CHEN Tingting
MARECHAL Anthony
MARECHAL Benoit

Réseaux

Rapport du projet “Weighted Clustering Algorithm”

Encadrant : N. Mikou

MASTER 2 MULTIMEDIA - 2007/2008

Table des matières

INTRODUCTION

Dans le cadre du module Réseaux de Master 2 informatique spécialité Multimédia, nous avons à réaliser un projet dont l'objectif est d'implémenter le "Weighted Clustering Algorithm", qui permet d'établir un réseau de communication optimisé pour les terminaux mobiles (ad hoc network), grâce notamment à l'élection de tête de cluster (clusterhead) permettant de relier les données échangées avec d'autres nœuds.

ANALYSE

1. Difficultés

Les difficultés de ce projet étaient essentiellement dues à la compréhension de l'algorithme que nous devions implémenter. La différence de langue et la concision des explications sont à l'origine d'une phase de compréhension allongée.

Pour la partie des tests de performances, nous devions, dans un premier temps, implémenter en C++ l'algorithme puis l'intégrer au logiciel de test de performances sous linux, mais ce logiciel ne disposant que de peu de documentations en ce qui concerne son développement et un code source très peu commenté nous avons préféré développer notre propre solution en Java. Nous nous sommes, par conséquent, concentré sur l'implémentation de l'algorithme WCA auxquelles à dû s'ajouter l'implémentation de jeux de tests du fait que nous n'utilisions pas le programme créer sous linux.

2. Architecture du projet

Le projet repose sur le langage Java et utilise cinq classes listées ci-dessous :

Main.java : Classe principale du projet. C'est elle qui appel, instancie, et exécute les différentes classes du projet. Nous reviendrons plus en détail sur cette classe plus tard dans ce rapport.

Nœud.java : Implémente la notion de nœud tel qu'il est défini par l'algorithme WCA. Nous reviendrons plus en détail sur cette classe dans la partie "*Détail de la classe Nœud*".

Constantes.java : Définie toutes les constantes de l'application. Vous retrouverez ci-dessous la liste de ces constantes :

int NBNŒUDS=15 : Nombre de nœud.

int NBTEMPS=10 : Temps pendant lequel les nœuds évolueront.

double w1=0.7 : Constante qui pondère le nombre de voisins d'un nœud.

double w2=0.02 : Constante qui pondère la somme des distances des voisins d'un nœud.

double w3=0.05 : Constante qui pondère la vitesse moyenne du nœud

double w4=0.18 : Constante qui pondère le nombre de fois qu'un nœud à été Clusterhead.

int gamma=4 : Nombre de voisin max d'un clusterhead.

int CANVASWIDTH= 800 : Taille de la fenêtre graphique en largeur.

int CANVASHEIGHT= 600 : Taille de la fenêtre graphique en hauteur.

JCanvas.java : Classe permettant la représentation graphique du projet.

GUIHelper.java : Classe utilisée pour afficher des dessins 2D en Java.

3. Détails de la classe Nœud

Nous allons dans cette partie expliquer comment l'implémentation de WCA à été réalisée. Pour cela nous verrons la classe Nœud avec ses attributs et ses méthodes. Puis nous expliquerons l'algorithme WCA avec le détail de l'implémentation des différentes étapes.

La première étape à réaliser est la définition de ce qu'est un nœud. Pour cela nous avons créé une classe "Nœud" possédant les caractéristiques définies ci-dessous.

Liste des variables d'instances de la classe "Nœud" :

int id : identifiant du nœud. Il s'agit d'un nombre unique assigné à un nœud afin de l'identifier.

int x : Entier représentant la position sur l'axe des abscisses du nœud.

int y : Entier représentant la position sur l'axe des ordonnées du nœud.

int tr : Correspond à la portée du nœud, appelé "*Transmission range*" dans le WCA.

int vx : Vélocité sur l'axe des *x* du nœud (car un nœud est mobile). Correspond au déplacement sur l'axe des abscisses du nœud à chaque nouveau temps "*t*".

int vy : Vélocité sur l'axe des *y* du nœud. Correspond au déplacement sur l'axe des ordonnées du nœud à chaque nouveau temps "*t*".

double[][] listeVoisinDistance : Tableau contenant la liste des nœuds voisins, identifiés par leur id, ainsi que la distance qui les séparent d'avec le nœud.

