UNIVERSITÉ DU QUÉBEC À MONTRÉAL

RÉALISATION D'UN OUTIL DE SIMULATION DE RÉSEAUX SOCIAUX MULTIPLEXES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
FRANCK GOUDJO

JUIN 2010

Remerciements:

J'aimerais remercier en premier lieu ma conjointe Valérie Bah pour son support constant pendant toute la durée de ma maitrise en informatique à l'Université du Québec à Montréal. J'aimerais également remercier mon directeur de recherche, Monsieur Daniel Memmi pour ses conseils et sa patience ainsi que Emmanuel Lazega qui a bien voulu mettre à notre disposition des matrices de réseaux multiplexes. Pour finir j'aimerais souligner la contribution de Mlle Mélanie Lord qui a pris le temps de m'expliquer en détail le fonctionnement de son logiciel Netsim.

Résumé

Nous présentons dans ce document notre projet de recherche dans le cadre de la maitrise en informatique de l'université du Québec à Montréal. Dans le cadre de cette maîtrise nous avons conçu un langage de modélisation de réseaux sociaux permettant de modéliser plusieurs réseaux sociaux simultanément. Nous voulions que ce langage soit proche du langage naturel afin d'être accessible aux néophytes en informatique. Nous avons également réalisé une plate-forme permettant l'exécution de ce langage. La conception de ce langage est motivée par le fait que généralement au sein d'un ensemble social donné il existe plusieurs types d'interactions sociales simultanément. Par exemple si nous considérons le groupe social constitué par l'ensemble des employés d'une entreprise, il existe souvent des relations de collaboration mais également des relations d'amitié et/ou des relations de conseil entre les employés de l'entreprise. L'étude des réseaux sociaux doit souvent prendre en compte tous les types de relations présentes dans un ensemble social afin de bien comprendre l'évolution de cet ensemble social. Dans le but de mettre en pratique le langage que nous avons conçu nous avons étudié la corrélation entre plusieurs réseaux sociaux. Nous avons également comparé notre plate-forme avec les outils d'analyse et de simulation de réseaux disponibles sur le marché actuellement.

Mots-clés : réseaux sociaux, analyse de réseaux, simulation, réseaux multiplexes

Table des matières

Index des illustrations

Introduction

Ces dernières années les réseaux sociaux semblent être sur toutes les lèvres. Cela est d'autant plus vrai dans le domaine de l'internet. Un grand nombre de sites internet dits de « réseautage » social ont fait leur apparition : Nous pouvons notamment citer Facebook et plus récemment Twitter. Mais que sont donc réellement les réseaux sociaux ? Les réseaux sociaux ont été un domaine de recherche en sociologie depuis de nombreuses années bien avant l'apparition des sites internet de réseautage social. Il existe une branche de la sociologie qui s'intéresse beaucoup aux réseaux sociaux: la sociologie structurale. La sociologie structurale, appelée maintenant analyse de réseaux a développé une grande panoplie de métriques pour caractériser les réseaux sociaux.

Dans le cadre de notre maîtrise en informatique à l'université du Québec à Montréal, nous avons cherché à proposer un langage de modélisation par génération de réseaux sociaux ainsi qu'un interpréteur destiné à exécuter ce langage de modélisation dans le but d'offrir aux analystes de réseaux un outil leur permettant de modéliser et simuler des réseaux sociaux. La spécificité de notre outil consiste dans le fait qu'il est destiné à permettre aux analystes de réseaux sociaux la possibilité de modéliser et manipuler plusieurs réseaux en même temps.

Dans ce document qui rend compte des résultats que nous avons obtenus, nous commencerons d'abord par le chapitre 1 qui est une mise en contexte de notre plate-forme de modélisation de réseaux sociaux et la problématique que nous nous proposons de résoudre. Le chapitre deux sera consacré aux outils que nous avons utilisés dans le cadre de ce projet de recherche. Dans le chapitre trois nous présenterons le langage que nous avons développé. Le quatrième chapitre est consacré au fonctionnement interne de l'interpréteur SNSimulator destiné à interpréter le langage présenté au chapitre trois. Dans le cinquième chapitre nous présenterons quelques exemples d'utilisation du langage SNSimulator dans le cadre de l'analyse de réseaux sociaux. Dans le chapitre six nous comparerons SNSimulator aux principaux outils de modélisation et de simulation disponibles actuellement. La dernière section de ce mémoire de recherche nous permettra de conclure après avoir proposé quelques pistes d'amélioration.

Chapitre 1 :

Mise en contexte et problématique

Dans ce chapitre nous présenterons le contexte des réseaux sociaux et les principaux concepts qu'il nous parait important de connaître afin de comprendre l'univers des réseaux sociaux. Nous parlerons d'abord des réseaux qui nous entourent. Ensuite nous définirons les réseaux sociaux et leurs caractéristiques et spécificités. Après nous présenterons la problématique à laquelle nous nous sommes attaqués dans le cadre de ce projet de recherche. Dans la dernière partie de ce chapitre nous énoncerons les objectifs que nous cherchons à atteindre.

Les réseaux

Les réseaux sont omniprésents dans notre entourage. Par exemple, lorsque nous nous déplaçons nous empruntons le réseau de transports: si nous prenons notre voiture alors nous utilisons le réseau routier, si nous prenons le train, nous empruntons le réseau de voies ferrées et si nous voyageons par avion alors nous utilisons le réseau des voies aériennes. Même notre corps est consitué de réseaux. Nos vaisseaux sanguins constituent par exemple un réseau de plusieurs kilomètres de longueur. Nous pouvons aussi citer les réseaux informatiques comme l'Internet qui regroupe plusieurs millions d'ordinateurs de par le monde. Tous ces réseaux sont très différents les uns des autres mais ils possèdent des caractéristiques communes : ils sont tous constitués de noeuds et de liens. Par exemple dans le cas des réseaux informatiques, les noeuds sont les ordinateurs connectés au réseau et les liens sont les câbles du réseau reliant ces ordinateurs, pour le réseau routier, les noeuds sont les villes et les liens sont les routes qui relient ces villes.

Nous pouvons donc dire qu'un réseau est un ensemble de nœuds ou sommets reliés par des liens. Les noeuds d'un réseau sont caractérisés par le nombre de liens qui les relient aux autres noeuds du réseau. Ce nombre de liens est appelé le degré du noeud.

Les réseaux peuvent être classés en trois groupes principaux : les réseaux aléatoires, les réseaux sans échelle (scale-free) et les réseaux hiérarchiques. Ces groupes de réseaux diffèrent notamment par leur distribution de degré.

La distribution de degré est une fonction qui permet d'obtenir la probabilité, p(k), qu'un nœud donné du réseau aie exactement k liens. Cette probabilité est donnée par $\mathbf{p(k) = X(k) / N}$ avec N le nombre de sommets (noeuds) dans le réseau et X(k) le nombre de sommets du réseau ayant un

degré k.

Dans la suite de ce chapitre nous allons présenter chacun des principaux types de réseaux:

Réseaux aléatoires:

Dans un réseau aléatoire les liens entre les noeuds du réseau sont distribués de manière aléatoire, c'est à dire qu'il existe une probabilité de lien entre deux noeuds du réseau pris au hasard. La distribution des degrés dans un réseau aléatoire suit une loi de Poisson, C'est à dire que la probabilité qu'un noeud quelconque possède un degré x diminue très vite lorsque x augmente. Cette probabilité est définie par :

$$P(x) = \frac{e^{-\lambda}\,\lambda^{x}}{x!}$$

La variable λ est un nombre réel strictement positif qui permet de paramétrer la distribution de probabilité. Pour des valeurs de λ grandes (notament λ =10), la loi de Poisson peut être approximé par la loi normale.

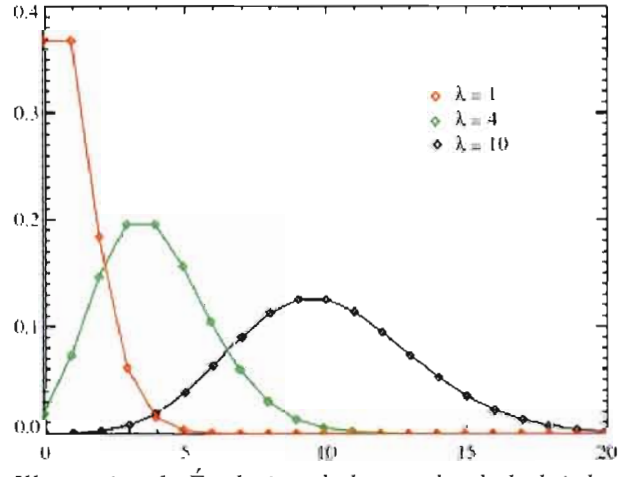

Illustration 1: Évolution de la courbe de la loi de Poisson en fonction de λ

Source : site web Wikipédia.

Les réseaux aléatoires ont été définis par Paul Erdős et Alfréd Rényi, deux mathématiciens

hongrois dans un article intitulé " On Random Graphs" paru en 1959 [25]. Les réseaux aléatoires présentent la propriété des "petits mondes" que nous abordons de manière plus détaillée plus dans ce chapitre, c'est à dire que le chemin entre deux noeuds quelconques est court, mais ils sont beaucoup moins structurés que les réseaux sociaux réels.

Réseaux sans échelle:

Les réseaux sans échelle (scale-free) sont caractérisés par une distribution dc degré suivant une loi de puissance. La loi de puissance est une relation entre deux quantités x et y telle que:

$$y = ax^k$$

La valeur "a" est une constante appelée constante de proportionnalité. L'élément "k" est aussi une constante appelée exposant de la loi. La probabilité de degré dans un réseau sans échelle est généralement donnée par :

$$p(k) \approx k^{-\alpha}$$

Généralement, le terme α est compris entre les valeurs deux et trois dans les réseaux sans échelle réels.

Plusieurs réseaux réels présentent des caractéristiques de réseaux scale-free: nous pouvons notament citer le World Wide Web.

Réseaux hiérarchiques:

Les réseaux hiérarchiques sont des réseaux construits selon une structure hiérarchique le plus souvent à partir d'un plan délibéré. Comme exemples de réseaux hiérarchiques nous pouvons citer certains réseaux techniques comme les réseaux informatiques.

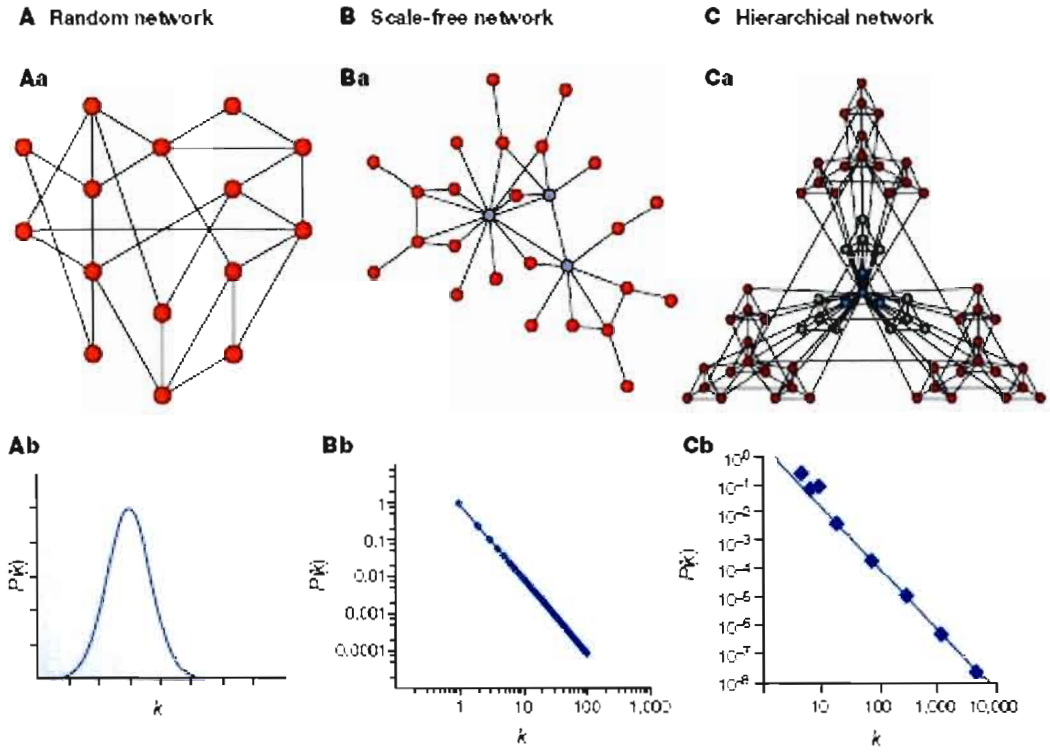

Illustration 2: Illustration de réseaux aléatoires, sans échelle et hiérarchiques.
Source : site web Wikipédia.

Les courbes Ab, Bb et Cb représentent la distribution de degré dans les trois types de réseaux. L'abscisse et l'ordonnée des courbes Bb et Cb sont en logarithme. Nous pouvons constater que la courbe Ab n'est pas un exemple typique de loi de Poisson et est plus proche d'une loi normale.

Les réseaux peuvent être représentés par des graphes qui sont des objets mathématiques constitués de noeuds et de liens entre les noeuds.

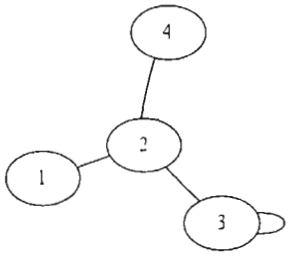

Illustration 3: Exemple de graphe

Les réseaux peuvent également être séparés en réseaux orientés et en réseaux non orientés. Dans les réseaux orientés les liens du réseau possèdent un sens de parcours et des nœuds de départ et d'arrivée. Dans un réseau non orienté les liens ne possèdent aucun sens de parcours particulier et les deux nœuds reliés par un lien sont équivalents. Les réseaux peuvent être représentés par des graphes. Dans le langage de modélisation que nous avons réalisé tous les réseaux créés sont par défaut des réseaux orientés. La plate-forme que nous avons réalisée permet de créer des réseaux sans échelle, aléatoire ou hiérarchique parce qu'elle permet de préciser des règles de création de liens dans les réseaux simulés. Par ces règles l'utilisateur peut obtenir une distribution de degré donnée.
Dans le cadre de la réalisation de notre outil de modélisation de réseaux nous nous sommes plus particulièrement intéressés à un type particulier de réseaux, les réseaux sociaux.

Les réseaux sociaux

Un réseau social est un ensemble d'individus ou groupes reliés entre eux par des liens créés lors d'interactions sociales. Une interaction au sens social du terme est un échange d'informations ou d'énergie entre deux agents au sein d'un système.
Emmanuel Lazega dans son ouvrage «Réseaux sociaux et structures relationnelles» [27] appelle réseau social un système de relations entre les membres d'un ensemble social.
Pour simplifier, on peut définir un réseau social comme tout ensemble constitué d'individus ou d'ensemble d'individus. Un réseau social est un type particulier de réseau. Les éléments d'un réseau social peuvent avoir ou ne pas avoir d'interactions les uns avec les autres. Par exemple un groupe d'amis, une entreprise, une classe dans une école primaire, l'ensemble des usagers d'un site internet sont tous des réseaux sociaux.
Les réseaux sociaux possèdent plusieurs caractéristiques très intéressantes qui méritent que nous les abordions: l'attachement préférentiel et l'effet du petit monde.

L'attachement préférentiel qui a été étudié par Barabasi & Albert [14] est une propriété souvent rencontrée surtout dans les réseaux de type sans échelle mais qui est également présente dans certains réseaux sociaux selon laquelle les nœuds du réseau établissent des liens de préférence avec les nœuds qui possèdent le plus de liens. Par exemple si nous considérons l'ensemble des chercheurs en informatique de la planète, un chercheur aura tendance à citer en priorité les

chercheurs qui sont le plus cités par les autres. Donc plus un chercheur est cité plus il aura tendance à être cité. Ce comportement donne des réseaux présentant une distribution de degré selon une loi de puissance.

L'effet du petit monde étudié par Stanley Milgram [35] est une hypothèse sociologique selon laquelle chaque individu sur terre peut être relié à n'importe quel autre individu par une courte chaine de relations sociales. L'effet du petit monde est illustré par la théorie des six degrés de séparation qui est une hypothèse élaborée par le hongrois Frigyes Karinthy selon laquelle toute personne peut être reliée à toute autre personne par une chaine de relations comprenant au plus six individus au total. L'effet du petit monde a des conséquences importantes pour la recherche sociale d'information, la diffusion des rumeurs, la propagation des épidémies etc.
On remarque que les réseaux sociaux réels sont difficiles à modéliser avec les modèles formels que nous avons vus précédemment. Une approche informatique est alors indiquée, mais on dispose aussi de mesures spécifiques sur ces réseaux.

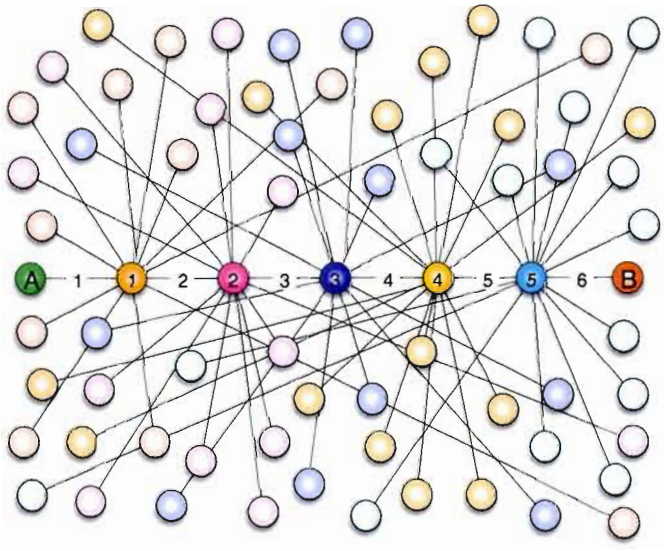

Illustration 4: Illustration de la théorie des six degrés de séparation

Source site web Wikipédia

L'analyse de réseau est une approche sociologique qui a pour objectif d'étudier les réseaux sociaux en tant qu'ensembles de nœuds représentant les entités sociales ou acteurs et de liens entre ces nœuds. L'analyse de réseau cherche à caractériser les propriétés spécifiques aux graphes

des réseaux sociaux. Pour cela l'analyse de réseau a développé de nombreuses métriques comme la centralité et le prestige. Ces deux métriques permettent de mesurer l'importance d'un acteur donné dans un réseau. La mesure de centralité est d'autant plus élevée que l'acteur est engagé directement ou indirectement dans beaucoup de relations. Le prestige quant à lui mesure la popularité d'un acteur. Le prestige est une métrique qui tient compte du sens des liens dans un réseau. Plus l'acteur considéré reçoit de liens des autres membres du réseau plus sa mesure de prestige sera élevée. Dans les lignes qui suivent nous allons examiner plus en détail quelques autres métriques d'analyse de réseau. Nous allons également présenter les principaux concepts de la théorie des graphes. Pour ces concepts nous nous sommes inspirés de l'ouvrage d'Emmanuel Lazega, « Réseaux Sociaux et Structures Relationnelles »[27].

Concepts et métriques en analyse de réseaux sociaux

La centralité :

La centralité d'une entité sociale dans un réseau social mesure l'importance relative de cette entité dans le réseau. Il existe différents types de centralité selon la définition de la notion d'importance d'une entité dans un réseau social.

La centralité de degré :

Ce concept mesure le nombre de liens entre un nœud et les autres nœuds du réseau. La centralité de degré d'une entité est d'autant plus élevée que cette entité a de liens avec les autres acteurs du réseau.

La centralité d'intermédiarité :

La centralité d'intermédiarité exprime le fait que les entités sociales du réseau dépendent de l'entité dont on cherche à déterminer la centralité d'intermédiarité pour interagir les uns avec les autres. La centralité d'intermédiarité d'une entité est d'autant plus élevée que les autres acteurs doivent passer par ce nœud pour avoir des interactions sociales.

La centralité de proximité :

La centralité de proximité mesure le nombre minimum de pas que doit effectuer une entité sociale pour entrer en contact avec les autres entités sociales du réseau.

La centralité de prestige:

La centralité de prestige mesure le nombre de liens entre le nœud considéré et les autres nœuds du réseau. À la différence des métriques précédentes la centralité de prestige tient compte de la direction des interactions sociales. Plus un acteur reçoit d'interactions plus il est prestigieux.

D'autres notions sont également utilisées en analyse de réseaux :

Les sous-groupes cohésifs :

Un sous-groupe cohésif est un sous-ensemble des entités d'un réseau entre lesquelles existent des relations fortes. Il existe essentiellement trois types de sous-groupes cohésifs: les sous-groupes basés sur la réciprocité, les sous-groupes basés sur l'accessibilité et le diamètre et enfin les sous-groupes basés sur le nombre de membres adjacents.

Les cliques:

Une clique dans un réseau social est un ensemble d'entités sociales ayant toutes des interactions directes les unes avec les autres. Cette notion peut être généralisée aux n-cliques qui sont des ensembles d'entités sociales ayant toutes des interactions directes ou indirectes de longueur maximale de n intermédiaires les unes avec les autres.

Les dyades :

Une dyade représente un ensemble de deux entités sociales à l'intérieur d'un réseau social.

Les triades :

Les triades sont un ensemble de trois entités sociales à l'intérieur d'un réseau social.

Équivalence structurale:

Deux acteurs d'un réseau sont structuralement équivalents s'ils ont des relations identiques avec les autres acteurs du réseau. Étant donné qu'il est quasiment impossible de trouver deux personnes dans un réseau ayant exactement les mêmes relations avec les autres membres du réseau, la détermination des acteurs structuralement équivalents s'effectue de manière statistique. Deux méthodes sont surtout utilisées : la méthode des *blockmodels* qui s'appuie sur la corrélation entre deux acteurs du réseau; la seconde méthode s'appuie sur la distance euclidienne entre deux acteurs.

Équivalence régulière:

Deux individus sont régulièrement équivalents s'ils ont au moins une relation avec des individus eux-mêmes équivalents. Par exemple, les étudiants et les professeurs sont deux ensembles équivalents.

L'analyse de réseaux sociaux pour être fiable doit tenir compte de la corrélation qui peut exister entre les différentes variables étudiées. Deux principaux modèles statistiques sont utilisés:

Le modèle P2:

Le modèle P2 est un modèle basé sur les attributs de chaque acteur du réseau en matière de propension à choisir d'autres acteurs, de propension à être choisi par d'autres, la propension à faire des choix réciproques et la tendance moyenne à interagir avec les autres. Ces attributs sont appelés respectivement α, β, ρ, μ. Le modèle P2 permet de formaliser ces caractéristiques de manière statistique.

Le modèle P :*

Le modèle p* est une extension au modèle p2 qui incorpore des caractéristiques structurales.

Théorie des graphes

Étant donné l'importance des graphes pour la modélisation des réseaux, voici un bref rappel des notions de base. C'est notions sont tirées de l'ouvrage « Théorie des graphes »de Jacques Labelle [2].

Graphe simple:

Un graphe simple est un ensemble de nœuds ou sommets et des liens ou arêtes reliant un nœud de l'ensemble à un autre nœud de l'ensemble.

Graphe non orienté:

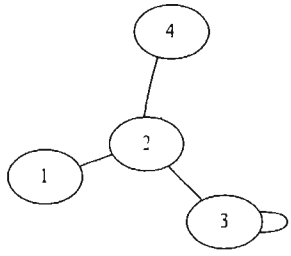

Illustration 5: Exemple de graphe non orienté

Graphe orienté:

Un graphe orienté est un graphe dont les arêtes sont orientées. Chaque arête possède une extrémité initiale et une extrémité finale.

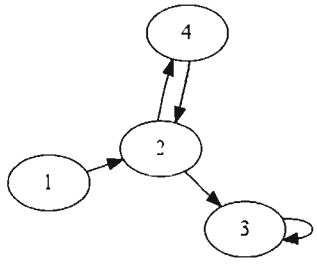

Illustration 6: Exemple de graphe orienté

Arêtes:

Une arête est un lien dans un graphe non orienté.

Arcs:

Un arc est un lien dans un graphe orienté. Un arc possède un nœud de départ et un nœud d'arrivée.

Sommets adjacents:

Deux sommets d'un graphe reliés par une arête A sont appelés sommets adjacents. On dit également que ces sommets sont incidents à A.

Ordre d'un graphe:

Le nombre de sommets d'un graphe G est appelé l'ordre de G.

Degré d'un sommet d'un graphe:

Le nombre d'arêtes d'un graphe G incidentes à un sommet x est appelé le degré du sommet x.

Sous-graphe:

Un sous-graphe H d'un graphe G est un graphe tel que tous les sommets de H sont aussi des sommets de G et tel que toutes les arêtes de H sont aussi des arêtes de G.

Graphe partiel :

Un graphe partiel H d'un graphe G est un graphe tel que tous les sommets de H sont aussi sommets et que tous les sommets de G sont aussi sommets de H et que toutes les arêtes de H sont aussi des arêtes de G. Il est possible que des arêtes de G ne soient pas aussi des arêtes de H.

Graphe régulier:

Un graphe simple G est dit régulier de degré r si tous les sommets de G sont de degré r.

Représentation informatique des graphes

Afin de pouvoir effectuer sur les graphes les traitements souhaités, il est nécessaire de les convertir dans un format accessible aux ordinateurs. Il existe quatre principaux formats de représentation de graphes:

Matrices d'adjacence

Une matrice d'adjacence est une matrice carrée dont les lignes et les colonnes représentent les nœuds du graphe correspondant. L'élément de la matrice au croisement de la ligne i et de la colonne j sera à 1 ou à vrai (V) s'il existe un lien entre les nœuds i et j et à zéro ou faux (F) sinon. Dans le cas d'un graphe non orienté, la matrice d'adjacence est une matrice symétrique alors que la matrice d'adjacence d'un graphe orienté n'est souvent pas symétrique. Dans une matrice d'adjacence d'un graphe orienté, les colonnes représentent les nœuds de départ des liens du graphe et les lignes représentent les nœuds d'arrivée des liens.

