Internet Topology Modeler Based on Map Sampling

Conference Paper *in* Proceedings - International Symposium on Computers and Communications · February 2002

DOI: 10.1109/ISCC.2002.1021797 · Source: IEEE Xplore

2 authors:

Damien Magoni
University of Bordeaux
143 PUBLICATIONS **1,454** CITATIONS

SEE PROFILE

Jean-Jacques Pansiot
University of Strasbourg
101 PUBLICATIONS **1,712** CITATIONS

SEE PROFILE

Internet Topology Modeler Based on Map Sampling

Damien Magoni, Jean-Jacques Pansiot
Université Louis Pasteur – LSIIT
Pôle API, Boulevard Sébastien Brant, 67400 Illkirch, France
{magoni, pansiot}@dpt-info.u-strasbg.fr

Abstract

Creating a network topology is the first step in building a scenario for a network protocol simulation. The simulation results usually depend on the topology layout, especially for routing and multicasting protocols. Therefore the topology used should be generated with the highest possible accuracy. In particular, protocols designed for the Internet should be simulated over Internet-like topologies. Many topology generators currently exist but the recent discovery of power laws in the Internet has brought new constraints upon the generated topologies. We introduce a flexible novel way to create Internet-like topologies. It is based on an algorithm that performs a sampling on a measured Internet map. The created topologies accurately comply with the newly found power laws as well as other more common topology properties such as the average path length.

1. Introduction

Network topologies are usually modeled by graphs. Hence topology generators are also called graph generators. In the same way, graph properties such as the mean degree and the diameter are called topology properties. We can check, to some extent, the accuracy of a generated graph *vs* a real network map by comparing their topology property values. The generated graph conformance to a real network is required by the increased use of simulators such as *ns-2* and *GloMoSim* by the research community. Although early simulation studies were satisfied with purely random graphs, today protocol simulation studies need realistic graphs based on the study of real networks. In this paper we present a topology modeler (it is not truly a generator so we use the term *modeler*) able to create small Internet topologies complying with the laws found in measured Internet maps. Although our modeler can create both Autonomous System (AS) or router level graphs, we focus here on the generation of graphs that model the Internet topology at the router level (*i.e.* each node of the graph is a router and each link is an IP adjacency). The Internet router level topology is very hard to capture because of its huge size and its dynamic changes so we only have currently three maps (available for free) at our disposal. Our method creates suitably sized router level Internet-like graphs by extracting them from one of the available Internet router level maps.

In section 3 we provide a short analysis of the Internet topology by using the three freely available Internet router maps in order to find common interesting topology properties. Section 4 explains our graph extraction method and gives a specification of our algorithm in pseudo-code. In short, it performs a randomized node sampling on a real Internet map to produce a tree. It then adds redundant links to produce graphs that have appropriate topological properties. Section 5 shows how the created graphs succeed in accurately modeling the topology properties found in the Internet maps.

2. Previous work

One of the earliest graph generation model was defined by Waxman [13] in 1988. As it was not trying to specifically model the Internet topology, it was replaced in 1996 by topology generation models enforcing a hierarchical graph construction such as Tiers [4] and Transit-stub [14]. These models were reflecting more accurately the multi-layered topology of Internet (*i.e.* host–router–Autonomous-System). However, the discovery of power laws in the Internet [5, 8] made these models obsolete. New topology models complying with the power laws were recently created. At the router level, some available graph generators are BRITE [10], PLOD [11] and our generator *nem* [9]. At the AS level a currently available graph generator is Inet2 [7]. Generic power law graph models have also been created by Aiello *et al.* [1] and Albert *et al.* [2]. Our sampling algorithm is implemented in our graph modeler and analyzer called *nem* (which stands for *ne*twork *m*anipulator). It is freely available for use and can be downloaded at http://www-r2.u-strasbg.fr/nem. In this paper we describe our algorithm in detail and we try to exhibit how it is able to create accurate Internet–like graphs particularly concerning the distance properties with respect to the ones measured in the Internet maps.

Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)
1530-1346/02 $17.00 © 2002 **IEEE**

3. Internet topology

Our first step is the study of the topology of Internet. As shown in table I we have three Internet router level maps at our disposal. The first one is a small map collected at ULP/LSIIT by Pansiot *et al.* in 1995 [12]. The second and the third ones were both collected in 1999, one at the Lucent labs by Burch *et al.* [3] and the other at USC/ISI by Govindan *et al.* [6]. These three maps give us three very different snapshots of Internet either by the creation date or by the number of nodes. In this section we study the values of the topology properties of these three maps to see if they comply with the power laws found at the AS level of Internet. We also try to find invariants among the distance properties of the maps. This will help us in defining criteria for assessing the fidelity of the generated graphs.

3.1. Degree properties

Faloutsos *et al.* have shown in [5] that the AS level maps of Internet have skewed degree distributions. They have defined two power laws that concisely describe this distribution shape:

- Power law 1 (rank exponent): The outdegree d of a node is proportional to the rank r of the node to the power of a constant R. So we have: $d \propto r^R$.
- Power law 2 (outdegree exponent): The frequency f of an outdegree d is proportional to the outdegree to the power of a constant O. So we have: $f \propto d^O$.

They have also defined another third power law and one approximate law but these last two laws do not provide interesting indicators for Internet similarity. Medina *et al.* have indeed shown in [10] that Waxman and Transit–stub graphs comply with power law 3 (eigen–value exponent) although these graphs do not comply with power laws 1 and 2. Furthermore the hop–plot exponent law is not yet proved to be a power law (it is derived by approximation). Thus we only use the first two power laws defined by Faloutsos *et al.* We have analyzed the three router level Internet maps and we have found that all of them comply with the rank and degree power laws (the case of the LSIIT router map was already verified in [5]). Table 1 shows the values of the Absolute Correlation Coefficients (ACC) of these two power laws for the three maps. As the values are all (but one) above 0.95 (the threshold of validation for the linear regression), it is clear that all three maps comply with the rank and degree power laws. It is striking to see how the rank and degree ACC do not depend on the size of the maps and thus provide good indicators. The rank ACC of the LSIIT map is just below the commonly used threshold of 0.95. The small size of this map *vs* the others may be the cause of the weak value of its rank ACC. To conclude we can say that checking the presence of these two power laws in the extracted graphs is meaningful to assess their similarity to the Internet maps.

Table 1. Degree properties

	LSIIT-95	Lucent-99	ISI-99
Number of nodes	3888	112969	228263
Rank ACC	0.945	0.967	0.975
Degree ACC	0.967	0.984	0.987

3.2. Tree properties

We have shown in [8] that the AS level maps of Internet have skewed tree size distributions. We have defined two power laws in the same way as Faloutsos *et al.* did:

- Power law 6 (tree rank exponent): The size s of a tree (*i.e.* number of nodes) is proportional to the rank r of the tree to the power of a constant T. So we have: $s \propto r^T$.
- Power law 7 (tree size exponent): The frequency f of an tree size s is proportional to the tree size to the power of a constant S. So we have: $f \propto s^S$.

We have also defined two other power laws related to the number of shortest paths between pairs but we haven't investigated them enough to use them. Indeed we want to correlate them to the hop–plot exponent approximate law. This is left for future work. We have analyzed the three router level Internet maps and we have found that all of them comply with the tree rank and tree size power laws. Table 2 shows the values of the ACC of these tree power laws for the three maps. Only the LSIIT map does not comply with the tree size power law. This is probably caused by its very small size. To conclude like the previous section, we can say that checking the presence of these two power laws in the extracted graphs is meaningful to assess their similarity to the Internet maps.

Table 2. Tree properties

	LSIIT-95	Lucent-99	ISI-99
Number of trees	391	15289	23136
Tree rank ACC	0.981	0.986	0.978
Tree size ACC	0.891	0.986	0.982

3.3. Distance properties

Throughout the paper we consider that the distance between two routers is the length of a shortest path measured in number of hops. The average path length of a router is the average of all the path lengths from it to every other router. The eccentricity of a router is the maximum

path length separating it from all the other routers. Fig. 1 shows the average path length distributions of the three maps. We can see that they are not skewed but are rather close to a Gaussian distribution shape. The average path length distributions of the Lucent and ISI maps seem to be nearly equal with a shift of 0.5 although the ISI map is twice as big as the Lucent map. The LSIIT distribution is centered a bit higher and is more spread but it is similar to the others although its size is only 3.4% of the Lucent map. This clearly proves that the average path length is not correlated to the graph size. We have found similar results for the eccentricity distributions.