int dv : Contient le nombre de voisins du nœud, aussi appelé degré ("*degree*"). C'est à dire le nombre de nœuds qui sont situés à une distance inférieur à la portée du nœud.

int deltav : Contient la différence de degré (“*degree-difference*”) résultant de la valeur absolu de la différence du degré du nœud (dv) par une constante.

double Dv : Contient la somme des distances des voisins du nœud (“*sum of the distances*”).

double Mv : Contient la vitesse moyenne du nœud. C’est à dire la quantité de déplacement du nœud par unité de temps *t*.

double Pv : Contient le temps cumulé du nœud passé à être la tête de cluster (“*clusterhead*”).

double Wv : Contient le poids combiné (“*combined weight*”) de toutes les variables précédentes par des coefficients constant.

boolean estClusterhead : Booléen spécifiant si le nœud est défini comme étant tête de cluster (“clusterhead”).

La principale activité de WCA consiste à définir les valeurs de *listeVoisinDistance*, *dv*, *deltav*, *Dv*, *Mv*, *Pv, Wv* et *estClusterhead* à chaque temps *t* pour un nœud donné, c’est à dire pour un nœud dont les valeurs *x*, *y*, *tr*, *vx* et *vy* ont été fixées. Pour cela, la classe Nœud implémente les méthodes suivantes :

Nœud (int id,int x,int y,int tr,int vx,int vy) : Constructeur de la classe Nœud, il défini les valeurs de *id*, *x*, *y*, *tr*, *vx* et *vy* lors de l’instanciation d’un nouveau nœud.

double calculDistanceVoisin(Nœud n) : Calcule la distance entre le nœud courant et le nœud donné en paramètre.

boolean estVoisinDe(Nœud n) : Détermine si le nœud donné en paramètre, est le voisin du nœud courant. La méthode renvoie vrai, lorsque la distance entre les deux nœuds est inférieure à la portée du nœud courant.

void creerVoisinDistanceSiVoisinAvec(Nœud n) : Ajoute dans le tableau *listeVoisinDistance* l’identifiant et la distance du nœud donné en paramètre si celui est dans la portée du nœud courant.

void calculerDeltav() : Calcule la différence de degré pour le nœud courant.

void calculerDv() : Calcule la valeur de Dv en faisant la somme des distances des voisins présent dans le tableau *listeVoisinDistance*.

void calculerMv(int[][] positions,int k) : Calcule de la vitesse moyenne du nœud courant à partir de la liste des positions de ce nœud au cours du temps.

void calculerWv() : Calcule le poids combiné (*combined-weight*) en fonction des paramètres calculés précédemment et des coefficients w1, w2, w3 et w4 fixés

respectivement à 0.7 , 0.2 , 0.05, 0.05 dans le fichier contenant les Constantes du programme.

La classe Nœud permet ainsi de calculer toutes les valeurs nécessaires au déroulement de WCA.

Maintenant que nous avons défini ce qu'est un nœud, nous allons montrer le déroulement de l'algorithme WCA implémenté par la méthode wca() du programme principale. Afin de rester clair et concis, l'algorithme à été simplifié.

4. Implémentation de l'algorithme WCA

Voici le pseudo-code et les étapes de l'implémentation de l'algorithme WCA :

```
Pour chaque temps t

Pour chaque nœud n1

// Etape 1 : Recherche et Calcul des voisins de chaque nœuds

       Pour chaque noeud n2

              Si n2 différent de n1

// Génération du tableau qui pour chaque voisin de n1 calcul

// La distance grâce à la

// Méthode creerVoisinDistanceSiVoisinAvec  de la classe Noeud.

              n1.creerVoisinDistanceSiVoisinAvec(n2)


// Etape 2 : Calcule la différence de degré (degree-difference) d'un

// noeud, grâce à la

// méthodecalculerDeltav de la classe noeud (qui soustrait une

// constante aux nombre

// de voisin du noeud n1).

              n1.calculerDeltav();


// Etape 3 : Calcule la somme des distances avec ses voisins, grâce à la méthode

// calculerDv() de la classe Noeud.

              n1.calculerDv();
```

```
// Etape 4 : Calcul de Mv la vitesse moyenne

// On stock toutes les positions précédentes du nœud courant

// dans un tableau que l'on nomme "positions"

// Pour chaque position on calcule la distance parcouru par le

// nœud, puis on divise par le temps T courant, afin d'obtenir la

// moyenne. Ceci correspond à la méthode calculerMv de la classe

// Noeud.



// Etape 5 : Calcule de Pv le temps cumulé durant lequel le noeud est clusterhead

// Pour chaque temps t jusqu'au temp courant, on incrémente un compteur à

// chaque fois que la valeur du boolean estClusterhead est vrai.


// Etape  6 : Calcule de Wv, le poid combiné de tous les paramètres précédent

// pondéré par les coefficients w1, w2, w3 et w4


//Fin des calculs sur chaque nœud


// Etape 7 : Election des Clusterheads pour cela on trie le tableau de nœuds à l'aide d'un trie rapide en fonction de la valeur Wv.

// Parcourt le tableau ainsi trié, on défini  estClusterhead à true pour le noeud au poids combiné Wv le plus faible

// On passe à true la variable estVoisinClusterhead des voisins de ce noeud afin qu'ils ne soient pas élu clusterhead même si leur poids est faible et on répète ainsi l'élection d'un autre clusterhead jusqu'à ce que soit la variable estCluster, soit la variable estVoisinClusterhead valent vrai pour chaque nœud.
```