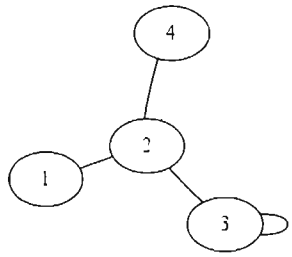

Illustration 7: Graphe non orienté et matrice d'adjacence correspondante (à droite)

	1	2	3	4
1	F	V	F	F
2	V	F	V	V
3	F	V	V	F
4	F	V	F	F

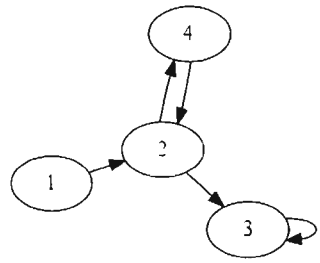

Illustration 8: Graphe orienté et matrice d'adjacence correspondante (à droite)

	1	2	3	4
1	F	V	F	F
2	F	F	V	V
3	F	F	V	F
4	F	V	F	F

Matrices d'incidence

Dans une matrice d'incidence, les lignes représentent les sommets alors que les colonnes représentent les arêtes du graphe représenté. L'élément de la matrice au croisement de la ligne i et de la colonne j sera à 1 ou vrai (V) si l'arête j est incidente au nœud i. Dans le cas d'un graphe orienté, l'élément de la matrice au croisement de la ligne i et de la colonne j sera à 1 si l'arc j possède comme nœud d'arrivée le nœud i et -1 si le nœud i est le nœud de départ. Les boucles (arcs dont le nœud d'arrivée et le nœud de départ sont les mêmes) peuvent être représentés par zéro. D'une manière générale, il est possible de choisir trois valeurs aléatoires a, b et d pour représenter les nœuds de départ, d'arrivée et les boucles dans une matrice d'incidence orientée.

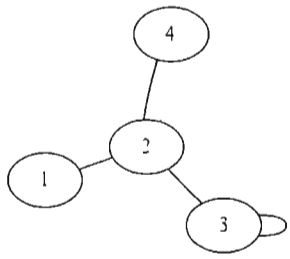

Illustration 9: Graphe non orienté et matrice d'incidence correspondante (à droite)

	(1,2)	(2,3)	(2,4)	(3,3)
1	V	F	F	F
2	V	V	V	F
3	F	V	F	V
4	F	F	V	F

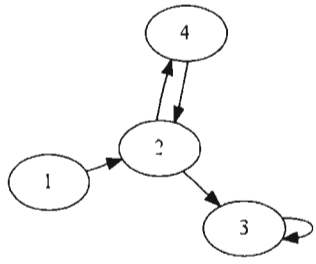

Illustration 10: Graphe orienté et matrice d'incidence correspondante (à droite)

	(1,2)	(2,3)	(2,4)	(3,3)	(4,2)
1	d	$-$	$-$	$-$	$-$
2	a	d	d	$-$	a
3	$-$	a	$-$	b	$-$
4	$-$	$-$	a	$-$	d

Listes d'adjacence

Une liste d'adjacence est un tableau (ou une liste chaînée) contenant autant de cases que le graphe représenté a de sommets. Chaque case de ce tableau pointe sur une liste de sommets qui représente la liste des sommets successeurs du nœud correspondant à la case considérée. Par exemple dans l'illustration 11 de la page suivante, la case 1 qui représente le sommet 1 pointe sur les cases 2,3 et 4 qui représentent les sommets 2,3 et 4 ce qui signifie que les sommets 2, 3 et 4 sont successeurs du sommet 1 dans le graphe. Dans une liste d'adjacence les sommets ne sont pas forcément consécutifs (un sommet y peut être séparé d'un autre sommet x par un troisième sommet z tout en restant un sommet successeur de x comme le montrent les sommets 3 et 4 de l'illustration 11 par rapport au sommet 1).

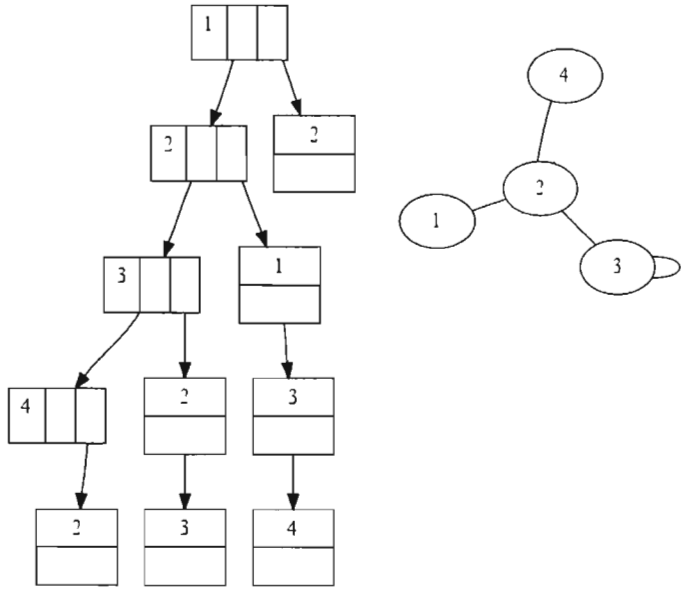

Illustration 11: Exemple de liste d'adjacence du graphe non orienté présenté à droite

Listes d'incidence

Dans une liste d'incidence chaque case du tableau pointe sur la liste d'arêtes incidentes au sommet considéré.

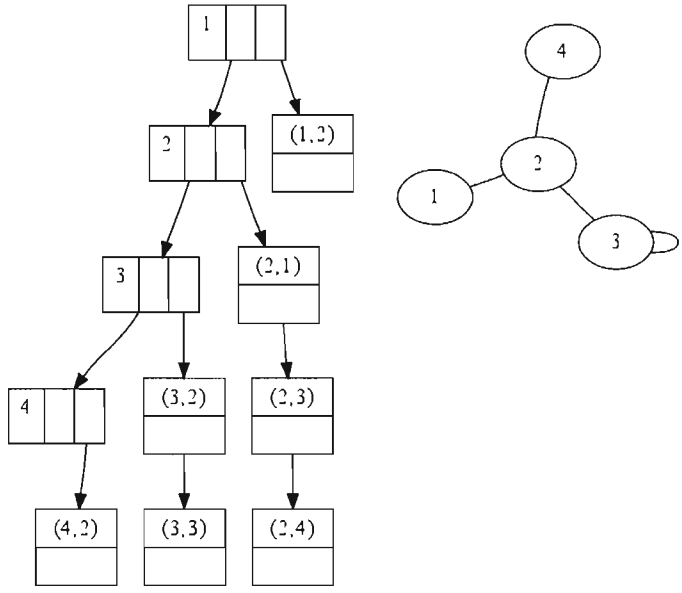

Illustration 12: Exemple de liste d'incidence du graphe non orienté présenté à droite

La représentation en liste d'incidence ou d'adjacence est plus économique en termes d'occupation mémoire dans le cas de graphes peu denses (graphes ayant un nombre peu élevé de liens). En effet la représentation matricielle de ce type de graphes donne une matrice creuse comportant beaucoup de zéros. Même si les zéros ne sont pas significatifs, leur représentation nécessite un espace mémoire non négligeable. Dans le cas de grands réseaux denses, la représentation matricielle est plus économique en terme d'espace mémoire. Pour des raisons de simplicité nous avons privilégié dans le cadre de ce travail de recherche une représentation sous forme de matrice d'adjacence des réseaux manipulés.

Dans les exemples de graphes que nous avons présentés jusqu'ici dans ce chapitre nous avions généralement un lien et un seul entre deux nœuds mais en général, plusieurs types d'interactions sociales prennent place au sein d'un réseau social donné. Par exemple si on considère les employés d'une entreprise comme un réseau social, ce réseau est le siège de relations de collaboration professionnelle mais également aussi d'autres relations comme des relations d'aide, de conseil etc. En fait plusieurs réseaux sociaux coexistent au sein de l'ensemble social que représente l'entreprise considérée. L'ensemble des réseaux sociaux existant au niveau d'un même système social est appelé un système d'échange généralisé ou encore réseau social multiplexe ou

multigraphe. Le logiciel que nous prévoyons de réaliser permettra de modéliser et de simuler les systèmes d'échanges généralisés. Dans le paragraphe suivant nous préciserons la problématique que nous souhaitons résoudre dans le cadre de notre projet de recherche.

La problématique

Dans le cadre de leurs travaux sur les réseaux sociaux, les sociologues ont à calculer et traiter de nombreuses informations comme les différentes valeurs de centralité, de prestige, etc. Pour ce faire ils s'appuient de plus en plus sur des moyens informatiques.

De plus, pour ajouter plus de réalisme dans l'étude de réseaux sociaux, il est nécessaire de s'intéresser à toutes les relations y compris les relations simultanées existant au sein d'un même ensemble social et à l'interaction entre ces relations. Par exemple, dans le cas d'une entreprise, les employés peuvent avoir plusieurs relations simultanées comme des relations d'amitié, de collaboration et de conseil. Il faut donc des outils permettant de manipuler ces réseaux (ensembles de relations) dans lesquels deux nœuds (par exemple deux employés) peuvent être reliés par plusieurs liens (relations). Nous avons examiné de nombreux logiciels de modélisation et de simulation, mais il apparait que les outils disponibles actuellement ne permettent à notre connaissance que de modéliser une seule relation sociale à la fois, nous pouvons notamment prendre le cas de Netsim qui est un logiciel de simulation de réseaux d'information réalisé par Mélanie Lord à l'Université du Québec à Montréal dans le cadre de sa maitrise en informatique qui permet de simuler des réseaux d'information mais qui ne traite qu'un seul réseau à la fois [40], et ne prennent pas en compte les interactions entre les relations. Les outils que nous avons étudiés et nos remarques à propos de chacun d'eux sont détaillés au chapitre 6 de ce mémoire. Sur la base de ce constat, nous pensons qu'il existe un besoin d'un outil simple d'usage qui permettrait d'étudier les interactions entres différente relations dans un ensemble social. Cet outil aiderait à répondre de la manière la plus simple possible à des questions qui font intervenir plusieurs relations au sein d'un groupe social telles que: Comment évoluent les relations d'amitié entre un groupe de collègues de bureau? Ou encore si nous considérons un groupe d'usagers de Facebook dont une partie est aussi utilisateur de Twitter, combien de temps prendront tous les membres du groupe pour devenir adeptes de Twitter? Nous pensons que l'étude de réseaux sociaux serait grandement facilitée par l'existence d'une plate-forme spécialisée dans la modélisation de réseaux et permettant de modéliser simultanément toutes les relations dans un ensemble social afin de faire une étude de leurs interactions. Dans le cadre de ce travail notre but est d'apporter un début de solution à ce problème par la réalisation d'une plate-forme de simulation de réseaux sociaux,

multiplexes.

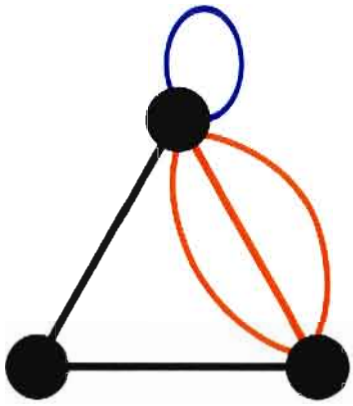

Illustration 13: Exemple de réseau comportant plusieurs relations

source site web Wikipédia

Maintenant que nous avons présenté la problématique à laquelle nous nous attaquerons, nous allons passer en revue les objectifs que nous désirons atteindre.

Objectifs

L'objectif poursuivi dans le cadre de ce projet de recherche est de proposer une plate-forme permettant de réaliser la modélisation, l'analyse, la simulation et la visualisation de structures relationnelles complexes comme des réseaux multiplexes. Notre but est de fournir aux chercheurs en réseaux sociaux un outil leur permettant dans le cadre de leurs travaux de générer et manipuler des réseaux multiplexes afin de pouvoir étudier de façon plus poussée les différentes interactions au sein d'un groupe social. Les réseaux multiplexes sont des réseaux au sein desquels deux nœuds peuvent être reliés par plusieurs liens simultanément. Nous allons donc réaliser un interpréteur en langage Java destiné à exécuter le langage spécialisé dans la manipulation de réseaux sociaux que nous avons conçu dans le cadre de ce projet de recherche. Notre approche de manipulation de multigraphes consistera à traiter chaque type de relation dans un ensemble social comme un réseau à part entière et l'outil que nous nous proposons de réaliser permettra de définir plusieurs réseaux simultanément et de définir les interactions entre ces réseaux.

Dans ce chapitre nous avons présenté quelques concepts utiles du domaine des réseaux sociaux ainsi que la problématique à laquelle nous nous sommes attaqués dans notre projet de recherche. Dans les chapitres suivants nous allons présenter la solution à la problématique discutée ci-haut c'est à dire le langage de modélisation de réseaux sociaux et l'interpréteur pour ce langage, SNSimulator. Dans le chapitre 2 nous allons présenter les outils qui nous ont aidés à réaliser SNSimulator. Le chapitre 3 sera consacré au langage interprété par SNSimulator et nous y présenterons les principales instructions de ce langage. Le chapitre 4 quant à lui présente le fonctionnement interne de SNSimulator. Dans le chapitre 5 nous allons présenter un exemple de programmation avec SNSimulator et dans le chapitre 6 nous ferons une revue des outils de simulation et de modélisation actuellement disponibles et nous les évaluerons par rapport à la problématique de notre projet de recherche et à notre outil SNSimulator.

Chapitre 2:

La plateforme SNSimulator

Dans ce chapitre nous allons présenter l'outil de modélisation et de simulation de réseaux sociaux que nous avons réalisé. Nous appellerons la plate-forme que nous avons développé *SNSimulator* comme *Social Networks Simulator.* Nous allons commencer par examiner les caractéristiques que notre outil doit satisfaire afin de pouvoir être utilisé pour la modélisation de réseaux sociaux multiplexes. Ensuite nous présenterons les outils dont nous nous sommes servis pour réaliser SNSimulator.

Caractéristiques attendues de la plate-forme proposée

Nous cherchons à réaliser un outil informatique répondant aux critères suivants :

Capacité à gérer plusieurs réseaux sociaux :

Notre SNSimulator doit pouvoir permettre à l'utilisateur de définir plusieurs réseaux définis sur les mêmes nœuds et pouvoir manipuler ces réseaux. L'utilisateur doit donc pouvoir accéder à ces réseaux par leur nom.

Capacité à définir des règles:

L'usager doit pouvoir définir des règles s'appliquant à un ou plusieurs des réseaux définis dans la simulation.

Capacité à définir des propriétés :

L'usager doit pouvoir définir des propriétés pour les éléments de la simulation. Notamment il doit être possible de définir des propriétés aux nœuds et aux liens du ou des réseaux de la simulation. L'usager doit également pouvoir affecter des valeurs aux propriétés définies, accéder aux valeurs

et les modifier.

Capacité à définir des fonctions :

L'usager doit pouvoir définir des fonctions afin de pouvoir regrouper plusieurs actions sur les éléments de la simulation ensemble. Les fonctions doivent pouvoir être appelées à l'intérieur des règles définies dans la simulation.

Capacité à définir des procédures:

Nous souhaitons également permettre à l'usager de pouvoir définir des procédures. Ces procédures peuvent par exemple permettre de regrouper plusieurs actions sur les éléments de la simulation. Les procédures pourront être appelées de l'intérieur des règles. La principale différence entre les fonctions et les procédures sera que les fonctions retourneront une valeur alors que les procédures non.

Capacité à définir des variables:

SNSimulator devra permettre à l'usager de définir des variables globales ou des variables locales au sein des fonctions ou des procédures ou des règles.

Capacité à définir des constantes :

SNSimulator devra également permettre à l'usager de définir des constantes globales ou locales.

Langage facile à apprendre et à utiliser:

Le langage que nous allons créer devra être le plus proche possible du langage naturel afin de pouvoir être utilisé facilement par une personne n'ayant aucune formation préalable en informatique.

Représentation des réseaux sociaux

Dans le cadre de la réalisation de l'outil SNSimulator, nous avons choisi d'utiliser le format de matrice d'incidence comme format par défaut des réseaux manipulés.

Dans le paragraphe précédent nous avons parlé des caractéristiques que nous voulons donner au langage SNSimulator. Dans la suite de ce chapitre nous allons présenter les principaux outils qui nous ont aidé à réaliser SNSimulator.

Outils utilisés pour réaliser SNSimulator

SNSimulator est un logiciel réalisé en langage Java. Afin de pouvoir réaliser une application répondant aux caractéristiques attendues de la plate-forme telles que précisées précédemment, nous avons utilisé un certain nombre d'outils existants notamment Javacc, la librairie Jung et l'environnement de développement intégré NetBeans. Dans les lignes qui suivent nous présenterons de manière plus détaillée chacun d'eux.

Javacc:

Javacc est un logiciel libre destiné à permettre la construction de compilateurs en langage Java. Javacc signifie Java Compiler Compiler ou compilateur de compilateurs java. Java. Il fournit un analyseur syntaxique et un analyseur lexical à partir de la description du langage passée en paramètre.

Un analyseur lexical ou lexer est un programme informatique qui à partir de la description d'un fichier texte en entrée ou de la ligne de commande est capable de reconnaitre certains mots appelés lexèmes. Les lexèmes à reconnaitre sont définis préalablement dans un fichier séparé selon une syntaxe ou grammaire propre à l'analyseur lexical.

L'analyseur syntaxique ou parser comme son nom l'indique reconnaît la syntaxe d'un langage. Cette syntaxe est définie dans un fichier texte passé en paramètre au parser.

L'analyseur syntaxique et l'analyseur lexical coopèrent de la manière suivante : Le parser essaye toujours de reconnaître un des éléments syntaxiques définis dans le fichier de syntaxe. Pour cela il appelle l'analyseur lexical pour obtenir un lexème. Si le lexème obtenu correspond à un des éléments syntaxiques attendus alors le parser appelle de nouveau le lexer afin d'obtenir un autre lexème. Si la succession de lexèmes obtenus ne correspond à la syntaxe attendue alors le parser affiche un message d'erreur.

JJTree :

JJTree est une extension au logiciel Javacc permettant à partir de la grammaire du langage à laquelle on a ajouté des annotations, de produire un arbre syntaxique abstrait.
Un arbre syntaxique abstrait est une représentation intermédiaire de la syntaxe d'un programme après son analyse par l'analyseur syntaxique et avant la vérification sémantique et/ou la production de code machine ou l'exécution dans le cas d'un interpréteur.

Jung:

Jung pour Java Universal Network/Graph framework est une librairie de visualisation de graphes écrite en Java. Il permet la visualisation aussi bien de graphes orientés que de graphes non orientés. Jung est disponible sous la licence open source.

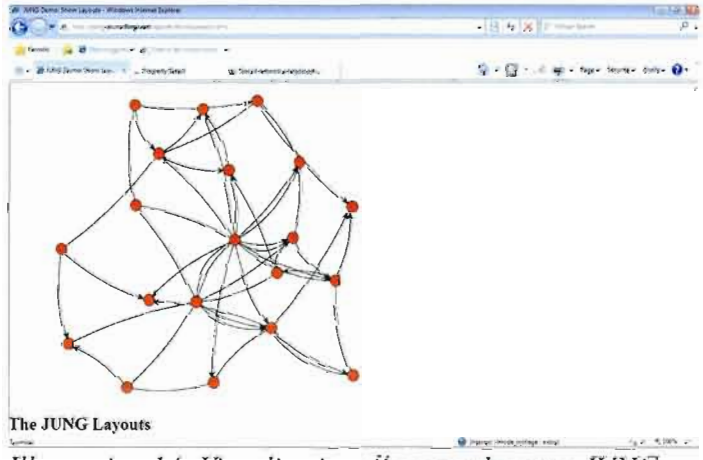

Illustration 14: Visualisation d'un graphe avec JUNG
Source site web de la librairie JUNG

NetBeans :

NetBeans est un environnement de développement intégré (IDE) open source disponible sous la licence Common Developpement and Distribution Licence (CDDL) de Sun Microsystems. Son rôle est de faciliter la programmation informatique notamment en langage Java.

Dans ce chapitre nous avons présenté les caractéristiques du langage SNSimulator et nous avons aussi présenté Javacc, Jung et NetBeans qui sont les outils dont nous nous sommes servis pour réaliser SNSimulator. Dans le chapitre 3 qui suit nous allons présenter en détail le langage SNSimulator que nous avons conçu.

Chapitre 3 :

Le langage de SNSimulator:

Le langage développé pour SNSimulator est inspiré du langage ADA et du langage PL/SQL d'Oracle. Ces deux langages ont été choisis comme base de travail car ils sont plus simples que plusieurs autres langages et sont donc selon nous plus faciles à apprendre.

Chaque programme SNSimulator est constitué de deux parties : La première partie contient les déclarations. Elle commence avec le mot-clé DECLARE et se termine lorsque la seconde partie du programme commence avec le mot-clé BEGIN. La seconde partie contient le corps du programme. La seconde partie d'un programme SNSimulator contient les instructions à exécuter avant et pendant la simulation. Cette partie commence par le mot-clé BEGIN et se termine par le mot-clé END. Dans les lignes ci-dessous nous allons examiner successivement la section déclaration et le corps des programmes SNSimulator et nous allons préciser pour chacune de ces deux parties les différents éléments qui la composent.

Section déclaration

Dans cette partie sont définies les variables globales, les constantes globales, les fonctions, les procédures et les règles. La partie des déclarations contient aussi les déclarations des réseaux manipulés dans la simulation.
La partie déclaration d'un programme SNSimulator commence par le mot-clé DECLARE. Cette partie se termine lorsque la seconde partie du programme commence avec le mot-clé BEGIN. La partie déclaration ne possède donc pas de mot-clé de fin.

Exemple de section déclaration dans un programme SNSimulator que j'ai conçu (le programme source est présenté de manière plus complète en annexe):

```
declare
 simulation sim1
network test using C:\Users\franck\Documents\NetBeansProjects\SNSimulator\lazega\franck.dat
 function carre(x number) return number is
```

```
begin
  return (x*x)
end
rule r1 is
begin
for i in 1 .. 2
  loop
    if(random()>=0.8) then
      add link lienfr3 in test between test1 and test2
    end if
  end loop
end
begin
```

Les différents éléments qui peuvent être déclarés dans la section déclaration sont les suivants :

Les variables :

SNSimulator permet à l'utilisateur de déclarer ses propres variables par le mot-clé VARIABLE. Les valeurs des variables peuvent être modifiées par l'utilisateur durant l'exécution du programme.
Exemple de déclaration de variable:

VARIABLE NUMBER test

Autre exemple de déclaration de variable

VARIABLE STRING test := "test"

Le premier exemple présente une variable déclarée sans être initialisée alors que dans le second exemple la variable est initialisée avec la chaine de caractères "test".

Dans les deux exemples les deux variables sont de types différents : NUMBER et STRING. SNSimulator possède trois types de données au total :

NUMBER pour les nombres,

STRING pour les chaines de caractères

BOOLEAN pour les booléens (valeurs true pour vrai ou false pour faux).

Les constantes :

À la différence des variables, les constantes doivent être obligatoirement initialisées.
Exemple de déclaration de constante :

CONSTANT NUMBER test := 0

Les réseaux :

Les réseaux dans SNSimulator sont déclarés avec le mot-clé NETWORK. Lors de la déclaration d'un réseau, l'utilisateur doit spécifier un fichier dans lequel SNSimulator doit chercher les informations sur le réseau. SNSimulator peut traiter par défaut les fichiers contenant une matrice décrivant le réseau. Il sera relativement simple d'ajouter d'autres formats de réseau.

Exemple de déclaration de réseau :
NETWORK reseau1 USING c:\reseau1.dat

Le mot-clé USING permet de préciser le fichier à charger.

Les nœuds :

SNSimulator permet à l'usager de déclarer des nœuds supplémentaires dans un des réseaux déjà déclarés. Cette déclaration correspond à une insertion du nœud dans le réseau. Le nom du réseau dans lequel doit être inséré le nœud doit donc être obligatoirement précisé. Le nom du nœud doit aussi être précisé.
Exemple de déclaration de nœud:

NODE noeud1 IN reseau1

Les liens :

Comme pour les nœuds l'usager peut déclarer un lien dans un réseau. Dans ce cas-ci l'usager doit préciser également les deux nœuds entre lesquels insérer le lien. L'usager doit aussi préciser le nom du lien.
Exemple de déclaration de lien :

```
LINK lien1 IN reseau1 BETWEEN noeud1 AND noeud2
```

L'exemple crée un lien appelé lien1 entre les nœuds noeud1 et noeud2 dans le réseau reseau1.

Les fonctions :

Les fonctions sont aussi déclarées dans la partie déclaration des programmes SNSimulator. Une fonction est caractérisée par son nom. La fonction peut avoir des paramètres ou ne pas avoir de paramètres. La fonction doit obligatoirement avoir un type de retour. La fonction peut aussi avoir une section déclaration. Toutes les variables et les constantes déclarées dans la section déclaration d'une fonction sont considérées comme des variables et constantes locales à la fonction. Lorsqu'une variable ou constante locale possède le même nom qu'une variable ou constante globale alors cette variable ou cette constante locale sera utilisée par le programme à la place de la variable ou de la constante globale chaque fois que l'utilisateur fera référence au nom de la variable ou de la fonction. La fonction a toujours une section corps. La section corps de la fonction commence par le mot-clé BEGIN et se termine par le mot-clé END. Les instructions permises entre le mot-clé BEGIN et le mot-clé END dans une fonction sont les mêmes que dans le corps du programme principal. Les fonctions sont déclarées avec le mot-clé FUNCTION.
Exemple de déclaration de fonction:

```
FUNCTION carre (x NUMBER) RETURN NUMBER IS
DECLARE
BEGIN
END
```

Dans l'exemple ci-dessus nous déclarons une fonction de nom carre qui prend en paramètre un nombre et qui retourne un nombre. Le corps de cette fonction ne comprend aucune instruction.

Exemple de fonctionnement des variables et constantes locales:

Les procédures :

Comme les fonctions, les procédures sont déclarées dans la section déclaration. Une procédure ne peut avoir le même nom qu'une fonction et vice-versa. La procédure ne peut avoir de type de retour. La procédure peut avoir des paramètres ou pas. La procédure peut avoir une section déclaration. La procédure a toujours une section corps. Les procédures sont déclarées avec le mot-clé PROCEDURE.
Exemple de déclaration de procédure:

```
PROCEDURE proc1 () IS
BEGIN
END
```

La procédure déclarée ne possède aucun paramètre et ne contient pas de section déclaration.

Les règles :

Comme les fonctions et les procédures, les règles sont déclarées dans la section déclaration. Une règle ne peut avoir le même nom qu'une fonction ou qu'une procédure et vice-versa. La règle ne peut avoir de type de retour. La règle ne peut avoir de paramètre. La règle peut avoir une section déclaration. La règle a toujours une section corps. Les règles sont déclarées avec le mot-clé RULE. Les règles sont automatiquement exécutées à chaque pas de temps. Une règle ne peut être appelée par l'utilisateur que ce soit dans une procédure ou dans une fonction ou même dans le corps du programme principal. Néanmoins au sein d'une règle les fonctions et procédures déclarées peuvent être appelées.
Exemple de déclaration de règle:

```
RULE rul1 IS
DECLARE
BEGIN
END
```

La règle rul1 possède une section de déclaration.

La simulation :

Un nom peut être attribué à la simulation en cours dans la section déclaration. Pour cela on utilise le mot-clé SIMULTION comme dans l'exemple ci-dessous:

SIMULATION sim1

Fonctions prédéfinies:

Un certain nombre de fonctions sont définies par défaut dans SNSimulator. Nous les présentons ci-dessous.

