NetSim: a Simulation and Visualization Software for Information Network Modeling

Mélanie Lord
Computer Science Dept. UQAM
melanie.lord@internet.uqam.ca

Daniel Memmi
Computer Science Dept. UQAM
memmi.daniel@uqam.ca

Abstract

For social and technical reasons, various information networks have become an important phenomenon, which it is useful to study and formalize. For this purpose we have developed NetSim, a generic software tool for modeling and simulating diverse network structures. We will see how to use this system to simulate and visualize two quite different network types: a preferential attachment model, and a typical social network. We will also present a modeling language we designed to express different network models to be interpreted by the NetSim system. We will show graphical simulation results that demonstrate the operation of the software.

1. Introduction

In a modern economy, most occupations have become highly informational and highly collaborative. Sociologists have thus described the emergence of a "network society" of sparse, highly varied and rapidly changing relationships [1][2]. Individual relationships as well as organizational structures within and between firms tend toward temporary social links.

At the same time, one has seen the explosive development of electronic communication networks, such as e-mail, instant messaging or cell phones. The Internet, which makes e-mailing possible between people, has also given rise to the Web, an enormous network of interrelated information sites.

The combination of social changes and technical advances has led to the recent advent of virtual communities, social groupings bound together mostly through electronic communication [3]. These communities are sometimes similar to traditional social groups, but they are often more impersonal, looser and goal-oriented [4].

These various, but related phenomena can be modeled by networks. Whether in mathematics, computer science or structural sociology, networks have been formalized as graphs of links and nodes [5][6][7]. A graph can in turn be easily represented by a matrix, on which the whole range of the methods of linear algebra may be brought to bear.

In a social network for instance, nodes stand for social actors (individuals or specific groups) and links represent social relationships (such as friendship ties or collaborations). The Web can also be described as a huge graph of hypertext links between Web pages. By exploring the corresponding matrix, one may learn a lot about the structure of various networks: the distribution of links, the centrality of specific actors, the existence of coherent subgroups…

To cite a well-known application, Google's famous ranking algorithm for Web pages was designed after careful study of the Web's hyperlink structure [8]. Studying the social structure of virtual communities and collaboration networks would similarly help design better collaborative software and more efficient communication networks.

Using social links to gather information is another way to exploit social networks [9]. This is particularly important in the case of tacit knowledge (expertise, know-how…), which is mostly found and acquired through personal contacts. "Social information retrieval", combining social networks with document search techniques, is becoming a useful and innovative addition to classical information retrieval [10].

For network modeling purposes, however, precise relational data is often hard to obtain and mathematical models [11] are usually very specific. For the study of dynamic phenomena, we believe that computer modeling and simulation offers a convenient and promising avenue of research.

Because of the variety of possible network structures in different domains, it would be useful to avail oneself of a general modeling platform, which would make it easy to define and simulate networks of all kinds. We have thus developed NetSim, a software system able to model and display the evolution of various types of networks [12]. For instance, triangle closure (in social networks) or preferential attachment (typical of Web sites) give rise to dissimilar structures, associated with very different patterns of behavior.

Starting from several dynamic network models found in the recent literature, we first designed a generic network modeling language. We then built a software tool with good visualization capabilities so as to generate diverse network structures, given basic generative rules expressed with our modeling language. The architecture of NetSim makes it possible to simulate and display many network types with reasonable ease and efficiency.

We will now explain in more detail two different (but quite typical) network models, a Web growth model and a dynamic friendship network. We will then describe in turn our modeling language and the architecture of NetSim. Lastly, we will simulate these two models with NetSim and present graphical simulation results. We thus hope to demonstrate the capabilities of this generic software tool.

2. Modeling information networks

2.1. A Web growth model

We will deal here with information networks in general, that is to say networks through which knowledge and information circulate (including notably social networks).

An interesting fact about some information networks has been shown in [13][14]. They noticed that their connectivity, notably in the Web, shows a degree distribution following a power law (the degree of a node being its number of links). This type of distribution decreases rapidly but without ever reaching an upper limit to the degree of nodes. Barabási's team also remarked that some networks continually grow with the constant addition of new nodes and links. We know this to be true of the Web but it's also true for sexual acquaintance networks for example. Moreover, new nodes tend to bind themselves with other nodes of high connectivity i.e. high degree. This property is called preferential attachment. In these networks, a high degree can be seen as a sign of popularity or prestige: new Web pages are more likely to link with already popular pages. So Barabási's team presented a scale-free model demonstrating that constant growth and preferential attachment are directly responsible for the power law degree distribution observed in this kind of networks.

Other Web-like growth models have been proposed to explain the power law degree distribution of such networks. For example, in [15], the authors have suggested a similar model integrating the idea that the propensity of a site to attract new links decreases with age. In [16], for their part, the authors have proposed a cost or a capacity constraint that slows down the power law degree distribution process. As a more detailed example, we will now explain a model presented in [17] which is an extension of the scale-free model previously introduced.