5. Modèle de navigation

Pour rendre plus visuelle l'exécution de WCA nous avons créé une interface graphique, avec la représentation de l'ensemble des nœuds et des clusterheads ainsi que leurs portées et les liens entre les voisins. Notez que nous générons autant de fenêtres qu'il y a de temps (par défaut nous effectuons 10 temps), afin de représenter l'évolution du réseau de communication mobile en fonction du temps.

Vous retrouverez *Fig. 1* une copie d'écran de cette représentation à un temps T donné.

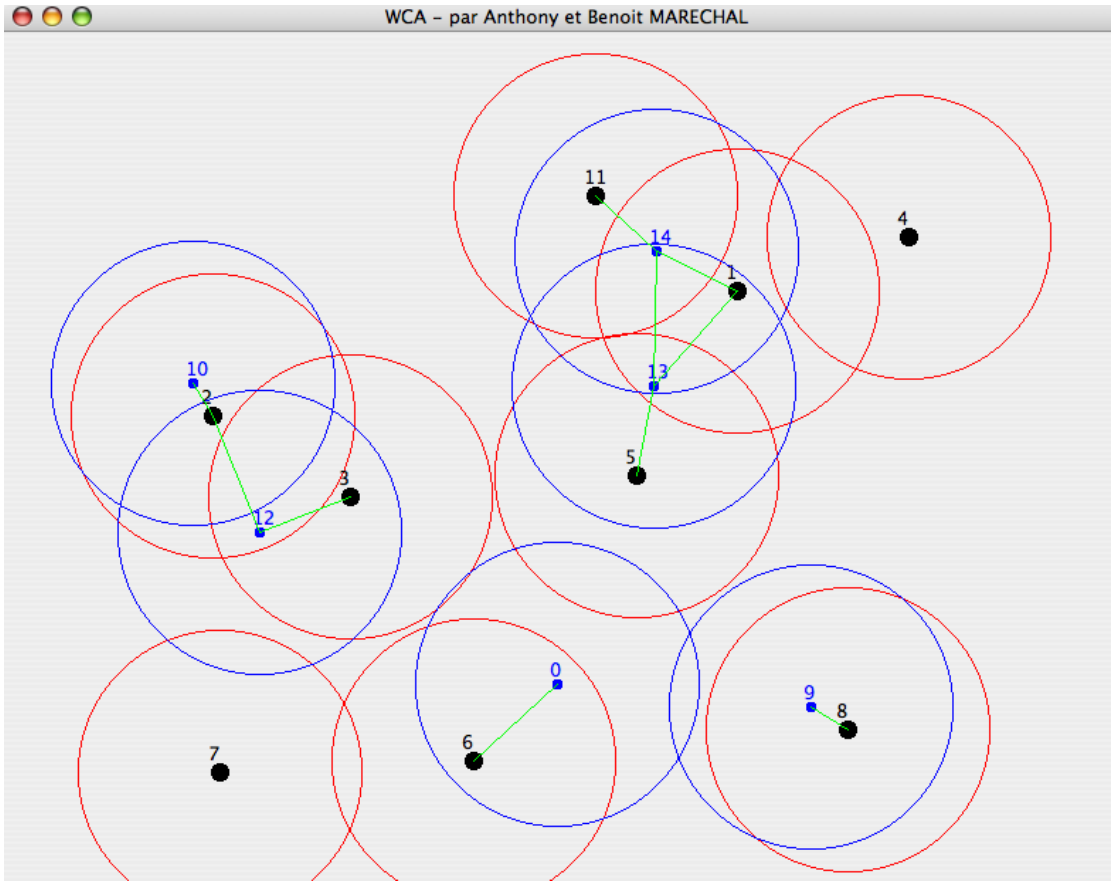

Fig1 : Représentation graphique des nœuds et des clusterheads d'un réseau de communications

Les ronds noirs représentent les nœuds élus Clusterhead selon l'algorithme WCA.

