois autres villes en viandes de première qualité comme.

Abattoir	Stock	Ville	Demande
Rabat	120	Casablanca	150
Fès	80	Agadir	70
Meknès	80	Marrakech	60

Les coûts de transport (en DH/ tonne) dépendent des contrats particulières avec les compagnies de chemin de fer et fret aérien sont résumés dans le tableau ci-dessous.

	Casablanca	Agadir	Marrakech
Rabat	8	5	6
Fès	15	10	12
Meknès	3	9	10

La question est : comment livrer au moindre coût ?

III. La résolution du problème de transport

La résolution du problème passe par deux étapes essentielles :

- La première c'est de trouvé une solution de base initiale.
- La deuxième étape est de trouvé la solution optimale à partir la solution de base.

1. *Recherche d'une solution de base initiale*

Définition 2- III-1:

On appelle solution de base une solution vérifiant les contraintes du problème et qui comporte exactement (m-1)(n-1) flux nuls.

- Obtention d'une solution de base par la méthode du coin Nord-Ouest :

C'est la méthode la plus rapide et la plus simple pour déterminé une solution de base car elle ne fait pas entré les coûts de transport c'est a cette raisons la que généralement la solution obtenue par cette méthode est loin de la solution optimale.

Règle du coin Nord-Ouest :

On considère à chaque étape, la case la plus Nord à l'Ouest de la matrice des coûts. On part donc de la route (i1 ; j1) ; on sature soit la ligne i1 soit la colonne j1. Puis on recommence sur la sous-grille formée des lignes et des colonnes non saturées.

Donc L'idée de la méthode est de remplir au maximum la case du tableau en haut, à gauche, puis compléter sur la ligne ou la colonne (de façon à atteindre l'offre ou la demande) et continuer ainsi à compléter les cases immédiatement à droite et en dessous alternativement.

On peut résumer la méthode dans l'algorithme suivant :

Méthode :

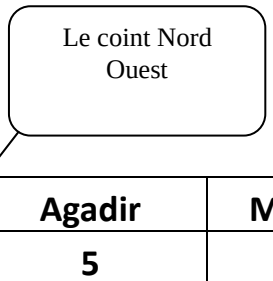
1- i=1, j=1

2- C_{ij}= min (a_i ; b_j). Si C_{ij} = a_i passer à (3) sinon passer à (4).

3- Poser b_j = b_j - a_i et i=i+1, si i ≤ n passer à (2) sinon fin.

4- Poser a_i = a_i – b_j et j=j+1, si j ≤ m passer à (2) sinon fin.

Pour bien comprendre l'idée de la méthode on va l'appliqué sur l'exemple précédant :



	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	8	5	6	120
Fès	15	10	12	80
Meknès	3	9	10	80
Demande	150	70	60	180

Dans le coin Nord-Ouest on mettre la plus grand valeur qui va soit satisfaire une demain ou bien épuisé une disponibilité. Dans l'exemple on va épuiser la 1ere disponibilité.

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	120			120-120=0
Fès				80
Meknès				80
Demande	150-120=30	70	60	160

En suite en refait la même chose mais sur le nouveau sous tableau qu'on obtient après la 1ere modification.

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	120			0
Fès	30			80-30=50
Meknès				80
Demande	30-30=0	70	60	130

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	120			0
Fès	30	50		50-50=0
Meknès				80
Demande	0	70-50=20	60	80

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	120			0
Fès	30	50		0

Meknès		**20**	→	**80-20=60**
Demande	**0**	**20-20=0**	**60**	**60**

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	**120**			**0**
Fès	**30**	**50**		**0**
Meknès		**20**	**60**	**60-60=0**
Demande	**0**	**0**	**60-60=0**	**0**

A la fin on obtient notre solution de base qui vérifie les contraintes de problème, donc la solution de base ainsi obtenue est représenté dans le tableau suivant :

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	**120**			**120**
Fès	**30**	**50**		**80**
Meknès		**20**	**60**	**80**
Demande	**150**	**70**	**60**	**180**

Le coût total de cette solution :

Z = (120*8) + (30*15) + (50*10) + (20*9) + (60*10) = 2690 DH/ tonne.

Comme on peut représente la solution de base dans le graphe suivant :

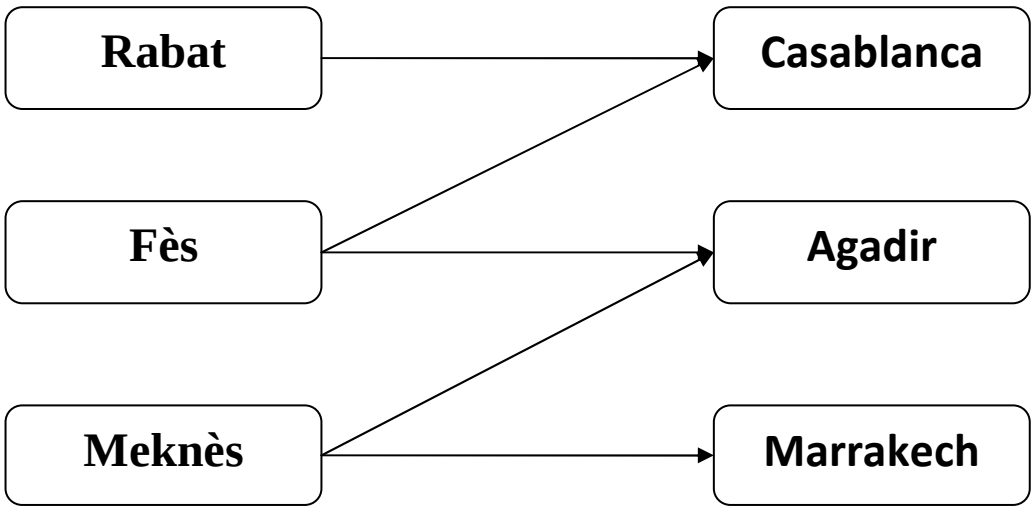


Figure 5 : la représentation de la solution de base sous forme d'un graphe bipartie.

2. Recherche d'une solution Optimal

- Méthode de Stepping-Stone :

Cette méthode est la plus connue parmi les méthodes de résolution du problème de transport, on peut l'applique à n'importe quelle solution de base de notre problème, ainsi on peut résumer l'algorithme dans les trois point suivant :

1- Calcul des potentiels associés aux origines et aux destinations.
2- Calcul des variations de coût unitaire (les coûts marginaux) pour chaque case vide (δ).
3- Calcul de la quantité maximale (q) qu'on peut ajouter à la case vide.

On va expliquer par la suit ces trois point.
On peut écrit l'algorithme de Stepping-Stone comme suit :

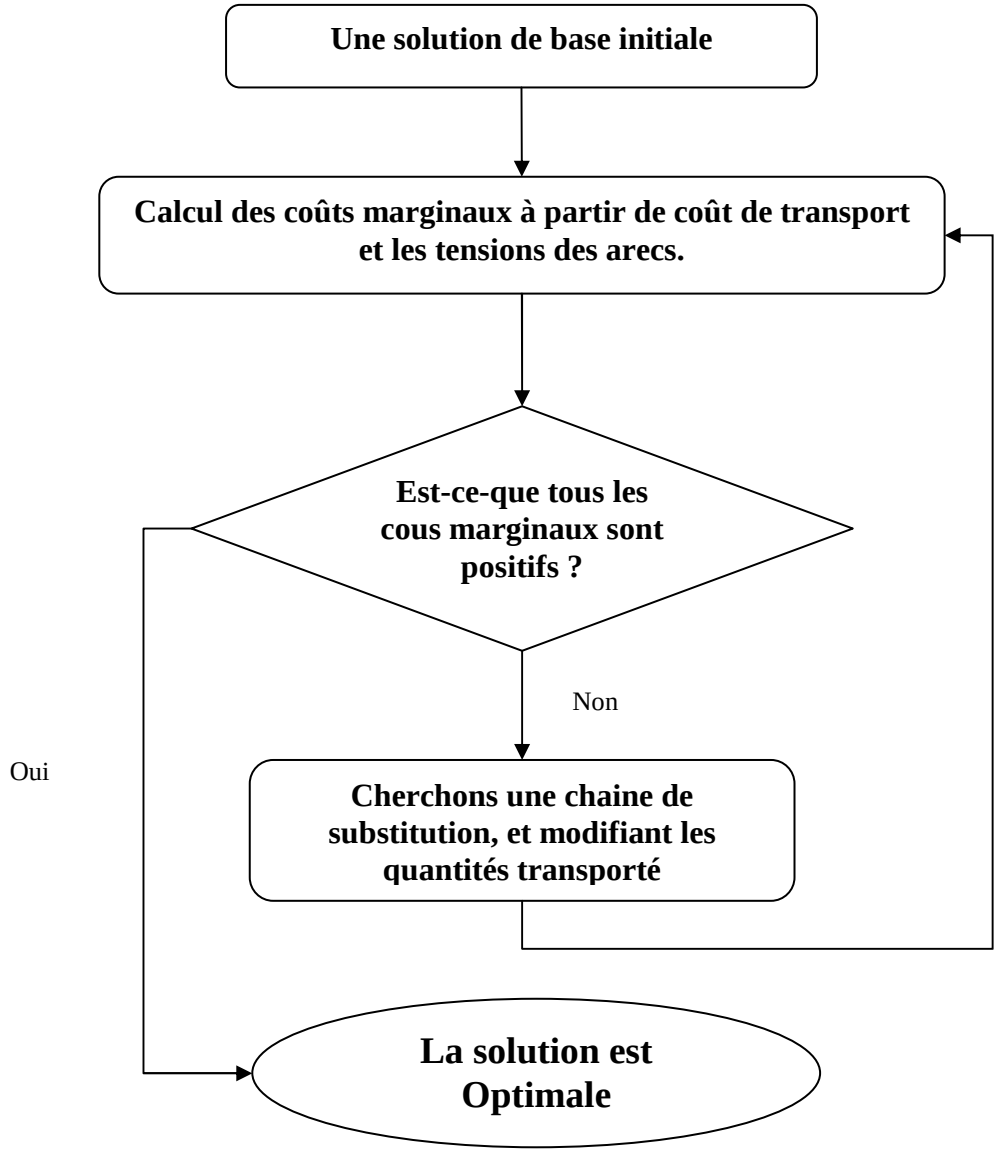


Figure 6: l'algorithme de Stepping-Stone.

Exemple :

On va expliquer les étapes de l'algorithme on utilisant l'exemple précédent de entreprise X du viande :

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	**8**	**5**	**6**	**120**
Fès	**15**	**10**	**12**	**80**
Meknès	**3**	**9**	**10**	**80**
Demande	**150**	**70**	**60**	**180**

Puisque on peut appliquer l'algorithme de Stepping-Stone à n'importe quelle solution de base donc on va considérer la solution de base obtenue par la méthode de coin Nord-Ouest puisque qu'elle est facile et aussi loin de la solution optimal donc elle nécessite beaucoup d'itération de l'algorithme Stepping-Stone.

Donc la solution de base obtenue par la méthode du coin nord-ouest pour cet exemple est représenté dans le tableau suivant :

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	**120**			**120**
Fès	**30**	**50**		**80**
Meknès		**20**	**60**	**80**
Demande	**150**	**70**	**60**	**180**

On commence par calculer les potentiels de chaque sommet (origine et destination) du graphe biparti correspond à la solution.

Pour cela on va donner une potentiels nul à une des origines du problème (par exemple ici $t_{Rabat} = 0$), on suit on calcul les autres potentiels on utilisant le principe suivant :

Soit t_j le potentiel au sommet, pour tout arc (i, j), on doit avoir $\delta_{ij} = 0$ en d'autre terme il faut que $C_{ij} = t_j - t_i$ donc pour calculer les potentiel il faut utiliser l'une de relation suivant :

- $t_j = C_{ij} + t_i$ pour calculer la potentiel d'une destination.
- $t_i = t_j - C_{ij}$ pour calculer la potentiel d'une origine.

Les potentiels calculer pour cette étape est représenté dans le graphe suivant :

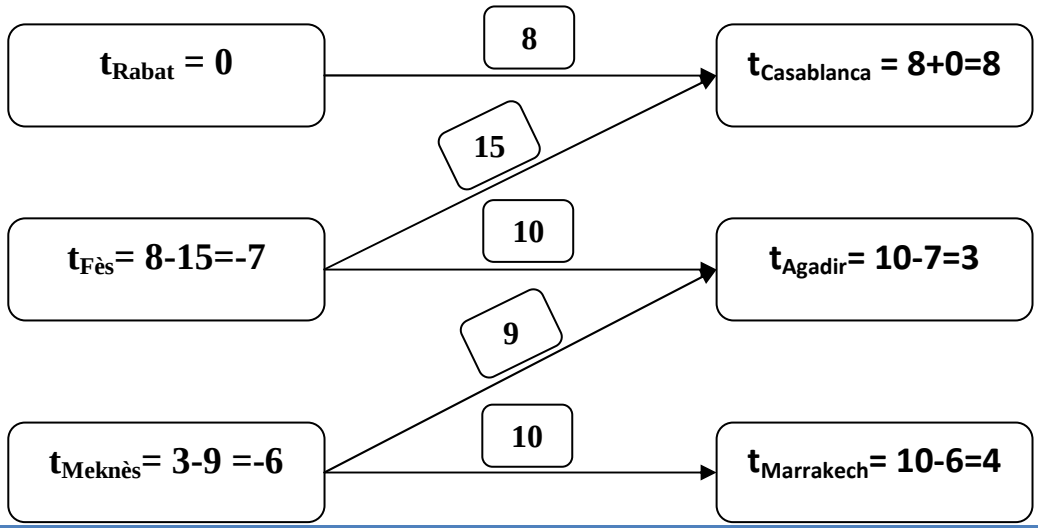


Figure 7 : graphe de potentiel.

Apres le calcul des potentiels on passe au calcul les coûts marginaux des cases vides dans le tableau c.-à-d. les arcs qui n'appartienne par au graphe précédant, donc on cherche s'il existe une nouvelle liaison qui permettrait d'améliorer la solution précédente.

Le calcul des coûts marginaux ce fait par la relation suivant : $\delta_{ij} = C_{ij} - (t_j - t_i)$ quelle que soit l'arc (i, j).