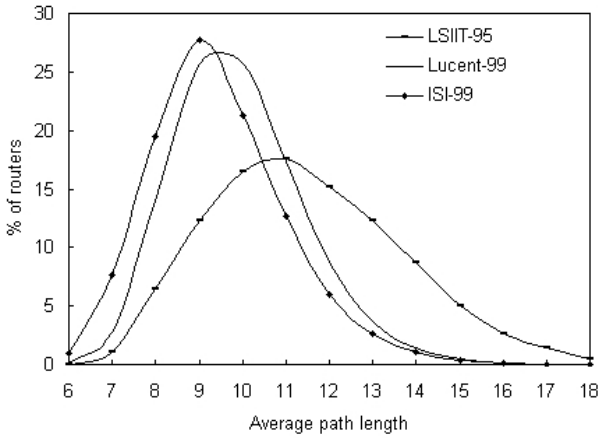

Fig. 1. Average path length distributions of the Internet router-level maps

The shape of the average path length distributions makes it interesting to look at the average values of these distributions shown in table 3. The average path length given in the table for each map is the average of the average path lengths of all routers. The average eccentricity given in the table is the average of the eccentricity of all routers. We can see that they are quite close to each other. This means that the average of an average path length or eccentricity distribution is a reliable indicator for assessing a graph realism. The path lengths are important as they permit an accurate evaluation of the number of packets created in a communication between two or more nodes. Again we can say that checking the presence of these two distance related gaussian laws and the values of their averages in the extracted graphs is meaningful to assess their similarity to the Internet maps.

Table 3. Path length properties

	LSIIT-95	Lucent-99	ISI-99
Average path length	11.59	9.94	9.51
Average eccentricity	21.86	19.19	20.79

4. Sampling algorithm

Previous Internet topology generators use techniques such as random placement or incremental growth and preferential connectivity or "reverse engineering" by first building an appropriate degree distribution. We introduce here a new way to create graphs. It makes use of an algorithm that randomly extract a subgraph of a real Internet map. As we focus on the router level of the Internet, we use router level Internet maps as input graphs (but it works in the same way for the AS level). By extracting nodes in a mostly random fashion, we ensure the creation of a wide diversity of subgraphs.

4.1. Description of the algorithm

The figure 2 shows how the tree creation algorithm works. Figure 2 (a) shows a small part of the map being sampled. In the first step (b), the node 1 is selected randomly among all the nodes of the map. Nodes 2 and 5 are then added to a candidate list (see below) with candidate link a and b respectively. In step (c), node 5 is selected randomly among the nodes of the candidate list. Having candidate link b, it is connected to the tree (composed only of node 1 for the moment) by this link. Nodes 4, 6 and 7 are added to the candidate list with candidate link e, f and g respectively. Node 2 is already in the candidate list so it is not added to it. As node 2 is connected to 5 by link d, a random roll is performed to check if its candidate link changes to d (it is currently a). The roll fails and the candidate link is not changed. In step (d), node 2 is selected randomly among the nodes of the candidate list. Having candidate link a, it is connected to the tree (composed of nodes 1 and 5 for the moment) by this link. Node 3 is added to the candidate list with candidate link c. As node 5 is already selected, link d is added to the redundancy link list for later use (after the tree construction). In step (e), node 6 is selected randomly among the nodes of the candidate list. Having candidate link f, it is connected to the tree (composed of nodes 1, 2 and 5 for the moment) by this link. Node 8 is added to the candidate list with candidate link i. Node 7 is already in the candidate list so it is not added to it. As node 6 is connected to 7 by link h, a random roll is performed to check if its candidate link changes to h (it is currently g). The roll succeeds and the candidate link is changed to h. In the last step shown (f), node 7 is selected randomly among the nodes of the candidate list. Having candidate link h, it is connected to the tree (composed of nodes 1, 2, 5 and 6) by this link. As node 5 is already selected, link g is added to the redundancy link list for use after the tree construction. This algorithm enables us to extract a tree having a number of nodes equal to the one of our future graph and to create a supply of redundancy links.