This model begin with m_0 nodes and, at each time step, the three following rules are executed according to a probability p, q or $1 - p - q$ where $0 \le p < 1$ and $0 \le q < 1 - p$.

1) Addition of new links: With a probability p, add $m < m_0$ links to the network. To do that, we first choose at random the start node for each new link. The end node of the link is then selected with a probability $\prod$, where k represents the node degree (its connectivity), thus taking into account the preferential attachment property (1). In the case of the Web, this rule corresponds, for instance, to a Webmaster that adds new hyperlinks to his Web site.

$$\prod (k_i) = (k_i + 1) / \sum_j (k_j + 1) \qquad (1)$$

Figure 1 explains this probability of creating a new link.

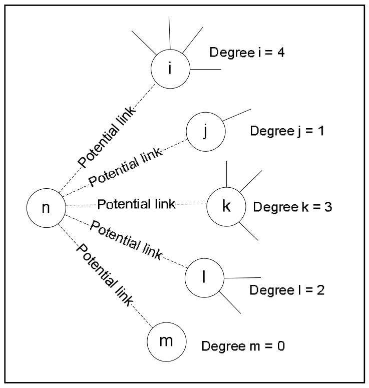

Figure 1. Probability of new link
Suppose the node *n*, chosen at random, as the start node of the link we want to add. We have to select the end node of this new link according to probability ∏. This probability increases with the degree of the end node.

2) Rewiring of existing links: With a probability q, rewire m links in the network. For this, we select at random a node i and a link l_{ij} that goes from i to another node j. Then, we remove this link and we replace it with a new link $l_{ij'}$ where the node j' is chosen with the same probability $\prod$ as in the first rule. In the Web, for example, this rule could mean the modification of an URL from a Web page to another one.

3) Addition of new nodes: With a probability $1 - p - q$, add a new node to the network. Then, we link this new node to m existing nodes in the network that are chosen with the same probability $\prod$. In the Web, this rule could represent the addition of a new Web page and its hyperlinks.

2.2. A friendship growth model

Other networks that do not conform to scale-free models were also investigated. For instance, structural sociologists [6][7] found that friendship or personal contact networks behave very differently from Web structure models. Contrary to the Web, those networks show a high clustering coefficient, i.e. the density of triangles (three nodes linked together) in the network. In a friendship network, this can be explained by the tendency for two of one's friends to be friends of each other. It seems realistic to say that two people having mutual friends are more likely to meet than two people without common contacts. In [18], Jin, Girvan and Newman proposed an interesting friendship network growth model that differs substantially from the scale-free model in many ways. The four rules in this model are the following:

1) Fixed number of nodes: The authors consider a closed population of fixed size. Given the fact that the timescale on which people make and break social relations is much shorter than the timescale on which people join or leave the network, the authors of the model believe that the addition and removal of nodes is negligible relative to the variation of the number and arrangements of connections. It means that, in the graph representation, there will be neither addition nor suppression of any node during the simulation. This is in sharp contrast to models of Web growth.

2) Limited degree: The probability of a person developing a new relation should decrease sharply once their number of friends reaches a certain level because of the time and effort needed to maintain a friendship. In the graph representation, this means that each node will have a limited degree (i.e. the number of links attached to this node). Contrary to Web growth models, the degree distribution in such social networks, does not follow a power law but fluctuates around a mean degree. The preferential attachment mechanism is not very important in friendship networks.

To implement this rule, the model suggests the parameter z^* as a fixed limit on the maximum number of friends that a person can have. When this limit is reached for an individual, the probability for this individual to make other friends decreases sharply.

3) Decay of friendships: A friendship must be maintained. If two friends don't meet often enough, their friendship will lapse. This rule models the fact that if two friends don't meet for a long period of time, as time passes, their friendship will weaken and eventually end. In terms of graph representation, it means that some links will disappear over time.

To implement this rule, each link has an attribute s, initialized to 1, that represents the strength of this friendship. Each time two friends, i and j, meet, the strength s_{ij} is reset to 1 but as time passes, if they don't meet again, their friendship decays exponentially according to (2), where Δt represents the interval of time since the last meeting of i and j and where k is an adjustable parameter of the model. If i and j meet again, the strength s_{ij} is reset to 1. It is then possible to set a minimal limit on s_{ij} to consider that below this limit, the friendship between i and j ends so that the link between them is removed.

$$s_{ij} = e^{-k\Delta t} \qquad (2)$$

4) Clustering: The probability for two individuals to meet is higher if they have mutual friends. This last rule is the most important one to explain the evolution of this type of network. It models the clustering property observed in many social networks but which is not so apparent in the Web.