Les coûts marginaux sont représenté dans le tableau suivant :

	Casablanca	Agadir	Marrakech
Rabat	8-(8-0)=0	5-(3-0)=2	6-(4-0)=2
Fès	15-(8+7)=0	10-(3+7)=0	12-(4+7)=1
Meknès	3-(8+6)=-11	9-(3+6)=0	10-(4+6)=0

Puisque il existe des coûts marginaux qui est strictement négatif donc la solution n'est pas optimale et on peut l'améliorée grâce aux route (Meknès, Casablanca) car c'est le coût marginal le plus négatif.

Donc on ajoute cette liaison dans le graphe.

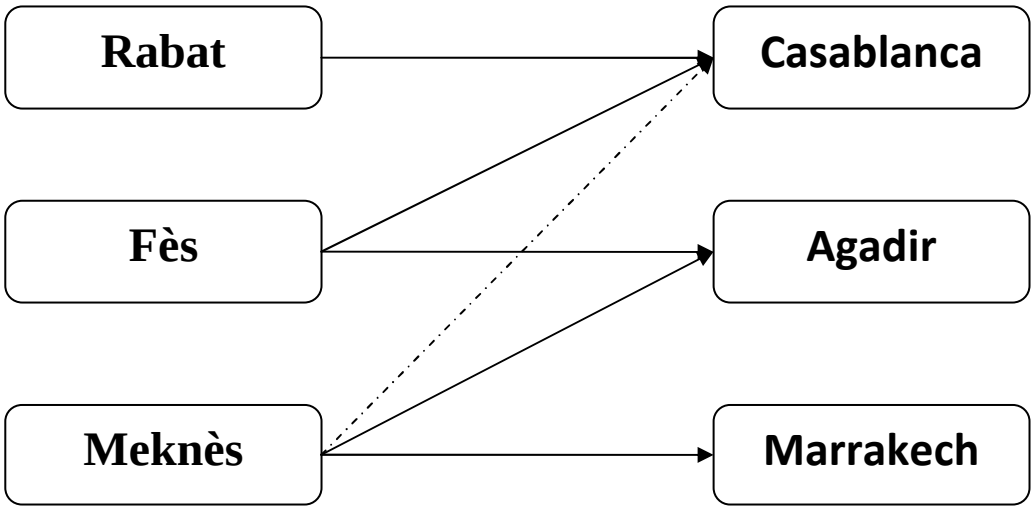


Figure 8 : liaison ajouté.

En modifiant les quantités transportées sur les liaisons du cycle suivant :

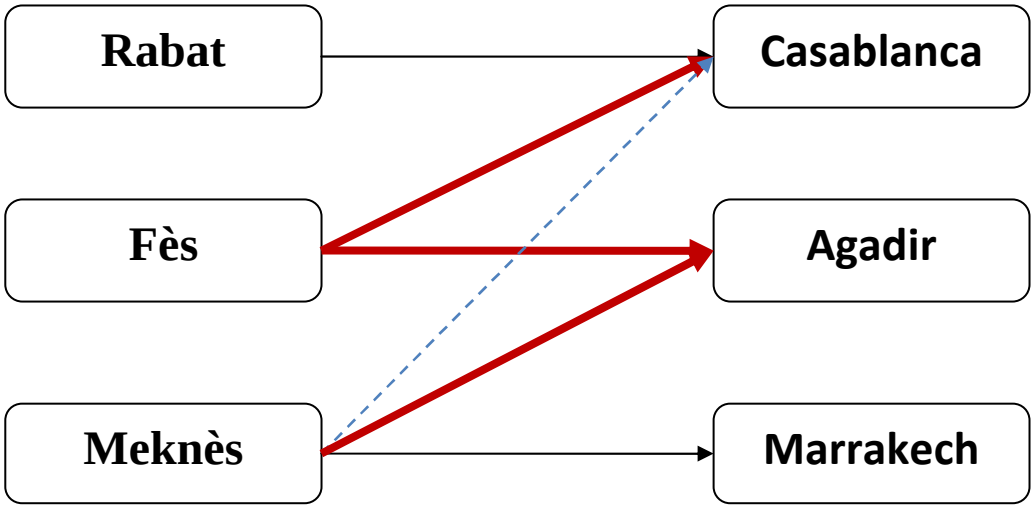


Figure 9 : la chaine de substitution.

Après la détermination de la chaine de substitution, il nous reste que trouver la quantité q qu'on peut transport par la liaison qu'on a ajouté, pour remplir une case vide il faut diminuer une case plaine, donc constituer un circuit de cases pleines qu'on vide et remplir alternativement :

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	120			120
Fès	30-q	50+q		80
Meknès	q	20-q	60	80
Demande	150	70	60	180

D 'ou q représente la quantité maximal qu'on peut transporter sur la liaison ajouté c.-à-d. q= min (20, 30) = 20, donc après la modification on obtient la nouvelle solution représenter dans le tableau suivant :

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	120			120
Fès	10	70		80
Meknès	20		60	80
Demande	150	70	60	180

Le coût total de transport :
Z_1 = (120*8) + (10*15) + (70*10) + (20*3) + (60*10) = 2470DH/ tonne.
Donc on remarque que le coût total a diminué par rapport à au coût de la solution de base $Z_1 < Z_0$ = 2690 DH/ tonne.

Etape 2 :

On refait les mêmes étapes précédant sur la nouvelle solution :

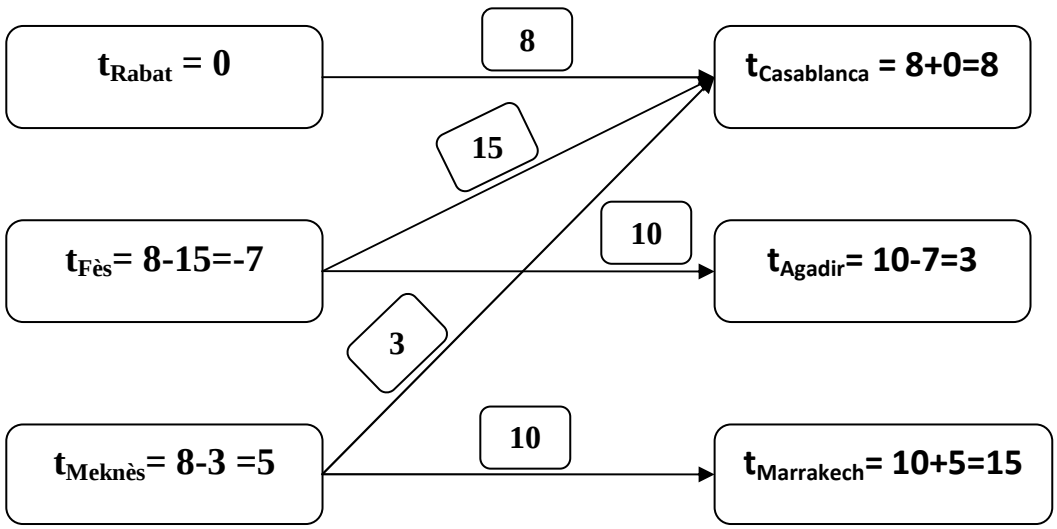


Figure 10 : graphe de potentiel.

Le tableau des coûts marginaux :

	Casablanca	Agadir	Marrakech
Rabat	8-(8-0)=0	5-(3-0)=2	6-(15-0)=-9
Fès	15-(8+7)=0	10-(3+7)=0	12-(15+7)=-10
Meknès	3-(8-5)=0	9-(3-5)=11	10-(15-5)=0

Les coûts marginaux ne sont pas tous positive, donc la solution précédente n'est pas optimale donc on continue, on va ajouter la liaison dont le coût marginal est le plus négatif, donc c'est la liaison (Fès, Marrakech) :

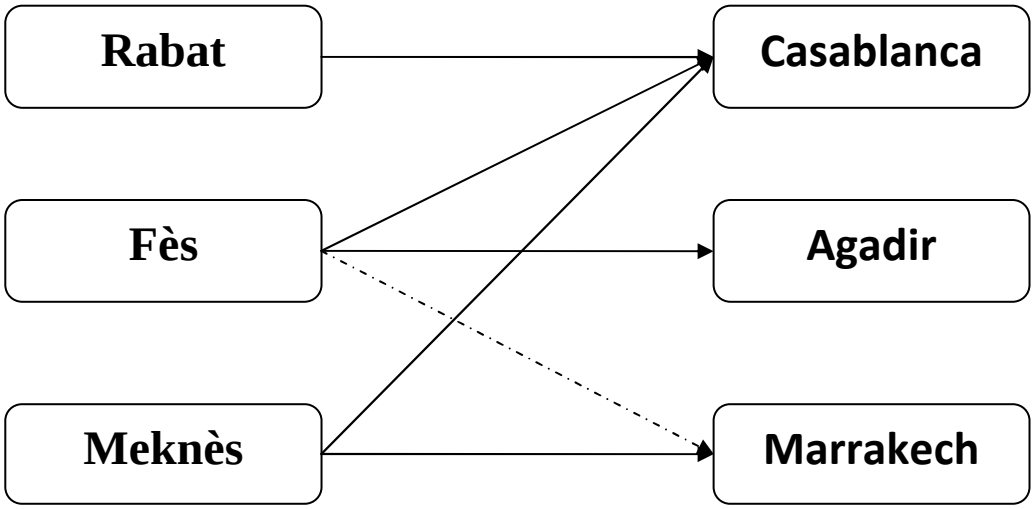


Figure 11 : liaison ajouté.

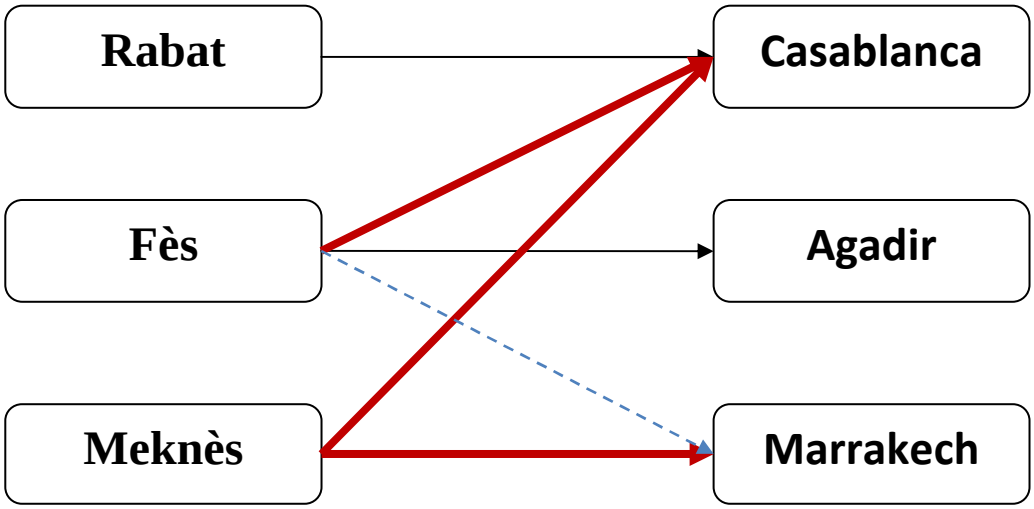


Figure 12 : la chaine de substitution.

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	120			120
Fès	10-q	70	q	80
Meknès	20+q		60-q	80
Demande	150	70	60	180

q = min (10, 60) = 10, donc après la modification on obtient la nouvelle solution représenter dans le tableau suivant :

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	**120**			**120**
Fès		**70**	**10**	**80**
Meknès	**30**		**50**	**80**
Demande	**150**	**70**	**60**	**180**

Le coût total de transport :
Z_2 = (120*8) + (70*10) + (10*12) + (30*3) + (50*10) = 2370 DH/ tonne.
Donc on remarque que le coût total a diminué $Z_2 < Z_1$ = 2470 DH/ tonne.

Etape 2 :

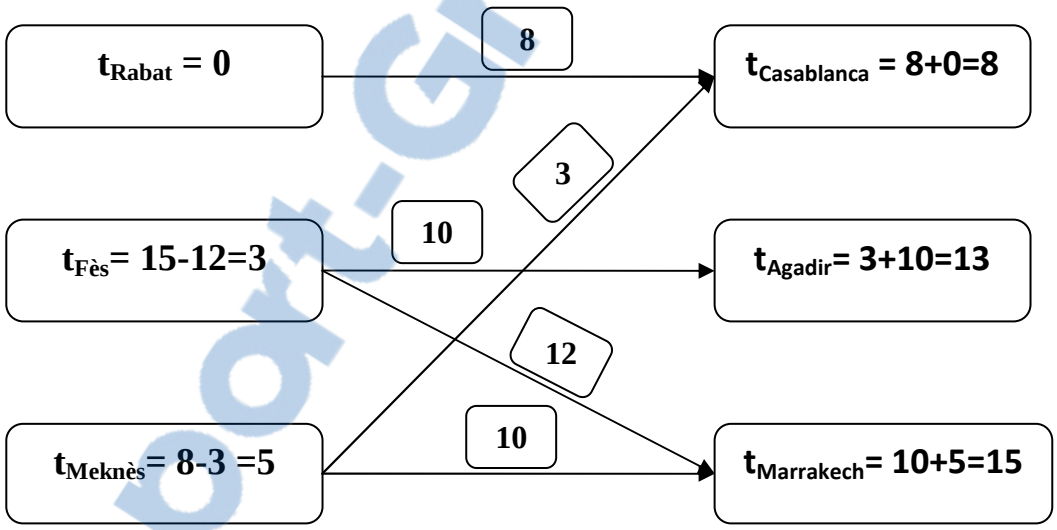


Figure 13 : graphe de potentiel.