Les ronds bleus sont les nœuds simples.

Les traits verts lient les nœuds identifiés comme étant voisin.

6. Démonstration

Retrouvez une démonstration en ligne de ce projet à l'adresse :

http://www.vimeo.com/673736

Ou télécharger la vidéo dans une meilleure qualité à l'adresse :

http://www.vimeo.com/download/video:31929662

7. Jeux de tests

Retrouvez ci-dessous le produit de l'exécution de l'algorithme WCA. Les résultats montrés ici, sont de la même forme tabulaire que ceux expliqués sur notre sujet en anglais.

id	dv	Deltav	Dv	Mv	Pv	Wv	Clusterhead	VoisinClusterhead	
T = 0									
0	2	2.0	163.31		0.0	0.0	4.67	true	false
1	1	3.0	50.22	0.0	0.0	3.1	true	false	
2	1	3.0	32.25	0.0	0.0	2.74	true	false	
3	2	2.0	159.73		0.0	0.0	4.59	false	true
4	1	3.0	50.22	0.0	0.0	3.1	false	true	
5	3	1.0	223.41		0.0	0.0	5.17	false	true
6	1	3.0	98.84	0.0	0.0	4.08	true	false	
7	0	4.0	0.0	0.0	0.0	2.8	true	false	
8	1	3.0	62.97	0.0	0.0	3.36	true	false	
9	3	1.0	229.79		0.0	0.0	5.3	false	true
10	1	3.0	32.25	0.0	0.0	2.74	false	true	
11	1	3.0	54.04	0.0	0.0	3.18	true	false	
12	2	2.0	176.01		0.0	0.0	4.92	false	true
13	2	2.0	158.28		0.0	0.0	4.57	true	false
14	1	3.0	54.04	0.0	0.0	3.18	false	true	

Comme on peut le voir l'ensemble des paramètres nécessaire à WCA ont été déterminés. On remarque que Mv et Pv valent tout deux zéros pour tous les nœuds ce qui est normal au temps T=0.

T = 1									
0	1	3.0	81.54	0.0	1.0	3.91	true	false	
1	1	3.0	58.69	0.0	0.0	3.27	true	false	
2	1	3.0	30.02	0.0	1.0	2.88	false	true	

3	2	2.0	174.89		0.0	1.0	5.08		false	true
4	1	3.0	58.69	0.0	0.0	3.27		false	true	
5	3	1.0	243.38		0.0	1.0	5.75		false	true
6	0	4.0	0.0	0.0	0.0	2.8		true	false	
7	0	4.0	0.0	0.0	1.0	2.98		true	false	
8	1	3.0	59.36	0.0	1.0	3.47		true	false	
9	2	2.0	148.64		0.0	1.0	4.55		false	true
10	1	3.0	30.02	0.0	0.0	2.7		true	false	
11	1	3.0	48.1	0.0	1.0	3.24		false	true	
12	1	3.0	76.22	0.0	0.0	3.62		true	false	
13	2	2.0	171.23		0.0	0.0	4.82		true	false
14	1	3.0	48.1	0.0	0.0	3.06		true	false	

T = 2

0	1	3.0	91.0	5.0	2.0	4.53		true	false	
1	1	3.0	67.19	128.0	1.0	10.02		false	true	
2	1	3.0	28.07	125.5	1.0	9.12		false	true	
3	1	3.0	75.27	133.0	2.0	10.62		false	true	
4	1	3.0	67.19	13.0	1.0	4.27		true	false	
5	2	2.0	160.78		4.5	1.0	5.02		false	true
6	0	4.0	0.0	18.0	1.0	3.88		true	false	
7	0	4.0	0.0	135.0	1.0	9.73		true	false	
8	1	3.0	55.76	5.0	2.0	3.83		true	false	
9	1	3.0	55.76	24.5	2.0	4.8		false	true	
10	1	3.0	28.07	54.0	1.0	5.54		true	false	
11	1	3.0	43.27	6.5	1.0	3.47		true	false	
12	1	3.0	75.27	27.5	1.0	5.16		true	false	
13	1	3.0	69.78	19.5	0.0	4.47		true	false	
14	1	3.0	43.27	26.0	0.0	4.27		false	true	