The probability p_{ij} by unit of time for two people i and j to meet depends on two factors (3): the number of friends z_i and z_j that i and j already have and the number m_{ij} of their mutual friends. These two factors are respectively represented by functions f and g.

$$p_{ij} = f(z_i)\, f(z_j)\, g(m_{ij}) \qquad (3)$$

The function $f(z)$ (4) must be large enough and roughly constant for small z but must decrease drastically when z approaches the transition value z^*. For f, the authors chose the Fermi function that shows this characteristic and where b controls the decreasing precision at z^*.

$$f(z) = 1 / (e^{b(z - z^*)} + 1) \qquad (4)$$

The function $g(m)$ (5) represents the expected increase of the probability that to individuals meet if they have one or more mutual friends. The parameter p_0 is the weak probability of an encounter between two people without mutual friends and a controls the rate of increase of $g(m)$.

$$g(m) = 1 - (1 - p_0)\, e^{-a\,m} \qquad (5)$$

This friendship growth model like the previous Web growth model is only an example of models that can be easily simulated with NetSim. In fact, our modeling language allows translating those models, with a minimum of efforts, in a language that can be then interpreted and simulated by the software. The next section introduces this modeling language.

3. NetSim modeling language

3.1. NetSim entities

The true contribution of NetSim lies in its modeling language. There already exist several software libraries, such as Jung [19] or Prefuse [20], providing functions to process and visualize network data. But they require the knowledge of a programming language such as Java to carry out a simulation, whereas NetSim hides programming details thanks to its high-level modeling language.

Stand-alone software tools are also available, but they are usually more specialized. Ucinet for example [21] is an excellent tool for social network analysis, but is not intended for modeling. The Pajek system [22] does offer a wide range of network processing functionalities, but NetSim put the emphasis on the clear expression of network models.

As we will see in this section, the NetSim modeling language is flexible enough to model various types of information networks with minimal time and effort. This language has been elaborated by observing several dynamical network models as in [17] [18]. We could notice that most of those models process similar entities: the actors or agents, the relations between those actors and finally, the relational structure composed of the actors and their relations. Therefore, in the NetSim modeling language, the network is represented as a graph in which the actors are represented by the nodes or vertices and the relations are represented by the links of the graph.

Table 1 lists all NetSim entities. For now, our simulation software can simulate the evolution of one network at a time but we are working on expanding the functionalities so that it can handle multigraphs i.e. several networks that may influence the evolution of one another. This is why we have the entity *world* that represents the environment in which networks evolve. This environment could then become a place for information exchanges between several networks.

We can define attributes and functions over these four NetSim entities. It's also possible to define constants, which are always defined over the entity *world* (by default). We have already defined attributes and functions on those entities but the language also allows the user to define some of their own. Constants represent some fixed parameters of the model that are initialized at the beginning of the simulation. These values stay unchanged till the end of the simulation.

Table 1. Netsim language entities

Entity	Description
world	The environment in which the networks evolve.
network	The relational structure composed of the actors and their relations.
link	A relation between two actors.
node	An actor.

Attributes are values characterizing the internal state of an entity which must be initialized at the beginning of the simulation but can be modified during the simulation. Functions allow computing some values as the simulation goes on. But in order to understand how to use the modeling language, we have to understand first the entity hierarchy that determines the visibility of one entity by another:

- Attributes and functions defined over the entity *world* are visible by all entities.
- Attributes and functions defined over the entity *network* are visible by the entities *network*, *link* and *node*.
- Attributes and functions defined over the entity *link* are only visible within this entity.
- Attributes and functions defined over the entity *node* are only visible within this entity.
- Constants, which are defined over the entity *world* (by default), are visible by all entities.

Figure 2 is a schematic representation of this hierarchy.

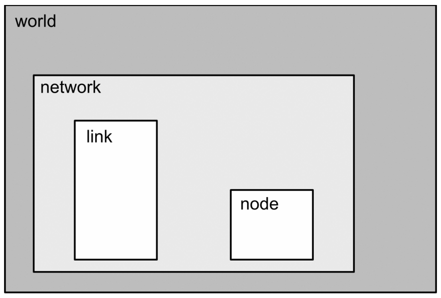

Figure 2. NetSim entity hierarchy
Each entity is visible by itself and by entities represented here with a lighter shade of grey. In other words, each entity can see itself and can see the entities that are represented with a darker shade of grey. For instance, the entity world can only see itself, the entity network can see itself and the entity world, the entity link can see itself, the entity network and the entity world and finally, the entity node can see itself, the entity network and the entity world.

In the following section, we will explain in detail the syntax of NetSim definitions i.e. constants, attributes and functions.

3.2. Syntax of model definitions

We have seen that the NetSim language allows us to define some characteristics of the model entities: constants, attributes and functions. Constants and attributes are always numerical and the computed value of a function is either numerical or boolean. In addition, except for constants that are visible from anywhere, each definition must be defined over a specific entity that determines its visibility.

The command to declare a constant is *DEFINE_CONST*. This command also needs two parameters that specify the name of the constant and the value of this constant. For example, the command line *DEFINE_CONST (pi, 3.1416)* declares a constant named *pi* initialized with the value 3.1416.