	Casablanca	Agadir	Marrakech
Rabat	**8-(8-0)=0**	**5-(13-0)=-8**	**6-(15-0)=-9**
Fès	**15-(8-3)=10**	**10-(13-3)=0**	**12-(15-3)=0**
Meknès	**3-(8-5)=0**	**9-(13-5)=1**	**10-(15-5)=0**

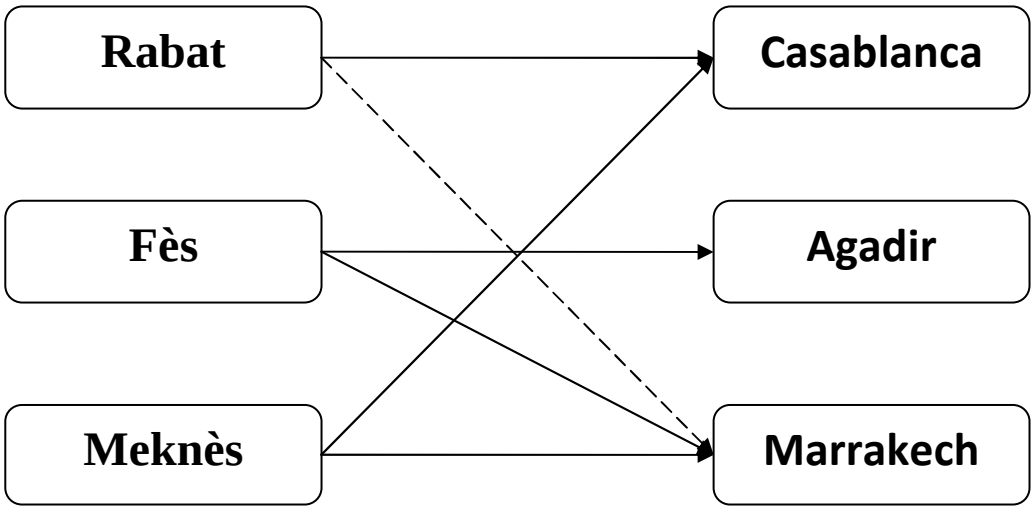


Figure 14 : liaison ajouté.

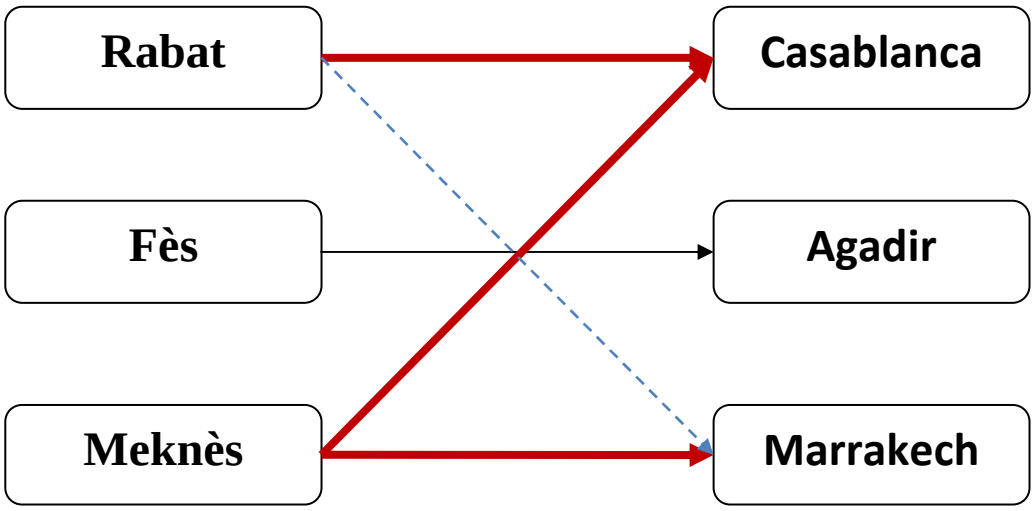


Figure 15 : la chaine de substitution.

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	120-q		q	120
Fès		70	10	80
Meknès	30+q		50-q	80
Demande	150	70	60	180

q = min (120, 50) = 50, donc après la modification on obtient la nouvelle solution représenter dans le tableau suivant :

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	70		50	120
Fès		70	10	80
Meknès	80			80
Demande	150	70	60	180

Le coût total de transport :
Z_3= (70*8) + (50*6) + (70*10) + (10*12) + (80*3) = 1920 DH/ tonne.
Donc on remarque que le coût total a diminué $Z_3 < Z_2$ = 2370 DH/ tonne.

Etape 3 :

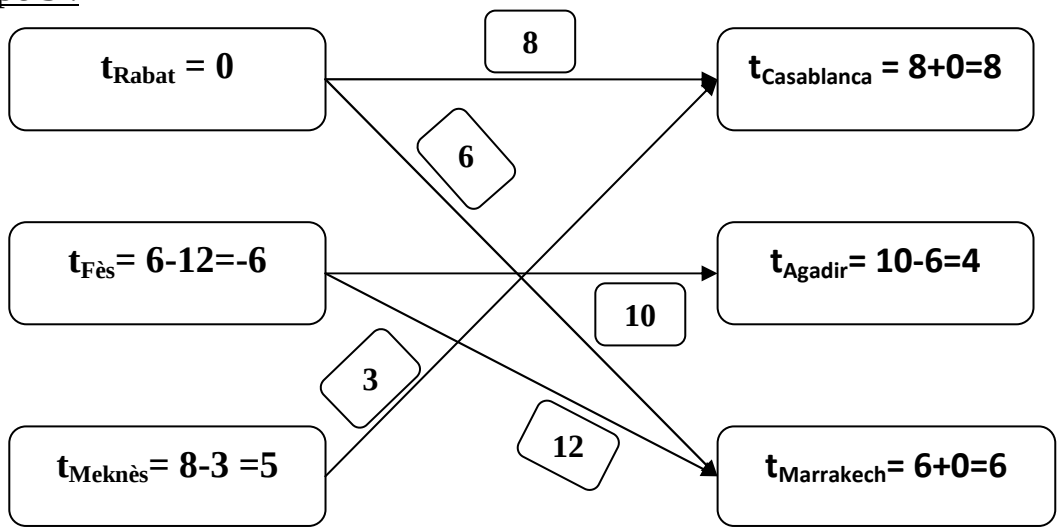


Figure 16 : graphe de potentiel.

	Casablanca	Agadir	Marrakech
Rabat	8-(8-0)=0	5-(4-0)=1	6-(6-0)=0
Fès	15-(8+6)=1	10-(4+6)=0	12-(6+6)=0
Meknès	3-(8-5)=0	9-(4-5)=10	10-(6-5)=9

Tous les coûts marginaux sont strictement positive donc la solution optimale :

	Casablanca	Agadir	Marrakech	Disponibilité
Rabat	70		50	120
Fès		70	10	80
Meknès	80			80
Demande	150	70	60	180

Le coût total de transport :

Z = (70 * 8) + (50 * 6) + (70 * 10) + (10 * 12) + (80 * 3) = 1920 DH/ tonne.

Remarque :

Si la 1ère contrainte n'est pas vérifier c.-à-d.

$$\sum_{i=1}^{m} a_i \neq \sum_{j=1}^{n} b_j$$

On pourra se ramener à l'énoncé précédent de la manière suivante :

- si

$$\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} b_j$$

il suffit d'ajouté une destination fictive de coût de transport infini dont la demande

$$b_{n+1} = \sum_{i=1}^{m} a_i - \sum_{j=1}^{n} b_j$$

- si

$$\sum_{i=1}^{m} a_i < \sum_{j=1}^{n} b_j$$

On ajoute une origine fictive de coût de transport infini, dont la disponibilité est :

$$a_{m+1} = \sum_{i=1}^{m} b_j - \sum_{j=1}^{n} a_i$$

Chapitre 3 : Problème de D'Affectation

I. Introduction

Le problème dit << problème d'affectation >> est très classique de la recherche opérationnelle et apparemment bien résolue. Ce problème s'agit d'affecté n tâche à n serveur en minimisant le coût totale d'affectation.

Chaque tâche a une charge dépendant du serveur auquel elle est affectée et il s'agit de ne pas dépasser la charge maximale de chaque serveur.

Deux serveurs n'ayant pas forcément les mêmes caractéristiques.

II. L'utilisation de l'affectation

1. L'occupation d'un bâtiment

La répartition du personnel dans un bâtiment est un problème d'affectation pour lequel il s'agit de déterminer la position idéale de chaque personne de façon à maximiser la satisfaction globale ou la productivité totale. Dans ce cas, les ressources sont les bureaux qui peuvent avoir des caractéristiques différentes en termes de surface ou d'équipement. Ce problème compte de nombreuses contraintes comme le regroupement géographique des équipes de travail ou leur position vis-à-vis des équipements communs comme les salles de réunion.

2. L'élaboration du plan annuel de mutation

Par extension, le problème précédent s'apparente à l'établissement du plan annuel de mutation (PAM). En effet, il s'agit dans ce cas d'affecter une personne à un poste en respectant les contraintes de gestion et les souhaits des administrés.

III. Formulation du problème :

Le problème d'affectation peut être représenté sous forme d'un graphe biparti de la manière suivant :

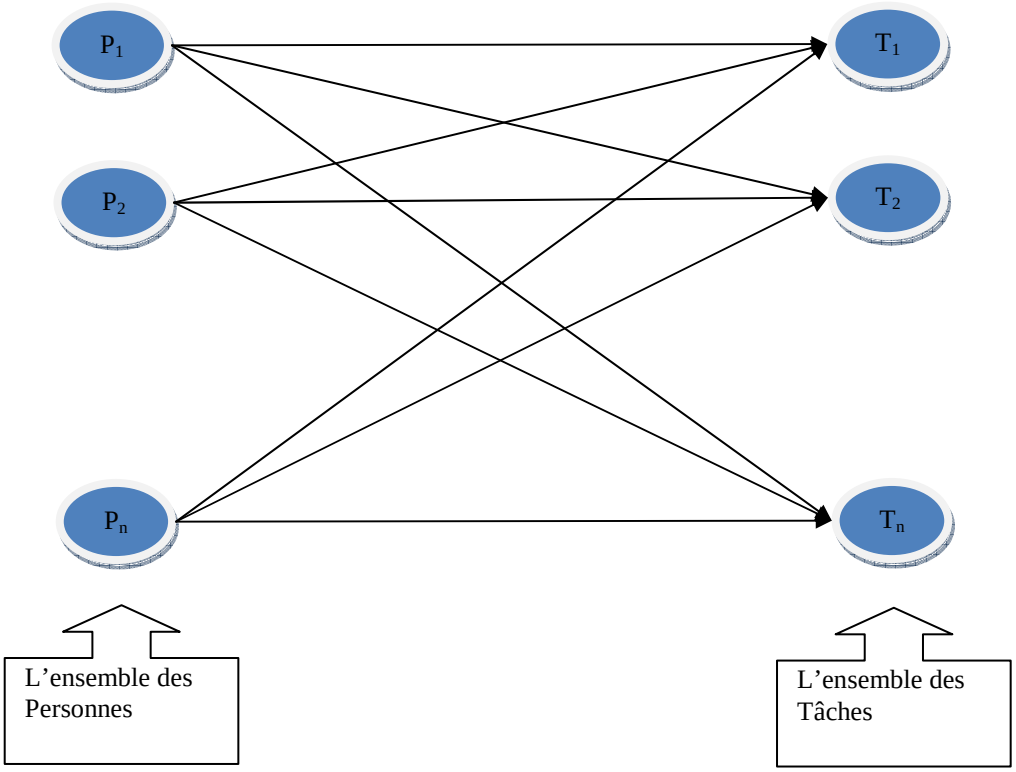


Figure 17 : la représentation du problème de d'affectation.

Modélisation d'un problème d'affectation standard :

Le problème d'affectation peut être modélisé comme suit :

$$\begin{cases} \text{Min } z = \sum_{1\le i\le n}\sum_{1\le j\le n} C_{ij}X_{ij} \\ \text{Sujet à} \\ \quad \sum_{1\le i\le n} X_{ij} = 1 \qquad 1 \le j \le n \\ \quad \sum_{1\le j\le n} X_{ij} = 1 \qquad 1 \le i \le n \\ X_{ij} = 0 \; ou \; 1 \qquad 1 \le i \le n \, , 1 \le j \le n \end{cases}$$

Ou : C = (c_{ij})➔ Matrice des coûts d'affectation

$$X_{ij} \begin{cases} 1 \text{ si la } i^{ème} \text{ personne est affectée à la } j^{ème} \text{ tâche} \\ 0 \text{ sinon} \end{cases}$$

Remarque :

Le problème d'affectation est dit non standard, si on a m personnes et n tâches avec m≠n. Mais on peut transformer ce problème non standard à un problème standard de la manière suivante :

- Si m > n alors nous créons m-n tâches fictives.
- Si n > m alors nous créons n-m tâches fictives.

Le coût d'affectation de ces élément fictive est égale à M(ou M>0).

Remarque :

Le problème standard d'affectation, peut être résolût comme un problème de transport ou $a_i = b_j = 1$, et les C_{ij} représente le coût unitaire de transport.

Remarque :

Le problème d'affectation peut être défini comme un problème de maximisation.

Ou la fonction objective devient :

$$\text{Max } z = \sum_{1 \le i \le n} \sum_{1 \le j \le n} Cij\ Xij$$

Ainsi, le nouveau problème consistera à affecter les n personnes aux n tâches de telle sorte à maximiser la rentabilité totale sous les mêmes contraintes du problème de minimisation.

Propriété :

Supposons $C_{ij} \geq 0$. L'affectation optimale n'est pas modifiée si on ajoute ou on soustrait une constante à tous les éléments d'une ligne ou d'une colonne quelconque de la matrice du coût du problème standard d'affectation.

Preuve :

Supposons, sans perte de généralité, que l'on ajoute une constante une constante K à tous les éléments de la première ligne de la matrice coût, autrement dit, on remplace C_{1j} par $C_{1j} + K$, alors le nouveau problème d'affectation aurait comme fonction objectif :

On sait que $z = \sum_{1 \le i \le n} \sum_{1 \le j \le n} C_{ij} X_{ij}$

On ajoute la constant K à la 1ere ligne on aura :

$$Z' = \sum_{1\le j\le n}(C_{1j}+K)X_{ij} + \sum_{2\le i\le n}\sum_{1\le j\le n} C_{ij}X_{ij}$$
$$= \sum_{1\le i\le n}\sum_{1\le j\le n} C_{ij}X_{ij} + K\sum_{1\le j\le n} X_{1j}$$
$$= \sum_{1\le i\le n}\sum_{1\le j\le n} C_{ij}X_{ij} + K \quad (\text{car} \sum_{1\le j\le n} X_{ij} = 1 \qquad 1 \le i \le n)$$

Donc Z' = Z+K.