T = 3

0	0	4.0	0.0	10.33	3.0	3.86		true	false	
1	1	3.0	75.69	87.67	2.0	8.36		false	true	
2	1	3.0	26.48	170.0	2.0	11.49		false	true	
3	1	3.0	74.33	189.67		2.0	13.43		false	true
4	1	3.0	75.69	14.67	2.0	4.71		true	false	
5	1	3.0	67.42	37.67	2.0	5.69		true	false	
6	0	4.0	0.0	94.67	2.0	7.89		true	false	
7	0	4.0	0.0	173.33		1.0	11.65		true	false
8	1	3.0	52.15	35.67	2.0	5.29		false	true	
9	1	3.0	52.15	16.33	3.0	4.5		true	false	
10	1	3.0	26.48	112.67		2.0	8.62		true	false
11	1	3.0	39.92	75.0	1.0	6.83		false	true	
12	1	3.0	74.33	74.33	2.0	7.66		true	false	
13	1	3.0	67.42	59.0	0.0	6.4		false	true	
14	1	3.0	39.92	28.33	0.0	4.32		true	false	

T = 4

0	1	3.0	98.37	73.5	4.0	8.46		true	false	
1	1	3.0	84.2	141.25		3.0	11.39		false	true
2	1	3.0	25.3	133.5	3.0	9.82		false	true	

3	1	3.0	73.39	145.75		3.0	11.4		false	true
4	1	3.0	84.2	83.25	2.0	8.31		true	false	
5	1	3.0	65.52	42.5	3.0	6.08		true	false	
6	1	3.0	98.37	109.25		2.0	9.89		false	true
7	0	4.0	0.0	194.25		1.0	12.69		true	false
8	1	3.0	48.55	27.25	3.0	4.97		true	false	
9	1	3.0	48.55	40.5	4.0	5.82		false	true	
10	1	3.0	25.3	88.75	2.0	7.4		true	false	
11	1	3.0	38.47	57.25	2.0	6.09		false	true	
12	1	3.0	73.39	118.0	2.0	9.83		true	false	
13	1	3.0	65.52	115.25		1.0	9.35		false	true
14	1	3.0	38.47	21.25	0.0	3.93		true	false	

T = 5

0	1	3.0	93.41	116.2	5.0	10.68		false	true	
1	2	2.0	191.07		175.4	4.0	14.71		false	true
2	1	3.0	24.6	108.2	3.0	8.54		false	true	
3	1	3.0	72.45	151.4	4.0	11.84		false	true	
4	1	3.0	92.72	126.8	2.0	10.65		true	false	
5	1	3.0	64.13	60.2	4.0	7.11		true	false	
6	1	3.0	93.41	111.6	3.0	10.09		true	false	
7	0	4.0	0.0	210.4	2.0	13.68		true	false	
8	1	3.0	44.94	36.4	3.0	5.36		true	false	
9	1	3.0	44.94	80.4	4.0	7.74		false	true	
10	1	3.0	24.6	103.8	3.0	8.32		true	false	
11	1	3.0	39.12	98.2	2.0	8.15		false	true	
12	1	3.0	72.45	100.2	2.0	8.92		true	false	
13	1	3.0	64.13	121.4	1.0	9.63		false	true	
14	2	2.0	137.47		47.8	1.0	6.72		true	false

T = 6

0	1	3.0	89.0	149.67		6.0	12.44		false	true
1	1	3.0	89.54	198.17		5.0	14.7		true	false
2	1	3.0	24.41	107.5	4.0	8.68		true	false	
3	1	3.0	71.51	143.67		4.0	11.43		false	true
4	0	4.0	0.0	107.17		2.0	8.52		true	false
5	1	3.0	63.29	51.17	5.0	6.82		true	false	
6	1	3.0	89.0	104.67		3.0	9.65		true	false
7	0	4.0	0.0	190.67		3.0	12.87		true	false
8	1	3.0	41.34	31.33	3.0	5.03		true	false	
9	1	3.0	41.34	109.67		5.0	9.31		false	true
10	1	3.0	24.41	118.5	4.0	9.23		false	true	
11	1	3.0	41.76	86.17	2.0	7.6		true	false	
12	1	3.0	71.51	106.83		2.0	9.23		true	false
13	1	3.0	63.29	128.17		2.0	10.13		false	true
14	2	2.0	131.3	89.0	1.0	8.66		false	true	