The attributes are variables defined over a specific entity. They are initialized at the beginning of the simulation but their value can be modified during the simulation. The NetSim modeling language already provides some attributes that characterize the appearance (color and size) of the node and links being displayed. In fact, those attributes can be useful to help visualize some nodes and links that exhibit specific properties during the simulation. For example, one can decide to increase the size of all nodes that reach a degree greater than six and to color in green all links with strength between six and ten.

Also, additional attributes can be defined by the user as well. The command to declare an attribute is *DEFINE_ATTRIB* which requires three parameters: the entity on which we want to define this attribute, the name of the attribute and the initialization value. For example, the command line *DEFINE_ATTRIB (link, strength, 1)* declares an attribute on the entity *link*, named *strength* and initialized to 1. This definition means that each time a new link is created during the simulation it will have an attribute *strength* initialized to 1. Of course, this value could change over time. According to the visibility rules, this attribute will be visible only by the entity *link*. The same principle applies when defining attributes over the entities *world*, *network* and *node*.

After writing the constants and attributes of the model we want to simulate, we can then define the model functions. A NetSim function is either an arithmetic or boolean expression that computes and returns a numerical or boolean value. As for the attributes, the NetSim modeling language already provides some functions over all entities. Those functions are described in table 2. We will see a more concrete example of the use of some of these predefined functions in section 3.4.

Table 2. Predefined NetSim functions

Entity *world*	
Function	Value returned
time_step()	Current time step
random_number()	Random number between 0 and 1
Entity *network*	
Function	Value returned
number_of_nodes()	Number of nodes in the network.
number_of_links()	Number of links in the network.
mean_degree()	Mean degree of the network.
max_degree()	Highest node degree.
min_degree()	Lowest node degree.
diameter()	Diameter of the network.
Entity *link*	
Function	Value returned
start_node()	First node of the link. If the graph is directed, this is the start node of this link.
end_node()	Second node of the link. If the graph is directed, this is the end node of this link.
common_contacts()	Number of neighbours that the first and second nodes of the link have in common.
have_common_contacts()	True if the first and second nodes of the link have at least a common neighbour.
a_common_contact()	A node that is a common neighbour of the first and second nodes of this link.
Entity *node*	
Function	Value returned
degree()	Node degree.
out_degree()	Number of outgoing links from the node.
in_degree()	Number of incoming links to the node.

Apart from NetSim predefined functions, the modeling language allows users to define their own functions. Like an attribute, a function is always defined over a specific entity that determines its visibility. A functional expression can contain arithmetic or boolean operators, constants, attributes and other functions provided that those attributes and other functions are visible to the entity over which we define this function. The command to define a function is *DEFINE_FUNCTION* and it requires two parameters: the entity on which we want to define the function and the name of the function. These parameters are then followed by the boolean or arithmetic expression of the function. Whenever we call a function, we must write its name followed by an opening parenthesis and a closing parenthesis in order to differentiate it from the use of an attribute.

For instance, suppose we have already defined attribute *strength* over the entity link and a constant *pi* as in the previous examples. We could then define a boolean function *f1* that returns true if the *strength* of the link is less than or equal to the constant *pi* with this function definition:

DEFINE_FUNCTION (link, f1) strength <= pi

One can also write an arithmetic function *f2* that computes, for example, the ratio of the degree of a node to the highest node degree in the network:

DEFINE_FUNCTION (node, f2)
degree() / max_degree()

After translating all the model definitions in the NetSim language, we can now write the rules.

3.3. Syntax of model rules

As we have seen in section 2, a dynamical model i.e. which can be simulated over time is a set of rules executed (or not) at each time step according to some conditions or probabilities which often depend on the state of the network (the set of all attribute values). These rules are essentially meant to influence the arrangement of links and nodes in the network i.e. its structural behavior in the hope that in time will emerge the expected structure.

NetSim proposes a set of actions to modify the network structure. These actions allow adding or deleting nodes and links in the network or enable the modification of any attribute of each entity. The model rules are thus defined with the actions listed in table 3.

First of all, each action can have a sequence of any number of parameters, separated by a comma. Each parameter must be an arithmetic expression, a boolean expression or an assignment. An arithmetic expression is an arithmetic (numerical) function, an attribute, a constant or an arithmetic expression proper which is built with arithmetic operators and/or numerical functions and/or constants and/or attributes. A boolean expression is either a boolean function or a boolean expression built with boolean operators and/or boolean functions and/or constants and/or attributes. Finally, an assignment is an expression replacing an attribute value with an arithmetic expression using the := operator. Only attribute values can be modified.

Action parameters are thus either numerical values, conditions or assignments. Numerical values are considered as probabilities if they are between 0 and 1 and as integers (decimals will be truncated) if they are greater than 1.

It is crucial to always respect the entity visibility (see figure 2). For instance, for an action defined over the entity *world*, it is strictly forbidden to use, in its parameter expressions, functions or attributes defined over entities *network*, *link* and *node* because those entities are not visible from the entity *world*. On the other hand, for an action defined over the entity *link*, one can use functions and attributes defined over entities *link*, *network* and *world* in the parameter expressions of this action.