IV. La résolution du problème d'affectation :

La méthode Hongrois :

L'algorithme hongrois ou méthode hongroise parfois appelé aussi algorithme de Kuhn-Munkres est un algorithme d'optimisation combinatoire, qui résout le problème d'affectation en temps polynomial. Il a été proposé en 1955 par le mathématicien américain Harold Kuhn[6], qui l'a baptisé « méthode hongroise » parce qu'il s'appuyait sur des travaux antérieurs de deux mathématiciens hongrois : Dénes Kőnig et Jenő Egerváry.

James Munkres a revu l'algorithme en 1957 et a prouvé qu'il s'exécutait en temps polynomial [7].

Cet algorithme, sert à résoudre les problèmes d'affectation, problèmes qu'on peut résumer de la manière suivante : considérant une matrice (appelée tableau de coûts), il faut choisir un seul élément par ligne et par colonne de façon à rendre la somme optimale (minimal ou maximal).

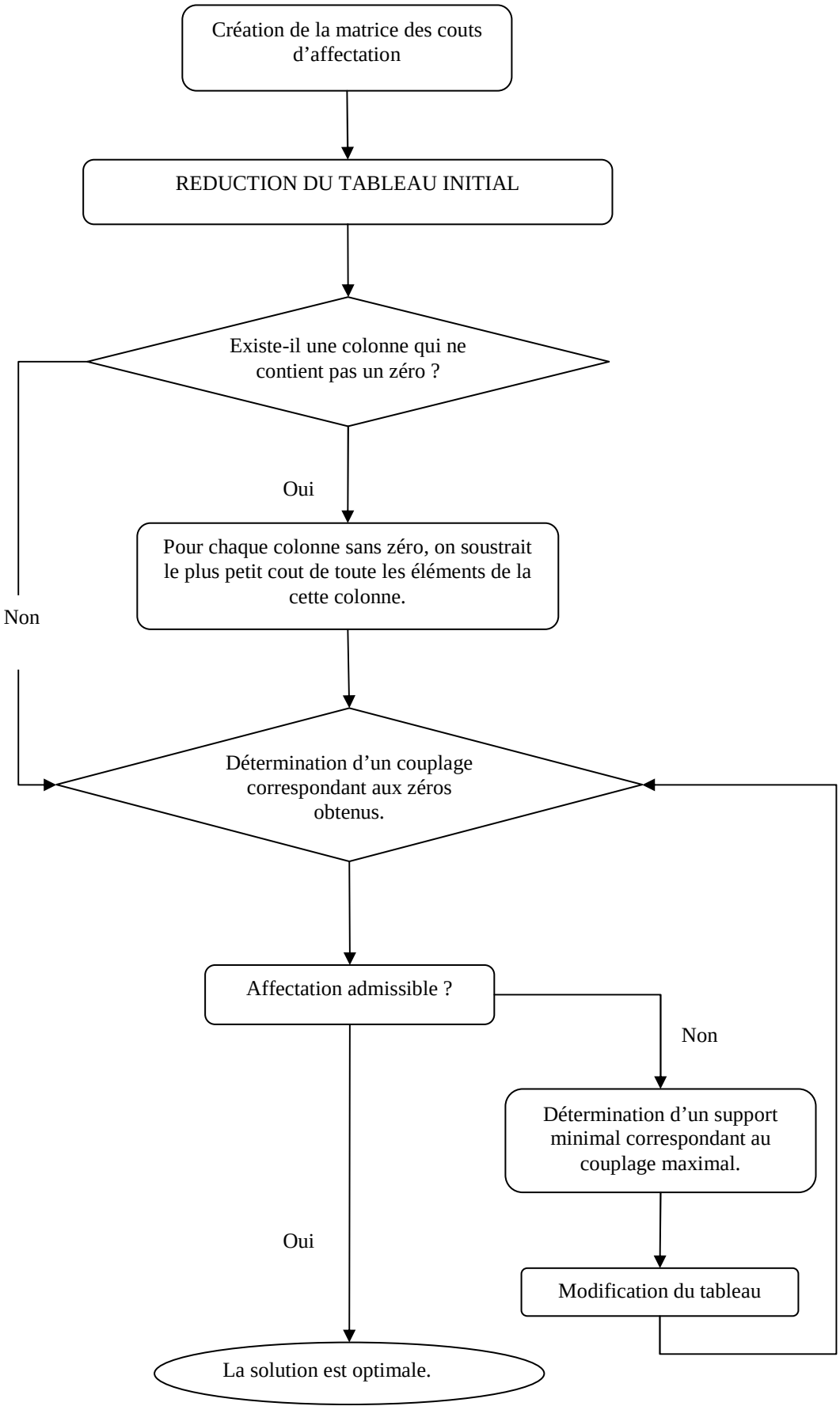


Figure 18 : l'algorithme de Hongroise.

On va explique l'algorithme étape par étape en utilisant l'exemple suivant :

Exemple :

On veut affecter 5 tâches à 5 machines. Les coûts des affectations sont donnés par le tableau suivant:

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5
Tâche 1	**15**	**40**	**5**	**20**	**20**
Tâche 2	**22**	**33**	**9**	**16**	**20**
Tâche 3	**40**	**6**	**28**	**0**	**26**
Tâche 4	**8**	**0**	**7**	**25**	**60**
Tâche 5	**10**	**10**	**60**	**15**	**5**

1ere étape : REDUCTION DU TABLEAU INITIAL

Faire apparaitre au moins un zéro dans chaque ligne et colonne de la matrice des coûts.

Pour cela on va soustrait le minimum de chaque ligne qui contient aucune zéro de tout les éléments de la ligne.

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5	Minimum
Tâche 1	**15-5**	**40-5**	**5-5**	**20-5**	**20-5**	**5**
Tâche 2	**22-9**	**33-9**	**9-9**	**16-9**	**20-9**	**9**
Tâche 3	**40**	**6**	**28**	**0**	**26**	**0**
Tâche 4	**8**	**0**	**7**	**25**	**60**	**0**
Tâche 5	**10-5**	**10-5**	**60-5**	**15-5**	**5-5**	**5**

Dans la nouvelle matrice obtenue après la soustraction de minimum de chaque ligne on va faire la même chose pour chaque colonne qui ne contient pas des zéros.

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5
Tâche 1	**10-5**	**35**	**0**	**15**	**15**
Tâche 2	**13-5**	**24**	**0**	**7**	**11**
Tâche 3	**40-5**	**6**	**28**	**0**	**26**
Tâche 4	**8-5**	**0**	**7**	**25**	**60**
Tâche 5	**5-5**	**5**	**55**	**10**	**0**
Minimum	**5**	**0**	**0**	**0**	**0**

Donc la matrice obtenue après la 1ere étape et la suivante :

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5
Tâche 1	**5**	**35**	**0**	**15**	**15**
Tâche 2	**8**	**24**	**0**	**7**	**11**
Tâche 3	**35**	**6**	**28**	**0**	**26**
Tâche 4	**3**	**0**	**7**	**25**	**60**
Tâche 5	**0**	**5**	**55**	**10**	**0**

2eme étape : ENCADRER ET BARRER DES ZEROS

La détermination d'un couplage correspondant aux zéros obtenue dans la phase 1.
Pour cela on va utiliser l'algorithme suivant :

1- Choisir la ligne de la matrice qui contient le moins de zéros ni encadrés ni barrés.
2- Encadrer le premier zéro de la ligne et puis barrer en ligne et colonne tous les zéros qui peuvent être affectés.
3- Revenir à (1) jusqu'à ce que tous les zéros soient barrés ou encadrés.

On applique cet algorithme a notre exemple en trouve le résultat suivant :

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5
Tâche 1	**5**	**35**	**[0]**	**15**	**15**
Tâche 2	**8**	**24**	**~~0~~**	**7**	**11**
Tâche 3	**35**	**6**	**28**	**[0]**	**26**
Tâche 4	**3**	**[0]**	**7**	**25**	**60**
Tâche 5	**[0]**	**5**	**55**	**10**	**~~0~~**

Obtention d'un couplage maximal :

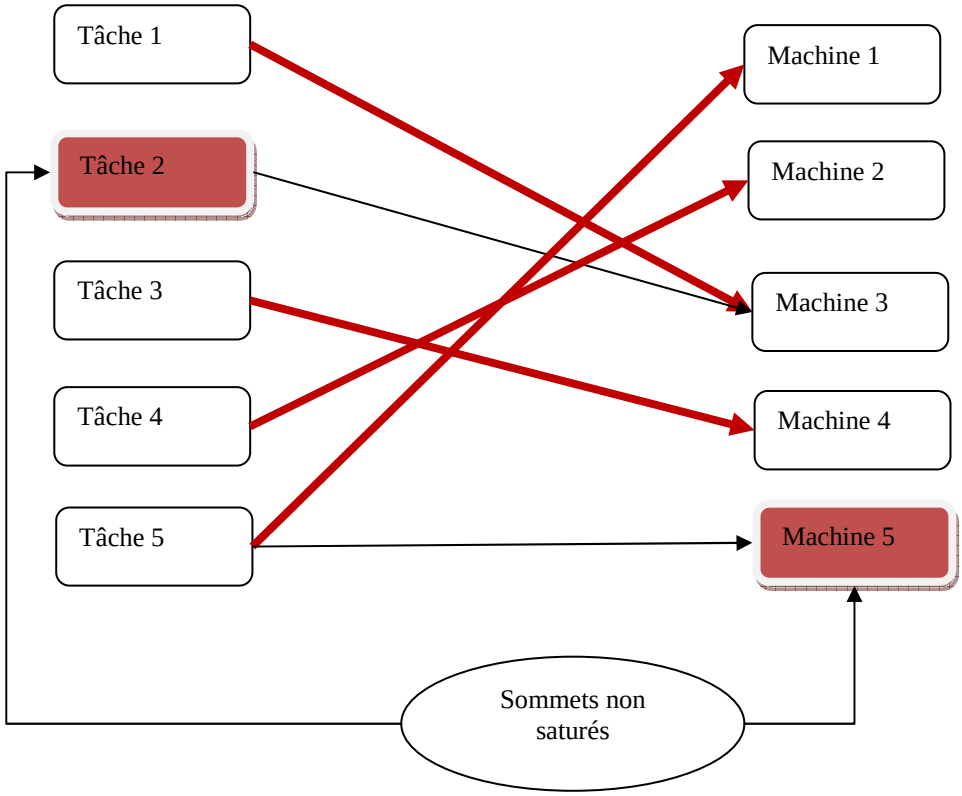


Figure 19 : la représentation de l'affectation obtenue par un couplage maximal.

Puisque il y a deux sommets non saturé alors l'affectation obtenue n'est pas totale.

3eme étape : MARQUER ET BARRER DES LIGNES ET DES COLONNES

Détermination d'un support minimal correspondant au couplage maximal :
Pour cela on va utiliser l'algorithme suivant :

1- Marquer toute ligne n'ayant pas de zéros encadrés.
2- Marquer toute colonne ayant un zéros barré sur une ligne marqué.
3- Marquer toute ligne ayant un zéros encadré sur une colonne marqué, puis revenir à (2). Jusqu'à qu'aucune marquage soit possible.

On utilisant ce procédure de marquage sur notre exemple en trouve :

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5	ligne marqué
Tâche 1	5	35	0	15	15	+
Tâche 2	8	24	~~0~~	7	11	+
Tâche 3	35	6	28	0	26	
Tâche 4	3	0	7	25	60	
Tâche 5	0	5	55	10	~~0~~	
Colonne marqué			+			

Pour obtenir le support correspondant on va rayer toute ligne non marquée et toute colonne marquée on a donc le tableau suivant :

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5	ligne marqué
Tâche 1	5	35	0	15	15	+
Tâche 2	8	24	~~0~~	7	11	+
~~Tâche 3~~	~~35~~	~~6~~	~~28~~	0	~~26~~	
~~Tâche 4~~	~~3~~	0	~~7~~	~~25~~	~~60~~	
~~Tâche 5~~	0	~~5~~	~~55~~	~~10~~	~~0~~	
Colonne marqué			+			

4eme étape : MODIFICATION DU TABLEAU

Modification du tableau on utilisant la technique suivante :

1- Déterminer le plus petit élément parmi les éléments non rayés du tableau.
2- Retrancher plus petit élément des éléments non rayés.
3- Ajouter ce plus petit élément a tous les éléments rayés deux fois.

Le plus petit élément non rayés

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5	ligne marqué
Tâche 1	5	35	0	15	15	+
Tâche 2	8	24	~~0~~	7	11	+
~~Tâche 3~~	~~35~~	~~6~~	~~28~~	0	~~26~~	
~~Tâche 4~~	~~3~~	0	~~7~~	~~25~~	~~60~~	
~~Tâche 5~~	0	5	~~55~~	~~10~~	~~0~~	
Colonne marqué			+			

Le nouveau tableau obtenu :

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5
Tâche 1	0	30	0	10	10
Tâche 2	3	19	0	2	6
Tâche 3	35	6	33	0	26
Tâche 4	3	0	12	25	60
Tâche 5	0	5	60	10	0

On revient à l'étape 2 jusqu'à trouver un couplage maximal qui sature tous les sommets c.-à-d. une affectation totale, et cette affectation sera forcément optimale.

Encadrer et barrer des zéros

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5
Tâche 1	0	30	~~0~~	10	10
Tâche 2	3	19	0	2	6
Tâche 3	35	6	33	0	26
Tâche 4	3	0	12	25	60
Tâche 5	~~0~~	5	60	10	0

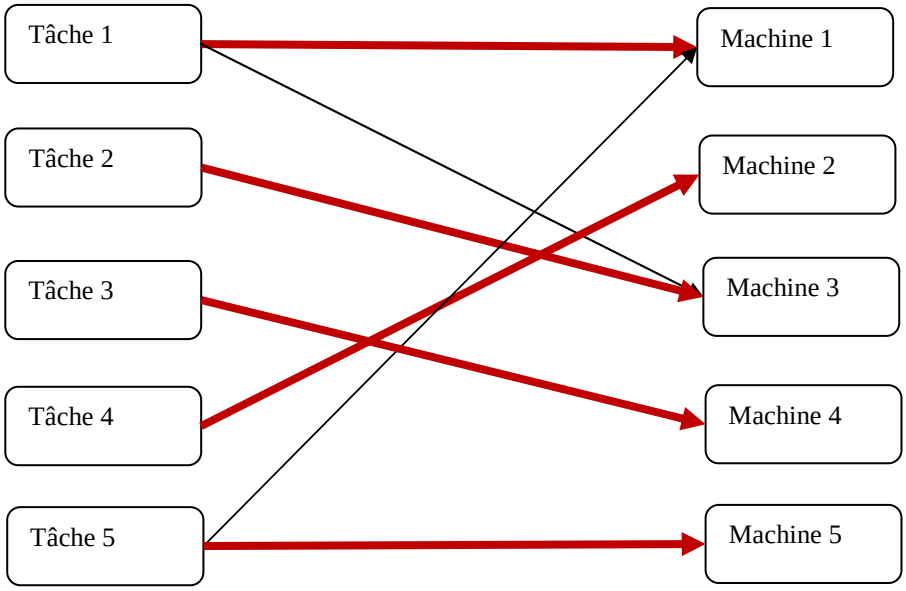


Figure 20 : la représentation de la solution du problème.