Fonction « link_between » : cette fonction prend deux paramètres, deux noms de nœuds d'un même réseau. La fonction link_between retourne vrai si un lien existe entre les deux nœuds passés en paramètre.

Fonction « random » : cette fonction qui ne prend aucun paramètre retourne une valeur aléatoire comprise entre zéro et un.

Fonction « print » : cette fonction prend un paramètre et imprime la valeur du paramètre à l'écran.

Fonction « racine » : cette fonction prend un paramètre et retourne la racine carrée du paramètre.

Propriétés prédéfinies:

Il existe également dans SNSimulator des propriétés prédéfinies. Les principales sont:

« nodes »: propriété définie pour chaque réseau déclaré dans un programme source ou passé en paramètre. Il contient le nombre de nœuds du réseau. Cette propriété peut également être appelée avec un numéro de nœud i pour accéder à ce nœud de la manière suivante : nodes[i]

« links »: propriété définie pour chaque réseau déclaré dans un programme source ou passé en

paramètre. Il contient le nombre de liens du réseau. Cette propriété peut également être appelée avec un numéro de lien i pour accéder à ce nœud de la manière suivante : links[i]

« degree »: propriété définie pour chaque nœud d'un réseau. Il contient le nombre de liens adjacents à ce nœud, c'est à dire le degré du nœud.

Corps du programme

La seconde partie d'un programme SNSimulator contient les instructions à exécuter avant et pendant la simulation. Cette partie commence par le mot-clé BEGIN et se termine par le mot-clé END. Entre ces deux mots-clés on peut retrouver une succession d'instructions qui seront exécutées dans l'ordre premier arrivé premier servi (FIFO).

Exemple de corps de programme SNSimulator:

```
BEGIN
END
```

Comme dans l'exemple ci-dessus, le corps du programme peut ne contenir aucune instruction.

Autre exemple de corps de programme (tiré du programme source présenté en annexe):

```
begin
 add node franck in test1
 print("reseau1")
 add node test1 in test
 add node test2 in test
 add link lientt in test between test1 and test2
 exit sim1 when ((carre(2))==4)
 add link lienfr1 in test between franck and test1
 print(link_between(test.test1,test.test2))
 print(random())
 print(random())
 if(test.test1.degree==1) then
```

add link lienfr2 in test between test2 and test1
end if
start sim1
end

Les différentes instructions permises entre les mots-clés BEGIN et END sont présentées dans les lignes qui suivent:

Ajout de nœud:

SNSimulator permet à l'utilisateur d'ajouter des nœuds dans un réseau en cours d'exécution du programme. Pour cela il est obligatoire de spécifier le réseau dans lequel on souhaite ajouter un nœud par son nom ainsi que le nom du nœud que l'on souhaite ajouter. Le nœud ajouté peut être utilisé dans les instructions qui suivent l'ajout. Pour ajouter un nœud dans un réseau on utilise le mot-clé ADD suivi du mot-clé NODE.
Exemple d'instruction d'ajout d'un nœud dans un réseau:

ADD NODE noeud2 IN reseau1

Dans cet exemple le nœud noeud2 sera ajouté dans le réseau reseau1. Chaque nœud d'un réseau possède reçoit un numéro unique attribué par SNSimulator qui correspond à l'ordre d'insertion des nœuds dans le réseau.

Ajout de liens:

Le langage de SNSimulator permet d'ajouter des liens dans un réseau en cours d'exécution du programme. L'ajout d'un lien nécessite de préciser le nom du réseau dans lequel on souhaite ajouter le lien. Il est aussi nécessaire de spécifier les nœuds entre lesquels le lien doit être ajouté.
Pour ajouter un lien dans un réseau on utilise le mot-clé ADD suivi du mot-clé LINK. L'ajout de lien en cours de programme est similaire à la déclaration de lien dans la section déclaration du programme.
Exemple d'ajout de lien dans un réseau.

ADD LINK lien2 IN reseau1 BETWEEN noeud1 and nocud2

Dans l'exemple ci-dessus nous ajoutons le lien lien2 dans le réseau reseau1 entre les nœuds nœud1 et nœud2.

Ajout de propriétés

Dans le cadre de l'étude des graphes en général et des réseaux sociaux en particulier, il est souvent nécessaire de pouvoir associer des valeurs à des nœuds ou des liens du réseau. Par exemple en analyse de réseau on calcule les degrés des nœuds, leur centralité etc. Il est aussi parfois utile de pouvoir associer des valeurs ou attributs aux liens du réseau. SNSimulator permet d'ajouter des propriétés aux nœuds et/ou aux liens des réseaux déclarés dans la simulation.
Pour ajouter une propriété à un lien ou un nœud d'un réseau, on utilise le mot-clé ADD suivi du mot-clé PROPERTY.
Exemple d'ajout de propriété à un nœud d'un réseau dans SNSimulator:

```
ADD PROPERTY propriete1 FOR noeud1 IN reseau1
```

Dans l'exemple ci-dessus nous définissons une propriété appelée propriété pour le nœud noeud1 du réseau reseau1.

Accès aux valeurs des propriétés

Les valeurs des propriétés des nœuds ou des liens peuvent être accédées dans les différentes instructions du programme SNSimulator en utilisant une notation pointée. Par exemple pour faire référence à la propriété propriete1 définie dans l'exemple précédent sur le nœud noeud1 du réseau reseau1 on utilisera la syntaxe suivante :

```
reseau1.noeud1.propriete1
```

Suppression de nœuds

SNSimulator donne la possibilité de supprimer un nœud d'un réseau durant l'exécution du programme. Pour supprimer un nœud dans un réseau on utilise le mot-clé DELETE suivi du mot-clé NODE.
Exemple de suppression de nœud:

DELETE NODE noeud1 IN reseau1

Suppression de liens

Comme avec les nœuds, il est également possible de supprimer un lien entre deux nœuds dans un réseau. SNSimulator offre deux méthodes de suppression de liens :

Exemple de suppression avec le nom du lien:

DELETE LINK link1 IN reseau1

Exemple de suppression sans le nom du lien :

DELETE LINK IN reseau1 BETWEEN noeud1 AND noeud2

Comme dans les deux exemples ci-dessus le montrent, la suppression d'un lien sans fournir le nom du lien nécessite de préciser les noms des nœuds adjacents au lien qu'on désire supprimer.

Suppression de propriété:

Pour supprimer une propriété pour un objet d'un réseau on utilise le mot-clé DELETE suivi du mot-clé PROPERTY:
Exemple de suppression de propriété dans SNSimulator:

DELETE PROPERTY propriete1 FOR noeud1 IN reseau1

Le langage de SNSimulator offre un ensemble d'instructions qui permettent d'organiser et de structurer les traitements. Il s'agit notamment des boucles et des branchements conditionnels.

Boucles:

Il existe trois types de boucles dans SNSimulator : une boucle inconditionnelle, la boucle WHILE et la boucle FOR.

La boucle inconditionnelle s'exécute sans fin. Elle ne possède aucune condition de fin. La

syntaxe de la cette boucle est la suivante :

LOOP
Suite d'instructions à exécuter
END LOOP

La boucle WHILE est une boucle qui s'exécute tant que la condition définie est vraie. Sa syntaxe est la suivante :

WHILE Condition
LOOP
Suite d'instructions à exécuter
END LOOP

La condition peut être n'importe quelle expression à évaluer. Elle peut porter par exemple sur la valeur d'une variable ou d'une propriété d'un lien ou d'un nœud dans un des réseaux définis.

La boucle FOR est une boucle qui s'exécute un nombre prédéterminé de fois. Sa syntaxe est la suivante :

FOR i in expression1 .. expression2
LOOP
Suite d'instructions à exécuter
END LOOP

Dans l'exemple ci-dessus la boucle s'exécutera un nombre de fois égal à la valeur de expression2 moins expression1 arrondie à l'entier inférieur. À la différence de la boucle WHILE expression1 et expression2 doivent être des nombres.

Branchement conditionnel

SNSimulator permet d'exécuter une suite d'instructions si une condition est vraie et d'exécuter une autre suite d'instructions si la condition n'est pas vraie. Cela est possible grâce aux mots clés IF et ELSE. La syntaxe de l'instruction de branchement conditionnel est la suivante :

```
IF expression1 THEN
        Suite d'instructions à exécuter
ELSIF expression2  THEN
        Suite d'instructions à exécuter
ELSIF expression3 THEN
        Suite d'instructions à exécuter
..
..
ELSIF expressionN THEN
        Suite d'instructions à exécuter
ELSE
        Suite d'instructions à exécuter
END IF
```

Le nombre d'instructions ELSIF n'est pas limité et il n'est pas obligatoire qu'il y ait une instruction ELSIF.

Affectations de variables

Les valeurs des variables globales ou locales définies dans le programme peuvent être modifiées par l'intermédiaire des instructions d'affectation. Le symbole ":=" permet d'affecter à la variable qui se trouve à sa gauche la valeur qui se trouve à sa droite.
Exemple d'affectation de la variable test1:

```
test1 := 123
```

Appels de sous routines

Les fonctions et procédures définies dans la section déclaration d'un programme SNSimulator peuvent être appelées dans le programme principal ou alors dans une autre fonction ou dans une autre procédure. Les fonctions et procédures peuvent être définies avec des paramètres et dans ce cas elles doivent être appelées avec des arguments correspondant aux paramètres déclarés.
Exemple d'appel de sous-routine dans SNSimulator:

test : = carre (2)

Dans l'exemple ci-dessus nous affectons à la variable test déclarée précédemment le résultat de l'appel de la fonction carre avec comme paramètre le chiffre 2.

Instruction de retour d'une fonction

SNSimulator propose une instruction permettant à l'usager de spécifier la valeur de retour d'une fonction. Cela se fait par le mot-clé RETURN.
Exemple d'instruction de retour de fonction dans SNSimulator:

RETURN (x*x)

Avec cette instruction nous pouvons finaliser la déclaration de la fonction carre qui devient:

```
FUNCTION carre (x NUMBER) RETURN NUMBER IS
DECLARE
BEGIN
RETURN (x*x)
END
```

Démarrage de la simulation

Le mot-clé START permet de déclencher la simulation. La syntaxe de cette instruction est simple.

START sim1

L'instruction start doit normalement être la dernière instruction du programme à exécuter car une fois que la simulation est démarrée les autres instructions du programme ne sont plus exécutées.

Arrêt de la simulation

Le mot-clé EXIT permet de spécifier la condition d'arrêt de la simulation. Sa syntaxe est:

EXIT sim1 WHEN expression3

L'évaluation de l'expression expression3 doit être une valeur booléenne. L'instruction EXIT est évaluée à chaque pas de simulation.

Commentaires

SNSimulator accepte deux types de commentaires: les commentaires uni ligne qui s'étendent sur une seule ligne et les commentaires multi-lignes qui peuvent s'étendre sur plusieurs lignes. Les commentaires uni lignes sont indiqués par le symbole "--" au début de la ligne de commentaires et se terminent à la fin de la ligne. Les commentaires multi lignes commencent par "/*" et se terminent par "*/".

Dans ce chapitre nous avons présenté le langage de SNSimulator que nous avons conçu pour la modélisation et la simulation de réseaux sociaux multiplexes. Ce langage se veut simple afin de pouvoir être utilisé facilement par une personne après une formation minimale en informatique. Dans le chapitre suivant nous allons aborder le fonctionnement interne de SNSimulator et nous expliquerons comment SNSimulator exécute le langage que nous venons de présenter.

Chapitre 4 :

Fonctionnement de SNSimulator

Après avoir présenté le langage de programmation SNSimulator dans le chapitre précédent, nous allons dans le chapitre actuel, décrire le fonctionnement interne de l'interpréteur que nous avons réalisé afin d'exécuter notre nouveau langage de programmation. Nous allons d'abord présenter le traitement des paramètres d'entrée du programme SNSimulator puis nous allons parler du chargement des réseaux dans SNSimulator et en enfin nous allons expliquer le traitement du programme source par SNSimulator.

SNSimulator est un programme écrit en langage Java. Il s'exécute à partir de la ligne de commande et ne possède pas d'interface graphique.

Comme tout programme java, SNSimulator est composé de classes java. Les principales classes java constituant SNSimulator sont la classe « Simulation », la classe « Interpreter », la classe « DirectedNetworks » et la classe « Main ». La classe Main est la première classe exécutée lors de l'appel de SNSimulator. Toutes les autres classes de SNSimulator sont exécutées par la classe Main. SNSimulator accepte des paramètres lors de son exécution. Le traitement effectué par SNSimulator à partir des paramètres est abordé dans le paragraphe suivant.

Traitement des paramètres par SNSimulator

Les paramètres de SNSimulator peuvent être séparés en deux groupes principaux: Les paramètres concernant les réseaux à simuler et les paramètres concernant le programme à exécuter.

L'utilisateur du programme SNSimulator a la possibilité de demander à ce que des descriptions de réseaux sociaux soient lues et chargées en mémoire avant même l'exécution de la simulation. Pour cela il faut placer la chaine de caractères "-n" dans la liste des paramètres passés à SNSimulator.
Cette chaine de caractères doit obligatoirement être suivie du nom du réseau à créer. Le nom du réseau peut être suivi d'un chemin d'accès vers un fichier contenant les informations du réseau à

charger en mémoire. Si l'utilisateur ne fournit pas une adresse de fichier valide alors un réseau vide sera créé en mémoire.

La chaine de caractères "-p" permet de préciser un fichier texte contenant des instructions à exécuter par SNSimulator. Ce paramètre est suivi du chemin d'accès du fichier à exécuter. Si aucun fichier à exécuter n'est fournit, SNSimulator affichera simplement les réseaux passés en paramètres.
Exemple d'appel de SNSimulator:

SNSimulator -n collaboration ./cowork.dat -n conseil ./advice.dat -n amitie ./friend.dat -p programme.txt

Extrait de la section de SNSimulator qui traite les paramètres:

```
boolean prog=false;
    for(int k=0;k<args.length;k++)
    {
      if(args[k].equals("-n")){
        if(k==(args.length-2))
          networks.AddNetwork(args[k+1], "");
        else{
        if((!args[k+2].equals("-n")) && (!args[k+2].equals("-p")) )
        networks.AddNetwork(args[k+1],args[k+2]);
        else
          networks.AddNetwork(args[k+1], "");
        }
      }
      else if(args[k].equals("-p")){
        if(!prog){
          prog=true;
          try{
          program=new FileReader(args[k+1]);
          }
          catch(Exception e){
            System.out.println(e.toString());
```

```
            }
        }
        else
            System.out.println("Only one program file is allowed");

    }

}
```

La classe Main exécute la méthode AddNetwork() de la classe DirectedNetworks qui charge en mémoire tous les réseaux passés en paramètre à SNSimulator. Dans le paragraphe suivant nous allons aborder le chargement en mémoire des réseaux transmis en paramètre.

Chargement de réseaux en mémoire

SNSimulator accepte trois types de fichiers de réseau :

a) Les fichiers texte contenant une matrice d'adjacence décrivant le réseau à créer.

Afin de pouvoir manipuler les graphes de manière informatique, il est nécessaire d'en avoir une représentation mémoire. Il existe deux principaux types de représentation des graphes : la représentation par une matrice et la représentation par une liste.
Les représentations matricielles des graphes peuvent être séparées en deux groupes: les matrices d'adjacence et les matrices d'incidence qui ont été définies au chapitre un.

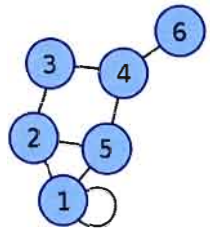

Illustration 15: Exemple de graphe

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Illustration 16: Matrice d'adjacence correspondante au graphe précédent

Source site web Wikipédia

Ce format de fichier est le format par défaut de SNSimulator.

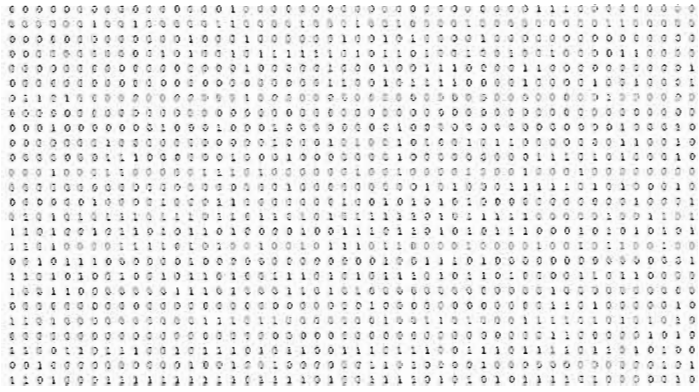

Illustration 17: Exemple de fichier de matrice d'adjacence pouvant être traité par SNSimulator

b) Les fichiers Pajek. Ce format de fichier qui est le format de fichier du logiciel d'analyse de réseau Pajek est traité par l'intermédiaire de la librairie JUNG. Les fichiers Pajek contiennent une représentation des réseaux sous une forme pouvant être facilement traduite en une liste d'incidence avec une liste des nœuds suivie par une liste des liens entre les nœuds.

```
*Vertices 14
1 I1
2 I3
3 W1
4 W2
5 W3
6 W4
7 W5
8 W6
9 W7
10 W8
11 W9
12 S1
13 S2
14 S4
*Edges
1 5
3 5
3 6
5 6
9 10
9 11
10 11
3 12
5 12
6 12
10 14
11 14
9 12
```

Illustration 18: Exemple de fichier PAJEK

Source site web de Pajek

c) Les fichiers GraphML sont traités grâce à la librairie Jung. GraphML est un format de fichier XML permettant de décrire des graphes. Le format GraphML est disponible sous licence open source.

Illustration 19: Exemple de fichier GraphML

Source site web de GraphML

Chaque réseau chargé en mémoire est enregistré dans la classe DirectedNetworks. Cette classe regroupe tous les réseaux définis dans SNSimulator ainsi que plusieurs méthodes permettant de les manipuler. Les réseaux de SNSimulator sont représentés dans la classe DirectedNetworks par des graphes orientés. Tous les réseaux créés dans SNSimulator sont donc par défaut des réseaux orientés. La classe DirectedNetworks contient plusieurs tables de hachage.

Une table de hachage est une structure de données informatique qui contient plusieurs éléments appelés valeurs ainsi que plusieurs clés permettant d'accéder aux éléments de la table. Chaque valeur est associée à une et une seule clé.

Ces tables de hachage permettent par exemple d'obtenir un réseau à partir de son nom ou de son numéro d'identification car chaque réseau reçoit automatiquement au moment de sa création un numéro d'identification qui est attribué par SNSimulator selon l'ordre de création des réseaux: Par exemple le premier réseau passé en paramètre ou déclaré dans le programme source à exécuter reçoit le numéro un et ainsi de suite. La classe DirectedNetworks est utilisée tout le long de l'exécution du programme SNSimulator: lors du traitement des paramètres, lorsqu'un réseau est déclaré par l'utilisateur dans le programme source et chaque fois qu'une action est effectuée sur un nœud ou sur un lien d'un réseau.

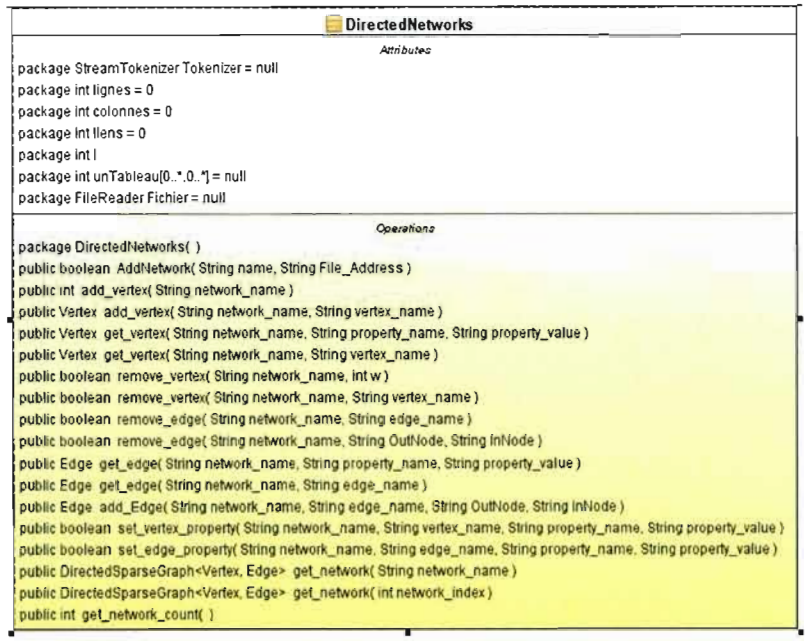

Illustration 20: Diagramme UML de la classe DirectedNetworks

Lorsque tous les paramètres de SNSimulator sont traités, la fonction Main exécute la classe Interpreter qui est l'analyseur syntaxique du langage présenté au chapitre trois. Le fonctionnement de cet analyseur syntaxique est abordé au paragraphe ci-dessous.

Traitement du programme source à exécuter

Une fois que les réseaux passés en paramètres sont chargés en mémoire, la classe Interpreter de SNSimulator effectue l'analyse syntaxique du programme source en paramètre.

L'analyseur syntaxique de SNSimulator essaie de reconnaître les différents éléments de la syntaxe du langage. L'élément syntaxique de plus haut niveau est *GetSimulationDefinition* qui représente le programme source à traiter. Le *GetSimulationDefinition* est composé soit de la déclaration d'une procédure, soit de la déclaration d'une fonction, soit d'un appel de sous-routine, soit enfin d'une séquence d'instructions. Pour reconnaître le *GetSimulationDefinition,* SNSimulator va cssayer de reconnaître chacun de ses éléments constitutifs.

De la même manière, pour reconnaître chacun des éléments constitutifs du *GetSimulationDefinition,* SNSimulator va essayer de reconnaître chacun de leurs propres constituants syntaxiques. Et ainsi de suite jusqu'aux différents tokens ou mots composant le programme source.

Lorsque SNSimulator réussit à reconnaître l'élément *GetSimulationDefinition*, c'est à dire que le programme source est correct selon la syntaxe, SNSimulator crée un arbre syntaxique abstrait représentant le programme source.

En linguistique, l'arbre syntaxique représente la structure syntaxique d'une phrase.

En informatique, l'arbre syntaxique abstrait est la représentation arborescente de la structure syntaxique d'un programme source écrit dans un certain langage de programmation.

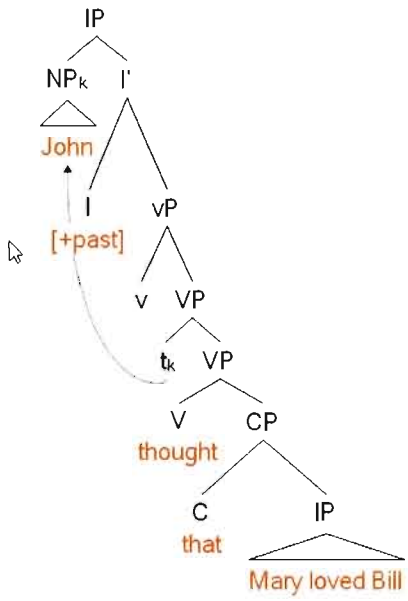

Illustration 21: Exemple d'arbre syntaxique de la phrase en anglais: "John thought that Mary loved Bill"

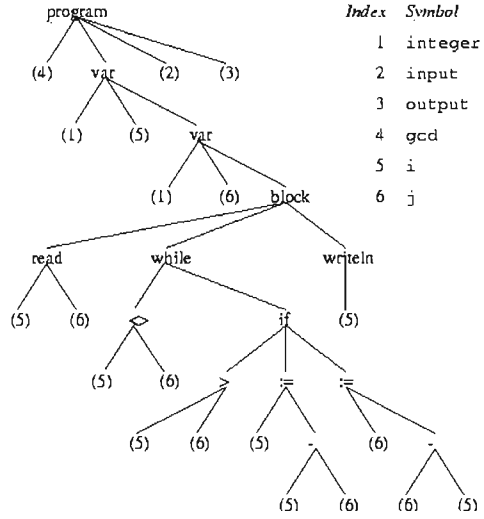

Illustration 22: Exemple d'arbre syntaxique abstrait d'un programme informatique

Chaque nœud de l'arbre créé par SNSimulator est une classe java qui représente un élément syntaxique reconnu. La racine de cet arbre est la classe ASTGetSimulationDefinition qui comme son nom l'indique correspond à *GetSimulationDefinition.*

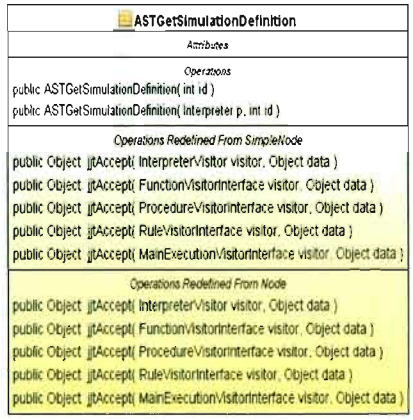

Illustration 23: Diagramme UML de la classe ASTGetSimulationDefinition

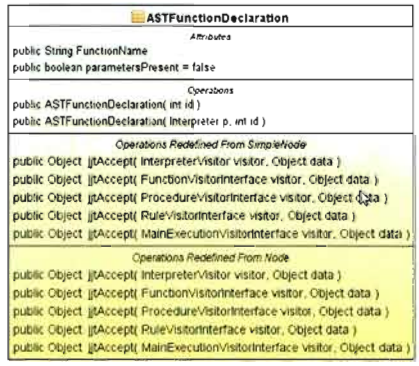

Illustration 24: Diagramme UML de la classe ASTFunctionDeclaration

Une fois que l'arbre syntaxique abstrait est créé, SNSimulator le parcours afin d'exécuter les instructions du programme source. Le parcours de l'arbre syntaxique s'effectue selon le modèle visiteur.

Le modèle visiteur est une méthode de traitement des arbres syntaxiques abstraits qui consiste à faire parcourir l'arbre syntaxique par une classe java appelée visiteur. En général le visiteur effectue un parcours en profondeur d'abord de l'arbre syntaxique (parcours depth-first) mais il est possible d'effectuer un ordre de parcours différent.

Le parcours en profondeur d'un arbre consiste à visiter d'abord tous les nœuds fils d'un nœud x de l'arbre avant de visiter le nœud x.

Le visiteur est une classe java qui possède une méthode *visit* pour chaque élément syntaxique de la grammaire utilisée.

Exemple de méthode de la classe visiteur pour l'élément syntaxique GetSimulationDefinition

```
public Object visit(ASTGetSimulationDefinition node, Object data){
    data = node.childrenAccept(this, data);
    return data;
```

}

Dans chaque méthode *visit* de la classe visiteur, un appel de la méthode *childrenAccept* est effectué. La méthode *childrenAccept* est la méthode responsable du parcours de l'arbre syntaxique abstrait.