Table 3. NetSim actions

Action	What does this action do
PICK_WORLD	Selects the entity *world* in order to modify its attribute values.
PICK_NETWORK	Selects the entity *network* in order to modify its attribute values.
PICK_LINKS	Selects links in order to modify their attribute values.
PICK_NODES	Selects nodes in order to modify their attribute values.
ADD_LINKS	Adds new links to the network.
CUT_LINKS	Deletes links from the network.
REWIRE_LINKS	Changes the second node of a link with another one chosen at random in the network.
ADD_NODES	Adds new nodes to the network
CUT_NODES	Deletes nodes from the network.

Action parameters are a way to select specific entities to process by this action. They can be seen as a succession of challenges or tests to undergo by concerned entities. Those who pass all challenges are chosen and processed accordingly to the action while others are not. These tests (parameters) apply sequentially, in the writing order.

In order to better understand the syntax of this modeling language, we will see, as an example, how to translate into this language the friendship network model described in a previous section.

3.4. A concrete example

As seen above, the friendship network model consists of four rules. Before we can write these rules, we have to write the definitions which are the constants, attributes and functions.

The first rule mentioned in this model was the fixed number of nodes. This rule requires no definitions at all because it simply means there will be neither addition nor suppression on any nodes in the network during the simulation.

The second rule specifies a maximum limit on the degree of each node in the network (because of time and effort to maintain a friendship, people can only have a limited number of friends). To do so, we have to define the model parameter z^*, as a constant, which is the fixed limit on the maximum number of friends that a person can have. This NetSim definition constant could be:

DEFINE_CONST (z_limit, 5);

The third rule (i.e. the decay of friendship) stated that for a friendship to continue, friends have to meet regularly. If not, their friendship become weaker and eventually ends. To implement this rule, the model proposes an attribute s_{ij} for each link in the network, representing the strength of a friendship link between i and j. We then have to define a NetSim attribute, over the entity link which is initialized to 1 (the maximum *strength* value):

DEFINE_ATTRIB (link, strength, 1);

Then, this rule states that if two friends don't meet, as time passes, the strength s_{ij} of their friendship decays exponentially according to $s_{ij} = e^{-k\Delta t}$ and when s_{ij} reaches a minimal limit, the friendship link is then cut. To write this function, in the NetSim language, we have to define first the constant representing the parameter e, the constant for k and the constant for the minimal limit on s_{ij}. We will also need to define another attribute over the entity link that will show the time elapsed (number of time steps) since the last meeting (i.e. Δt). Finally, we can write the NetSim function representing the model function s. These definitions are the following:

```
DEFINE_CONST (e, 2.718);
DEFINE_CONST (k, 0.01);
DEFINE_CONST (limit_min_for_friendship, 0.5);
DEFINE_ATTRIB (link, time_since_last_meeting, 0);
DEFINE_FUNCTION (link, s)
        e ** (-k * time_since_last_meeting));
```

The last rule which is the most important one to explain network behavior is the clustering property saying that two people, i and j, have a higher probability p_{ij} to meet if they have mutual friends:

$$p_{ij} = f(z_i)\, f(z_j)\, g(m_{ij})$$
where $f(z) = 1 / (e^{\,b\,(z - z^*)} + 1)$
and $g(m) = 1 - (1 - p_0)\, e^{-a\,m}$

Before we can define the NetSim function for p_{ij}, we must define NetSim functions for $f(z_i)$, $f(z_j)$ and $g(m)$. We first write the constants:

```
DEFINE_CONST (b, 5);
DEFINE_CONST (a, 0.5);
DEFINE_CONST (p0, 0.006);
```

Then, we can write the three NetSim functions for $f(z_i)$, $f(z_j)$ and $g(m)$:

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

DEFINE_FUNCTION (link, fj)
1 /(e ** (b * (end_node().degree() – z_limit)) + 1);

DEFINE_FUNCTION (link, g)
1 – (1 – p0) * e ** (-a *number_of_common_contacts());
```

And finally, we can write the NetSim function for probability p_{ij}:

*DEFINE_FUNCTION (link, pij) fi() * fj() * g();*

The initialization values we used to define constants or attributes are for the most part the same as those used by the authors of the model. We have just written all definitions we need to finally write the model rules with NetSim actions. These actions are the following:

```
ADD_LINKS (pij());

PICK_LINKS (time_since_last_meeting :=
time_since_last_meeting + 1, strength := s());

PICK_LINKS (pij(), time_since_last_meeting := 0,
strength := 1);