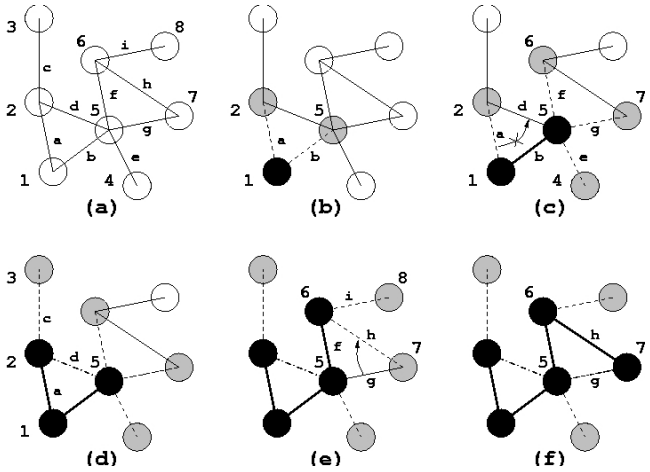

Fig. 2. Tree creation process

4.2. Algorithm data types and variables

In addition to the usual types (*i.e. int*, *float*, …) we use two data structure types described in table 4. The variables in table 5 are considered global variables of the algorithm and thus are not defined in any function. They keep their state between function calls. The *list* type is supposed to be a typical template container implemented by a linked list.

Table 4. Object members

Object name	Members	Description
link	orig_node	Reference to the node at the beginning of the link
	end_node	Reference to the node at the end of the link
	selected	Boolean indicating if the link is in the graph
node	address	Unique identifier
	links	List of the links of the node
	selected	Boolean indicating if the node is in the graph
	candidate	Boolean indicating if the node is in the "candidates" list (see below)
	candidate_link	Reference to a link connecting a selected node to this one

Table 5. Algorithm global variables

Variable name	Type	Description
map	*list<node>*	A collected router-level map of (a part of) the Internet from which a graph will be extracted
node_count	*int*	The current number of selected nodes for the graph
link_count	*Int*	The current number of selected links for the graph
candidates	*list<node>*	A list of the nodes candidate for selection in the graph
red_links	*list<link>*	A list of redundant links candidate for selection in the graph

4.3. Algorithm specification

The algorithm is given below in pseudo–code. Regular operations such as randomly choosing a node or adding a node to a list are described literally (in italic). At the end of the algorithm, all nodes and links marked as *selected* are composing the extracted graph. The final step which consists in copying them in a new memory data structure is not shown here. The main function of the algorithm is called **GraphExtraction** and it takes two input parameters: the nb_of_nodes_wanted and the mean_degree_wanted which have both self–explanatory names. It calls three functions, each corresponding to an important step of the algorithm.

```
1 GraphExtraction (nb_of_nodes_wanted : int,
mean_degree_wanted : float)
2       Initialization ( )
3       TreeExtraction (nb_of_nodes_wanted)
4       RedundancyLinkCreation (nb_of_nodes_wanted,
mean_degree_wanted)
```

The **Initialization** function chooses the first node of the graph. As already said, each node of the map marked as selected will be considered as in the graph (also true for each link). A regular node can be at first a candidate node and then a selected node but it can't be both at the same time.

```
1 Initialization ( )
2       FOR_EACH node IN map
3          node.candidate ← FALSE
4          node.selected ← FALSE
5          node.candidate_link ← 0
6       FOR_EACH link IN map
7          link.selected ← FALSE
8       Pick a node in the map and assign it to first_chosen
9       first_chosen.selected ← TRUE
10      candidates ← {∅}
11      FOR_EACH link IN first_chosen.links
12         neighbor ← link.end_node
13         neighbor.candidate ← TRUE
14         neighbor.candidate_link ← link
15         Add neighbor in the candidates list
```

The aim of the **TreeExtraction** function is to create the graph incrementally in the form of a tree. We do this in order to ensure the connectivity of the future graph. By building a tree we do not have to check afterwards if the extracted graph is connected. At the end of this function the desired number of nodes in the graph is reached.

```
1 TreeExtraction (nb_of_nodes_wanted)
2     red_links ← {∅}
3     WHILE node_count < nb_of_nodes_wanted DO
4       Pick a node in the candidates list and assign it to chosen
5       chosen.selected ← TRUE
6       chosen.candidate_link.selected ← TRUE
7       FOR_EACH link IN chosen.links
8         neighbor ← link.end_node
9         IF neighbor.selected = FALSE THEN
10          IF neighbor.candidate = FALSE THEN
11            IF FilterCandidate (neighbor) = FALSE THEN
12              neighbor.candidate ← TRUE
13              neighbor.candidate_link ← link
14              Add neighbor in the candidates list
15            ELSE IF Random (1, 100) > 50 THEN
16              neighbor.candidate_link ← link
17        ELSE IF neighbor ≠ chosen.candidate_link.orig_node
18          Add link in the red_links list
```