Lors du parcours de l'arbre syntaxique abstrait, le visiteur exécute les instructions de la méthode *visit* spécifique pour chaque type d'élément syntaxique. Étant donné que nous avons choisi de réaliser un interpréteur, ces instructions consistent à exécuter les instructions du programme source. Dans le cas d'un compilateur, le visiteur produit le code machine.

Illustration 25: Diagramme UML de la classe visiteur

SNSimulator parcourt l'arbre syntaxique du programme source à deux reprises. Lors du premier passage, SNSimulator exécute toutes les instructions de déclaration qu'il rencontre. Toutes les fonctions, variables, procédures, règles sont donc créées lors de ce passage.

Lors du second passage SNSimulator exécute toutes les autres instructions du programme source. Lorsque SNSimulator rencontre l'instruction START il démarre l'exécution de la simulation.

Extrait de la description au format BNF du langage SNSimulator présentée en annexe

GetSimulationDefinition ::= (ProcedureDeclaration <EOF> | FunctionDeclaration <EOF> | SequenceOfStatements <EOF> | "CALL" SubroutineCall <EOF> | <EOF>)

DeclarationSection ::= "DECLARE" Declarations

*Declarations ::= (VariableDeclaration | ConstantDeclaration | FunctionDeclaration | NetworkDeclaration | NodeDeclaration | NodeSetDeclaration | LinkDeclaration | LinkSetDeclaration | PropertyDeclaration | ProcedureDeclaration | RuleDeclaration | SimulationDeclaration)**

ProcedureDeclaration ::= "PROCEDURE" <S_IDENTIFIER> ("(" ")" | "(" ParameterList ")") "IS" ProcedureBody

NetworkDeclaration ::= "NETWORK" <S_IDENTIFIER> ("USING" <FILE_ADDRESS>)?

NodeDeclaration ::= "NODE" <S_IDENTIFIER> "IN" <S_IDENTIFIER>

Exécution de la simulation

Il existe différents types de simulation: la simulation continue, la simulation discrète elle-même séparée en simulation discrète synchrone et en simulation discrète asynchrone.

La simulation continue est basée sur la modélisation du système étudié à l'aide d'équations différentielles.

Dans une simulation discrète le système simulé est soumis à une succession d'évènements qui le modifient. La simulation discrète se divise en deux grandes catégories :

Dans une simulation discrète synchrone on simule à chaque fois le passage d'une unité de temps sur tout le système.

Dans une simulation discrète asynchrone on calcule l'arrivée du prochain événement, et on ne simule qu'événement par événement.

Enfin la simulation par agents est basée sur le concept d'agent. Plutôt que de simuler les événements qui affectent le système, dans la simulation par agents, le système est plutôt considéré comme un ensemble d'agents autonomes. Chaque agent possède un comportement qui lui est propre et qui est basé sur des règles qui peuvent être différentes pour les autres agents du système.

L'évolution du système simulé est le résultat du comportement de tous les agents.
SNSimulator est un simulateur discret asynchrone.

L'exécution de la simulation est effectuée par la classe java Simulation. La classe Simulation contient plusieurs tables de hachage qui stockent les variables, les constantes, les procédures, les fonctions et les règles de la simulation. Lorsque la simulation démarre, la classe Simulation exécute une boucle qui ne s'arrête que lorsque la condition précisée dans l'instruction EXIT WHEN est vraie.

Boucle exécutée lors de la simulation

```
public boolean start(){
    while(!end_of_simulation()){
      rules=simulation_rules.values().iterator();
      apply_rules();
    }
    return true;
 }
```

La fonction end_of_simulation est une fonction qui retourne la valeur booléenne vrai lorsque la condition d'arrêt de la simulation est atteinte.

Si l'utilisateur n'a pas spécifié de condition d'arrêt alors la boucle devient une boucle infinie. À chaque pas de la boucle, SNSimulator exécute toutes les règles définies dans le programme source l'une après l'autre. Chaque pas de la boucle constitue un pas de temps dans la simulation. Une fois que la condition d'arrêt de la simulation est atteinte, SNSimulator affiche tous les réseaux sociaux utilisés comme dans les exemples de visualisation de réseau par SNSimulator présentés ci-dessous.

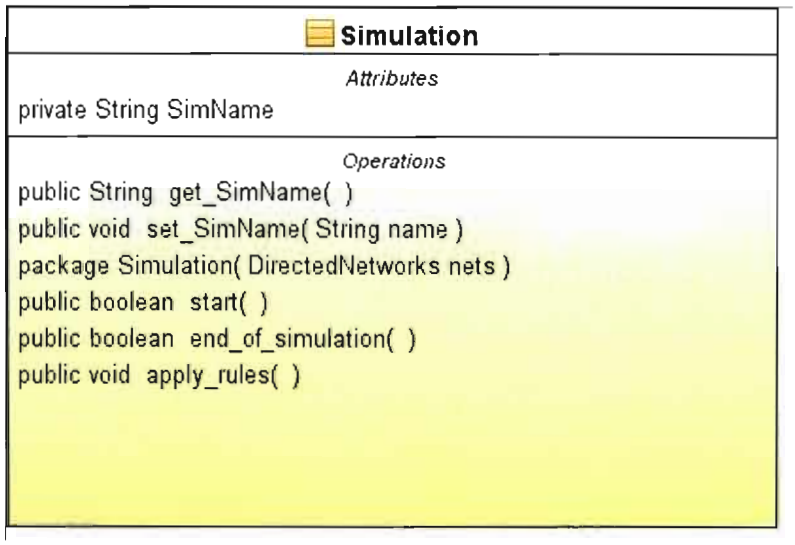

Illustration 26: Diagramme UML de la classe Simulation

Exemples de visualisation de réseau par SNSimulator

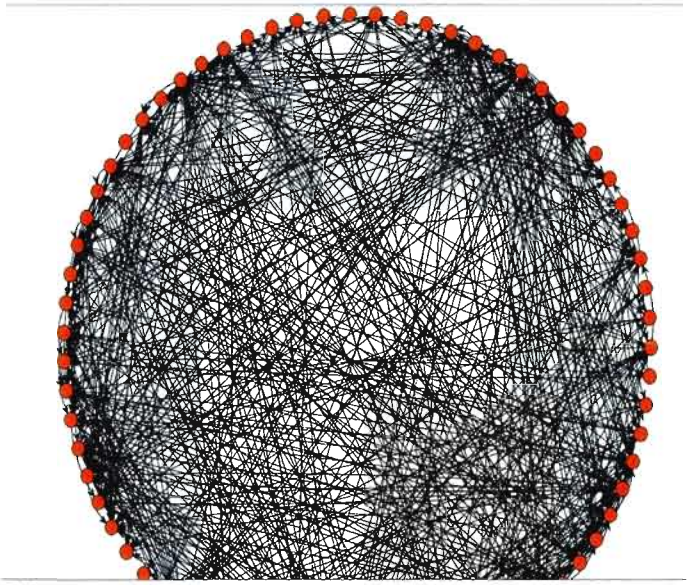

Illustration 27: Réseau d'amitié du cabinet d'avocats décrit dans [27]

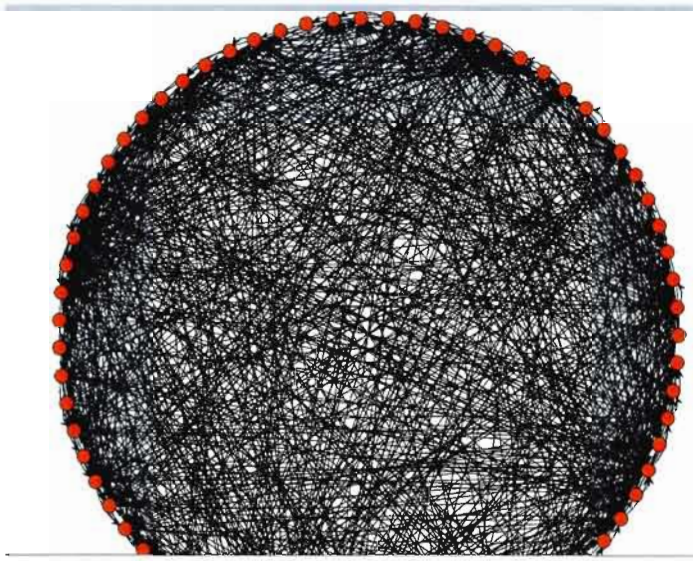

Illustration 28: Réseau de conseil du cabinet d'avocats décrit dans [27]

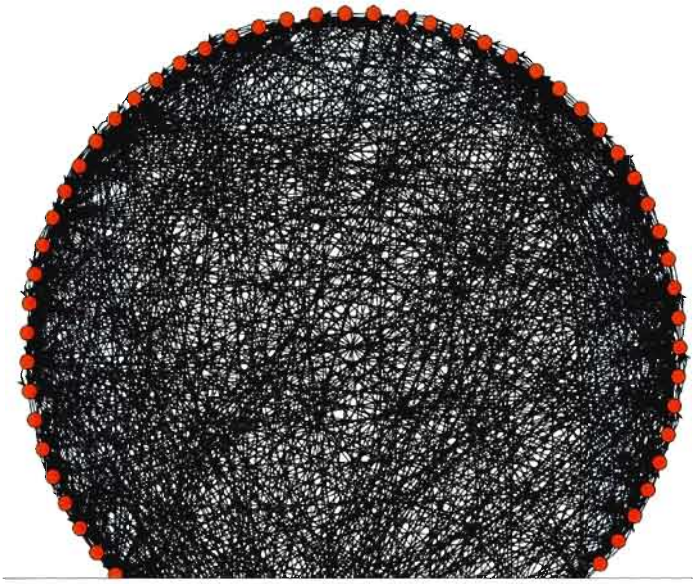

Illustration 29: Réseau de collaboration du cabinet d'avocats décrit dans [27]

Dans ce chapitre nous avons abordé le fonctionnement interne de l'interpréteur SNSimulator. Nous avons examiné le traitement des paramètres, du programme source et l'exécution de la simulation. Dans le prochain chapitre nous mettrons le langage SNSimulator en pratique et nous allons programmer une fonction permettant de déterminer le coefficient de corrélation entre deux réseaux sociaux simultanés.

Chapitre 5 :

Utilisation du langage SNSimulator

Dans ce chapitre nous allons mettre en pratique SNSimulator. Pour cela nous allons étudier l'évolution de la corrélation entre deux réseaux. Nous allons appliquer à chacun des réseaux le modèle de croissance de réseaux proposé par Newman, Jin et Girvan dans [16]. Dans la suite de ce chapitre nous allons d'abord présenter les réseaux sur lesquels nous allons travailler, puis nous parlerons du modèle de croissance de réseaux de Newman, Jin et Girvan. En troisième partie nous allons montrer la réalisation du modèle de croissance de réseaux à l'aide de SNSimulator. Dans la dernière partie de ce chapitre nous allons présenter les résultats que nous avons obtenus et nous allons les interpréter.

Réseaux utilisés

Dans le cadre de cette application de SNSimulator, nous avons utilisé les résultats de l'étude d'un cabinet d'avocats de la Nouvelle-Angleterre aux États-Unis d'Amérique par Emmanuel Lazega dans [27]. Le personnel de ce cabinet est constitué de 71 avocats. Ces avocats sont répartis entre 36 associés et 35 collaborateurs. Les avocats du cabinet pratiquent deux domaines du droit : Le contentieux et le conseil. Le conseil regroupe toutes les activités qui ne sont pas du domaine du contentieux.

Dans le cadre de son étude sociologique, Emmanuel Lazega a observé trois relations différentes au sein du cabinet d'avocats : Le conseil entre avocats, la collaboration et l'amitié. Pour réaliser son étude, Emmanuel Lazega a soumis chaque avocat à un questionnaire sociométrique sur l'existence ou non de relations entre la personne interrogée et les autres membres du cabinet.

Le résultat final de l'étude est constitué de trois matrices d'incidence, une pour chaque relation étudiée, que nous allons exploiter dans les paragraphes suivants.

Le modèle d'évolution des réseaux sociaux de Newman, Jin et Girvan

Emily M. Jin, Michelle Girvan et M. E. J. Newman ont proposé dans [16] un modèle permettant

de reproduire les processus d'évolution des réseaux sociaux. Selon eux les réseaux sociaux suivent un certain nombre de règles :

1) Le nombre de nœuds des réseaux sociaux peut être considéré avec une approximation acceptable comme fixe: À la différence de certains types de réseaux tels que le World Wide Web dans lesquels le nombre de nœuds du réseau croit au fur et à mesure du temps par l'addition de nouveaux nœuds et liens dans le réseau, les réseaux sociaux sont souvent constitués d'une population de taille fixe car l'arrivée de nouveaux individus au sein de la population considérée se fait à un rythme beaucoup plus faible que, par exemple, dans le cas du World Wide Web.

2) Un degré limité: Dans les réseaux sociaux, la probabilité pour qu'une personne du réseau considéré établisse un nouveau lien diminue fortement lorsque cette personne possède déjà des liens dont le nombre approche d'une certaine valeur limite.

3) Regroupement de nœuds: La probabilité que deux personnes du réseau établissent un lien entre eux est proportionnelle au nombre de personnes avec lesquelles elles ont toutes deux un lien (proportionnelle au nombre d'amis communs)

4) Décroissance des relations: La force des liens du réseau peut diminuer avec le temps et des liens peuvent même se briser mais aussi se créer ce qui permet au réseau de ne pas rester statique.

Dans leur article les auteurs proposent deux modèles qui respectent les quatre règles que nous avons présentées ci-dessus. Dans le cadre de ce projet de recherche nous nous sommes surtout intéressés au premier des deux modèles et nous allons l'aborder dans les paragraphes suivants.

Le modèle de Newman, Jin et Girvan est fondé sur la probabilité que deux personnes d'un réseau se rencontrent. Cette probabilité doit être proportionnelle à la fois au nombre de liens entre chacune des deux personnes et les autres personnes du réseau mais également proportionnelle au nombre de personnes du réseau qui ont un lien à la fois avec les deux individus considérés. Les auteurs proposent deux fonctions dont la combinaison permet d'obtenir une distribution de probabilité qui satisfait aux conditions de proportionnalité énoncées:

a) La fonction de Fermi

$f(zi) = 1/(\exp(\beta(zi - zmax)) + 1)$.

Dans la fonction de Fermi le terme zi représente le nombre de liens entre l'individu i et les autres membres du réseau considéré. zmax est le nombre maximal de liens qu'un individu peut avoir conformément à la règle numéro 2 relative au degré limité que nous avons présentée précédemment. L'élément β de la fonction contrôle la vitesse de diminution du nombre de liens d'un individu lorsqu'il approche de la valeur zmax.

b) La seconde fonction représente l'augmentation de la propension de deux individus à établir des liens entre eux lorsqu'ils ont déjà des liens avec plusieurs autres individus en commun. Les auteurs de l'article expriment cette fonction de la manière suivante:

$$g(mij) = 1 - (1 - p0)\exp(-\alpha\, mij)$$

Dans cette formule mij représente le nombre de personnes liées à la fois aux individus i et j. Le terme p0 quant à lui représente la probabilité d'établissement de lien lorsqu'il n'y a personne dans le réseau considéré qui a des liens à la fois avec i et avec j. Le terme α est un paramètre qui permet de contrôler l'intensité d'augmentation de la fonction g.

La probabilité par unité de temps que deux individus i et j se rencontrent est donnée par :

pij() = f(zi)*f(zj)*g(mij)

Pour respecter la règle numéro 4 ci-dessus, le modèle de Newman, Jin et Girvan attribue à chaque lien une force qui diminue petit à petit d'une valeur sij donnée par

$$sij = \exp(-K\,\Delta t)$$

lorsque les deux individus i et j reliés par ce lien ne se rencontrent pas. Le terme Δt représente le temps écoulé depuis la dernière fois que les deux individus reliés par un lien se sont rencontrés. Le terme K est un paramètre qui permet de contrôler la valeur de sij. Dans le modèle lorsque les individus i et j se rencontrent de nouveau la force du lien entre eux est réinitialisée avec la valeur

1. S'il n'existait pas de lien entre i et j et que i et j se rencontrent, alors un lien avec une force de 1 est crée. Les auteurs ont défini 0.3 comme valeur minimale de force pour l'existence d'un lien. Lorsque la valeur de la force d'un lien devient inférieure à 0.3, ce lien est supprimé.

Pour simuler leur modèle d'évolution de réseaux sociaux, Newman, Jin et Girvan ont utilisé la méthode de Monte Carlo cinétique qui fait l'objet du paragraphe suivant.

L'Algorithme de Monte Carlo cinétique

Les algorithmes de Monte Carlo sont une famille d'algorithmes basés sur l'utilisation de nombres aléatoires. L'algorithme de Monte Carlo cinétique est un algorithme permettant la simulation informatique de processus qui se reproduisent avec un taux connu. L'algorithme de Monte Carlo cinétique est très utilisé pour simuler des réactions chimiques au niveau microscopique.

L'exécution de l'algorithme de Monte Carlo cinétique pour un système comprend les sept étapes suivantes:

1) On note t_0 le temps initial du système.
2) Calculer les taux d'occurrence Γ_i de toutes les évolutions possibles du système pour i=1,....., N avec N le nombre total de transitions possibles.

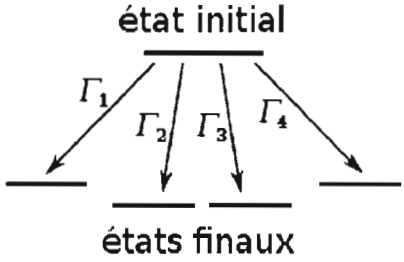

Illustration 30: Algorithme de Monte Carlo: transitions du système

Source Wikipédia

3) Calculer $\Gamma_{tot}=\Sigma \Gamma_i$ le taux de transfert total.
4) Le système quitte son état initial au bout du temps $T= -\ln(U)/\Gamma_{tot.}$ où U est un nombre choisi aléatoirement compris entre 0 et 1.
5) On choisit un second nombre aléatoire U' compris entre 0 et Γ_{tot} qui servira à déterminer parmi les transitions possibles du système celle qui sera exécutée.
6) Si U' est compris entre 0 et Γ_1 alors le système transite vers l'état 1. Si U' est compris entre Γ_1 et $\Gamma_1+\Gamma_2$ alors l'état 2 sera choisi. D'une manière générale,

$$S_i = \sum_{j=1}^{i} \Gamma_j$$

avec $S_0=0$ et si U' est compris entre S_{i-1} et S_i alors la transition i sera exécutée.

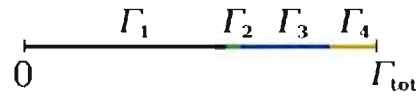

Illustration 31: Algorithme de Monte Carlo: choix de la transition à exécuter

Source Wikipédia

7) Changer le temps en t0+T et recommencer à l'étape 1.

Réalisation du modèle de Newman, Jin et Girvan avec SNSimulator

Nous allons présenter dans cette partie la manière dont nous avons créé le modèle de croissance de réseaux sociaux de Newman, Jin et Girvan avec le langage que nous avons conçu.

Tout d'abord nous allons rappeler les principales caractéristiques du modèle que nous nous proposons de réaliser:

a) Une probabilité par unité de temps que deux individus i et j se rencontrent :

pij() = f(zi)*f(zj)*g(mij)

avec f(zi) = $1/(\exp(\beta(zi-zmax))+1)$

et $g(mij)=1-(1-p0)\exp(-\alpha mij)$

b) Diminution de la force des liens par $sij=\exp(-K\Delta t)$

c) Simulation par l'algorithme de Monte Carlo cinétique

Étant donné que nous avons conçu le langage SNSimulator de manière à ce qu'il soit le plus simple possible, il est relativement simple de traduire la fonction pij. Pour cela nous avons besoin des fonctions f() et g(). Nous avons aussi besoin d'un moyen de déterminer le nombre d'individus ayant des liens avec deux individus donnés dans un réseau. Dans le cas d'un réseau d'amitié, il s'agirait de déterminer le nombre d'amis communs à deux individus.
Nous avons donc créé quatre fonctions dans SNSimulator, p, f, g et common_friends_number:

```
function common_friends_number(x string,y string,z string) return number is
declare
        variable number t := 0
begin
        for i in 1 .. z.nodes.count
        loop
                if(link_between(z.x, z.nodes[i])or link_between(z.nodes[i], z.x)) then
                        if(link_between(z.y, z.nodes[i])or link_between(z.nodes[i], z.y)) then
                                t:=t+1
                        end if
                end if
        end loop
        return t
end
```

Cette fonction reçoit trois paramètres : le premier paramètre est le nom du réseau auquel appartiennent les deux individus. Les deux paramètres restants sont les noms des individus dans le réseau. Dans cette fonction nous utilisons la fonction prédéfinie link_between que nous avons présenté au chapitre 3 et qui permet de savoir si un lien existe entre deux nœuds d'un réseau.

La fonction g elle est programmée comme ci-dessous :

```
function g(x number,y number,z string) return number is
declare
        variable number resultat
        variable number temp
begin
        temp := 0
         resultat := 0
        resultat := common_friends_number(z.nodes[x].name,z.nodes[y].name,z)
        resultat := resultat * alpha
        resultat := - resultat
        resultat := exp(resultat)
        temp := p0
        temp := -temp
        temp := temp + 1
        resultat := resultat * temp
        resultat := -resultat
         resultat := resultat + 1
         return resultat
end
```

La fonction g reçoit comme paramètres le nom d'un réseau ainsi que les numéros de deux nœuds dans ce réseau.

La fonction f devient :

```
function f(x number,z string) return number is
declare
        variable number numerateur
        variable number denominateur
        variable number resultat
begin
        numerateur := 1
```

```
        denominateur := z.nodes[x].degree
        denominateur := denominateur - z_star
        denominateur := denominateur * beta
        denominateur := exp(denominateur)
        denominateur := denominateur + 1
        resultat := numerateur/denominateur
        return resultat
end
```

Les deux paramètres de cette fonction sont le nom d'un réseau et le numéro d'un nœud dans le réseau.

Et la probabilité de rencontre entre individus est donnée par :

```
function p(x number,y number, z string) return number is
declare
        variable number f1
        variable number f2
        variable number g1
        variable number resultat
begin
        f1 :=0
        f2 :=0
        g1:=0
        resultat := 0
        f1 := f(x,z)
        f2 := f(y,z)
        g1 := g(x,y,z)
        resultat := f1 * f2 * g1
        return resultat
end
```

Nous avons également créé une procédure pour l'algorithme de Monte Carlo cinétique:

Cette procédure comporte deux étapes principales :

1) Nous calculons la valeur gamma_tot qui représente la probabilité de transfert total du réseau comme nous l'avons présenté dans le paragraphe précédent sur l'algorithme de Monte Carlo cinétique.
2) Nous parcourons les couples de nœuds du réseau et pour chaque couple nous calculons les valeurs gamma_sum et gamma_sum_minus qui représentent respectivement S_i et S_{i-1} les probabilités cumulatives des transitions i et i-1 avec

$$S_i = \sum_{j=1}^{i} \Gamma_j$$

Dans le cas de ce mémoire de maîtrise, nous considérons que chaque rencontre entre deux individus représente une transition du système, donc chaque pij() est une probabilité de transition et la probabilité cumulative S_k devient donc:

$$S_k = \Sigma^{k}_{i=1,j=1} \text{Pij}()$$

3) Pour chaque couple de nœuds du réseau nous tirons au hasard un nombre aléatoire u_prime entre zéro et gamma_tot. Ensuite nous comparons ce nombre aléatoire avec gamma_sum et gamma_sum_minus qui correspondent à S_k et S_{k-1}. Si gamma_sum_minus < u_prime < gamma_sum alors un lien est créé entre les individus i et j en cours de traitement. Si un lien existe déjà entre i et j alors la force de ce lien est initialisée avec la valeur 1. Lorsque u_prime n'est pas compris entre gamma_sum_minus et gamma_sum alors cela signifie qu'il n'y a pas de rencontre entre les individus i et j, alors s'il y a un lien entre ces individus nous diminuons la force de ce lien conformément au modèle de Newman, Jin et Girvan.

La traduction finale de l'algorithme de Monte Carlo cinétique dans le langage SNSimulator est reproduite ci-dessous:

```
procedure n_fold_way_monte_carlo_conseil(x string) is
declare
variable boolean fait
begin
fait := false
gamma_tot := 0
gamma_sum_minus := 0
gamma_sum:= 0
```

```
for i in 1 .. x.nodes.count
loop
 for j in 1 .. x.nodes.count
 loop
  if(i!=j) then
   gamma_tot:=gamma_tot+p(i,j,x)
  end if
 end loop
end loop
gamma_tot := gamma_tot/2
u_prime := random()* gamma_tot
delta_temps := -ln(random())/gamma_tot
temps := temps + delta_temps
temps_discret := temps_discret + 1
for i in 1 .. x.nodes.count
loop
 for j in 1 .. x.nodes.count
 loop
    if(i!=j) then
     gamma_sum_minus := gamma_sum
     gamma_sum:=gamma_sum+p(i,j,x)
      if((u_prime>gamma_sum_minus) and (u_prime<gamma_sum)) then
       if(fait==false) then
        if((link_between(x.nodes[i],x.nodes[j]))) then
         x.links[i,j].strength := 1
         x.links[i,j].last_meeting :=0
         fait := true
        elsif((link_between(x.nodes[j],x.nodes[i]))) then
         x.links[j,i].strength := 1
         x.links[j,i].last_meeting :=0
         fait := true
        else
         add link in x between x.nodes[i] and x.nodes[j]
         add property number strength for x.links[i,j] in x as 1
```

```
        add property number last_meeting for x.links[i,j] in x as 0
        fait := true
       end if
      else
       if((link_between(x.nodes[i],x.nodes[j]))) then
          x.links[i,j].strength := exp(-ka * x.links[i,j].last_meeting)
        x.links[i,j].last_meeting := x.links[i,j].last_meeting + delta_temps
        if(x.links[i,j].strength<strength_limit) then
         delete link in x between x.nodes[i].name and x.nodes[j].name
        end if
       elsif((link_between(x.nodes[j],x.nodes[i]))) then
            x.links[j,i].strength := exp(-ka * x.links[j,i].last_meeting)
          x.links[j,i].last_meeting := x.links[j,i].last_meeting + delta_temps
          if(x.links[j,i].strength<strength_limit) then
           delete link in x between x.nodes[j].name and x.nodes[i].name
          end if
         end if
        end if
       else
        if((link_between(x.nodes[i],x.nodes[j]))) then
          x.links[i,j].strength := exp(-ka * x.links[i,j].last_meeting)
        x.links[i,j].last_meeting := x.links[i,j].last_meeting + delta_temps
        if(x.links[i,j].strength<strength_limit) then
         delete link in x between x.nodes[i].name and x.nodes[j].name
        end if
       elsif((link_between(x.nodes[j],x.nodes[i]))) then
            x.links[j,i].strength := exp(-ka * x.links[j,i].last_meeting)
          x.links[j,i].last_meeting := x.links[j,i].last_meeting + delta_temps
          if(x.links[j,i].strength<strength_limit) then
           delete link in x between x.nodes[j].name and x.nodes[i].name
          end if
        end if
      end if
     end if
end loop
```

end loop

end

Dans la suite de ce chapitre nous allons d'abord présenter le protocole expérimental que nous avons respecté puis nous allons présenter les résultats que nous avons obtenus et les conclusions que nous pouvons en tirer.