CUT_LINKS (strength <= limit_min_for_friendship);
```

This is the NetSim implementation model of the friendship network growth model of Jin, Girvan and Newman. We could now simulate and visualize it with the NetSim simulation software designed to interpret the NetSim modeling language. The next section presents a general view of the NetSim architecture and some interesting functionalities.

4. NetSim interpreter

4.1. General architecture

NetSim is first a modeling language but it is also a software capable of interpreting the modeling language, yet it is not a real interpreter. In fact, it is a Java-based application which uses the Java virtual machine to interpret the NetSim modeling language first translated into Java code.

The first step in building a simulation is to write, in the NetSim language, the model definitions and rules as we did above. The application then verifies the syntax of these definitions and rules. If there are syntax errors, it warns the user to correct them before going on. When there are no errors, the program loads the NetSim model into memory and translates it into a Java model. This Java model will then be turned into Java code, which will be dynamically compiled and finally interpreted by the Java virtual machine to execute the simulation. Figure 3 illustrates the main steps to build and run a simulation.

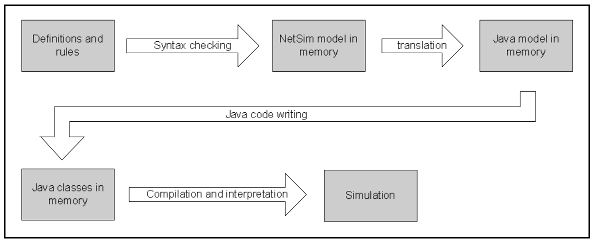

Figure 3. General NetSim architecture

Because of space limitations, we will not discuss here specific implementation details of NetSim but we must mention that we designed NetSim so it can be easily enhanced. In fact, our software provides diverse configuration files that could be modified easily to add new functionalities. For example, to implement another predefined NetSim function we simply have to code a Java method that implements this function in a specific Java class called the *SimulationController* and then add this new function in the corresponding configuration file. This new function will then be available as a NetSim predefined function. We could also easily add new initial network configurations (see section 4.2) offered by NetSim in a similar way.

NetSim also proposes interesting features in order to facilitate the implementation of a simulation and to better control the simulation progress. We will now explain briefly some of them but as for implementation details, more explanations can be found in [12].

4.2. Initializing network configuration

NetSim allows the user to choose an initial network configuration. This could be accomplished in two ways. First, we can choose a configuration among those offered by NetSim via its graphical user interface. These configurations are typical degree distributions following, for example, a random, a normal or a power law. We could also simply choose the number of nodes and the number of links we want in the initial network. The second way to choose an initial network configuration is to describe it in a special NetSim XML file format. This format allows describing all nodes and links of the network. NetSim then reads this file and builds the corresponding initial network. The use of a NetSim XML file is a convenient way to use external network data for building the initial configuration network.

In addition, it is always possible to save the resulting network data of a simulation in a NetSim XML file. This file can then be re-used as an initial network configuration for another simulation.

4.3. Controlling the simulation progress

A rule set is a group of rules (NetSim actions) which are executed together or not at all. It is possible, in NetSim, to define several rule sets within a simulation and to specify a condition for each rule set to be executed. For example, one can decide to build an initial network from scratch with a certain rule set and then, when the given condition is reached, the simulation continues with a different set of rules and so on. We could also define several rule sets that would be executed or not according to some probability, at each time step. This is, in fact, what we did to simulate the Web growth model previously presented since each rule of this model had to be executed according to some probability. In this case, each rule was considered as a rule set.

4.4. Visualizing network evolution

NetSim also provides graphical display capabilities to visualize the network evolution as the simulation goes on. In order to implements this functionality, we used a friendly Java library providing useful Java classes to manipulate and display graphs. (See http://www.prefuse.org/).