The candidate_link field is used by candidate nodes to store a reference to a link that connect them to a selected node. For efficiency, we do not store a list but only one instance. If a candidate node is selected, its candidate_link is also selected for inclusion in the graph. As an heuristic, if a new candidate_link is found for a candidate node, it will replace its predecessor with a 50% probability. The **FilterCandidate** function allows discarding some of the candidate nodes having a high degree thus increasing the distance property values. Furthermore, this function also increases the algorithm's speed as a side–effect. It uses two parameters σ and ε. If the candidate list size is below σ, the candidate node is added to the list without further checks. The threshold σ is used to prevent the list from becoming empty (although it can not ensure this). σ is typically set to 10 and its only purpose is to make the algorithm work smoothly (it is a practical parameter that should remain constant in most situations). If the candidate list size is equal or above σ, a random roll is performed and compared to a formulae which takes the node's degree as argument. The formulae is designed to reduce the chances of nodes having a high degree from being taken. The higher ε is, the lower the chance of a node, with a given degree, is to be chosen as a candidate. ε acts like a stretch parameter for the extracted graphs therefore obtaining higher values for distance properties. However the value of ε must not be too high otherwise the graphs will not match some of the other topology properties anymore. Usually $\varepsilon \approx 1$ gives suitable results for graphs of a few thousand nodes (ε does not have to be changed for each specific size).

```
1 FilterCandidate (node)
2     IF Size (candidates) < σ OR
IF Random (1, 100) < 100 – (Size (node.links))^ε THEN
3       RETURN FALSE
4     ELSE
5       RETURN TRUE
```

Finally the **RedundancyLinkCreation** function adds links one by one from the redundancy link pool created during the tree generation. The red_links list can sometimes be too small to provide enough links for the graph to reach the desired mean degree. Creating new links in the graph not existing in the map could alleviate this limitation but we have not implemented this option yet.

```
1 RedundancyLinkCreation (number_of_nodes_wanted,
mean_degree_wanted)
2     link_count ← number_of_nodes_wanted – 1
3     WHILE TRUE
4       Pick a link in red_links list and assign it to chosen_link
5       chosen_link.selected ← TRUE
6       link_count ← link_count + 1
7       IF red_links is empty OR link_count ≥
(mean_degree_wanted × number_of_nodes_wanted) ÷ 2
8         BREAK
```

5. Performances

We have implemented our new sampling algorithm in our *nem* graph manipulator software. We have extracted 20 graphs for each size (ranging from 125 to 4000), for each model (see below) and from each of the three Internet router level maps as input. The topology property values are averaged for the 20 graphs. It is worth noticing that it only takes, on average, 1.7s to extract a 4000–node graph.

5.1. Methodology

To study the efficiency of our algorithm concerning the distance properties, we have defined four graph creation models. The first model is composed only of the tree extraction part of our algorithm (it is called *Tree*). The second model is a modified version of our algorithm that creates all the links existing in the map and connecting the nodes of the graph. This means that we do not stop when the desired mean degree is reached but when all possible redundant edges are added (it is called *All edges*). These first two models give us boundaries for our algorithm's performances concerning the distance properties. The third model is composed of our algorithm without the use of our filtering function (called *nem w/o filter*) and the fourth model is the complete algorithm itself (*called nem w/ filter*).

Concerning the values of the two parameters of the filtering function in the fourth model, we have used σ = 10 and ε = 1.1 for all extractions. The average degree wanted was fixed to 2.3 for graphs under 1000 nodes and 2.4 for

graphs of 1000 or more nodes. As stated in the section describing the algorithm, the parameters are quite universal and usually work for any input map. They can be changed in special cases where the extraction often fails which was not the case in our experiments.

5.2. Results

We have found similar results for graphs extracted from the three Internet maps. Therefore we show only the results of the graphs extracted from the ISI map. We have looked at the average path length and eccentricity distributions of a sample of the extracted graphs and they all have a gaussian shape (overlooking the small oscillations due to the sampling scale). This means that we can consider the average values of these distributions as being interesting distance indicators. Fig. 3 clearly demonstrates that it is possible to extract graphs that have the same average path length than a 200 times bigger map (shown by the dotted line). All graphs of size equal or above 500 nodes have average distances between 8 and 10 which are typical values measured in Internet router level maps. For graphs of less than 500 nodes it is hardly possible to obtain satisfying results as even the first model (*Tree*) graphs have low distance values. Our filtering function is necessary to achieve these results. The basic extraction is not enough to obtain accurate average path lengths at these size levels (a few thousand nodes).