Protocole expérimental appliqué et résultats attendus

Dans ce paragraphe nous discuterons du protocole expérimental que nous avons utilisé pour mettre en pratique SNSimulator.

Nous avons réalisé cinq expérimentations dont les résultats permettent selon nous d'évaluer la capacité de SNSimulator à donner des résultats crédibles.

Expérimentation 1 :

Nous avons créé deux copies du réseau d'amitié du cabinet d'avocats étudié par Emmanuel Lazega. Nous avons appliqué le modèle d'évolution des réseaux sociaux de Newman, Jin et Girvan à une des deux copies, la seconde restant intacte. Nous avons répété ce processus un grand nombre de fois en appliquant le modèle de croissance de réseaux sociaux à la copie de réseau d'amitié résultant de l'itération précédente. À chaque itération nous avons calculé le coefficient de corrélation de degré entre la copie du réseau d'amitié du cabinet d'avocats de Emmanuel Lazega sur laquelle le modèle d'évolution de réseau a été appliqué et la copie restée intacte.

Résultats attendus :

À la suite de l'expérimentation 1, nous nous attendons à ce que le coefficient de corrélation entre les deux versions du réseau d'amitié diminue de manière continue. Cette diminution du coefficient de corrélation est due au fait que étant donné que des modifications sont appliquées à un seul des réseaux, les deux copies du réseau d'amitié sont de plus en plus différentes l'une de l'autre à chaque itération.

Expérimentation 2 :

Nous avons appliqué le modèle de croissance de réseaux sociaux de Newman, Jin et Girvan au réseau de collaboration d'Emmanuel Lazega mais également au réseau d'amitié. Comme dans le cas de l'expérimentation 1 nous calculons la corrélation entre le réseau de collaboration modifié par le modèle de croissance de réseau et le réseau d'amitié modifié. Nous calculons aussi le degré moyen des nœuds du réseau de collaboration et du réseau d'amitié.

Résultats attendus :

Nous n'avons pas d'attente particulière pour cette expérience. Cette expérimentation est surtout destinée à obtenir des résultats que nous allons comparer avec les résultats de l'expérimentation 3 ci-dessous.

Expérimentation 3 :

La principale différence entre l'expérimentation 3 et l'expérimentation 2 est que dans ce cas ci nous avons modifié le modèle de Newman, Jin et Girvan afin d'y ajouter une règle supplémentaire : Si un lien existe entre deux individus dans le réseau d'amitié et qu'un lien existe entre les mêmes individus dans le réseau de collaboration alors la force du lien dans le réseau d'amitié ne diminue pas si les deux individus ne se rencontrent pas et le lien dans le réseau de collaboration ne perd pas non plus de sa force. La raison pour laquelle nous avons procédé ainsi est que nous voulons comparer l'évolution de la corrélation par rapport aux résultats de l'expérimentation 2.

Résultats attendus :

Nous pensons que la corrélation entre le réseau d'amitié et le réseau de collaboration devrait être plus forte que dans le cas de l'expérimentation 2 parce que dans ce cas ci nous avons introduit dans le modèle de croissance de réseau un lien explicite entre deux réseaux sociaux.

Expérimentation 4 :

L'expérimentation 4 est similaire à l'expérimentation 2 à l'exception près que nous avons remplacé le réseau de collaboration par le réseau de conseil d'Emmanuel Lazega.

Résultats attendus :

Les résultats devraient être similaires à ceux obtenus pour l'expérimentation 2.

Expérimentation 5 :

Cette expérimentation correspond à l'expérimentation 3 dans laquelle le réseau de collaboration est remplacé par le réseau de conseil.

Résultats attendus :

Les résultats devraient être similaires à ceux obtenus pour l'expérimentation 3.

Pour ces cinq expérimentations nous avons utilisé les valeurs suivantes pour le modèle de croissance de réseau. Ces valeurs sont celles utilisées par Newman, Jin et Girvan dans leur article:

$\beta = 5$

$p0 = 0.1$

$\alpha = 1$

$zmax = 5$

$ka = 0.01$

la force minimale d'un lien est fixée à 0.3

Afin de réaliser ces cinq expérimentations nous avons besoin de pouvoir calculer le coefficient de corrélation entre deux réseaux sociaux. Dans la suite de ce chapitre nous allons d'abord donner la définition du coefficient de corrélation et ensuite nous allons programmer le calcul de ce coefficient de corrélation en langage SNSimulator.

Définition du coefficient de corrélation

Le coefficient de corrélation entre deux variables permet de mesurer la liaison qui existe entre ces deux variables. Le coefficient de corrélation est donné par le rapport de la covariance entre les deux variables au produit des écarts-type de chacune des deux variables. C'est à dire si Rp est le coefficient de corrélation et x et y sont les deux variables considérées, alors Rp est donné par:

$$r_p = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

avec la covariance entre x et y:

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x}) \cdot (y_i - \bar{y})$$

Les écarts-types sont respectivement pour x et y:

$$\sigma_x = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

$$\sigma_y = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

La formule du coefficient de corrélation entre x et y, après remplacement des termes devient donc:

$$r_p = \frac{\sum_{i=1}^{N}(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

Ce coefficient est compris entre -1 et +1. Lorsque sa valeur est nulle alors les deux variables étudiées sont indépendantes l'une de l'autre. Dans le cas des réseaux sociaux nous allons définir les variables x et y de la manière suivante:

x_i = degré du nœud i du réseau x. De la même manière y_i = degré du nœud i dans le réseau y.

La moyenne de la variable x sera définie comme le degré moyen d'un réseau. La même moyenne est définie sur le réseau y. Le coefficient de corrélation nous permettra de mesurer la relation

entre deux réseaux définis sur les mêmes nœuds. Dans le cadre de travail de recherche nous avons choisi dans la définition de notre coefficient de corrélation de ne pas tenir compte du sens des liens lorsque nous mesurons le degré d'un noeud.

Réalisation du coefficient de corrélation

Avant de programmer notre coefficient de corrélation, nous allons avoir besoin d'une fonction qui calculera le degré moyen d'un réseau. Nous allons donc programmer une fonction que nous appellerons moyenne. Le degré moyen d'un réseau est donné par le rapport de la somme des degrés des nœuds du réseau le nombre de nœuds du réseau.

Avec le langage SNSimulator la fonction moyenne se programme facilement:

```
function moyenne(x string) return number is
declare
variable number temp := 0
begin
 for i in 1 .. x.nodes.count
   loop
    temp :=temp + x.nodes[i].degree
   end loop
 return (temp/x.nodes.count)
end
begin
end
```

Dans cette fonction nous parcourons tous les nœuds du réseau x passé en paramètre et nous faisons la somme des degrés de ces nœuds. Ensuite nous divisons cette somme par le nombre de nœuds et nous retournons le résultat comme valeur de retour de la fonction moyenne.

Maintenant que nous disposons d'une manière d'obtenir le degré moyen d'un réseau, nous pouvons programmer la formule de calcul du coefficient de corrélation entre deux réseaux.
Nous allons utiliser la fonction *racine* que nous avons présentée au chapitre 3 de ce mémoire afin d'obtenir la racine carrée d'un nombre. Nous utiliserons aussi la fonction carre présentée au

chapitre 3. Nous utiliserons aussi dans notre fonction trois variables pour stocker les différentes parties du numérateur et du dénominateur de la formule du coefficient de corrélation. La fonction corrélation prend en paramètres les noms des deux réseaux dont nous voulons calculer la corrélation. Comme pour la fonction moyenne, pour obtenir le résultat attendu nous parcourons les nœuds des réseaux passés en paramètres. Nous obtenons le programme ci-dessus :

```
function correlation(x string, y string) return number is
declare
        variable number  numerateur := 0
        variable number  denominateur1 := 0
        variable number  denominateur2 := 0
        variable number resultat := 0
begin
        for i in 1 .. x.nodes.count
        loop
                numerateur := numerateur + (x.nodes[i].degree –
        moyenne(x))*(y.nodes[i].degree – moyenne(y))
                denominateur1 := denominateur1 + carre(x.nodes[i].degree –
        moyenne(x))
                denominateur2 := denominateur2 + carre(y.nodes[i].degree –
        moyenne(y))
        end loop
        resultat := numerateur/(racine(denominateur1)*racine(denominateur2))
        return resultat
end
```

Les deux fonctions définies dans ce chapitre peuvent être utilisées dans n'importe quel programme SNSimulator.

Résultats obtenus et interprétation

Dans ce paragraphe nous allons présenter les résultats que nous avons obtenus à la suite de nos expérimentations et nous essayerons d'expliquer ces résultats par rapport aux résultats que nous attendions. Nous présentons également pour chacune de nos expériences l'état des réseaux utilisés après l'expérimentation. Dans les illustrations de corrélation entre réseaux et de degrés moyens de réscaux des pages qui suivent, l'axe d'abscisse représente le nombre d'itérations de l'expérimentation et l'axe d'ordonnée représente soit le coefficient de corrélation soit le degré moyen du réseau selon le cas.

Expérimentation 1:

Dans cette expérimentation nous avons appliqué le modèle de Newman, Jin et Girvan au réseau d'amitié et nous avons calculé la corrélation avec l'état initial du réseau d'amitié.

Nous constatons (illustration 32) que la courbe de corrélation de degré entre le réseau d'amitié initial et le réseau modifié par le modèle de Newman, Jin et Girvan connaît une évolution décroissante allant de 1 au début de l'expérimentation 1 à une valeur d'environ 0.084. Cette évolution est conforme aux résultats que nous attendions. Les illustrations 33 et 34 présentent le réseau d'amitié avant l'expérimentation et après l'expérimentation. Nous pouvons remarquer que à la fin de l'expérimentation 1 le réseau d'amitié comporte beaucoup moins de liens qu'avant le début de l'expérimentation. Nous pensons que cette évolution du nombre de liens dans le réseau était prévisible étant donné que le modèle de Newman, Jin et Girvan comporte une règle d'éliminations de liens lorsque les individus du groupe social étudié ne se rencontrent pas.

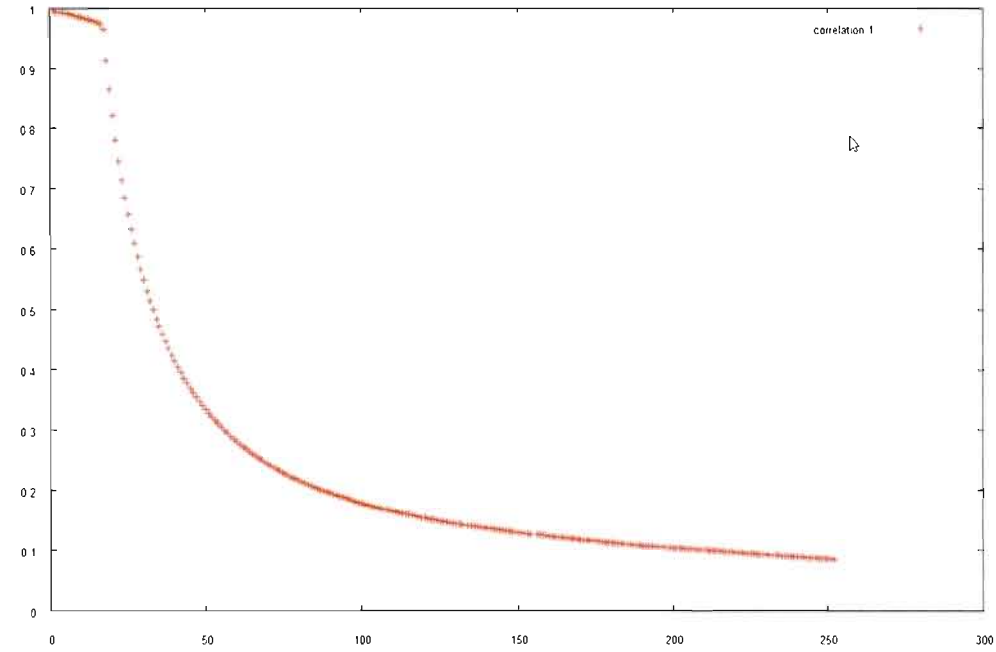

Illustration 32: Évolution de la corrélation de degré entre les deux copies du réseau d'amitié

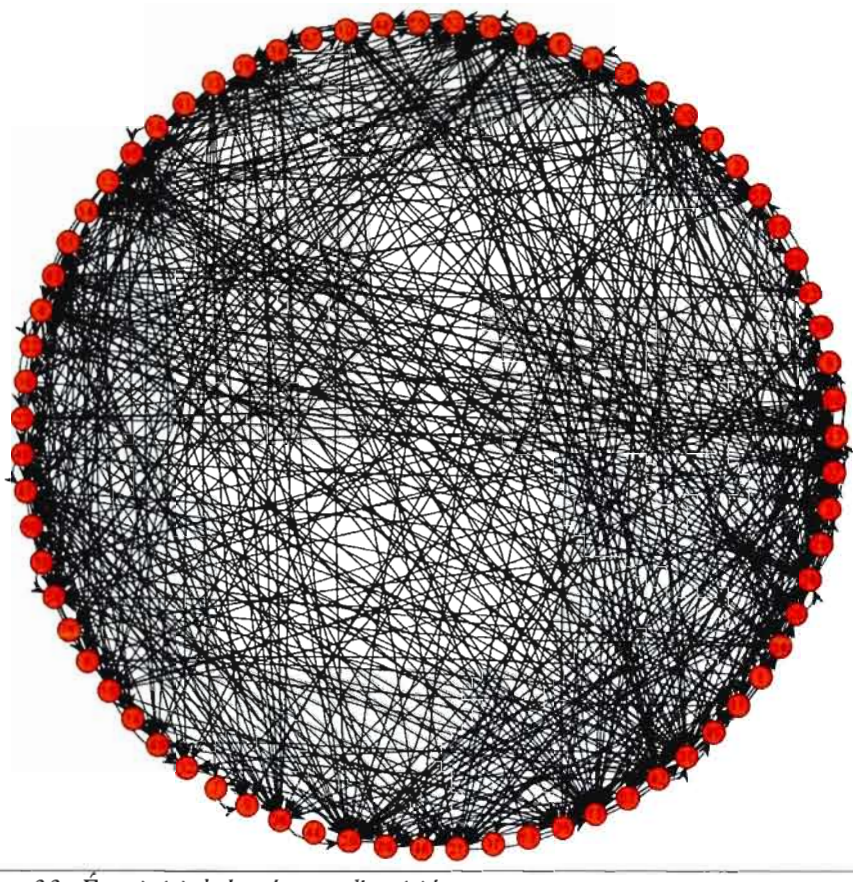

Illustration 33: État initial du réseau d'amitié

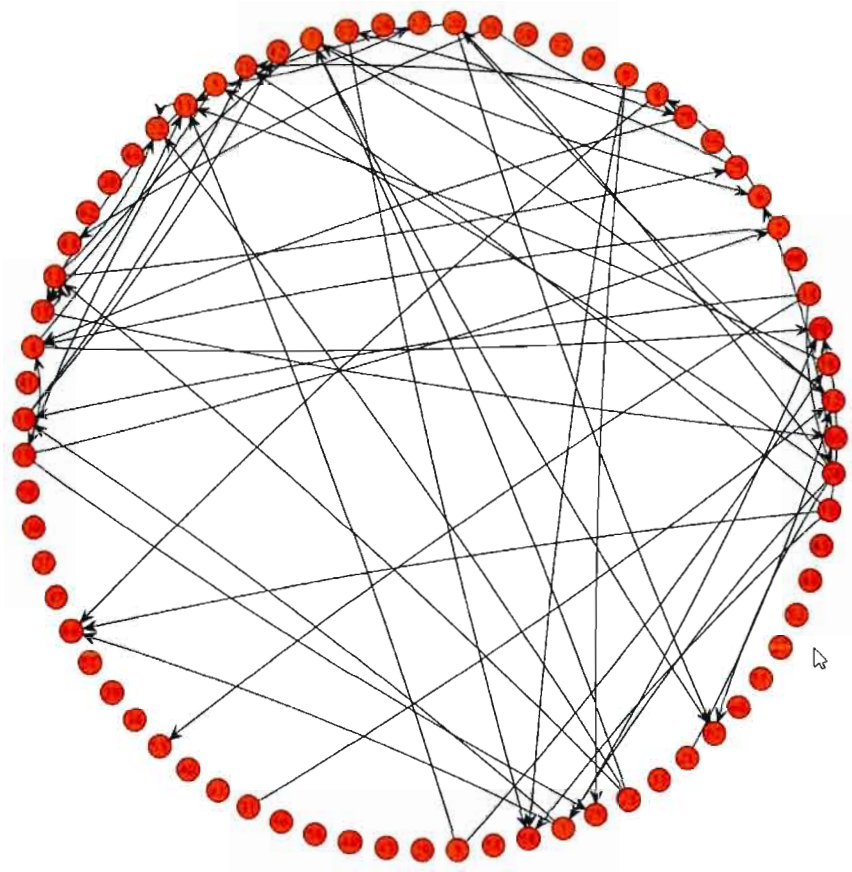

Illustration 34: État final du réseau d'amitié à la suite de l'exécution de l'expérimentation 1

Expérimentation 2 :

Dans cette expérimentation nous avons appliqué le modèle de Newman, Jin et Girvan aux réseaux d'amitié et de collaboration et nous avons calculé la corrélation des deux réseaux résultants. Les deux réseaux évoluent de manière indépendante.

Nous constatons (illustration 35) que la courbe de corrélation entre les réseaux d'amitié et de collaboration connait une évolution décroissante similaire aux résultats obtenus après l'expérimentation 1 mais la valeur minimale de la corrélation est beaucoup plus élevée avec une valeur d'environ 0.59 contre 0.084 c'est à dire près de cinq fois plus. Ce résultat est conforme à nos prévisions et s'explique par le fait que le même modèle est appliqué aux deux réseaux. Nous pouvons également constater que le degré moyen des deux réseaux présente une courbe descendante en escalier. Les degrés moyens des nœuds des deux réseaux diminuent graduellement puis semblent se stabiliser à une valeur commune d'environ 2 liens. Nous pensons que la structure en escalier de la courbe des degrés moyens (illustrations 36 et 37) des réseaux est liée à la règle de diminution de la force des liens du modèle de croissance des réseaux de Newman, Jin et Girvan que nous avons présentée plus tôt dans ce chapitre. En effet après un certain nombre d'itérations pendant lesquelles les nœuds de départ et d'arrivée d'un lien ne se rencontrent pas, la valeur de la force de ce lien diminue jusqu'à devenir inférieure à la valeur minimale de force nécessaire en dessous de laquelle le lien est détruit et que nous avons fixée à 0.3. La structure en escalier montre que plusieurs liens voient leur force devenir inférieure à 0.3 au même moment et sont donc supprimés au cours de la même itération ce qui fait chuter le degré moyen du réseau. Cette évolution du degré moyen des réseaux observée à la suite de cette expérimentation est donc conforme selon nous à la définition du modèle de croissance de réseaux de Newman, Jin et Girvan. Les illustrations 38 et 39 présentent les réseaux d'amitié et de collaboration après l'expérimentation 2. Nous constatons que le nombre de liens dans le réseau d'amitié a diminué par rapport au début de l'expérimentation mais le nombre de liens dans le réseau de collaboration est beaucoup plus élevé. Nous pensons que cela et dû au fait qu'initialement le degré moyen du réseau d'amitié est inférieur à celui du réseau de collaboration (un peu moins de 20 contre un peu plus de 30 comme on peu voir dans les illustrations 36 et 37).Étant donné que le modèle de Newman, Jin et Girvan est fondé sur une probabilité de rencontre entre deux individus dans un réseau, la probabilité qu'un lien de collaboration existe entre les individus est plus élevée que celle d'avoir un lien d'amitié.

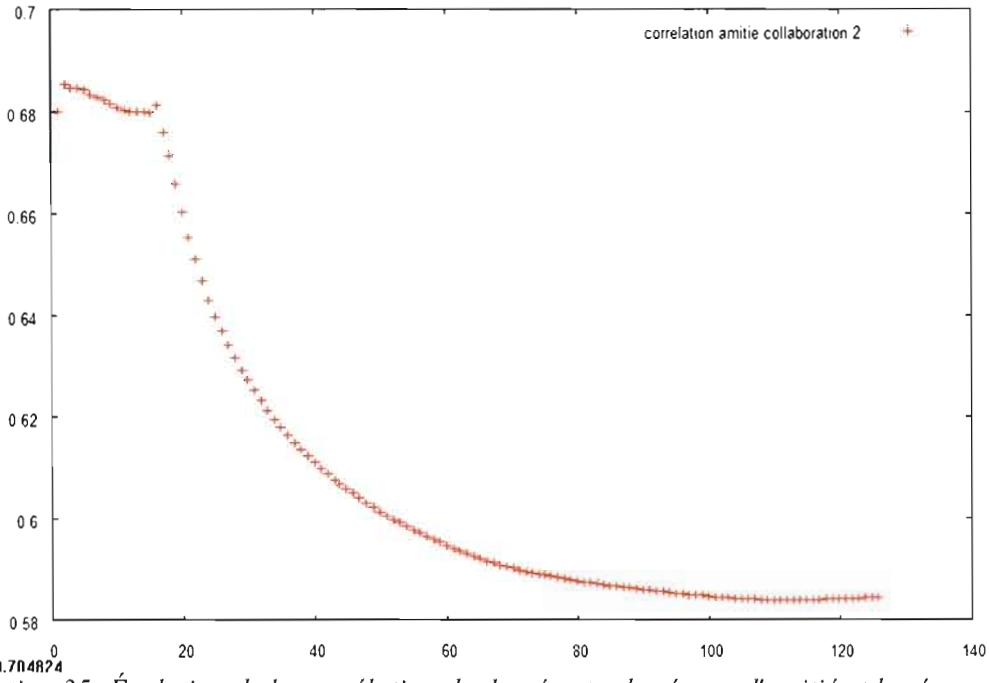

Illustration 35: Évolution de la corrélation de degré entre le réseau d'amitié et le réseau de collaboration