Graphic visualization can be useful for quickly spotting special structural patterns that could emerge from the network during a simulation. For example, suppose we have a social network structure showing a central actor surrounded by other peripheral actors as in figure 4 (a). One could then say without further analysis that this central actor can have a big influence on the information circulation between other actors because to communicate with one another, the peripheral actors always have to go through this central actor. Figure 4 (b) shows a tree structure which induces the idea of a hierarchy and figure 4 (c) illustrates the idea of the formation of distinct subgroups or communities within the network. All these patterns could be spotted at a glance.

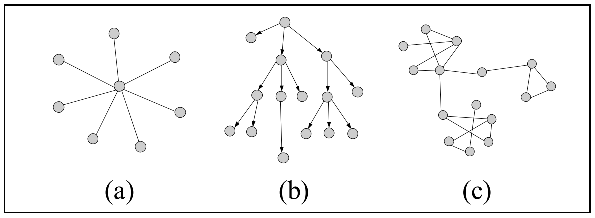

Figure 4. Typical structural patterns

Such visual data is also useful to quickly detect if the model we are simulating appears to have the expected behaviour. If not, we can immediately stop the simulation, modify some model definitions or rules and re-start the simulation. When we study some phenomena by modeling, the simulation of our models is a crucial step. It allows testing our hypothesis under different constraints, adjusting parameters to make our models explicit.

In order to test our modeling language and our simulation software, we have simulated with NetSim the friendship growth model and the Web growth model previously discussed. The next section presents those simulation results.

5. Simulation results

5.1. Simulating the friendship growth model

As the authors of this model did, we initialized the friendship network with 250 nodes and the simulation adds links to the network until the mean degree of the network reaches the *z_limit* constant value. To do so, we used the same rules as the original model but with a constant *k* initialized to 0 in order to prevent links from being removed from the network. Figure 5 shows the initial network configuration we obtained.

Then, we start the real simulation of the NetSim implementation model described in section 2.2. Figure 6 shows the resulting network after 1000 time steps.

As in the original model, we can notice the formation of well defined communities within the network. The observed clusters mostly stayed linked together, but some of them have parted form the principal graph component during the simulation.

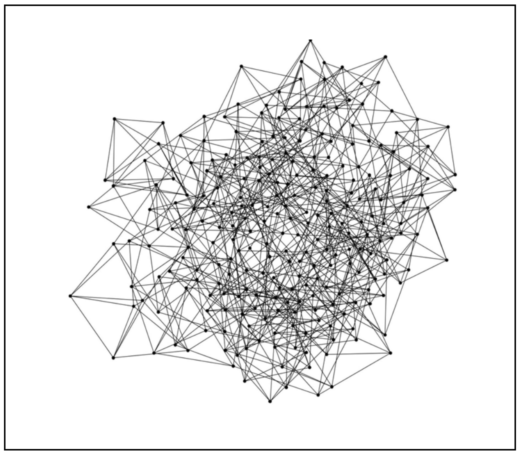

Figure 5. Initial network configuration (friendship model)

This phenomenon which was not originally stated in the model is certainly observable in the real world. This can be explained by the fact that when a region of the network begins to show a higher density of connections than the mean density of the whole network, there is a higher number of pairs of nodes that have mutual friends. Thus, according to the clustering property, new friendships will preferably form between those pairs of nodes, favoring again the connection density in this region. It appears that a little initial fluctuation in network density can cause the evolution of cohesive communities within the whole graph.

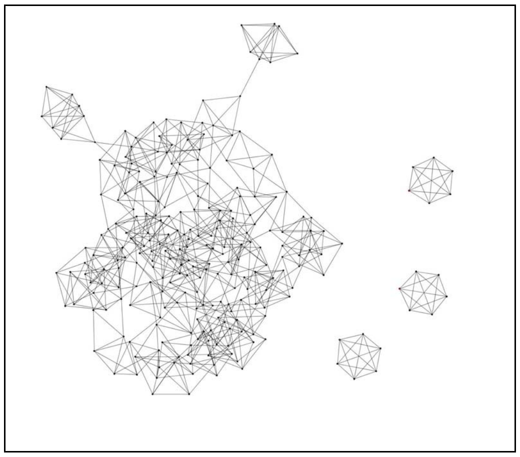

Figure 6. The network after 1000 time steps (friendship model)

In addition, we notice that these communities seem to be self-sustaining. We observe a lot of triangular structures in the network and those structures are self-sustaining in this model. In a triangle, each pair of nodes has necessarily a common neighbour so they have a good chance to meet and the strength of their friendship is continually renewed. This means that within a community, friendship links are more likely to last than those that link communities together.

Finally, we computed the clustering coefficient of the resulting network and we obtained a high value of 0.544. This clustering coefficient is very high if we compare it with random graphs of the same size and with the same number of links that would have a clustering coefficient turning around $z^* / N = 5 / 250 = 0.02$ where N is the number of nodes in the network. Our model clearly exhibits the clustering property.

So NetSim modeling and simulation of the friendship growth model produces results very similar to those obtained by the authors of the original model. Now, we will see if the NetSim simulation of the Web growth model is also conclusive.

5.2. Simulating the Web growth model

This model, as we have seen, takes into account the preferential attachment property which state that nodes with a higher degree tend to attract more links than nodes with a lower degree. Contrary to the friendship growth model, this takes also into account the constant addition of nodes and links to the Web. Due to this constant growth and to preferential attachment, it has been shown that the distribution of the Web follows a power law. We have simulated this model with NetSim in order to see if our resulting network follows a power law degree distribution as well.