On remarque que tous les sommets sont saturés donc l'affectation et total et optimal.
Le coût optimal de cette affectation :
Z = 15 + 9 + 0 + 0 + 5 = 29.

Partie 3 : Application

Chapitre 1 : Application

Dans ce chapitre on va expliquer comment utiliser le programme qu'on a développé pour résoudre les deux problèmes (transport et affectation).

I. Présentation du programme

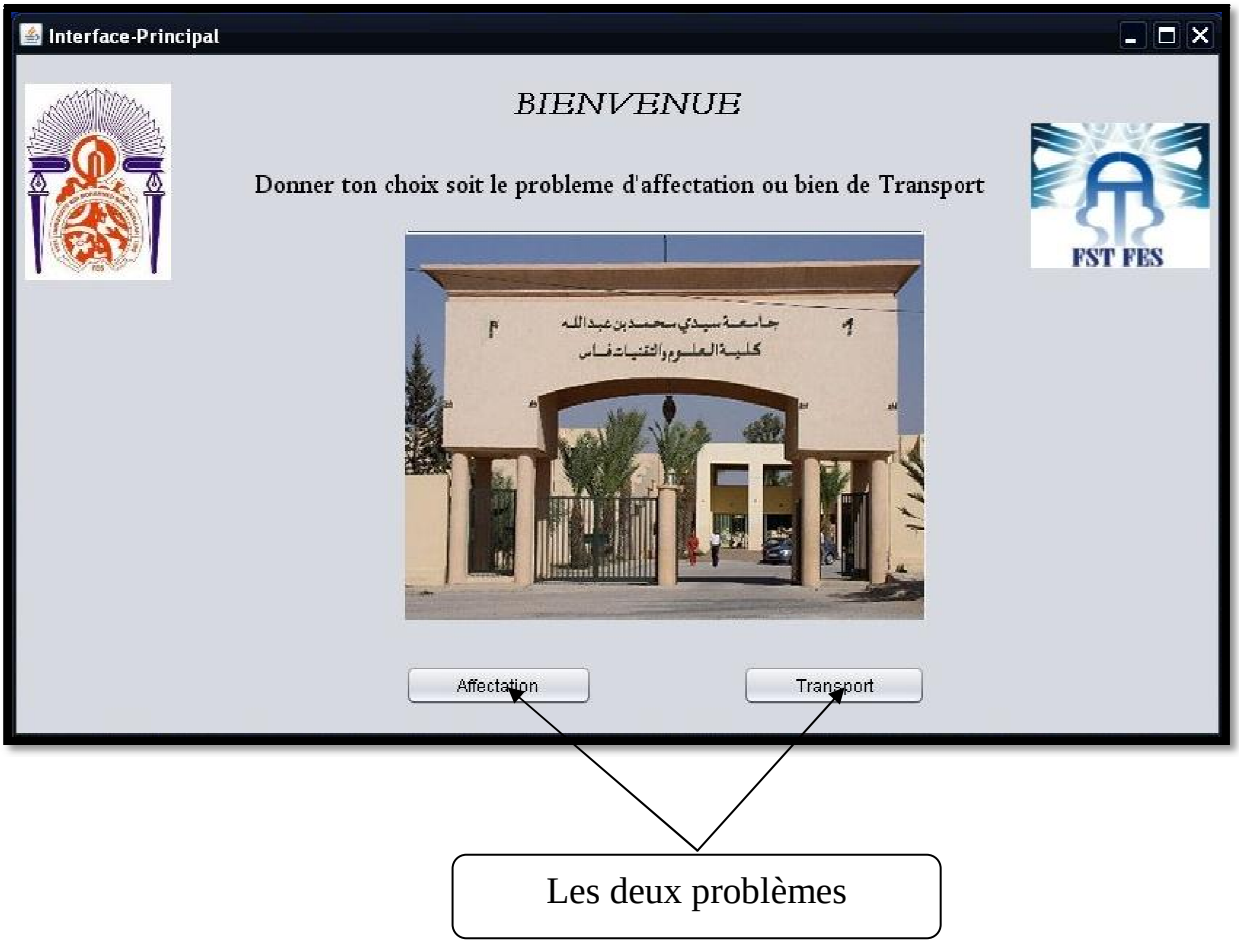


Figure 21 : Interface principal de programme

Dans la 1ere interface le programme vous donne la possibilité de choisir le problème qui tu veux le résoudre, on clique sur le bouton de votre problème, une nouvelle fenêtre sera afficher directement.

II. Problème d'affectation

Si on clique sur le bouton du problème d'affectation la fenêtre suivante s'affiche :

Figure 22 : Interface du problème d'affectation.

On arrivants ici, vous avez la possibilité de résoudre le problème d'affectation par deux méthode différente, soit par l'algorithme de Hongrois qu'on a déjà expliqué son principe dans la 3eme chapitre, soit par la méthode Cplex qu'on va l'explique après dans ce chapitre.

1. Méthode Hongrois

Si on clique sur la méthode de Hongrois la fenêtre qui va s'afficher et la suivante :

Figure 23 : Interface de la résolution par Hongrois.

Dans cette interface qu'on va insérer nous donnée et lire le résultat (l'affectation), donc on va montrer comment on insère les donnée :

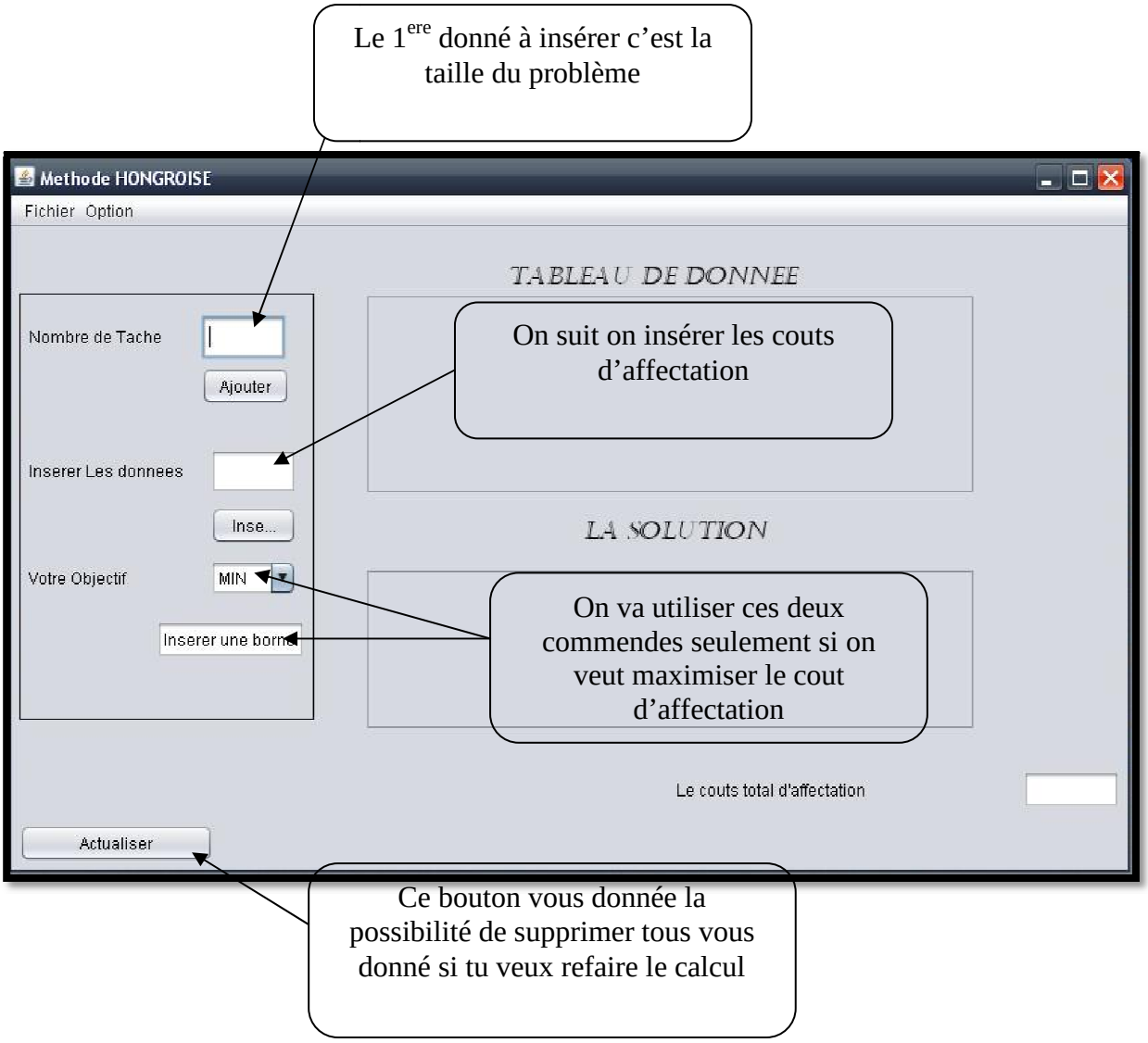


Figure 24 : Insertion des données dans l'interface.

Remarque :

Ci vous avez déjà vous donnée sur un fichier texte il faut seulement lire a partir de ce fichier.

La lecture de la solution :

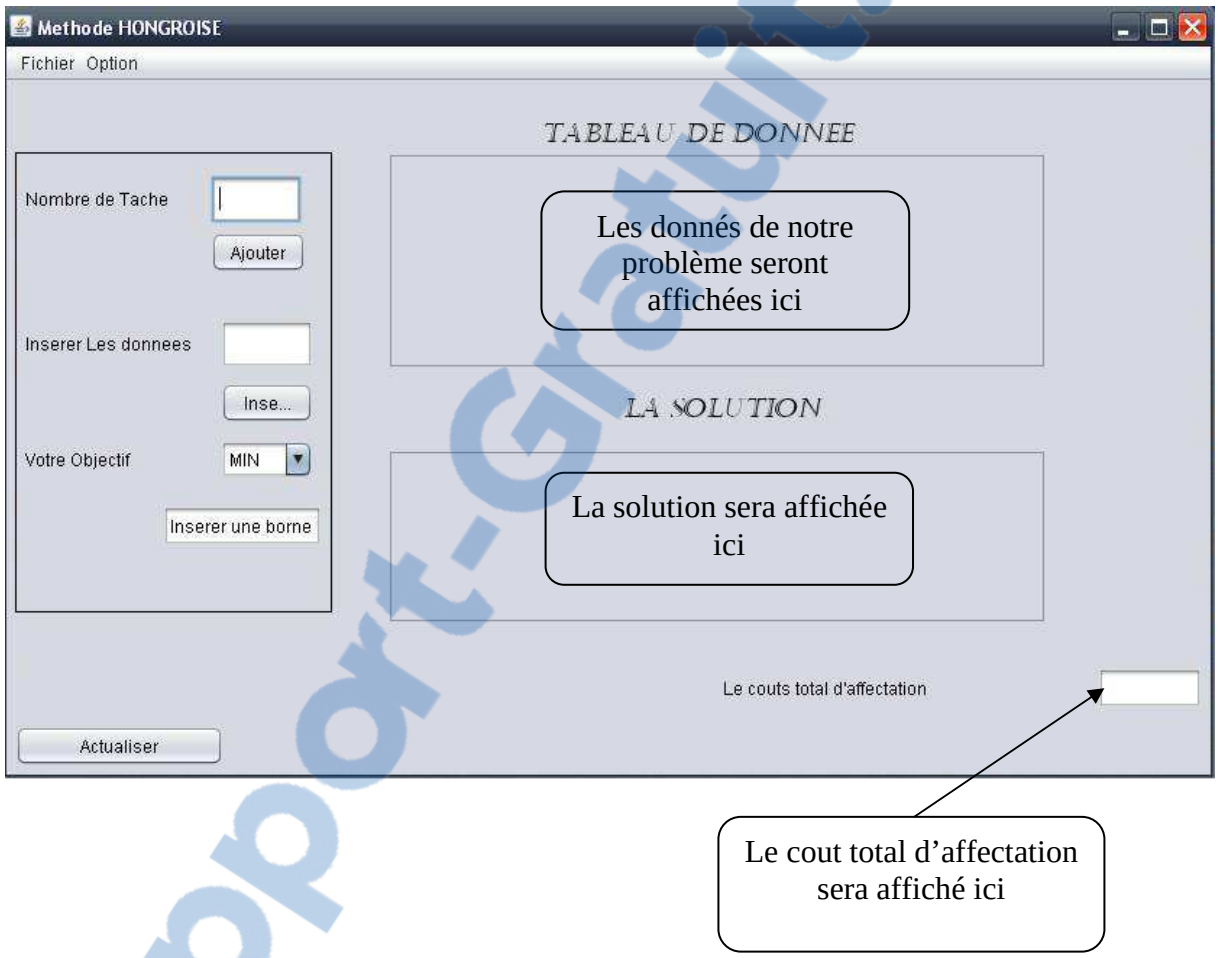


Figure 25 : La lecture de la solution.

Exemple :

On considère l'exemple suivant ou on peut affecter 3personnes à 3tâches différentes :

	M1	M2	M3
P1	12	2	33
P2	51	6	91

P3	41	23	65

Apres l'insertion des donnes la solution obtenue par la méthode Hongroise :

Figure 26 : La solution du problème.