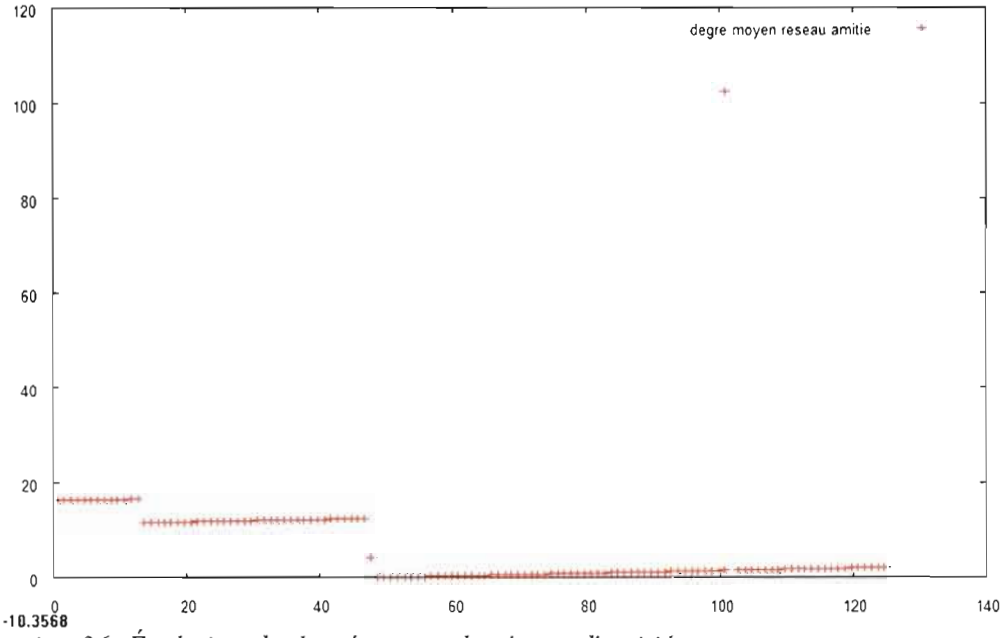

Illustration 36: Évolution du degré moyen du réseau d'amitié

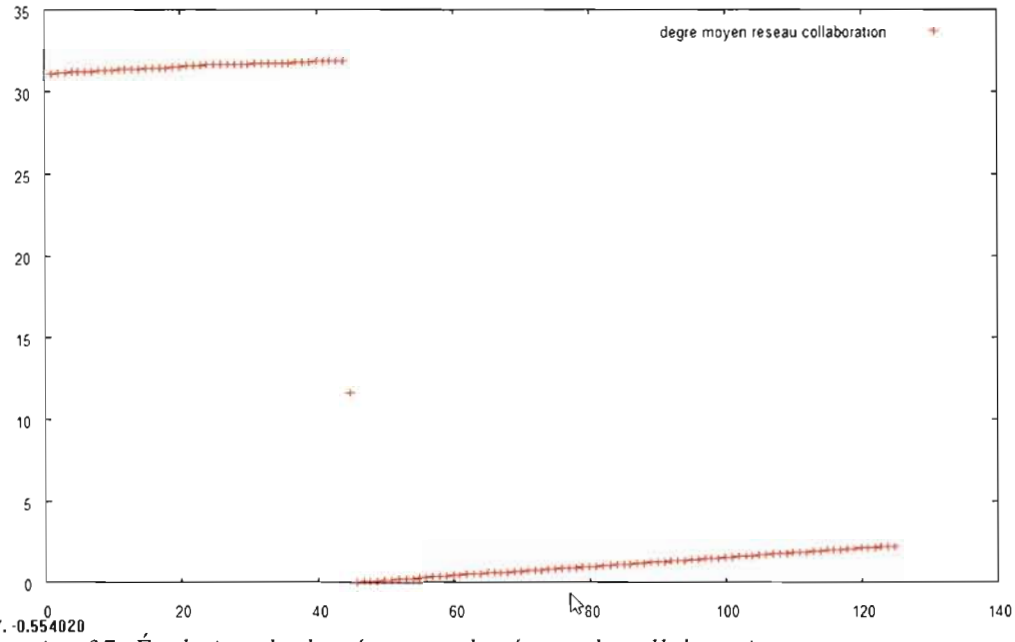

Illustration 37: Évolution du degré moyen du réseau de collaboration

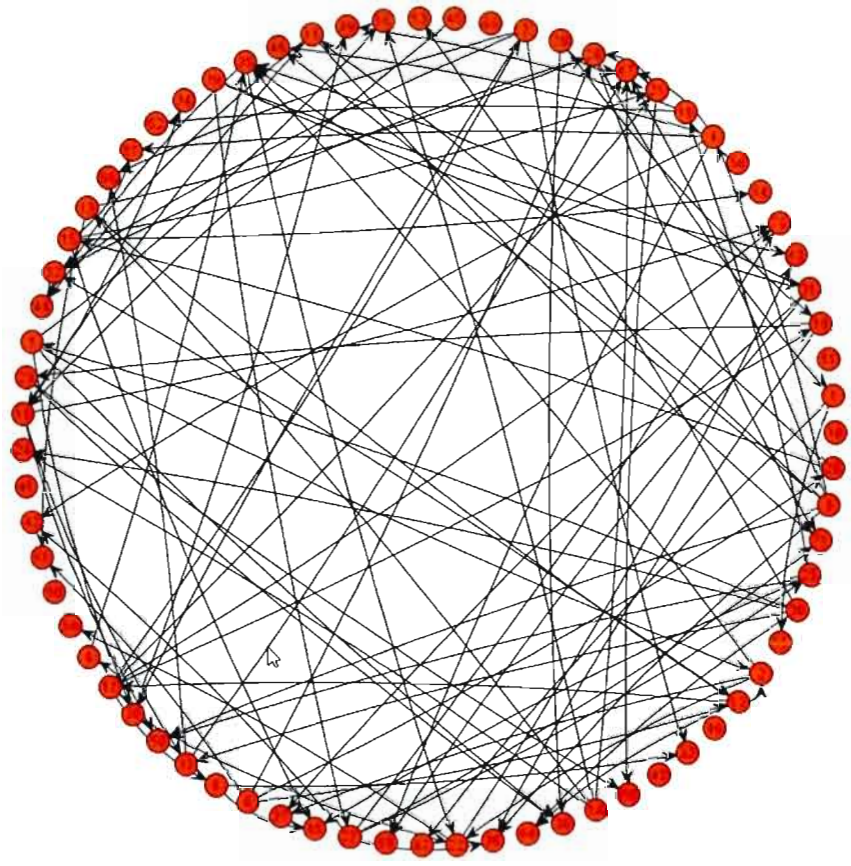

Illustration 38: État final du réseau d'amitié après l'expérimentation 2

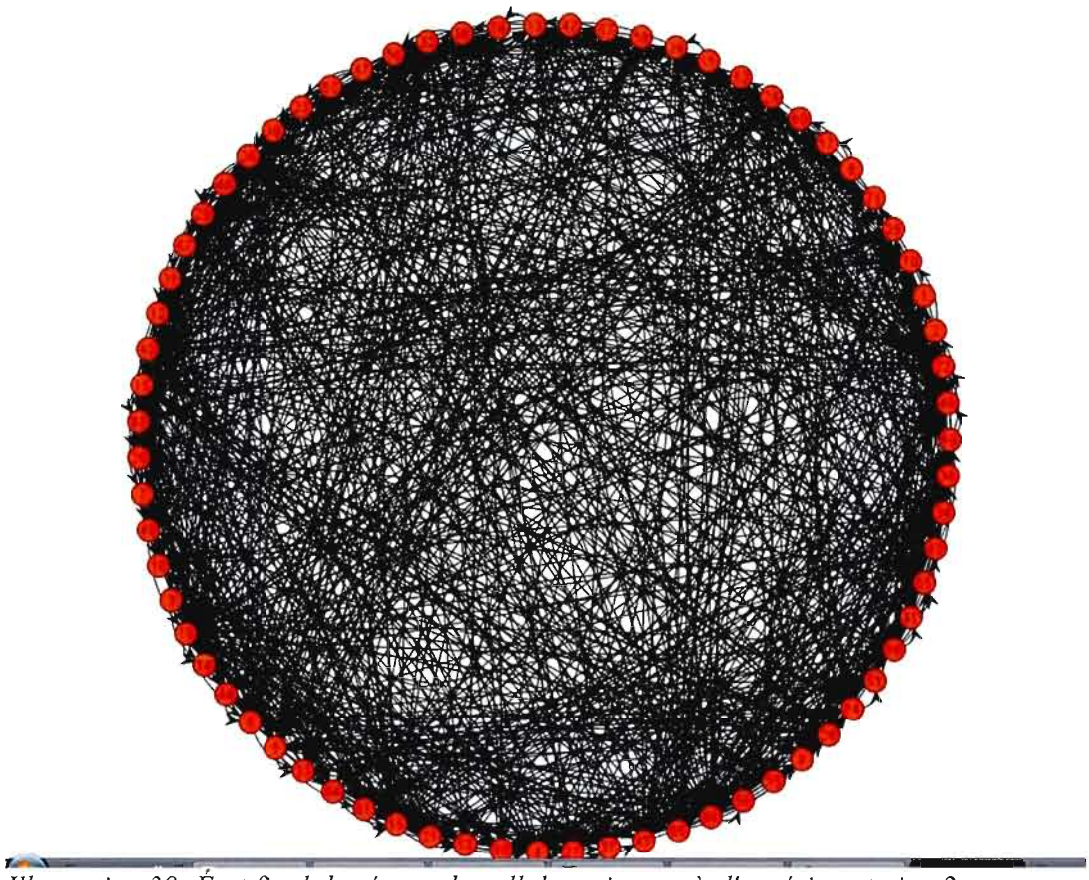

Illustration 39: État final du réseau de collaboration après l'expérimentation 2

Expérimentation 3:

Dans cette expérimentation nous avons appliqué le modèle de Newman, Jin et Girvan aux réseaux d'amitié et de collaboration. Les deux réseaux évoluent de manière corrélée.

La courbe de la corrélation (illustration 40) évolue en phases : d'abord une décroissance de la valeur 0.68 jusqu'à environ 0.6 puis la corrélation entre les réseaux de collaboration et d'amitié augmente jusqu'à atteindre 0.64 à la fin de notre expérimentation. Nous pensions que la corrélation entre les réseaux aurait une évolution ascendante tout le long de l'expérimentation 3 mais les résultats obtenus diffèrent un peu. Néanmoins nous pensons que si notre expérimentation avait duré un plus longtemps, l'augmentation de la corrélation se serait poursuivie. À la différence de l'expérimentation 2, le degré moyen du réseau d'amitié (illustration 41) ne présente pas la structure décroissante en escalier rencontrée précédemment mais plutôt une courbe croissante comprise entre les valeurs 16 et 17.5 nœuds. Nous pensons que cela est dû au fait que nous avons ajouté dans le cadre de cette expérience la règle selon laquelle la force d'un lien dans le réseau d'amitié ne diminue pas lorsqu'il existe un lien correspondant dans le réseau de collaboration. Cette règle diminue le nombre de liens dans le réseau d'amitié qui sont détruits parce que leur force devient inférieure à la valeur minimale que nous nous sommes donnés et qui est de 0.3 ce qui permet l'augmentation du degré moyen des nœuds du réseau d'amitié tout le long de cette expérience. Le degré moyen des nœuds du réseau de collaboration (illustration 42) conserve la forme en escalier et décroît d'environ 31 liens par nœud jusque à presque 9 liens par nœud. Les illustrations 43 et 44 présentent le réseau d'amitié et le réseau de collaboration après l'expérimentation 3. À la différence de l'expérimentation 2, il semble que dans le cas de cette expérimentation le nombre de liens dans le réseau d'amitié soit supérieur au nombre de liens dans le réseau de collaboration, Nous pensons plutôt que le nombre de liens dans le deux réseaux deviendrait de plus en plus proches de l'égalité étant donné que nous avons ajouté dans cette expérimentation un lien entre les deux réseaux (voir la description de l'expérimentation 3).

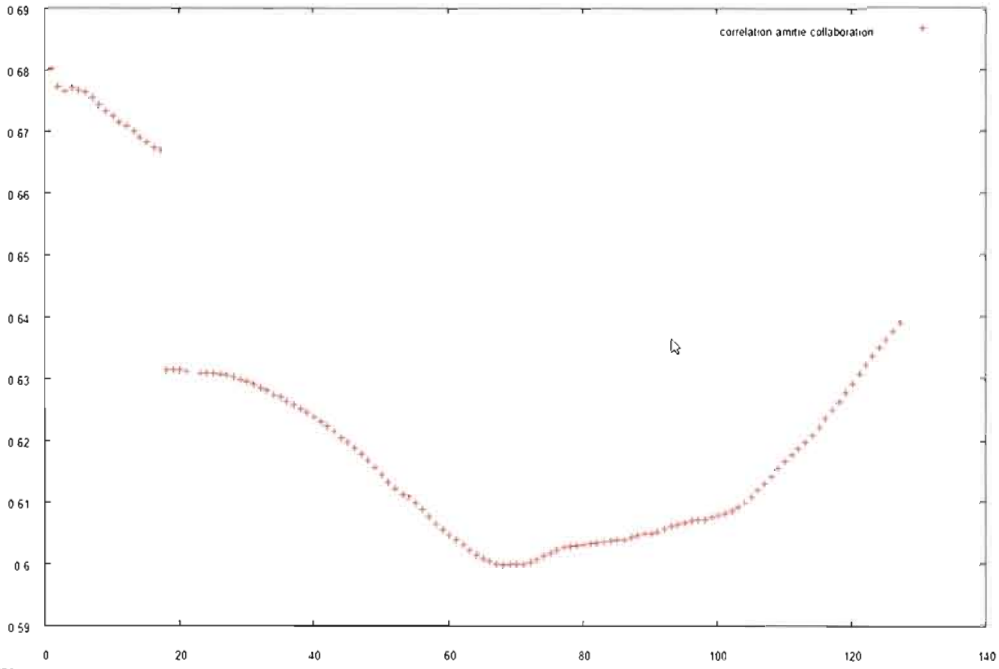

Illustration 40: Évolution de la corrélation entre le réseau d'amitié et le réseau de collaboration

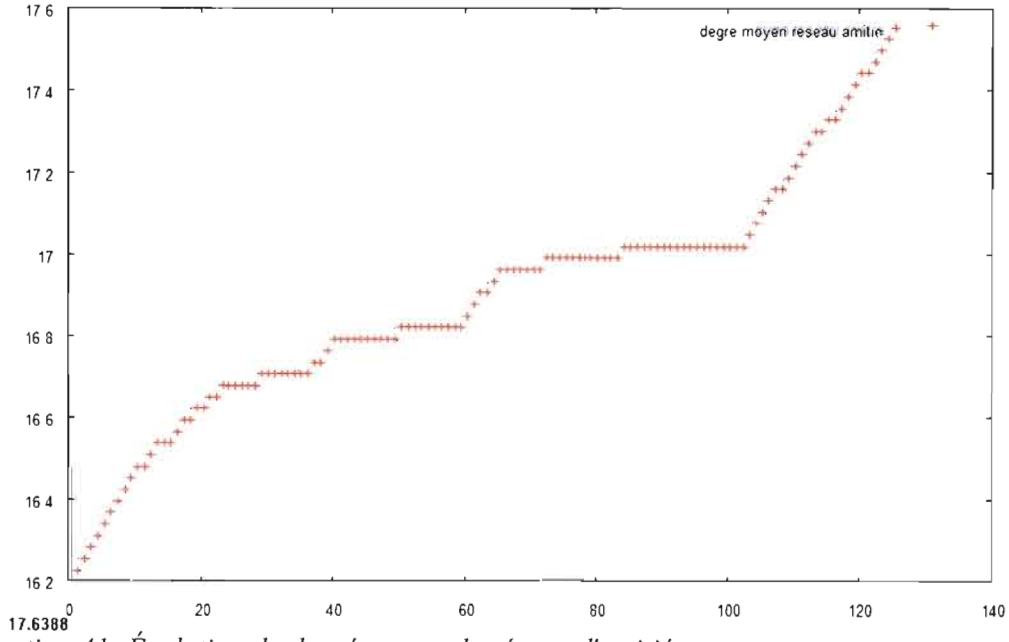

Illustration 41: Évolution du degré moyen du réseau d'amitié

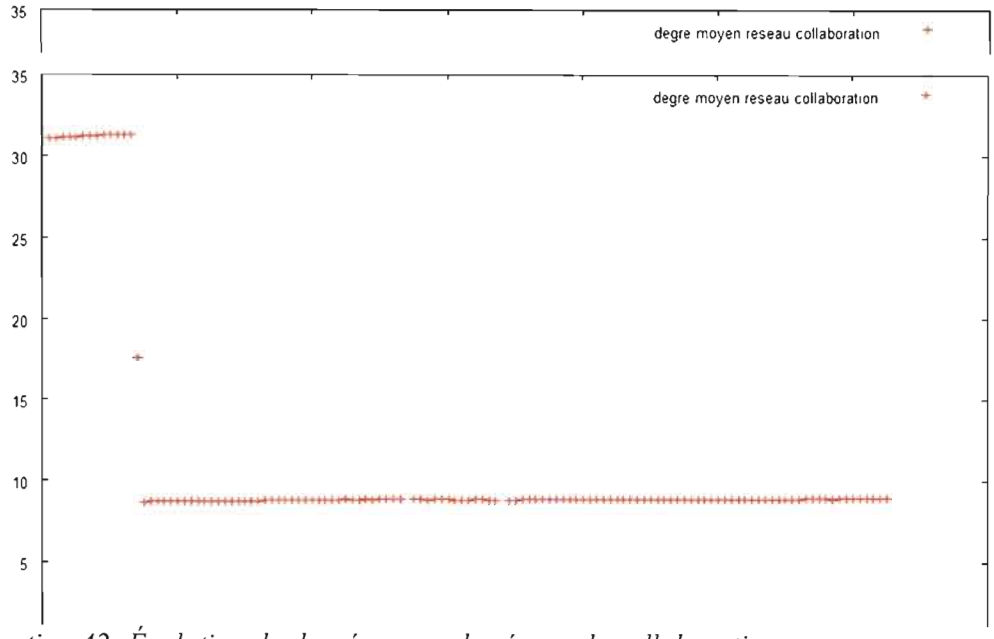

Illustration 42: Évolution du degré moyen du réseau de collaboration

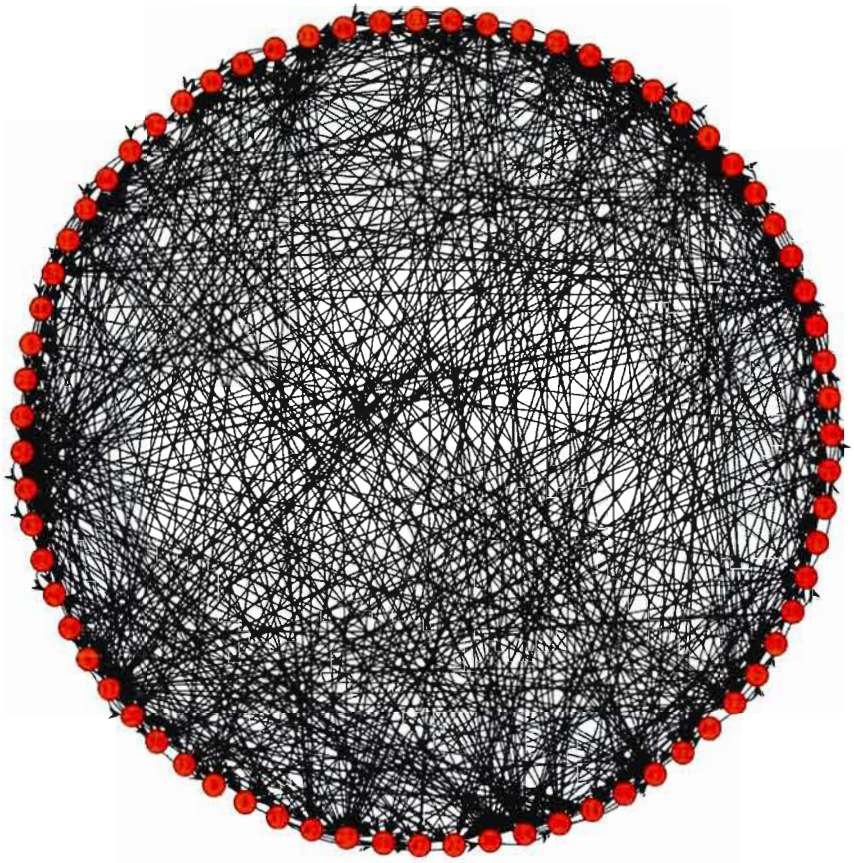

Illustration 43: État final du réseau d'amitié après l'expérimentation 3

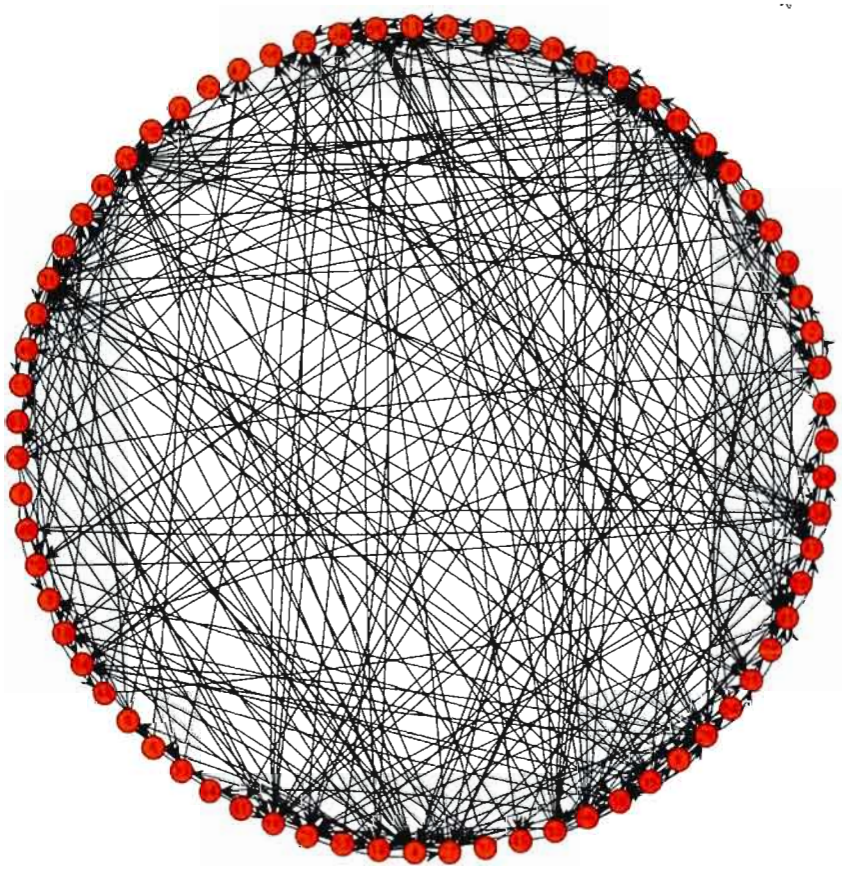

Illustration 44: État final du réseau de collaboration après l'expérimentation 3

Expérimentation 4:

Nous avons appliqué le modèle de Newman, Jin et Girvan aux réseaux d'amitié et de conseil du cabinet d'avocats. Les deux réseaux évoluent de manière indépendante.

Comme nous le prévoyions, nous obtenons une courbe de corrélation (illustration 45) semblable à celle obtenue pour l'expérimentation 2. L'évolution des degrés moyens des nœuds des réseaux d'amitié (illustration 46) et de collaboration (illustration 47) présentent un aspect en escalier et se stabilisent autour de 3 liens par nœud. Les illustrations 48 et 49 présentent le réseau d'amitié et le réseau de conseil après l'expérimentation 4. Nous pouvons constater que le nombre de liens dans les deux réseaux semble égal ce qui confirme la constatation que le degré moyen des deux réseaux est égal à environ 3 liens par nœud.

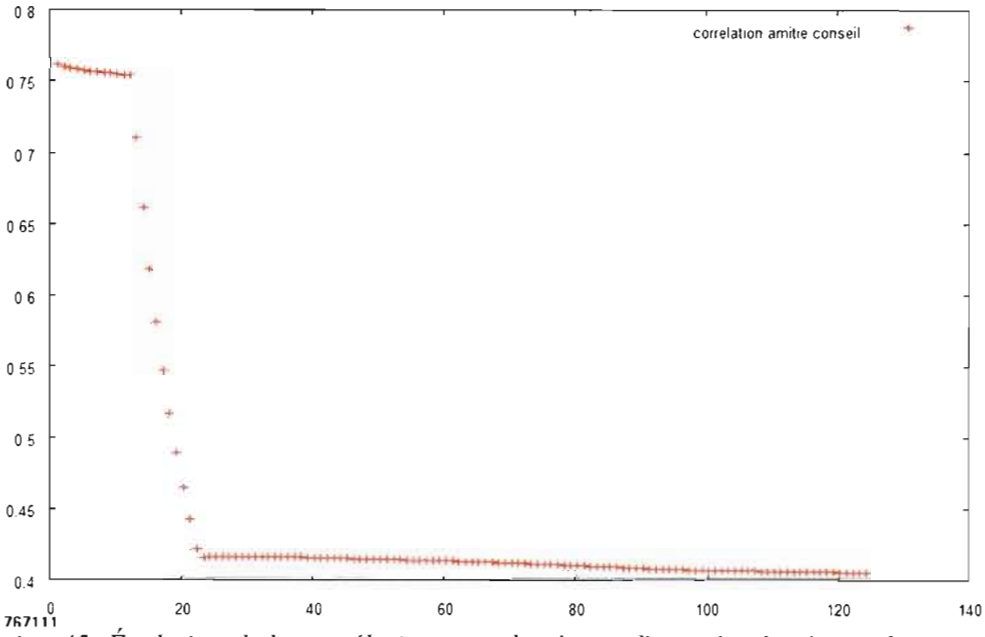

Illustration 45: Évolution de la corrélation entre le réseau d'amitié et le réseau de conseil