We started the simulation with an initial newtork of 6 free nodes and we then executed the simulation for 600 time steps. Figure 7 shows the resulting network.

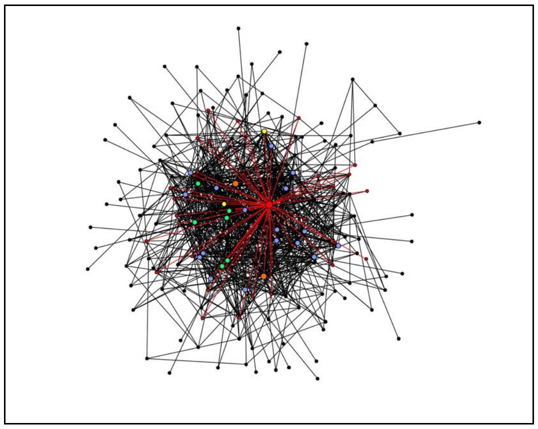

Figure 7. The network after 600 time steps (Web model)

We then plotted the degree distribution of our resulting network, omitting all null degrees or probabilities. Figure 8 presents this plot where the *x* axis corresponds to the degree of all nodes and where the *y* axis represents the probability of each degree in the network. We see that the dots in the graph seem to trace an exponential curve. We know that if this curve really follows a power law, we should obtain a straight line by using logarithmic values for each axis. Figure 9 shows this plot. Using a linear regression technique, we obtain a straight line with a significant correlation coefficient equal to -0.879. So the model simulated with NetSim actually seems to produce a network structure following a power law as mentioned in the literature.

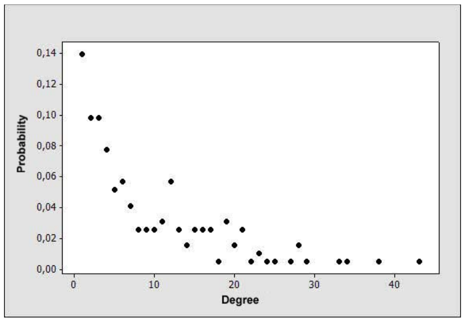

Figure 8. Degree distribution of the resulting network (Web model)

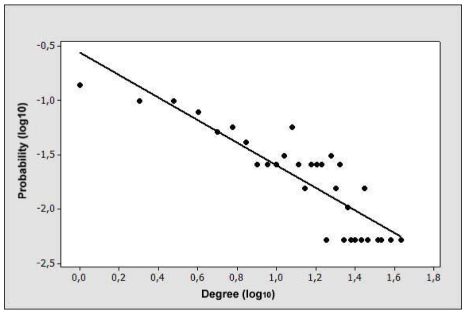

Figure 9. Logarithmic degree distribution of the resulting network (Web model)

In short, we have obtained significant results regarding the modeling and simulation, with NetSim, of two well-known but quite dissimilar models to be found in the literature. We think these results have started to validate the design of our software system and modeling language.

6. Conclusion

We have presented here and demonstrated NetSim, a generic software tool for modeling and simulating various network structures. We have seen how this system could be used to simulate and visualize two quite different network types: a preferential attachment model following a power law, and a typical social network characterized by a tendency to triangle closure.

The modeling language we first had to develop for modeling purposes is probably of interest in itself, as a way of expressing and comparing different network types. But this high-level language also allows users to define and test network models without having to write computer programs, because it is then interpreted by the NetSim system.

7. References

[1] Castells, M. (1996) *The Rise of the Network Society*, Blackwell, Oxford.
[2] Wellman, B. (ed) (1999) *Networks in the Global Villlage*, Westview Press, Boulder, Co.
[3] Rheingold, H. (2000) *The Virtual Community*, MIT Press, Cambridge, Mass.
[4] Memmi, D. (2006) The nature of virtual communities, *AI and Society* 20 (3).
[5] Lazega, E. (1998) *Réseaux Sociaux et Structures Relationnelles*, PUF, Paris.
[6] Wassermann, S. & Faust, K. (1994) *Social Network Analysis: Methods and Applications*, Cambridge University Press, Cambridge.
[7] Degenne, A. & Forsé, M. (1994) *Les Réseaux Sociaux*, Armand Colin, Paris.
[8] Brin, S. & Page, L. (1998) The anatomy of a large-scale hypertextual search engine, Computer Science Dept., Stanford University.
[9] Memmi, D. & Nérot, O. (2003) Building virtual communities for information retrieval, in *Groupware: Design, Implementation and Use*, Favela et Decouchant (eds), Springer, Berlin.
[10] Goh, D. & Foo, S. (eds) (2007) *Social Information Retrieval Systems*, IGI, Hershey, PA.
[11] Watts, D.J. (1999) *Small Worlds*, Princeton University Press, Princeton, NJ.
[12] Lord, M. (2007) NetSim: un logiciel de modélisation et de simulation de réseaux d'information, Maîtrise en Informatique, UQAM, Montréal.
[13] Albert, R., Jeong, H. & Barabási, A.L. (1999) Diameter of the World-Wide Web, *Nature*, vol. 401, 130-131.
[14] Barabási, A.L. & Albert, R. (1999) Emergence of scaling in random networks, *Science* 286, 509-512.
[15] Dorogovtsev, S. N. & Mendes, J. F. F. (2000) Evolution of reference networks with aging, *Physical Review E*, vol. 62, 1842.
[16] Amaral, L. A. N., Scala, A., Barthélémy, M. & Stanley H. E. (2000) Classes of small-world networks, *Proceedings of National Academy of Sciences USA*, vol. 97, 11149-11152.
[17] Albert, R.H. & Barabási, A.-L. (2000) Topology of evolving networks: local events and universality, *Physical Review Letters*, vol. 85, 5234.
[18] Jin, E.M., Girvan, M. & Newman, M.E.J. (2001) The structure of growing social networks, *Phys. Rev. E* 64, 046132.
[19] http://jung.sourceforge.net/
[20] http://prefuse.org/
[21] http://www.analytictech.com/ucinet/ucinet.htm
[22] http://vlado.fmf.uni-lj.si/pub/networks/pajek/