La résolution du problème est base sur quatre méthodes principales représentées comme suit :

La méthode getMatricetranch, cette méthode retranche le minimum de chaque ligne, et de chaque colonne :

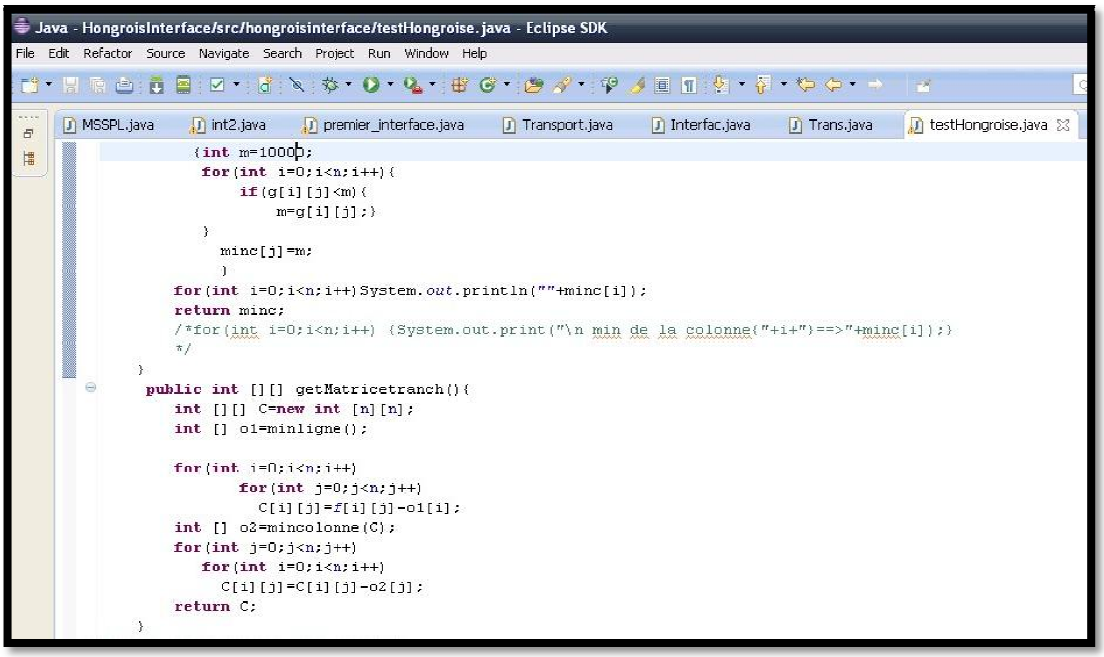


Figure 27 : La méthode getMatricetranch.

La méthode marquezero, cette méthode fait le marquage des zéros :

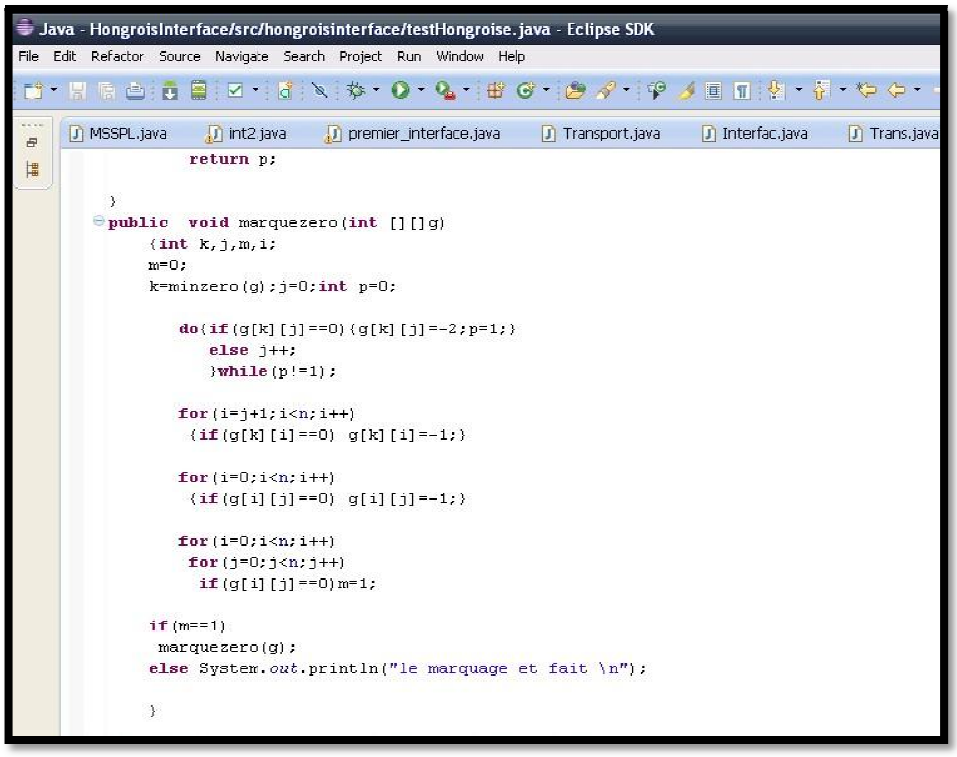


Figure 28 : La méthode marquezero.

La méthode getLigneColonne, cette méthode a comme objectif le marquage ligne colonne pour améliorer la solution :

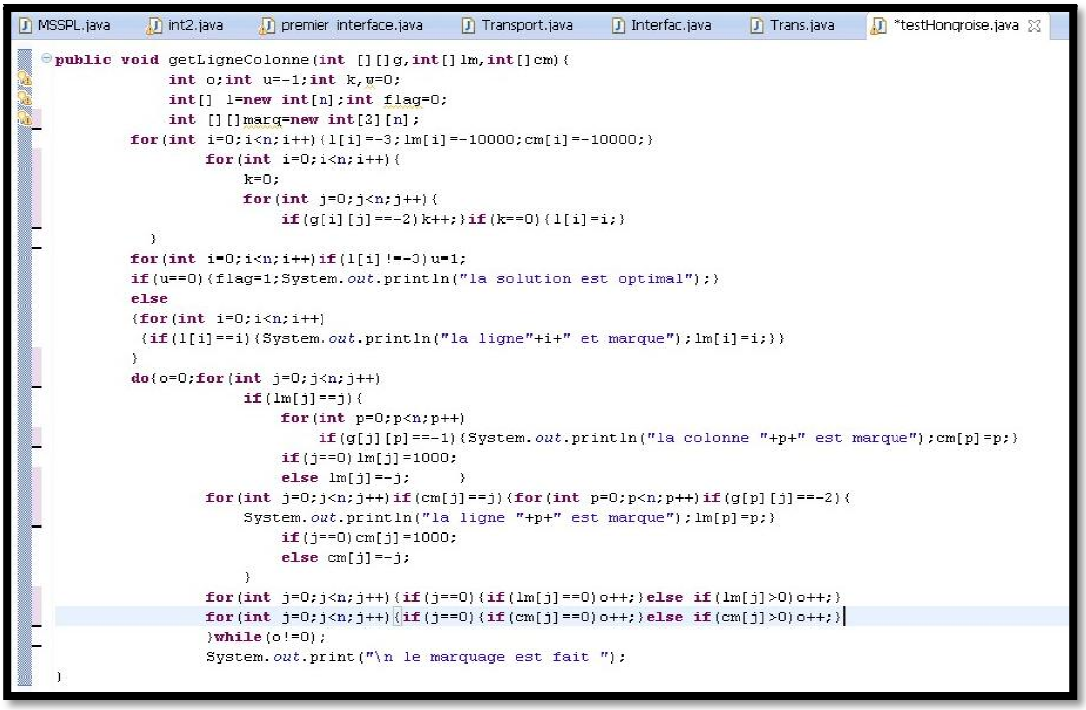


Figure 29 : La méthode getLigneColonne.

La méthode min_sous_tab, cette méthode améliore la solution :

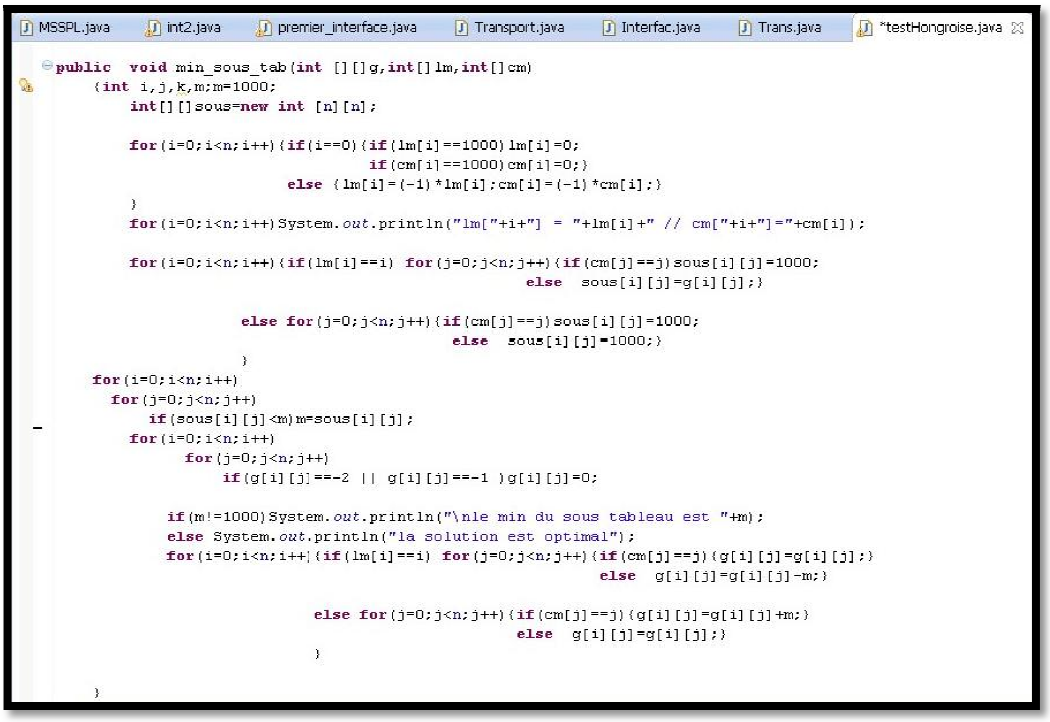


Figure 30 : La méthode min_sous_tab.

On peut résumer le principe de résolution dans le schéma suivant :

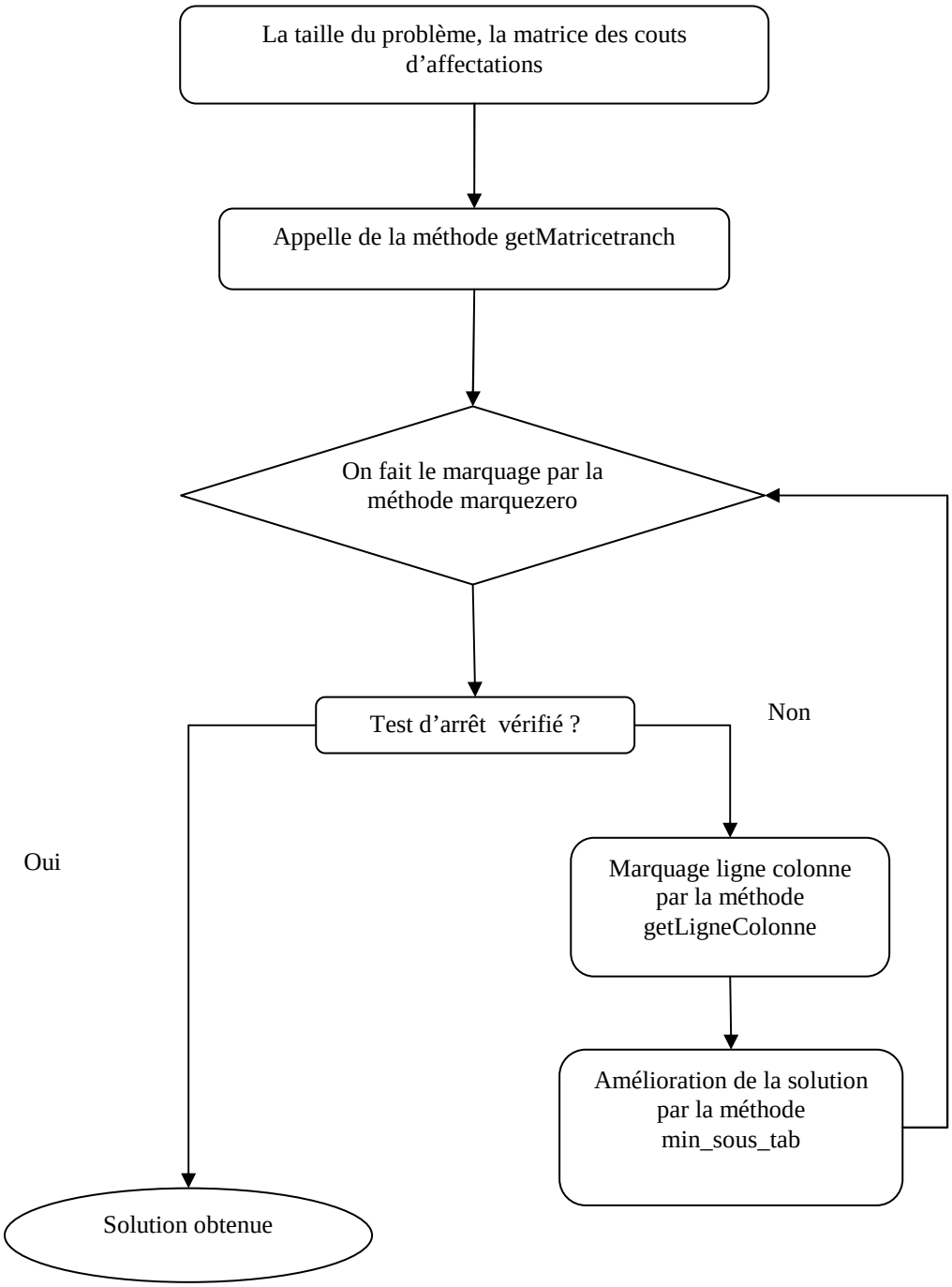


Figure 31 : principe de la résolution.

2. Méthode Cplex

Pour l'insertion et la lecture dans l'interface c'est la même chose que dans l'interface de Hongrois la seule différence et dans la méthode de la résolution.