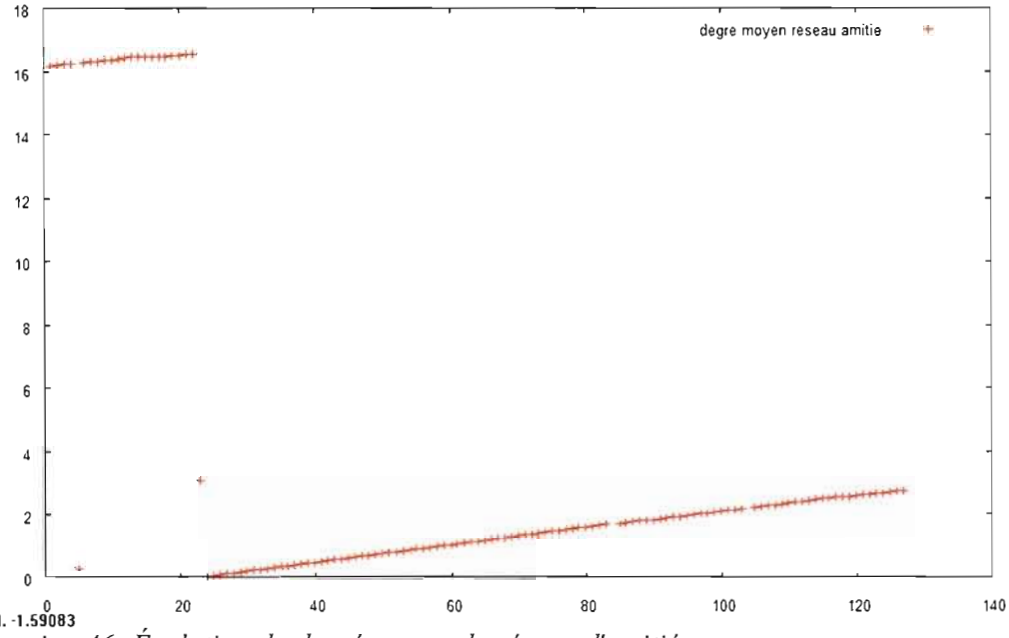

Illustration 46: Évolution du degré moyen du réseau d'amitié

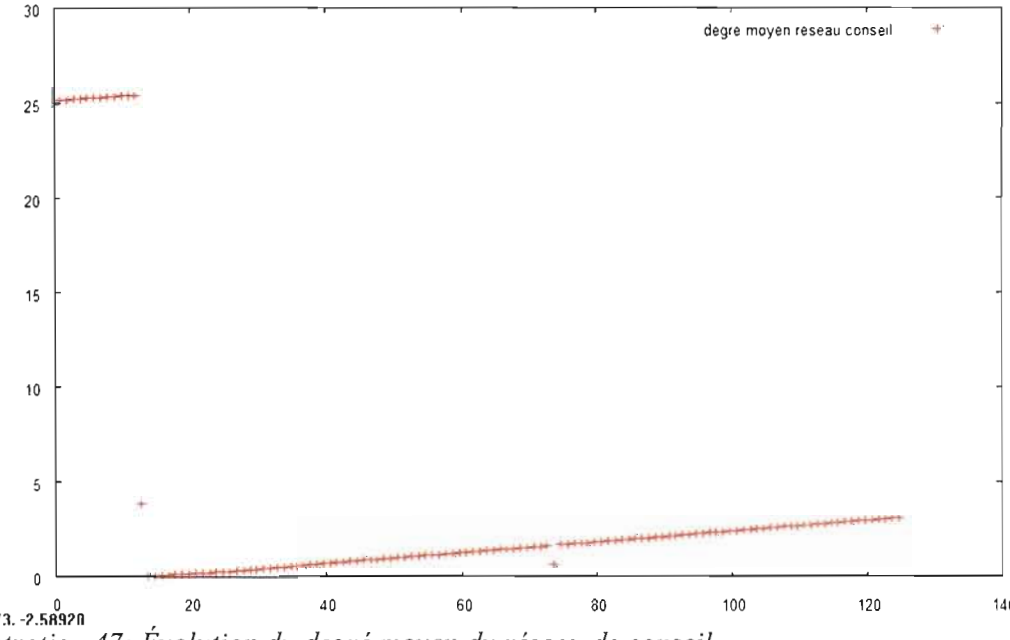

Illustration 47: Évolution du degré moyen du réseau de conseil

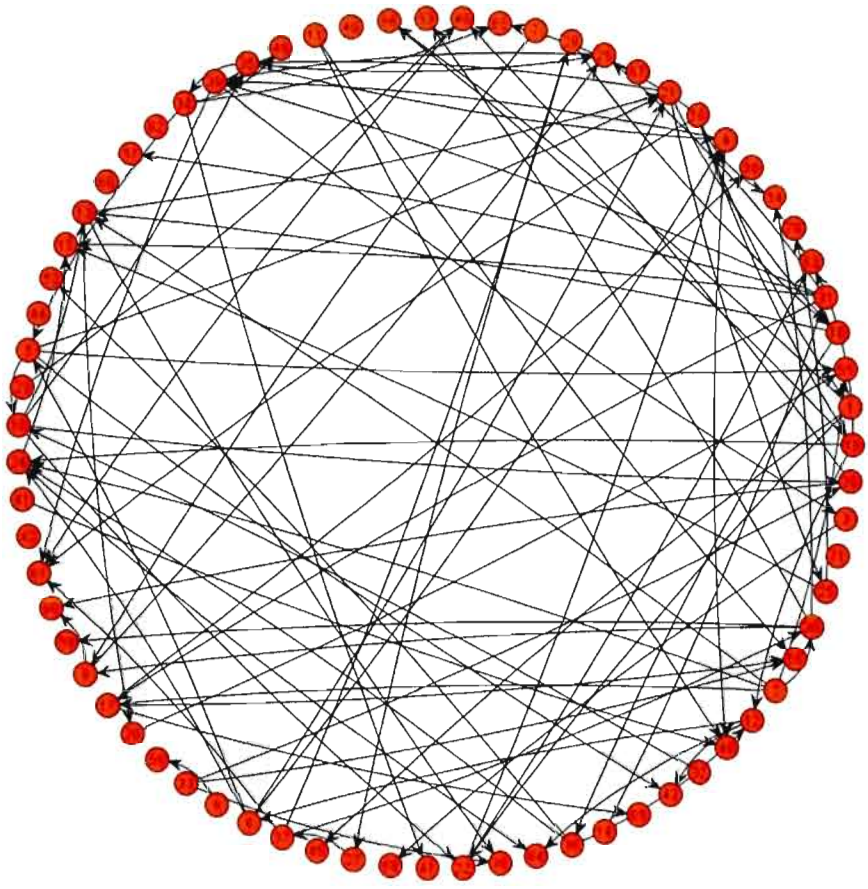

Illustration 48: État final du réseau d'amitié après l'expérimentation 4

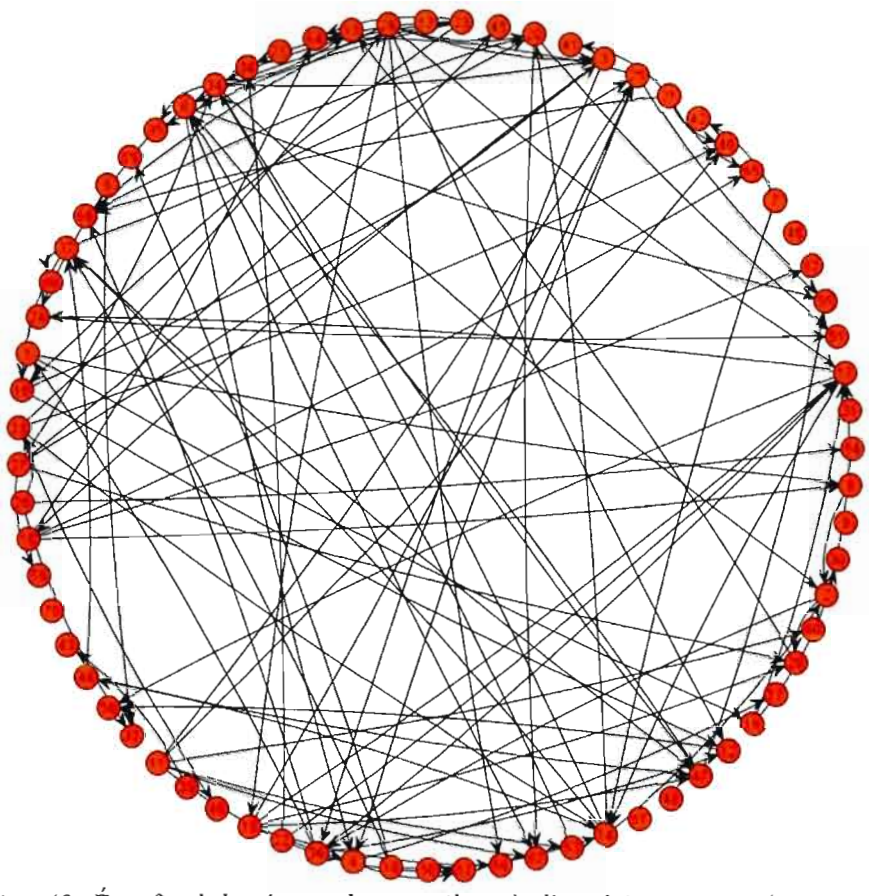

Illustration 49: État final du réseau de conseil après l'expérimentation 4

Expérimentation 5:

Nous avons appliqué le modèle de Newman, Jin et Girvan aux réseaux d'amitié et de conseil. Dans le cadre de cette expérience nous avons établi un lien explicite entre les deux réseaux.

La courbe de corrélation entre le réseau d'amitié et le réseau de conseil (illustration 50) présente deux phases : une phase descendante jusqu'à l'itération 20 puis une phase croissante. Nous pensons que la phase de croissance de la corrélation préfigure les résultats que nous aurions obtenus lors de l'expérimentation 3 si nous avions poursuivi l'expérimentation 3. Les illustrations 51 et 52 présentent le degré moyen des réseaux d'amitié et de conseil et nous retrouvons la même forme d'escalier que dans la majorité des expérimentations précédentes. Les deux réseaux ont des degrés moyen qui semblent tendre vers la valeur 12, Les illustrations 53 et 54 présentent le réseau d'amitié et le réseau de conseil après l'expérimentation 5. Les deux réseaux sont très semblables ce qui confirme le constat que les degrés moyens des deux réseaux tendent vers la même valeur.

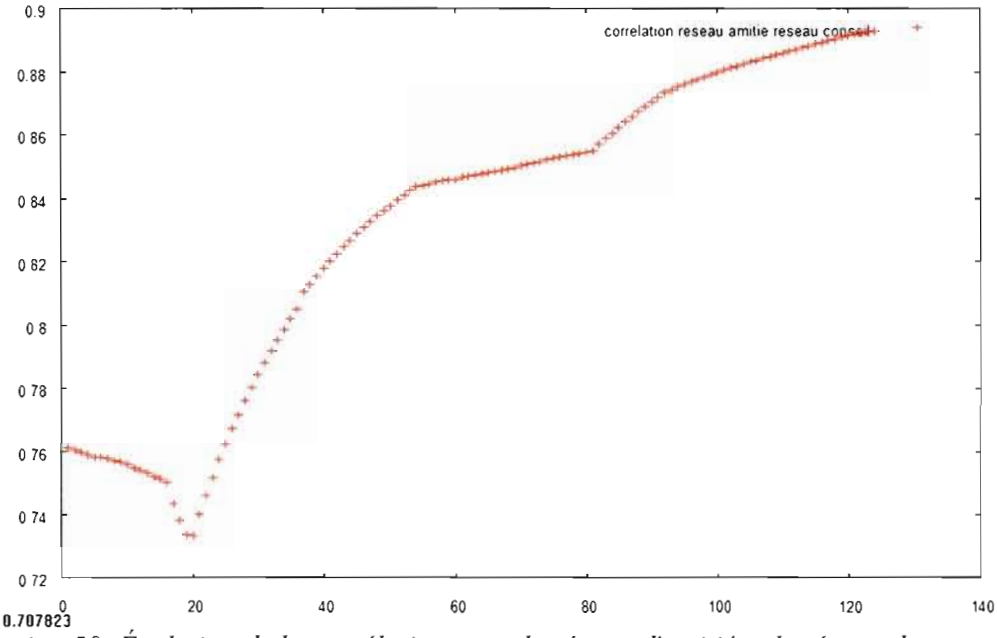

Illustration 50: Évolution de la corrélation entre le réseau d'amitié et le réseau de conseil

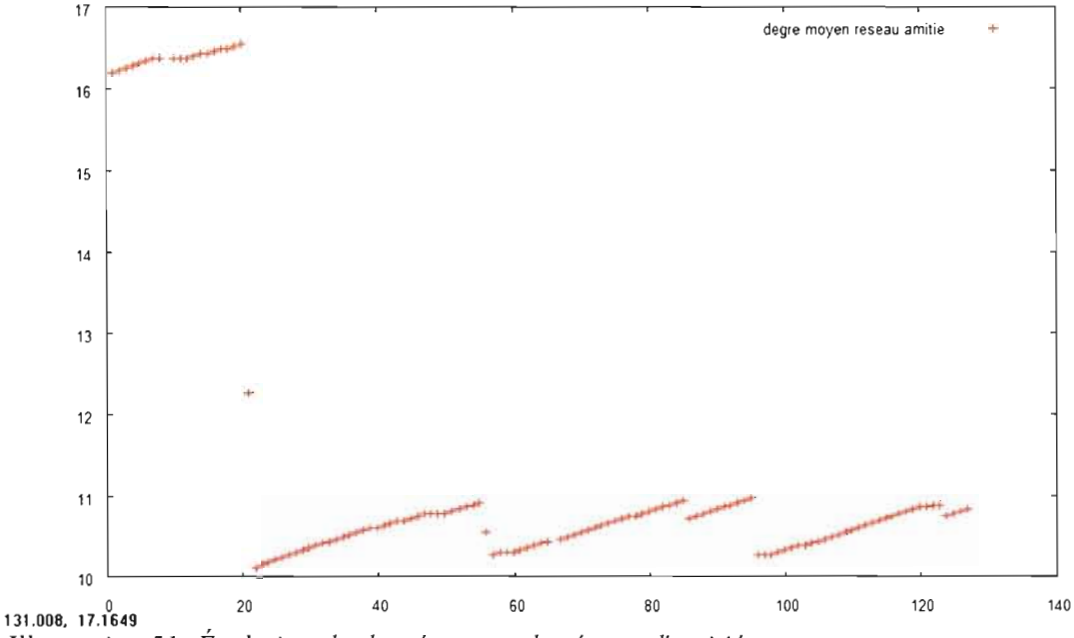

Illustration 51: Évolution du degré moyen du réseau d'amitié

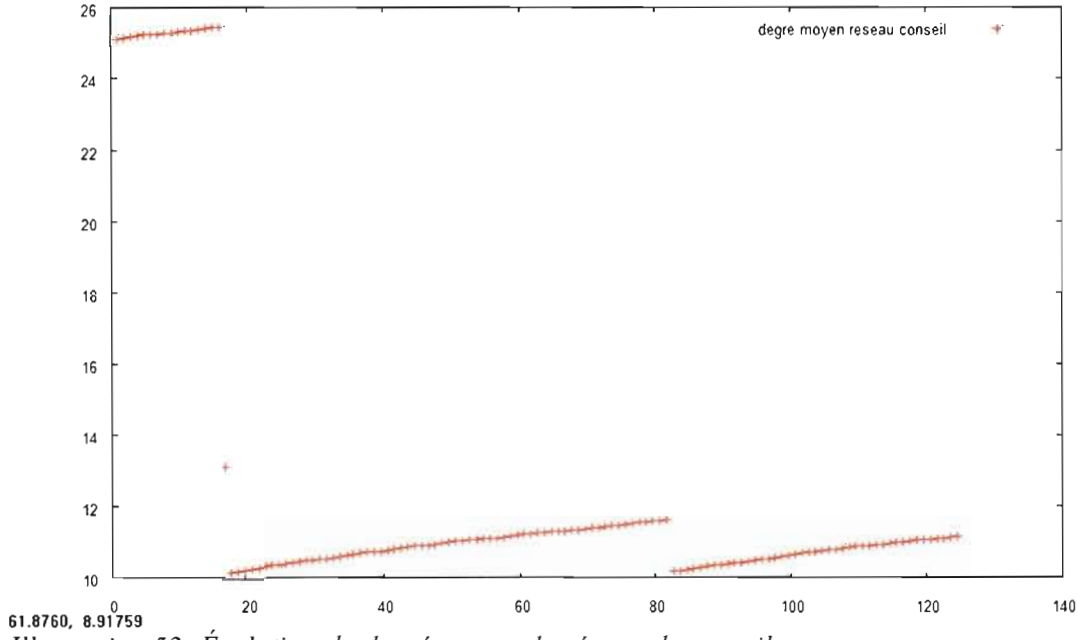

Illustration 52: Évolution du degré moyen du réseau de conseil

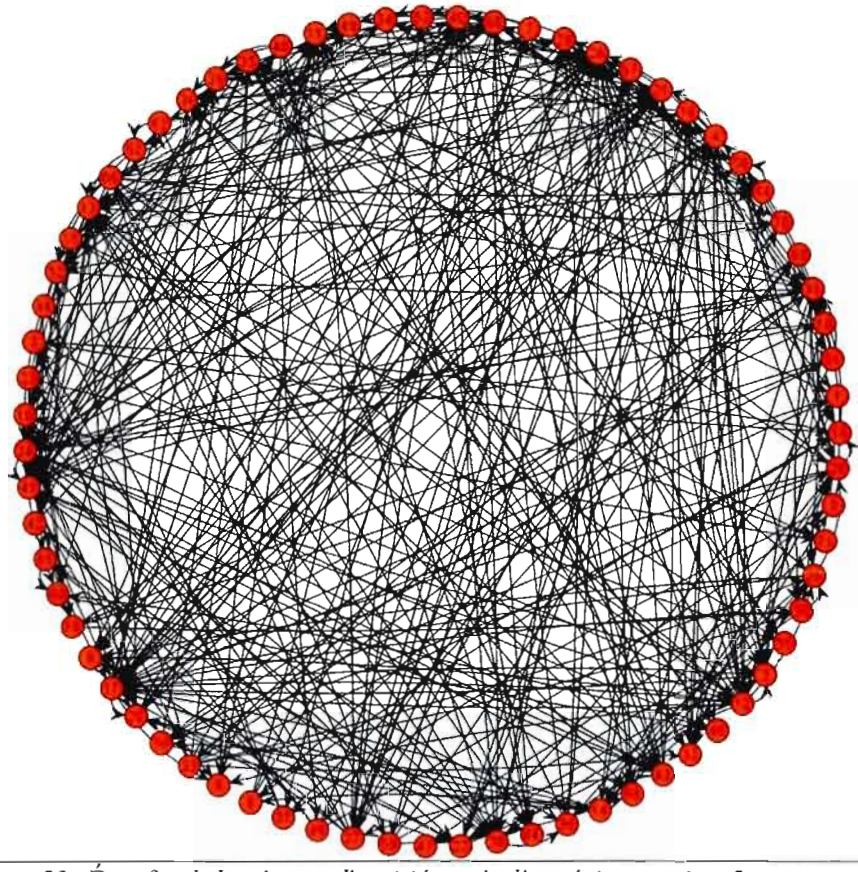

Illustration 53: État final du réseau d'amitié après l'expérimentation 5

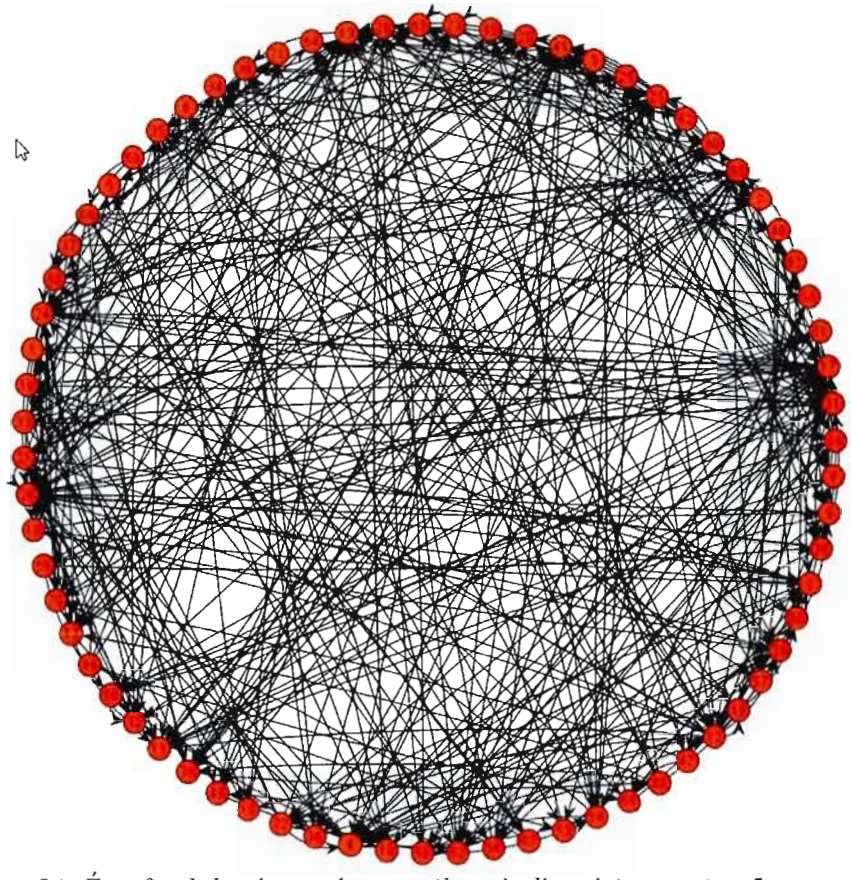

Illustration 54: État final du réseau de conseil après l'expérimentation 5

Les expériences que nous avons effectuées dans les pages précédentes nous ont permis de montrer que nous pouvions faire évoluer plusieurs réseaux de relation qui s'influencent les uns les autres et mesurer la corrélation entre ces réseaux.

À la suite de cette mise en pratique de SNSimulator nous pensons que notre interpréteur pour le langage SNSimulator permet de simuler l'évolution de plusieurs réseaux au sein d'un même ensemble social et peut permettre de réaliser des expérimentations qui seraient beaucoup plus complexes à faire avec d'autres outils. Nous pensons également que le langage que nous proposons à la suite de ce projet de recherche de maitrise en informatique est assez simple pour être accessible à quelqu'un qui n'est pas un spécialiste en informatique.

Dans le chapitre suivant nous allons faire une revue des principaux outils de modélisation de réseaux sociaux et nous allons examiner leurs fonctionnalités par rapport aux objectifs de ce mémoire.

Chapitre 6:

Évaluation de SNSimulator par rapport aux outils existants

Dans ce chapitre nous allons présenter quelques logiciels disponibles actuellement et nous allons préciser leurs fonctionnalités par rapport à notre problématique qui consiste à proposer un outil de modélisation et de simulation de réseaux multiplexes et à notre plate-forme SNSimulator. Nous présenterons d'abord des logiciels de modélisation et ensuite nous présenterons des logiciels de simulation.

Logiciels de modélisation

De nombreux outils de modélisation de réseaux existent. Nous allons surtout nous intéresser aux logiciels suivants : Pajek, R, Ucinet, NetSim, NEGOPY, Stocnet et Graphviz. Nous comparerons la capacité de ces logiciels à permettre la modélisation de plusicurs réscaux sociaux simultanés par rapport aux possibilités de notre logiciel, SNSimulator.

Pajek :

Pajek est un logiciel d'analyse et de visualisation de réseaux sociaux développé initialement pour la faculté des sciences sociales de l'université de Ljubljana en Slovénie. Il s'agit d'un logiciel fonctionnant dans l'environnement Windows et gratuit pour un usage non commercial. Pajek permet d'analyser de réseaux sociaux et de les visualiser. Pajek permet de déterminer pour un réseau social les centralités de proximité et d'intermédiarité, de partitionner les réseaux afin déterminer les nœuds structurellement équivalents. À la différence de SNSimulator qui n'est pas limité dans le nombre de réseaux simultanés, Pajek peut manipuler au maximum deux réseaux simultanément. Pajek n'offre aucune possibilité de simulation de l'évolution dynamique de réseaux sociaux.

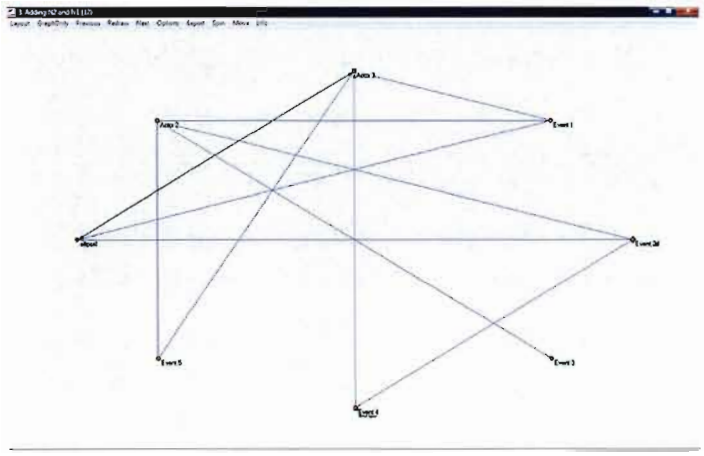

Illustration 55: Visualisation d'un réseau dans Pajek

R:

R est une librairie et un langage de calculs statistiques qui fait partie du projet GNU. R est une implémentation open source de S un langage de programmation statistique développé par les laboratoires Bell. R est une librairie statistique et ne se spécialise pas dans la manipulation de réseaux sociaux. R est surtout fait pour un usage à partir de la ligne de commande.

Étant donné que R n'est pas spécifiquement conçu pour les réseaux sociaux (il existe néanmoins une extension de R appelée sna spécialisée dans l'analyse de réseaux sociaux) et du fait de son usage de la ligne de commande, il est nécessaire d'avoir une connaissance poussée du langage afin de pouvoir utiliser R. Cela rend R complexe à utiliser pour par exemple un sociologue qui n'a aucune formation en programmation. Il ne permet pas de simulation dynamique de réseaux. Il n'est donc pas aussi bien adapté à la modélisation et à la de réseaux multiplexes que notre plate-forme SNSimulator.

NEGOPY: Logiciel d'analyse de réseaux réalisé par l'université Simon Fraser de Vancouver en Colombie-Britannique. Il fonctionne à partir de la ligne de commande et offre plusieurs fonctionnalités comme la recherche de cliques. Il ne permet pas de simulation dynamique de réseaux contrairement à SNSimulator.

Stocnet : Logiciel libre d'analyse de réseaux. Il permet d'appliquer les modèles P2 et P* (appelé SIENA). Il permet de simuler l'application de ces deux modèles sur un réseau. Il ne permet pas de simulation dynamique de réseaux comme le permet SNSimulator

Ucinet :

Ucinet est un logiciel commercial d'analyse de réseaux sociaux. Une extension gratuite à Ucinet appelée NetDraw permet la visualisation de réseaux sociaux. Ucinet de permet pas de simulation dynamique de réseaux à la différence de SNSimulator.

GraphViz:

Logiciel de visualisation de réseaux. Il s'agit d'une librairie généraliste permettant de visualiser toutes sortes de graphes. Il n'offre pas de fonctionnalités d'analyse de réseaux au sens sociologique du terme. Il ne permet pas de simulation dynamique de réseaux comme SNSimulator.

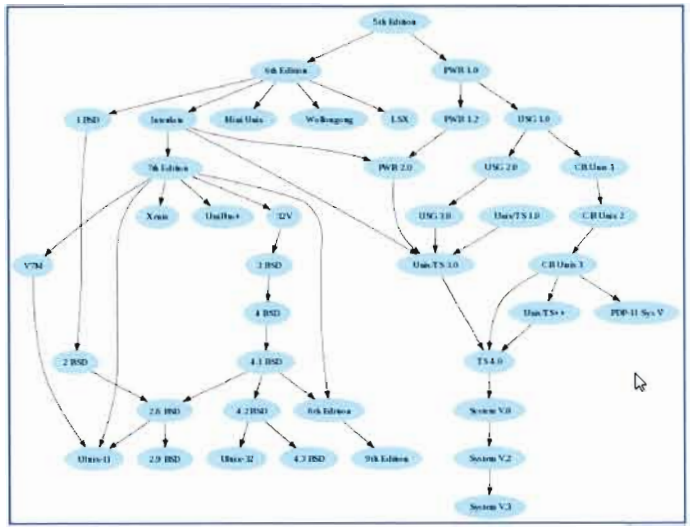

Illustration 56: Visualisation de graphe avec GraphViz

NetSim :

Logiciel de simulation et de visualisation de réseaux sociaux réalisé par Mlle Mélanie Lord à l'université du Québec à Montréal [40]. Ce logiciel permet surtout la simulation dynamique de réseau social. Il offre également quelques fonctionnalité d'analyse de réseau notamment il permet d'obtenir le degré d'un nœud du réseau. Il est développé en java et utilise la librairie Prefuse pour la visualisation des réseaux sociaux.
Les principaux inconvénients de NetSim résident dans le fait que NetSim ne peut gérer qu'un seul réseau à la fois et que le langage de NetSim pourrait paraître un peu ardu pour un néophyte.

Exemple de déclaration de la fonction de nom fi dans NetSim :

```
DEFINE_FUNCTION (link, fi)
1 /(e ** (b * (start_node().degree() - z_limite)) + 1);
```

Le langage du logiciel NetSim est aussi relativement rigide, n'offrant pas de boucles, de conditions etc. L'avantage de notre outil SNSimulator par rapport à NetSim consiste dans le fait que SNSimulator permet à l'utilisateur d'avoir plusieurs réseaux dans la simulation et de pouvoir programmer des interactions entre les différents réseaux simulés. Nous avons réalisé SNSimulator comme une continuation de NetSim.

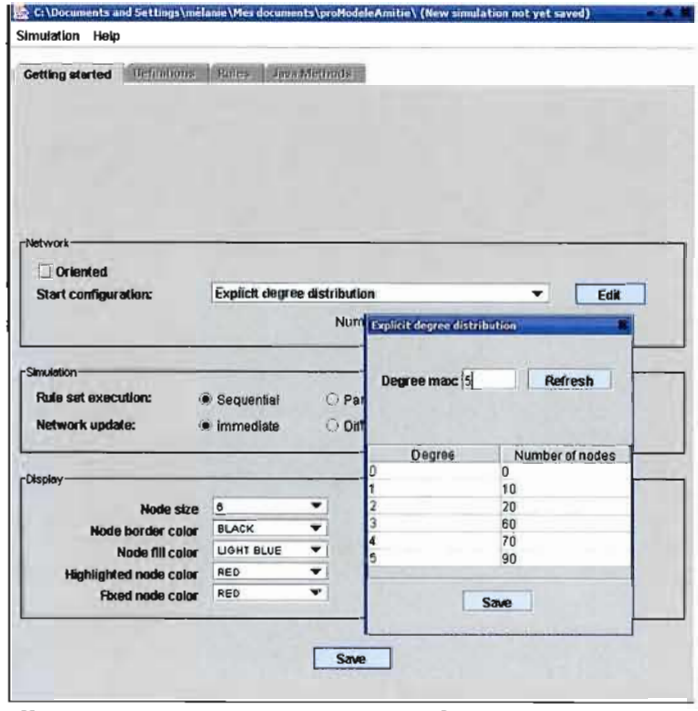

Illustration 57: Fenêtre principale de NetSim

Logiciels de simulation disponibles

Nous allons maintenant présenter quelques logiciels de simulation de systèmes. Nous allons surtout nous intéresser aux logiciels suivants : GPSS, Simula, Simscript et Simulink. Nous allons comparer la capacité de ces logiciels à permettre la simulation de plusieurs réseaux sociaux simultanés par rapport aux possibilités de notre logiciel, SNSimulator.