T = 7

0	1	3.0	85.23	129.0	7.0	11.51		false	true	
1	1	3.0	80.78	195.0	6.0	14.55		true	false	
2	1	3.0	24.76	119.86		5.0	9.49		false	true

3	1	3.0	70.58	164.71		5.0	12.65		false	true
4	0	4.0	0.0	92.29	2.0	7.77		true	false	
5	1	3.0	63.0	46.71	6.0	6.78		true	false	
6	1	3.0	85.23	106.57		4.0	9.85		true	false
7	0	4.0	0.0	181.0	3.0	12.39		true	false	
8	1	3.0	37.74	52.71	3.0	6.03		true	false	
9	1	3.0	37.74	95.14	6.0	8.69		false	true	
10	1	3.0	24.76	116.86		4.0	9.16		true	false
11	1	3.0	46.07	91.0	2.0	7.93		true	false	
12	1	3.0	70.58	109.14		2.0	9.33		true	false
13	1	3.0	63.0	111.0	3.0	9.45		false	true	
14	2	2.0	126.84		76.43	2.0	8.12		false	true

T = 8

0	1	3.0	82.2	113.5	8.0	10.86		false	true	
1	1	3.0	72.09	171.38		7.0	13.37		false	true
2	2	2.0	121.43		106.13		6.0	10.21	false	true
3	1	3.0	69.64	147.0	6.0	11.92		false	true	
4	0	4.0	0.0	80.75	2.0	7.2		true	false	
5	1	3.0	63.29	68.13	6.0	7.85		true	false	
6	1	3.0	82.2	107.88		5.0	10.04		true	false
7	0	4.0	0.0	173.75		4.0	12.21		true	false
8	1	3.0	34.13	70.25	3.0	6.84		true	false	
9	1	3.0	34.13	112.88		6.0	9.51		false	true
10	1	3.0	25.61	131.0	5.0	10.06		true	false	
11	1	3.0	51.61	95.38	2.0	8.26		false	true	
12	2	2.0	165.46		102.38		3.0	10.37	true	false
13	1	3.0	63.29	118.38		3.0	9.82		false	true
14	2	2.0	123.7	67.0	3.0	7.76		true	false	

T = 9

0	1	3.0	79.98	101.44		9.0	10.39		false	true
1	2	2.0	152.78		171.89		8.0	14.49	true	false
2	2	2.0	115.32		94.56	7.0	9.69		true	false
3	1	3.0	68.71	151.22		7.0	12.3		true	false
4	0	4.0	0.0	75.11	2.0	6.92		true	false	
5	1	3.0	64.13	61.33	6.0	7.53		true	false	
6	1	3.0	79.98	98.11	5.0	9.51		true	false	
7	0	4.0	0.0	171.44		5.0	12.27		true	false
8	1	3.0	30.53	64.67	4.0	6.66		true	false	
9	1	3.0	30.53	101.33		6.0	8.86		false	true
10	1	3.0	26.93	132.56		6.0	10.35		false	true
11	1	3.0	58.05	85.11	2.0	7.88		true	false	
12	2	2.0	157.1	110.78		3.0	10.62		false	true
13	3	1.0	248.43		125.89		4.0	12.68	false	true
14	3	1.0	216.58		59.67	3.0	8.55		false	true

CONCLUSION

Ce projet nous a permis de comprendre les enjeux des réseaux de communications avec l'exemple du « Weighted Clustering Algorithm ». L'implémentation en Java et la création d'une interface graphique et à été très pratique pour comprendre et vérifier l'intégrité de l'algorithme que nous avons mis en place ce qui nous à permis de nous assurer de répondre exactement au sujet qui nous été demandé en implémentant l'algorithme de manière à ce qu'il suive les mêmes étapes que ce qui nous était expliqué dans le sujet.

BIBLIOGRAPHIE

Au-delà de ce que nous avons pu apprendre et réaliser en CM, TD et TP, plusieurs sources nous ont aidées durant la réalisation de ce projet et du rapport associé, notamment :

Scientific Literature Digital Library
http://citeseer.ist.psu.edu/chatterjee02wca.html

Springerlink
http://www.springerlink.com/content/knme635wbqmhvpx5/