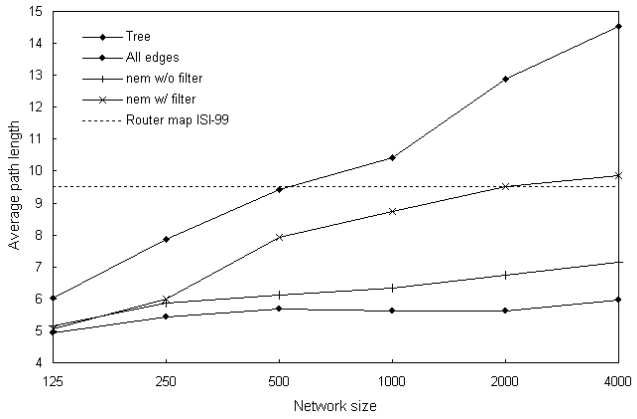

Fig. 3. Average path length of the graphs

Fig. 4 exhibits a similar behavior for the average eccentricity than for the average path length although the eccentricity of the collected map could not be reached. However *nem* graphs of 1000–node size or above have average eccentricity values that are within 20% of the ISI map average eccentricity. It is worth noticing that fig. 4 also shows that the eccentricity values of the *nem* graphs are closer to the measured map value when the filtering function is used. Again this is mainly due to the strong restriction on the sizes of the extracted graphs. It is possible to have eccentricity values equal to the one of the measured map without filtering if the graph size has more than 10000 nodes for instance. However these graphs are not interesting as most of the current simulators are not able to handle such big graphs.

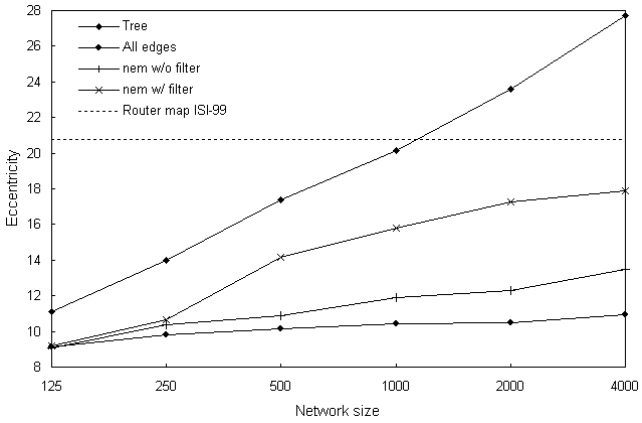

Fig. 4. Average eccentricity of the graphs

We saw that the distance properties of the graphs extracted by the complete algorithm are satisfactory, however we must still check if these graphs do comply with the power laws found in the Internet maps they were extracted from. Fig. 5 and 6 show the power law 1, 2, 6 and 7 ACCs of the graphs extracted by the complete *nem* algorithm (*i.e.* the *nem w/ filter* model). Fig. 5 shows the ACCs for the rank and degree power laws. All extracted graphs comply with these laws for any size with values above the 0.95 commonly used threshold.

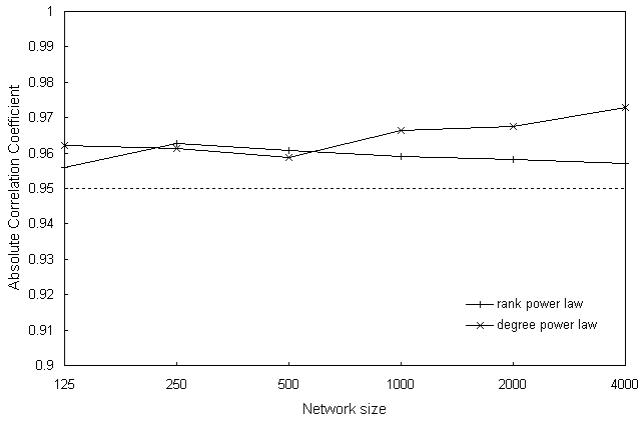

Fig. 5. Rank and degree ACC of the graphs

Fig. 6 shows the ACCs of the *nem* graphs for the tree rank and tree size power laws. All extracted graphs comply with the tree rank power law with values above 0.95 for any size. However all *nem* graphs below 2000 nodes do not comply with the tree size power law (2000–node graphs

being just under the threshold) although the ACC values are not too bad (all above 0.9).