GPSS:

GPSS pour General Purpose Simulation System est un langage de simulation discrète. Il nécessite selon nous une importante adaptation afin de pouvoir simuler des réseaux sociaux alors que nous

avons conçu SNSimulator spécifiquement dans ce but.

```
; GPSS World Sample File - ORDERPNT.GPS, by Gerard F. Cummings
************************************************************************
*
* Order Point Inventory System
*
************************************************************************
* Initialize and define
INITIAL X$EOQ,500 ;Economic order qty.
INITIAL X$Point,600 ;Order point
INITIAL X$Stock,700 ;Set initial stock=700
Inventory TABLE X$Stock,0,50,20 ;Table of stock levels
Sales TABLE P$Demand,38,2,20 ;Table of sales levels
Var2 VARIABLE RN1@24+40
************************************************************************
GENERATE ,,,1
Again TEST L X$Stock,X$Point ;Order placed on successful test
ADVANCE 5 ;Lead time = 1 week
SAVEVALUE Stock+,X$EOQ ;Economic order
TRANSFER ,Again ;Cycle transaction again
************************************************************************
        GENERATE  1             ;Daily demand xact
        ASSIGN    Demand,V$Var2 ;Assign daily demand
        TABULATE  Inventory     ;Record inventory
        TEST GE   X$Stock,P$Demand ;Make sure order can be filled
        SAVEVALUE Stock-,P$Demand ;Remove demand from stock
        SAVEVALUE Sold,P$Demand ;X$Sold=Daily demand
        TABULATE  Sales         ;Record daily sales
        TERMINATE 1             ;Daily timer
************************************************************************
```

Illustration 58: Exemple de programme GPSS

Simula:

Langage de simulation discrète conçu dans les années 60 en Norvège.

```
Begin
   Class Glyph;
      Virtual: Procedure print Is Procedure print;;
   Begin
   End;

   Glyph Class Char (c);
      Character c;
   Begin
      Procedure print;
        OutChar(c);
   End;

   Glyph Class Line (elements);
      Ref (Glyph) Array elements;
   Begin
      Procedure print;
      Begin
         Integer i;
         For i:= 1 Step 1 Until UpperBound (elements, 1) Do
            elements (i).print;
         OutImage;
      End;
   End;

   Ref (Glyph) rg;
   Ref (Glyph) Array rgs (1 : 4);

   ' Main program;
   rgs (1):- New Char ('A');
   rgs (2):- New Char ('b');
   rgs (3):- New Char ('b');
   rgs (4):- New Char ('a');
   rg:- New Line (rgs);
   rg.print;
End;
```

Illustration 59: Exemple de programme Simula

Source site web Wikipédia

Le langage du simulateur Simula est relativement proche du langage naturel et est à notre avis d'un accès plus facile pour un néophyte mais il n'est pas conçu pour la manipulation de réseaux sociaux ce qui nécessitera une programmation poussée afin de l'adapter aux réseaux sociaux. Cette nécessité d'adaptation n'existe pas avec SNSimulator car SNSimulator traite les réseaux sociaux par défaut.

Simscript:

Logiciel commercial de simulation. Simscript permet de réaliser selon le constructeur à la fois des simulations discrètes et des simulations continues.

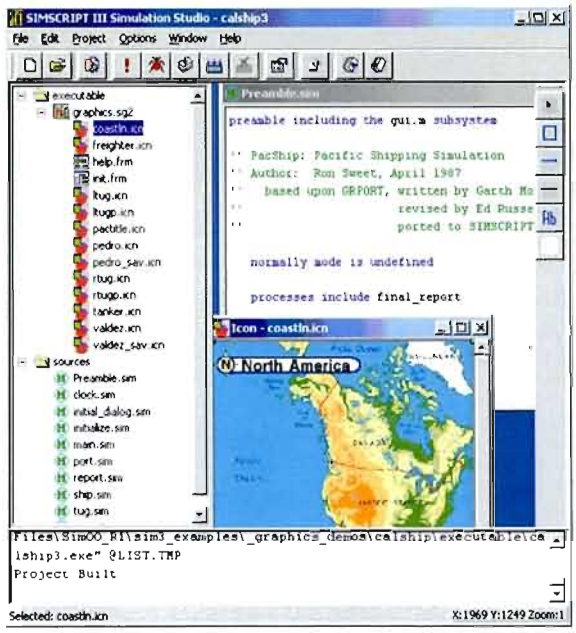

Illustration 60: Fenêtre de Simscript

Source site web de Simscript

Il nous a été impossible d'évaluer Simscript du fait de son caractère commercial. Nous pouvons quand même penser qu'étant donné qu'il s'agit d'un logiciel généraliste, il faudra un important effort pour l'adapter à la simulation de réseaux sociaux alors que SNSimulator est spécialisé dans la simulation de réseaux sociaux.

Simulink:

Logiciel commercial de simulation discrète et continue. Conçu pour fonctionner avec l'outil MATLAB, Simulink est surtout spécialisé dans la simulation de systèmes de traitement du signal et demandera donc selon nous un important effort pour pouvoir simuler des réseaux sociaux. SNSimulator ne nécessite aucune adaptation pour simuler des réseaux sociaux.

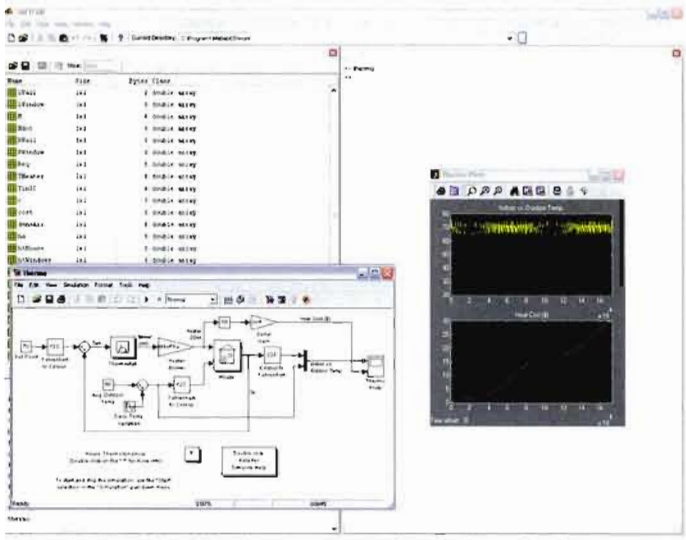

Illustration 61: Fenêtre principale de Simulink
Source site web Wikipédia

Pour conclure ce chapitre nous pouvons dire qu'à la suite de l'évaluation des différents outils disponibles pour l'analyse et/ou la simulation de réseaux sociaux, force est de constater que peu d'outils permettent de simuler des réseaux sociaux et parmi les rares qui existent presque aucun ne permet de simuler plusieurs réseaux sociaux à la fois ainsi que les interactions entres ces réseaux. De plus ces logiciels bien qu'étant très élaborés nécessitent une certaine période d'apprentissage. Cet apprentissage est d'autant plus difficile que les chercheurs en réseau sont parfois néophytes en informatique.

Il n'existe à notre connaissance pas encore d'outils alliant à la fois simplicité d'usage, capacité à modéliser des réseaux multiplexes, capacité d'analyse de réseaux et possédant aussi des fonctionnalités de visualisation et de simulation de réseaux. Nous pensons donc que SNSimulator peut représenter une alternative intéressante pour les personnes désirant étudier des réseaux sociaux et ne disposant pas de l'expertise nécessaire pour adapter les outils existants.

Apport de SNSimulator à l'étude des réseaux sociaux

Dans ce paragraphe nous essayerons de présenter à notre avis les apports de notre outil de simulation de réseaux sociaux à l'étude des réseaux sociaux.

Dans ce mémoire de maitrise en informatique, nous avons cherché à proposer un logiciel de simulation de réseaux sociaux capable de simuler plusieurs relations simultanément. Comme nous l'avons expliqué dans le chapitre 1 de ce document, notre objectif était de combler un vide qui existait à notre avis dans le domaine de l'étude des réseaux sociaux à savoir la possibilité de pouvoir simuler de manière plus réaliste l'évolution de groupes sociaux en tenant compte de la totalité des relations entre les membres d'un même groupe social. Nous avons tenté de réaliser avec les outils de modélisation et de simulation présentés dans ce chapitre les mêmes expérimentations que nous avons présentées au chapitre 5. Malheureusement nous avons été incapables de réaliser ces expérimentations soit parce que l'outil ne permettait simplement pas de le faire comme dans le cas de Netsim qui ne permet pas de gérer plusieurs relations ou alors parce que l'effort d'adaptation de l'outil nécessaire s'est avéré extrêmement élevé comme dans le cas de GPSS.

Nous pensons que SNSimulator rend possibles la simulation de groupes sociaux et de relations complexes entre les membres de ces groupes à un public non informaticien ce qui à notre avis n'existait pas jusqu'ici.

Conclusion

Dans ce mémoire de maîtrise en informatique nous avons présenté notre projet de recherche qui consistait en la réalisation d'un logiciel de modélisation et de simulation de réseaux sociaux multiplexes nous avons appelé SNSimulator. Nous avons d'abord présenté l'univers des réseaux et la problématique que nous allions essayer de résoudre. Nous avons ensuite présenté le langage de programmation de SNSimulator. Ce langage est un langage que nous avons voulu à la fois simple, proche du langage naturel et souple afin de permettre aux utilisateurs auxquels nous destinons cet outil de pouvoir modéliser aisément des réseaux sociaux. Nous avons ensuite expliqué le fonctionnement interne de SNSimulator en présentant les paramètres d'appel, le chargement des réseaux passés en paramètre et le traitement des programmes sources. Après cela nous avons présenté un exemple d'utilisation de notre outil SNSimulator en étudiant la corrélation entre deux réseaux sociaux au sein d'un cabinet d'avocats de la Nouvelle Angleterre. Ensuite nous avons étudié les différents logiciels de modélisation et de simulation de réseaux disponibles. Nous avons constaté que bien que de nombreux outils de modélisation et de simulation sont offerts, aucun d'entre eux ne permet de manière simple et accessible la modélisation de réseaux sociaux multiplexes ce qui nous confirme qu'une plate-forme telle que SNSimulator a sa place dans l'univers des outils d'analyse et de simulation.

Nous pensons que la plate-forme que nous proposons sera très utile dans le domaine de l'analyse de réseaux sociaux multiplexes parce qu'il permet de manipuler des multigraphes à savoir plusieurs réseaux simultanément au sein d'un même ensemble social et permet de simuler les interactions non seulement entre deux nœuds d'un même réseau comme le permettent la plupart des outils existants, mais également entre deux nœuds appartenant à deux réseaux différents comme l'influence entre la collaboration entre deux individus et l'amitié entre ces deux individus.

Néanmoins, il reste encore beaucoup de place à l'amélioration. SNSimulator pourrait être amélioré en ajoutant plus de fonctionnalités d'analyse de réseau. Pour le moment seul le degré des nœuds ainsi que la corrélation entre deux réseaux sont disponibles mais nous pensons qu'il serait relativement facile d'implémenter d'autres métriques d'analyse de réseaux telles que la centralité d'intermédiarité. Une autre voie pourrait consister à ajouter à SNSimulator une interface graphique ce qui permettrait de le rendre encore plus accessible à un public non informaticien. Les résultats de ce mémoire de recherche feront l'objet d'un article que nous espérons publier.

Annexes

Annexe 1 : Grammaire BNF du langage SNSimulator

GetSimulationDefinition = (ProcedureDeclaration <EOF> | FunctionDeclaration <EOF> | SequenceOfStatements <EOF> | "CALL" SubroutineCall <EOF> | <EOF>)

DeclarationSection = "DECLARE" Declarations

Declarations = (VariableDeclaration | ConstantDeclaration | FunctionDeclaration | NetworkDeclaration | NodeDeclaration | NodeSetDeclaration | LinkDeclaration | LinkSetDeclaration | PropertyDeclaration | ProcedureDeclaration | RuleDeclaration | SimulationDeclaration)*

ProcedureDeclaration = "PROCEDURE" <S_IDENTIFIER> ("(" ")" | "(" ParameterList ")") "IS" ProcedureBody

NetworkDeclaration = "NETWORK" <S_IDENTIFIER> ("USING" <FILE_ADDRESS>)?

NodeDeclaration = "NODE" <S_IDENTIFIER> "IN" <S_IDENTIFIER>

NodeAdditionStatement = "ADD" "NODE" <S_IDENTIFIER> "IN" <S_IDENTIFIER>

PropertyAdditionStatement = "ADD" "PROPERTY" <S_IDENTIFIER> "FOR" <S_IDENTIFIER> "IN" <S_IDENTIFIER>

PropertyDeleteStatement = "DELETE" "PROPERTY" <S_IDENTIFIER> "FOR" <S_IDENTIFIER> "IN" <S_IDENTIFIER>

NodeDeleteStatement = "DELETE" "NODE" <S_IDENTIFIER> "IN" <S_IDENTIFIER>

NodeSetDeclaration = "NODESET" <S_IDENTIFIER>

LinkDeclaration = "LINK" <S_IDENTIFIER> "IN" <S_IDENTIFIER> "BETWEEN" <S_IDENTIFIER> "AND" <S_IDENTIFIER>

LinkAdditionStatement = "ADD" "LINK" <S_IDENTIFIER> "IN" <S_IDENTIFIER> "BETWEEN" <S_IDENTIFIER> "AND" <S_IDENTIFIER>

LinkDeleteStatement = "DELETE" "LINK" ((<S_IDENTIFIER> "IN" <S_IDENTIFIER>) | ("IN" <S_IDENTIFIER> "BETWEEN" <S_IDENTIFIER> "AND" <S_IDENTIFIER>))

LinkSetDeclaration = "LINKSET" <S_IDENTIFIER>

PropertyDeclaration = "PROPERTY" <S_IDENTIFIER> "FOR" <S_IDENTIFIER> "IN" <S_IDENTIFIER> "IS" Expression

RuleDeclaration = "RULE" <S_IDENTIFIER> "IS" RuleBody

SimulationDeclaration = "SIMULATION" <S_IDENTIFIER>

RuleBody = (DeclarationSection BeginEndBlock | BeginEndBlock)

ProcedureBody = (DeclarationSection BeginEndBlock | BeginEndBlock)

FunctionDeclaration = "FUNCTION" <S_IDENTIFIER> ("(" ")" | "(" ParameterList ")") "RETURN" TypeDeclaration "IS" FunctionBody

FunctionBody = (DeclarationSection BeginEndBlock | BeginEndBlock)

VariableDeclaration = "VARIABLE" TypeDeclaration <S_IDENTIFIER> (":="

Expression)?

ConstantDeclaration = "CONSTANT" TypeDeclaration <S_IDENTIFIER> ":=" Expression

TypeDeclaration = ("STRING" | "NUMBER" | "BOOLEAN") ("[" <S_NUMBER> ("," <S_NUMBER>)? "]")?

BeginEndBlock = "BEGIN" SequenceOfStatements "END"

SequenceOfStatements = (Statement)+

Statement = (SubroutineCall | NodeAdditionStatement | LinkAdditionStatement | AssignmentStatement | PropertyAdditionStatement | PropertyDeleteStatement | ExitStatement | IfStatement | LoopStatement | ReturnStatement | LinkDeleteStatement | NodeDeleteStatement | GroupDeleteStatement | InsertStatement | QueryStatement | UpdateStatement | (DeclarationSection)? BeginEndBlock | StartStatement)

StartStatement = "START" <S_IDENTIFIER>

SubroutineCall = <S_IDENTIFIER> ("(" ")" | "(" Arguments ")")

AssignmentStatement = (<S_IDENTIFIER> ":=" Expression | PropertyReference ":=" Expression)

ExitStatement = "EXIT" <S_IDENTIFIER> ("WHEN" Expression)?

IfStatement = "IF" Expression "THEN" SequenceOfStatements ("ELSIF" Expression "THEN" SequenceOfStatements)* ("ELSE" SequenceOfStatements)? "END" "IF"

LoopStatement = (NormalLoop | WhileLoop | ForLoop)

NormalLoop = "LOOP" SequenceOfStatements "END" "LOOP"

WhileLoop = "WHILE" Expression NormalLoop

ForLoop = "FOR" <S_IDENTIFIER> "IN" SimpleExpression ".." SimpleExpression NormalLoop

ReturnStatement = "RETURN" Expression

UpdateStatement = "UPDATE" PropertyReference ("(" PropertyReference ("," PropertyReference)* ")")? "SET" PropertyValues ("WHERE" Expression)?

PropertyValues = PropertyReference "=" Expression ("," PropertyReference "=" Expression)*

InsertStatement = "INSERT" "INTO" PropertyReference ("(" PropertyReference ("," PropertyReference)* ")")? ("VALUES" "(" ExpressionList ")" | SelectStatement)

GroupDeleteStatement = "DELETE" ("FROM")? ObjectReference ("WHERE" (Expression))?

QueryStatement = SelectStatement

Expression = AndExpression ("OR" AndExpression)*

AndExpression = UnaryLogicalExpression ("AND" UnaryLogicalExpression)*

UnaryLogicalExpression = ("NOT")? RelationalExpression

RelationalExpression = SimpleExpression (Relop SimpleExpression | InClause | BetweenClause)?

ExpressionList = Expression ("," Expression)*

InClause = ("NOT")? "IN" "(" ExpressionList ")"

BetweenClause = ("NOT")? "BETWEEN" SimpleExpression "AND" SimpleExpression
SimpleExpression = MultiplicativeExpression (("+" | "-") MultiplicativeExpression)*

MultiplicativeExpression = ExpotentExpression (("*" | "/") ExpotentExpression)*
ExpotentExpression = UnaryExpression ("**" UnaryExpression)*
UnaryExpression = ("-" PrimaryExpression | PrimaryExpression)
PrimaryExpression = (<S_QUOTED_IDENTIFIER> | "TRUE" | "FALSE" | PropertyReference | ObjectReference | <S_NUMBER> | "(" Expression ")" | SubroutineCall | <S_IDENTIFIER>)
PropertyReference = ObjectName ("[" <S_NUMBER> "]")? "." ObjectName ("[" <S_NUMBER> "]")? "." ObjectName
ObjectName = <S_IDENTIFIER>
Relop = ("==" | "!=" | ">" | ">=" | "<" | "<=")
ObjectReference = ObjectName "." ObjectName
ParameterList = Parameter ("," Parameter)*
Parameter = <S_IDENTIFIER> TypeDeclaration
SelectStatement = (Select | Select ForUpdateClause)
Select = "SELECT" ("ALL")? SelectList (IntoClause)? FromClause (WhereClause)? (SetClause)?

SelectList = ("*" | SelectItem ("," SelectItem)*)
SelectItem = (ObjectName ".*" | ObjectName "." ObjectName ".*" | SimpleExpression (<S_IDENTIFIER>)?)
IntoClause = "INTO" IntoItem ("," IntoItem)*
IntoItem = (<S_IDENTIFIER> ("." <S_IDENTIFIER>)?)
FromClause = "FROM" FromItem ("," FromItem)*
FromItem = (PropertyReference | PropertyReference <S_IDENTIFIER>)
WhereClause = "WHERE" Expression
SetClause = (("UNION" ("ALL")?) | "INTERSECT" | "MINUS") (("(" Select ")") | Select)
ForUpdateClause = "FOR" "UPDATE" "OF" PropertyReference ("," PropertyReference)*
Arguments = ExpressionList

Annexe 2 : Exemple de programme SNSimulator

```
declare
 simulation sim1
 network test using C:\Users\franck\Documents\NetBeansProjects\SNSimulator\lazega\franck.dat
 function carre(x number) return number is
 begin
  return (x*x)
 end
 rule r1 is
 begin
 for i in 1 .. 2
  loop
   if(random()>=0.8) then
    add link lienfr3 in test between test1 and test2
   end if
  end loop
 end
begin
 add node franck in test1
 print("reseau1")
 add node test1 in test
 add node test2 in test
 add link lientt in test between test1 and test2
 exit sim1 when ((carre(2))==4)
 add link lienfr1 in test between franck and test1
 print(link_between(test.test1,test.test2))
 print(random())
 print(random())
 correlation("a","b")
 if(test.test1.degree==1) then
  add link lienfr2 in test between test2 and test1
 end if
 start sim1
end
```

Bibliographie

Graphes

[1] Bondy & Murty. *Graph Theory with Applications.* The Macmillan Press Ltd, 1976.

[2] Jacques Labelle. *Théorie des graphes.* éditions Modulo, 2003.

[3] Reinhard Diester. *Graph Theory.* Springer-Verlag Heidelberg, 2005.

Compilateurs

[4] Alfred V. Aho, Ravi Sethi & Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1986.

Simulation

[5] Jerry Banks,John S. Carson,II & Barry L. Nelson. *Discrete-Event System Simulation.* Prentice Hall, 1996.

[6] S. Hartmann. *The World as a Process: Simulations in the Natural and Social Sciences.* in Modelling and Simulation in the Social Sciences from the Philosophy of Science Point of View, Theory and Decision Library. Dordrecht: Kluwer, 1996, 77-100.

[7] James R. Emshoff & Roger L. Sisson. *Design and use of computer simulation models.* The Macmillan Company, 1970.

Programmation informatique

[8] Michael Morgan. *Java 2 for Professional Developpers.* Sams Publishing, 1999.

[9] Bruce Eckel. *Thinking in Java.* Prentice Hall, 2002.

[10] Dan Pilone. *UML 2.0 Pocket Reference.* O'Reilly, 2006.

Réseaux sociaux

[11] Ainhoa de Frederico de la Rua. « Analyse longitudinale de réseaux sociaux totaux avec

SIENA-Methode, discussion et application ». *Bulletin de Méthodologie Sociologique*, Num. 84, Octobre 2004.

[12] Revues *Connections, Social Networks*.

[13] Site web de International Network for Social Network Analysis (INSNA): *http://www.insna.org*.

[14] Barabási & Albert-László. «Scale-Free Networks». *Scientific American*, 288, May 2003, pp. 60-69.

[15] Alexandre Steyer, Jean-Benoît Zimmermann. « Influence sociale et diffusion de l'innovation ». *Mathematics and Social Sciences*, n° 168, p. 43-57, 2004.

[16] Emily M. Jin, Michelle Girvan, and M. E. J. Newman. «The structure of growing social networks». *Phys. Rev. E 64*, 046132, 2001.

[17] M. E. J. Newman and Juyong Park. « Why social networks are different from other types of networks». *Phys. Rev. E 68*, 036122, 2003.

[18] M. E. J. Newman and G. T. Barkema. *Monte Carlo Methods in Statistical Physics*. Oxford University Press,1999.

[19] G. T. Barkema and M. E. J. Newman. « New monte carlo algorithms for classical spin systems». *Monte Carlo Methods in Chemical Physics*, D. Ferguson, J. 1. Siepmann, and D. G. Truhlar (eds.), Wiley, New York, 1999.

[20] G. T. Barkema and M. E. J. Newman. « Monte carlo simulation of ice models». *Phys. Rev. E 57*, 1155–1166, 1998.

[21] M. E. J. Newman. « Mathematics of networks». *The New Palgrave Encyclopedia of Economics*, 2nd edition, L. E. Blume and S. N. Durlauf (eds.), Palgrave Macmillan, Basingstoke, 2008.

[22] M. E. J. Newman. « The structure of scientific collaboration networks». *Proc. Natl. Acad. Sci. USA 98*, 404–409, 2001.

[23] M. E. J. Newman. « Clustering and preferential attachment in growing networks». *Phys. Rev. E 64*, 025102, 2001.

[24] M. E. J. Newman. « Coauthorship networks and patterns of scientific collaboration». *Proc. Natl. Acad. Sci. USA 101*, 5200–5205, 2004.

[25] Erdős, P. Rényi, A.. « On random graphs I». *in Publ. Math. Debrecen 6*, 1959, p. 290–297

[26] M. Girvan et M. E. J. Newman. « Community structure in social and biological networks».

Lawrence A. Shepp, Rutgers, State University of New Jersey–New Brunswick, Piscataway, NJ, Avril , 2002.

[27] E.Lazega. *Réseaux Sociaux et Structures Relationnelles.* Collection Que sais-je?, éditions PUF, 1998.

[28] Alain Degenne. « Quelques modèles en analyse de réseaux sociaux». *Mathématiques, Informatique et Sciences Humaines,* No. 137, 1997, pp. 5-9.

[29] M. Goldberg, S. Kelley, M. Magdon-Ismail, K. Mertsalov, and W. A. Wallace. « Communication dynamics of blog networks». *in Proc. SIGKDD Workshop on Social Network Mining and Analysis*, ACM Digital Library , August 2008

[30] M. Goldberg, M. Hayvanovych, A. Hoonlor, S. Kelley, M. Magdon-Ismail, K. Mertsalov, B. Szymanski and W. Wallace. « Discovery,analysis and monitoring of hidden social networks and their evolution». *IEEE International Conference on Technologies for Homeland Security*, Waltham, MA, 2008.

[31] M. Goldberg, R. Rivenburgh. « Constructing cliques using restricted backtracking». *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Cliques, Coloring, and Satisfiability,* 26, 1996, pp. 75-88.

[32] N. Henry et J-D. Fekete. « MatLink: enhanced matrix visualization for analyzing social networks». *In C. Baranauskas, P. Palanque, J. Abascal and S. D. J. Barbosa, editors, Human-Computer Interaction – INTERACT 2007,* volume 4663 of LNCS, Springer. (Brian Shackel Award), 2007, pages 288–302.

[33] N. Henry, J-D. Fekete et M. J. McGuffin. « NodeTrix: a hybrid visualization of social networks». *IEEE Transactions on Visualization and Computer Graphics,* 13(6), 2007, pp. 1302-1309.

[34] N. Henry et J-D. Fekete. « MatrixExplorer: un système pour l'analyse exploratoire de réseaux sociaux». *In Proceedings of IHM 2006, International Conference Proceedings Series*, ACM Press, Montréal, Canada, April 2006, pages 67–74.

[35] Stanley Milgram, J. Travers. « An experimental study of the small world problem». *Sociometry*, 1969, Vol. 32, No. 4. (1), pp. 425-443

[36] A. Degenne & A. Forsé. *Les Réseaux Sociaux.* Paris : Armand Colin, 1994.

[37] E. Lazega. *The Collegial Phenomenon.* Oxford University Press, 2001.

[38] D.J. Watts. *Six Degrees.* New York : Norton , 2003.

[39] M. E. J. Newman. « The structure and function of complex networks». *SIAM Rev.*,vol. 45, 2003, pp. 167–256.

[40] Mélanie Lord. « NETSIM: Un logiciel de modélisation et de simulation de réseaux d'information.». Mémoire de maîtrise, UQAM, Janvier 2007.