- La solution par la méthode Cplex :

Figure 32 : La solution du problème par Cplex.

Une petite comparaison montre que les deux méthodes donnent le même résultat.

Le Cplex utilise dans la résolution les deux méthodes suivant :

```
static IloLPMatrix populateByRow(IloMPModeler model) throws IloException {

    int [] t=new int[n*n];
   IloLPMatrix lp = model.addLPMatrix();
   IloNumVar[] x  = model.boolVarArray(n*n);
   test2 tes=new test2();
   int [][] A=tes.Construire_matrice(n);

  for(int l=0; l<A.length; l++){
      for(int j=0;j<A[0].length;j++){
          t[j]=A[l][j];
      }
         lp.addRow(model.eq(model.scalProd(x,t),1.0));
     }

     //  aid2 =model.sum(aid2,model.prod(x[i],E[i]));
  //IloNumExpr Q=model.sum(aid1,aid2);

/* la fonction objective*/

    int y[] = tes.fonction_objectif(f);
    model.add(model.minimize(model.scalProd(x,y)));
    return (lp);
}
```

Figure 33 : La méthode IloLPMatrix populateByRow.

```
public void solve(){

     try {
         IloCplex cplex = new IloCplex();
         IloLPMatrix lp = populateByRow(cplex);
         cplex.setParam(IloCplex.IntParam.RootAlg, IloCplex.Algorithm.Primal);
         if ( cplex.solve() ) {
            double[] x     = cplex.getValues(lp);
            double[] slack = cplex.getSlacks(lp);
            System.out.println("Solution status = " + cplex.getStatus());
            System.out.println("Solution value  = " + cplex.getObjValue());
            cout=(int)cplex.getObjValue();

            int nvars = x.length;
            for (int j = 0; j < nvars; ++j) {
               System.out.println("Variable " + j +
                                  ": Value = " + x[j]);
            }
            int[][] tableau=new int[n][n];
            int k=0;
            for(int y=0;y<n;y++)
                for(int r=0;r<n;r++)
                    {tableau[y][r]=(int)x[k];k++;}
            for(int y=0;y<n;y++){
                for(int r=0;r<n;r++)
                    {System.out.print("\t"+tableau[y][r]);
                    }
                System.out.println();
            }
            d=new int [n];
            int ncons = slack.length;
            for(int y=0;y<n;y++)
                for(int r=0;r<n;r++)
                    {if(tableau[y][r]==1)d[y]=r;}
```

Figure 34 : La méthode solve.

Si on sauvegarde la solution on obtient le fichier suivant :

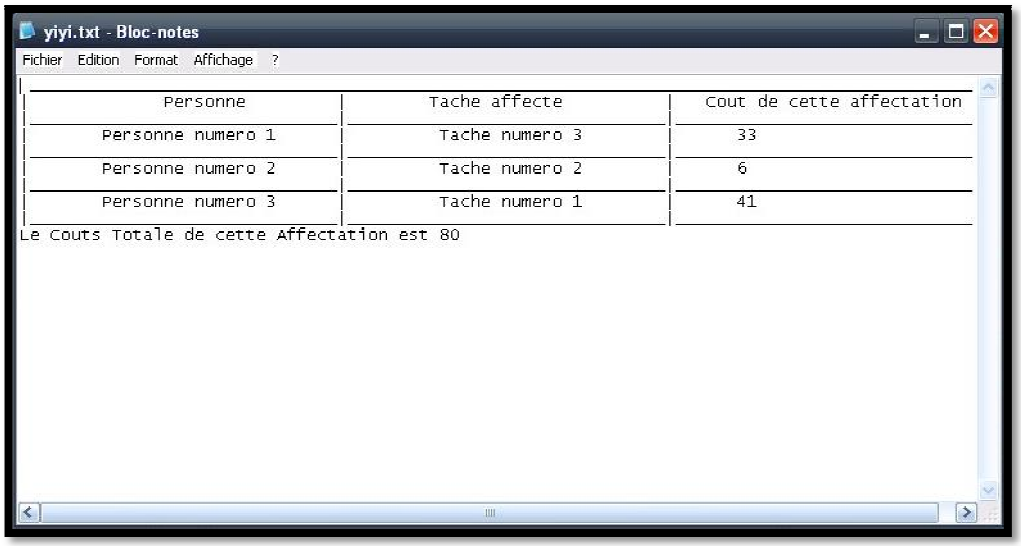


Figure 35 : La solution.

III. Problème de transport :

Si on clique sur le bouton du problème de transport dans l'interface principal la fenêtre suivante s'affiche :

Figure 36 : Interface du problème de transport.

Pour la résolution du problème de transport notre programme utilise l'algorithme de Stepping-Stone qu'on a déjà expliqué son principe dans la 1ere chapitre.

On clique sur le bouton START le programme commence, et on insère nous donné de la manière suivant :

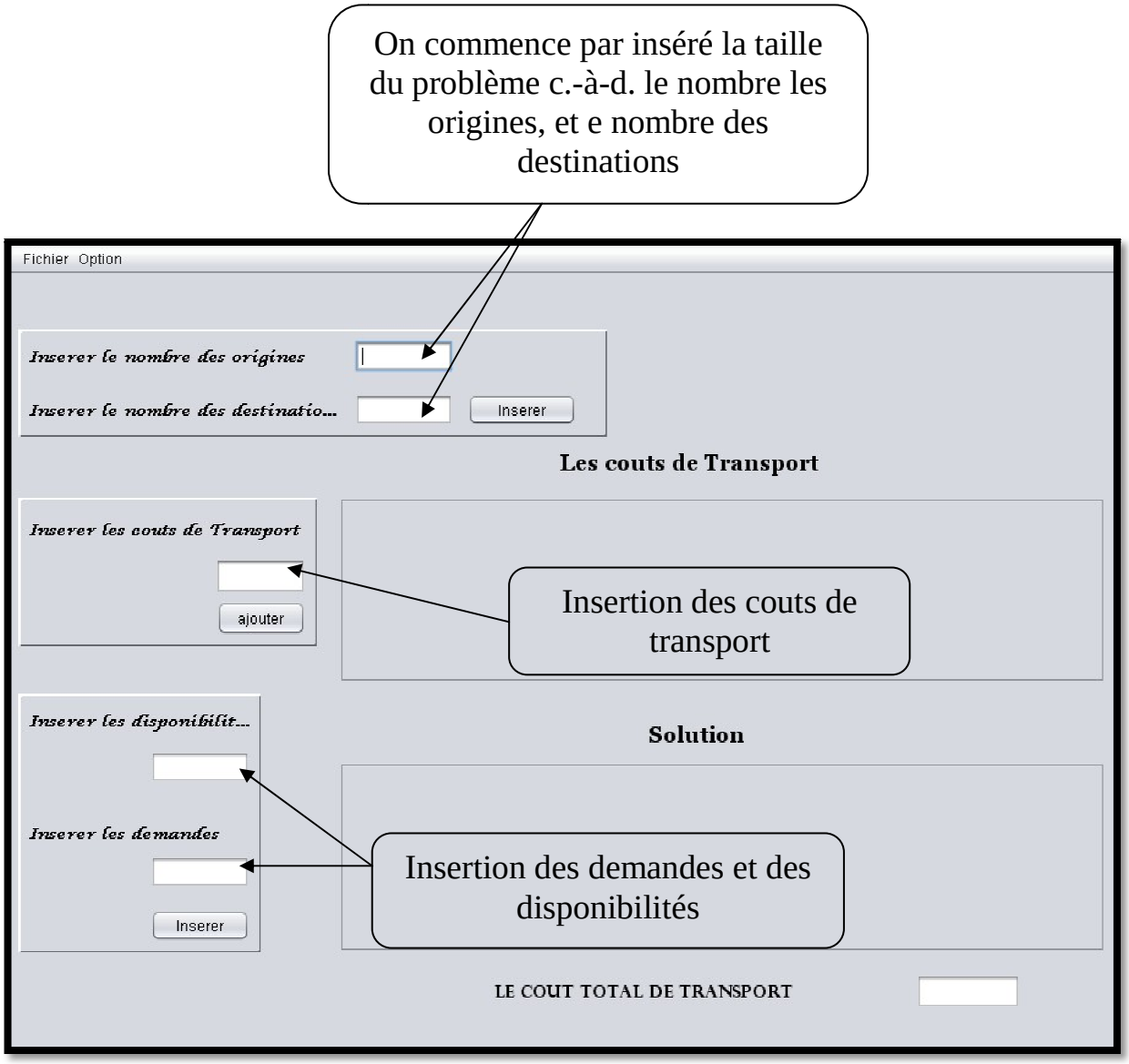


Figure 37 : Insertion des données.

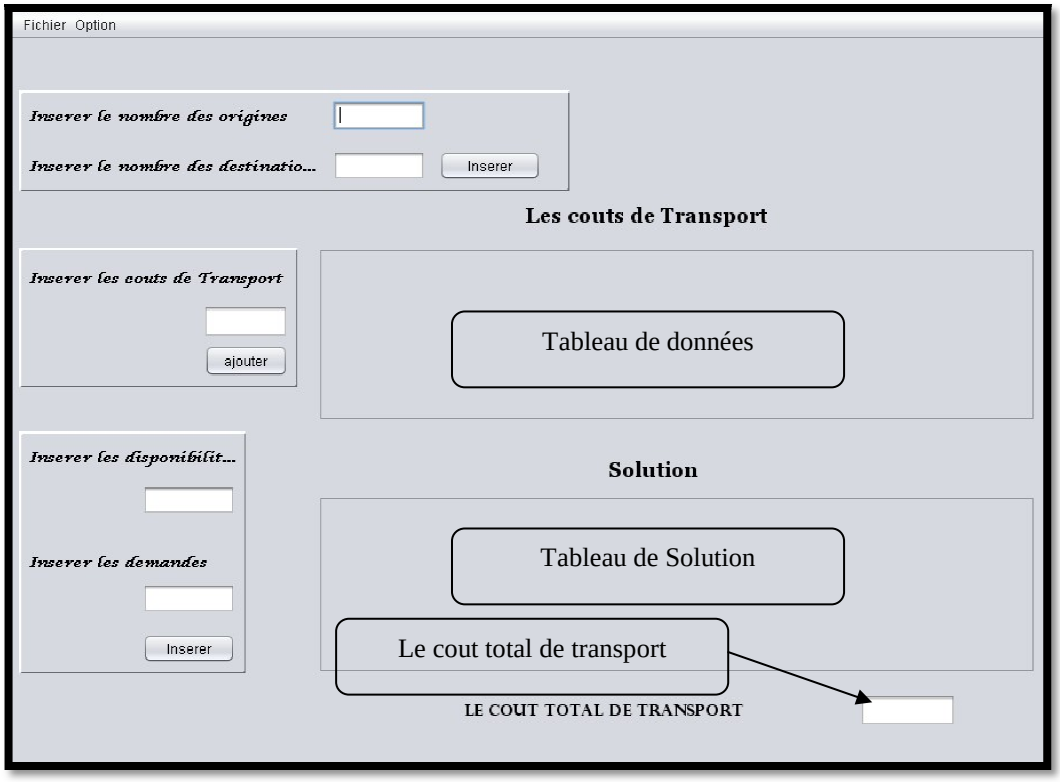


Figure 38 : la lecture

Exemple :

On applique notre programme sur l'exemple suivant :

	Client 0	Client 1	Client 2	Client 3	Disponibilité
Usine 0	11	12	10	10	60
Usine 1	17	16	15	18	30
Usine 2	19	21	20	22	90
Demande	50	75	30	25	180

On trouve le résultat suivant :

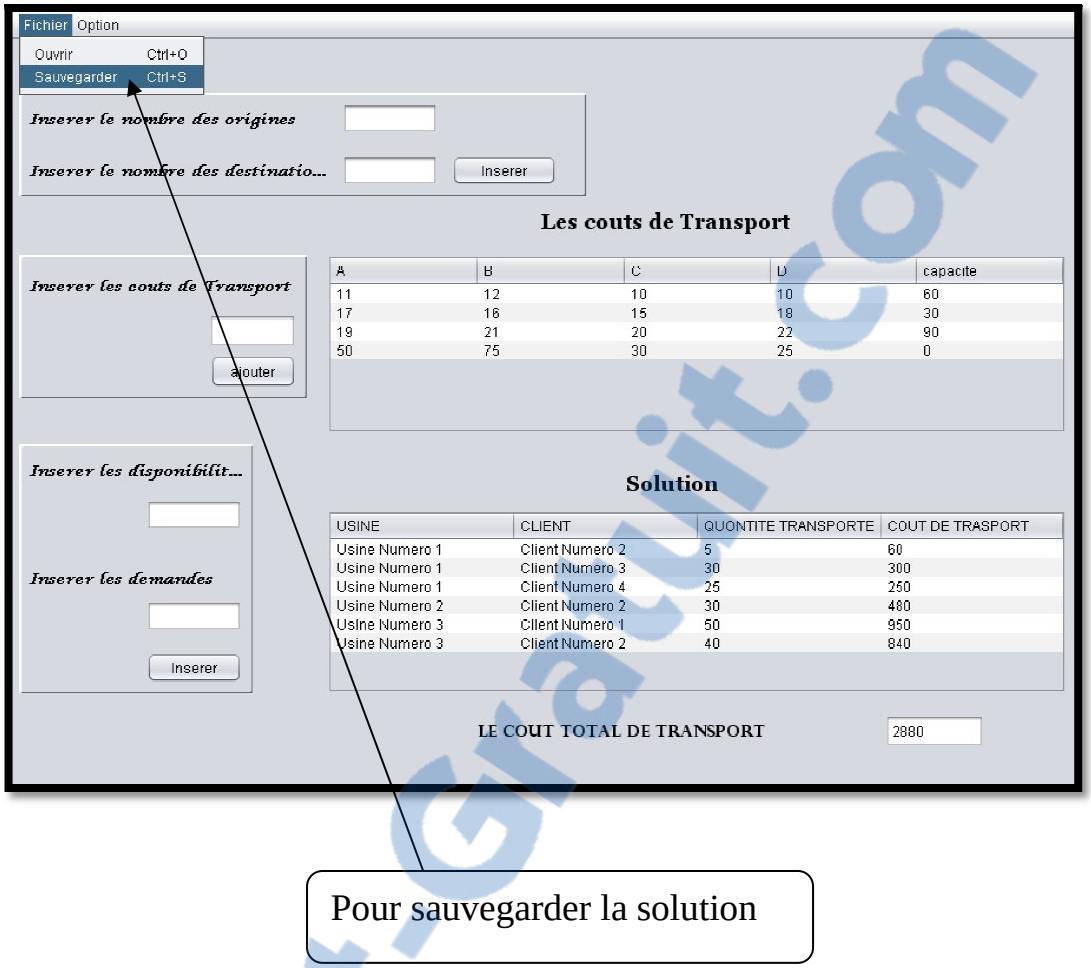


Figure 39 : la solution du problème.