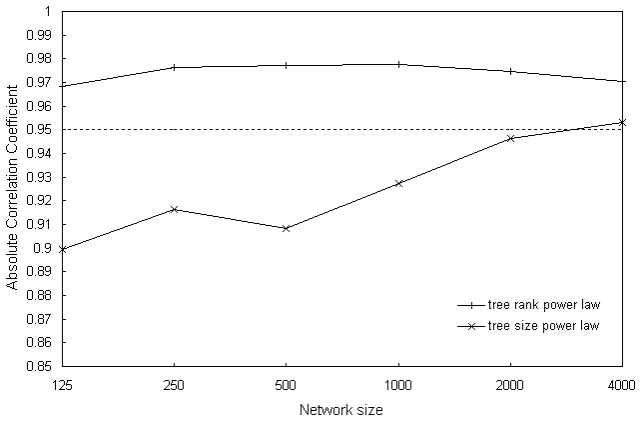

Fig. 6. Tree rank and tree size ACC of the graphs

To conclude the results section, we saw that the *nem* extracted graphs comply with power law 1, 2 and 6 for all sizes and comply with the power law 7 for sizes above 1000 nodes. Concerning the distance properties (average path length and average eccentricity), we saw that accurate results can be achieved for graph sizes of at least 1000 to 2000 nodes.

6. Conclusions

We have shown that distance properties are not correlated to the size of the graphs as long as the size is above a given threshold of a few thousand nodes. We are able to extract graphs two orders of magnitude smaller than their originating map that still have identical distance property values. Furthermore the extracted graphs do comply with the recently found power laws concerning degree and tree distributions. The comparison of the accuracy of our graph modeler with the accuracy of the other power law complying graph generators will be presented in a future paper. However as we already have promising results, we encourage researchers to try our *nem* software for network topology analysis and modeling.

7. References

[1] W. Aiello, F. Chung, and L. Lu, "A random graph model for massive graphs", in Proc. *ACM STOC'00*, pp. 171–180, 2000.

[2] R. Albert, and A-L. Barabási, "Topology of evolving networks: local events and universality", *Physical Review Letters*, no. 85, pp. 5234, 2000.

[3] H. Burch, and B. Cheswick, "Mapping the Internet", *IEEE Computer*, vol. 32, no. 4, pp. 97–98, April 1999.

[4] M. Doar, "A better model for generating test networks", in Proc. *IEEE Globecom'96*, November 1996.

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology", in Proc. *ACM SIGCOMM'99*, September 1999, Cambridge, Massachusetts, USA.

[6] R. Govindan, and H. Tangmunarunkit, "Heuristics for Internet map discovery", in Proc. *IEEE Infocom'00*, June 2000, Tel Aviv, Israël.

[7] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet topology generator", Technical Report CSE–TR–433–00, University of Michigan, 2000.

[8] D. Magoni, and J.–J. Pansiot, "Analysis of the Autonomous System network topology", *ACM Computer Communication Review*, vol. 31, no. 3, pp. 26-37, July 2001.

[9] D. Magoni, and J.–J. Pansiot, "Internet topology analysis and modeling", In Proc. *IEEE Computer Communications Workshop*, October 14–17, 2001, Charlottesville, Virginia, USA.

[10] A. Medina, I. Matta, and J. Byers, "On the origin of power laws in Internet topologies", *ACM Computer Communication Review*, vol. 30, no. 2, April 2000.

[11] C. Palmer, and G. Steffan, "Generating network topologies that obey power laws", in Proc. *IEEE Globecom'00*, 2000.

[12] J.–J. Pansiot, and D. Grad, "On routes and multicast trees in the Internet", *ACM Computer Communication Review*, vol. 28, no. 1, pp. 41–50, January 1998.

[13] B. Waxman, "Routing of multipoint connections", *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, December 1988.

[14] E. W. Zegura, K. L. Calvert, M. J. Donahoo, "A quantitative comparison of graph-based models for internetworks", *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 770–783, December 1997.