Le fichier correspondant à cette solution :

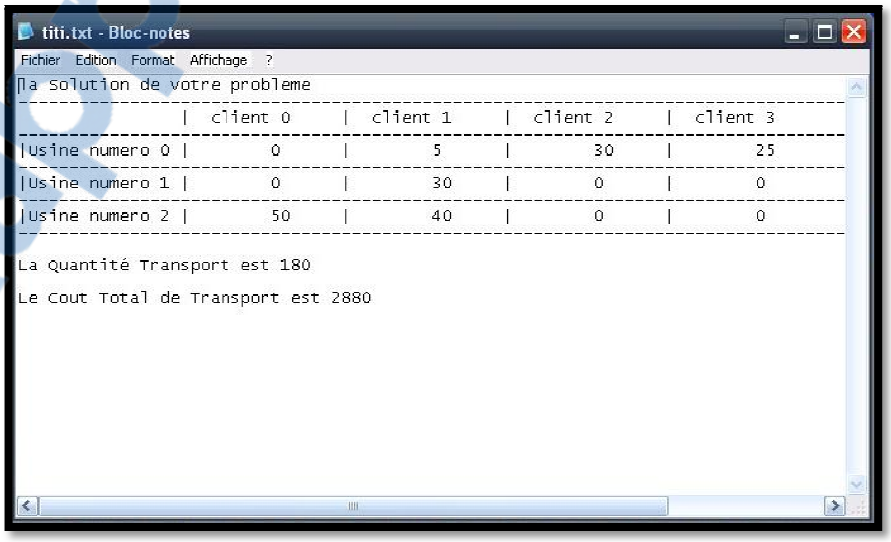


Figure 40 : le fichier de solution.

Donc la solution et bien présenter et elle représente la quantité transporté de chaque Usines vers les clients au moins coûts.

La résolution du problème est base une quatre méthode principal représenter comme suit :

La méthode soldebase, cette méthode cherche une solution de base pour le problème :

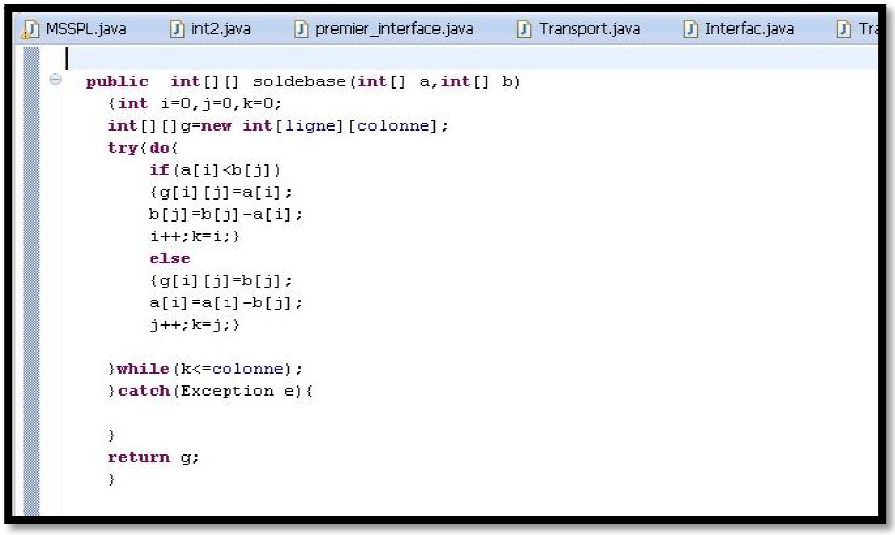


Figure 41 : la méthode soldebase.

La méthode tension, cette méthode a comme objectif le calcul des tensions de chaque sommet (origine et destination)

Figure 42 : la méthode tension.

La méthode coutmarg, cette méthode fait le calcul des coûts marginaux :

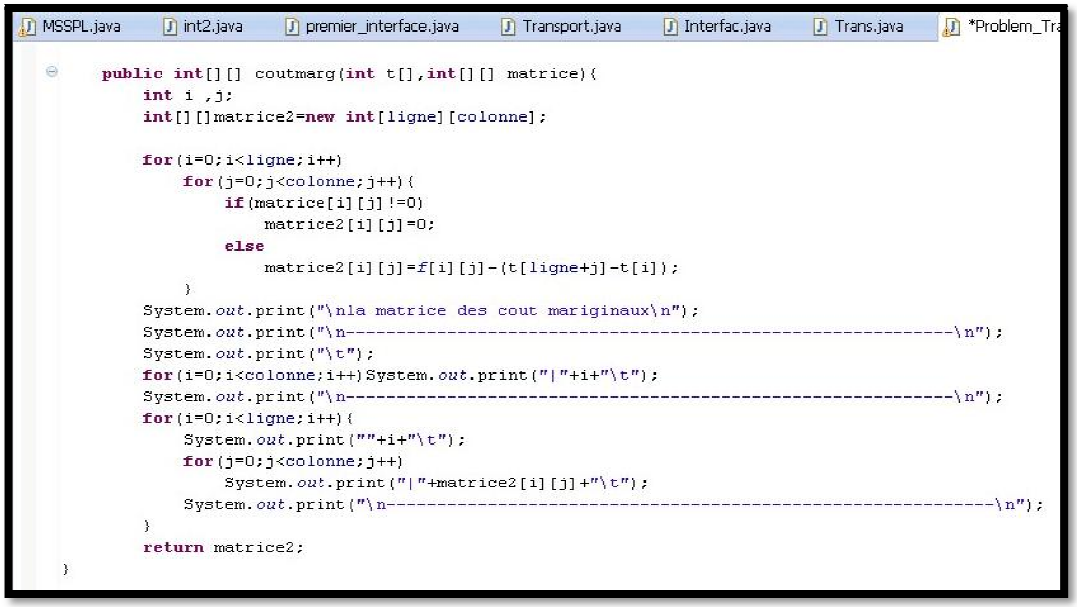


Figure 43 : la méthode coutmarg.

La méthode stepping, cette méthode améliore la solution :

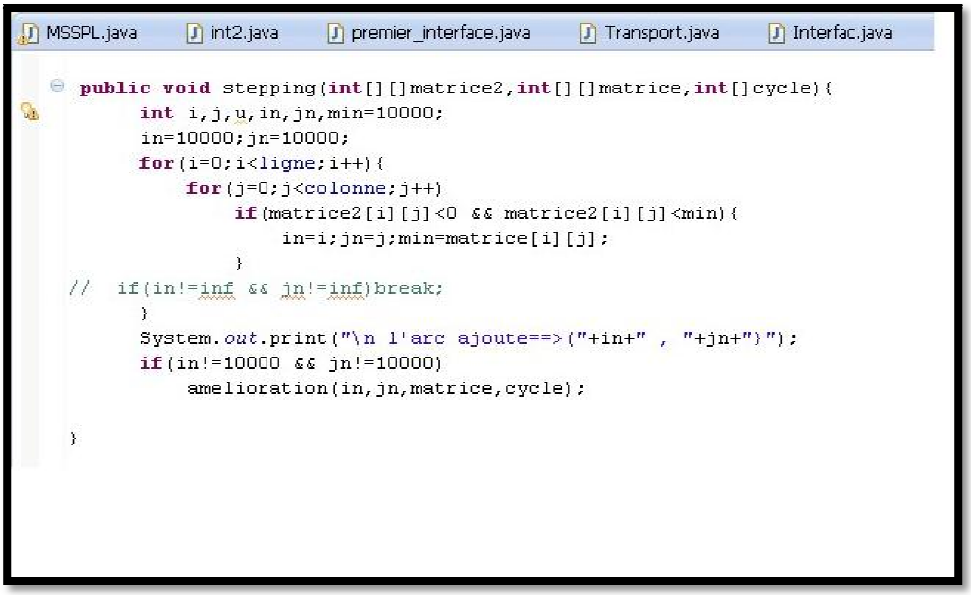


Figure 44 : la méthode stepping.

On peut résume le principe de résolution dans la figure suivant :

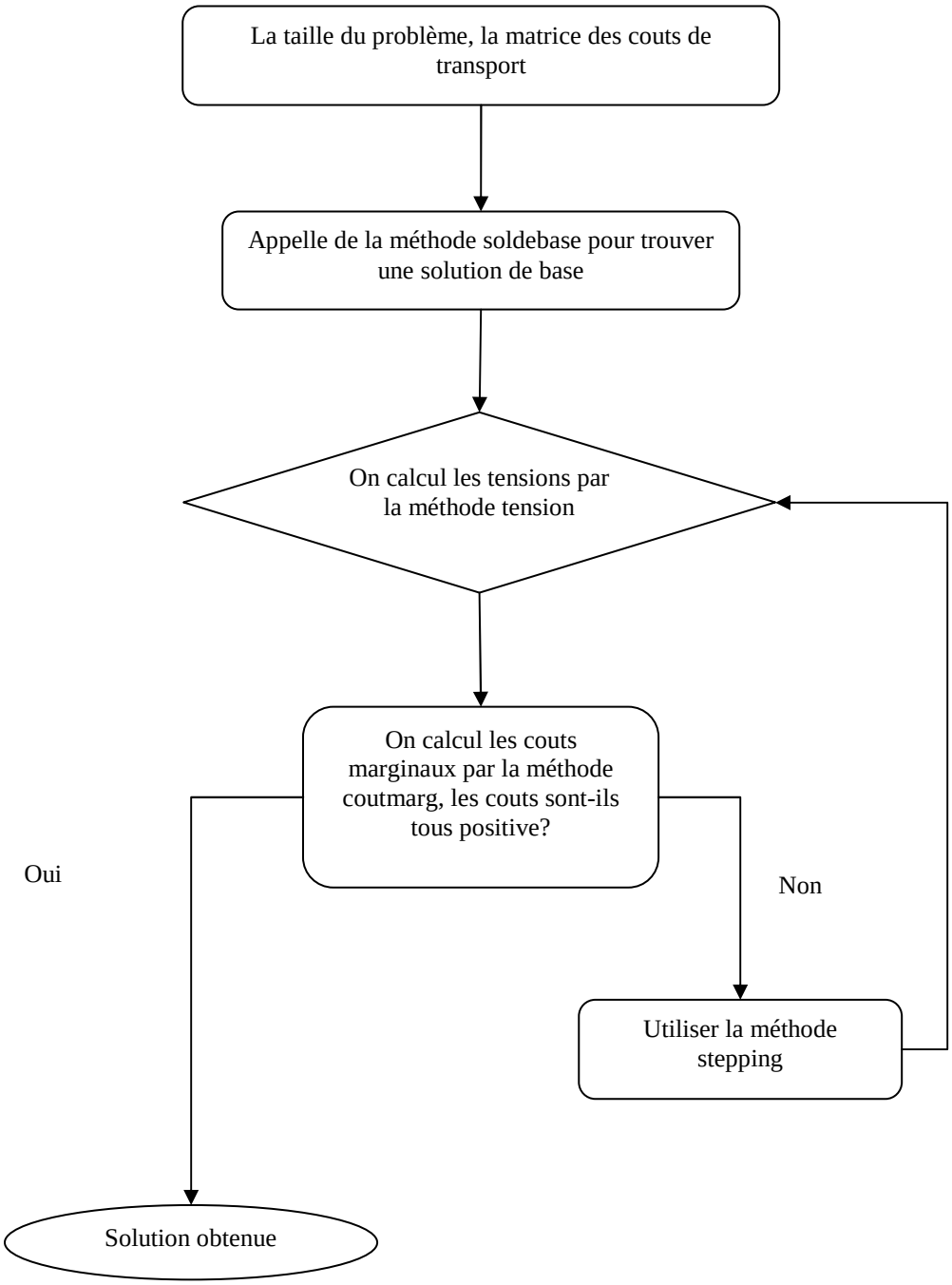


Figure 45 : le principe de la résolution.

Conclusion

Il n’est pas possible de tirer une conclusion définitive sur un sujet en plein développement à l’heure actuelle. Dans ce projet on a traité seulement les cas linéaires des problèmes de transport et d’affectation, mais dans le domaine économique les contraints de ces deux problèmes changes d’une entreprise a l’autre, donc c’est un peu difficile de donner un programme générale qui peut être utilisé par tous les entreprises pour optimiser les coûts.

A la fin je peux dire que ce travaille représente une base de départ pour résoudre les problèmes générale de transport et d’affectation.

Références bibliographiques

- Livre : Programmation Mathématique (Pr. Ahmed El Hilali Alaoui, Pr Youssef Benadada)
- Livre : Méthodes de planification en transport (Yves Nobert, Roch Ouellet, Régis Parent)
- Livre : Optimisation appliquée (YadolahDodge)
- Livre : Apprenez à programmer en JAVA
- http://books.google.fr/
- http://www.doc-etudiant.fr/
- [1]http://www.iecn.unancy.fr/~garet/cours/graphes/graphes.pdf
- [2]http://cours.ensem.inpl-nancy.fr/cours-dm/graphes/glossary.html
- [3]Article : G. Monge. *Mémoire sur la théorie des déblais et de remblais. Histoire de l'Académie Royale des Sciences de Paris, avec les Mémoires de Mathématique et de Physique pour la même année*, pages 666–704
- [4]Article: Frank L. Hitchcock, *The distribution of a product from several sources to numerous localities*, j. Math. Physics, 20(1941), 224-230.
- [5]Article: L. Kantorovich. *On the translocation of masses.* C.R. (Doklady) Acad. Sci. URSS (N.S.)
- [6] : http://www.math.psu.edu/vstein/Hungarian.pdf
- [7] : http://fr.wikipedia.org/wiki/Algorithme_hongrois