NETWORKS AND HETEROGENEOUS MEDIA
©American Institute of Mathematical Sciences
Volume **3**, Number **2**, June **2008**

Website: http://aimSciences.org

pp. **277–294**

AUGMENTING k-CORE GENERATION WITH PREFERENTIAL ATTACHMENT

Michael Baur, Marco Gaertler, Robert Görke, Marcus Krug and Dorothea Wagner

Faculty of Informatics, Universität Karlsruhe (TH)
78131 Karlsruhe, Germany

Abstract. The modeling of realistic networks is of prime importance for modern complex systems research. Previous procedures typically model the natural growth of networks by means of iteratively adding nodes, geometric positioning information, a definition of link connectivity based on the preference for nearest neighbors or already highly connected nodes, or combine several of these approaches.

Our novel model brings together the well-know concepts of k-cores, originally introduced in social network analysis, and of preferential attachment. Recent studies exposed the significant k-core structure of several real world systems, e.g., the AS network of the Internet. We present a simple and efficient method for generating networks which at the same time strictly adhere to the characteristics of a given k-core structure, called core fingerprint, and feature a power-law degree distribution. We showcase our algorithm in a comparative evaluation with two well-known AS network generators.

1. **Introduction.** The interest in modeling classes of graphs has significantly increased by recent studies of complex systems such as the Internet, biological networks, river basins, or social networks. While random graphs have been studied for a long time, the standard models appear to be inappropriate because they do not share certain abstract characteristics observed for those systems. One of these characteristics is the k-core structure which can be interpreted as a nested decomposition separating parts of the network based on their density. This decomposition is commonly applied in order to identify central parts of the networks since it peels the network layer by layer, filtering out less important parts that are sparsely connected with the remaining graph. Example applications are network fingerprinting with LaNet-vi [3], protein network analysis [27], or the exploration of modern social networks [12].

A crucial field of application of graph generators is the simulated evolution of a given network, granting insights in both its past development and its anticipated future behavior. One prominent example is the Internet at the Autonomous System (AS) level where various models have emerged over the last few years, including BRITE [20], Inet [17], nem [18], and various models presented by Pastor-Satorras

2000 *Mathematics Subject Classification.* Primary: 05C85, 05C80; Secondary: 05C75.

Key words and phrases. graph generation, k-core structure, preferential attachment.

This work was partially supported by the DFG under grant WA 654/14-3 and EU under grant DELIS (contract no. 001907). A previous version appeared as Generating Graphs with Predefined k-Core Structure, at the European Conference on Complex Systems (ECCS 2007) [7].

and Vespignani [22]. While this network has been observed to possess a very distinct k-core structure [2, 9], kept track of over a long period of time, all generating tools so far ignore this structure, and thus largely fail to do justice to this significant and stable property [11]. Overall, up to our knowledge an approach to create networks with a given k-core structure is missing so far.

To address this issue we refine the abstract measurement of core sizes to a *core fingerprint* that additionally includes information on the inter-connectivity of each pair of shells. This allows us to design a simple and efficient method to incrementally generate randomized networks with a predefined k-core structure, starting with the maximum core. By utilizing two results on edge rewiring we thus achieve a structure that precisely matches the core fingerprint.

Predefining the core fingerprint of a network still leaves many degrees of freedom open. Since we focus on the network of Autonomous Systems as a case study we exploit this fact and optionally bias the randomness in the adjacency of nodes towards *preferential attachment*, as described by Barabási and Albert [1]. This paradigm of setting up links in a network has been proven to introduce a power-law degree distribution, which has first been observed by Faloutsos et al. [14] for the Internet. Our approach imposes almost no modifications on a vanilla realization of preferential attachment, a fact that is reflected by our experimental results. We thus manage to coalesce two of the most fundamental concepts in the theory of complex networks of the recent past.

This paper is organized as follows: first we clarify the preliminaries and state some basic properties for k-core structures and on preferential attachment in Section 2, then we give the description of the network generator in Section 3. In Section 4 we evaluate our model in comparison to two well-known generators with respect to commonly used network properties. Finally we give some concluding remarks.

2. Preliminaries.

Let $G = (V, E)$ be a simple, undirected graph. A subset $V' \subseteq V$ of the node set induces a subgraph $G[V'] := (V', E')$ where the edge set E' is defined by $E' := \{\{u, v\} \mid u, v \in V', \{u, v\} \in E\}$. The degree $\deg(v)$ of the node v is the number of its incident edges. A nested decomposition of G is a finite sequence $(V_0, \ldots, V_k)$ of subsets of nodes such that $V_0 = V$, $V_{i+1} \subseteq V_i$ for $i < k$, and $V_k \neq \emptyset$.

2.1. Core decomposition.

Cores are a widely used realization of nested decompositions. The concept was originally introduced by Seidman [25] and generalized by Batagelj and Zaversnik [6]. Constructively speaking, the *i-core* of an undirected graph is defined as the unique subgraph obtained by iteratively removing all nodes of degree less than i. This procedural definition immediately gives rise to a construction algorithm that can easily be implemented. Moreover, it is equivalent to the closed definition of the i-core as the set of all nodes with at least i adjacencies to other nodes in the i-core. The *core number* of a graph is the smallest i such that the $(i+1)$-core is empty, and the corresponding i-core is called the *core* of a graph. Figure 1 depicts the core decomposition of an example graph with a core number of 4. The core decomposition can be computed in linear time with respect to the graph size [5].

A node has *coreness* i, if it belongs to the i-core but not to the $(i+1)$-core. We call the collection of all nodes having coreness i the *i-shell*. An edge $\{u, v\}$ is an

intra-shell edge if both u and v have the same coreness, otherwise it is an *inter-shell edge*.

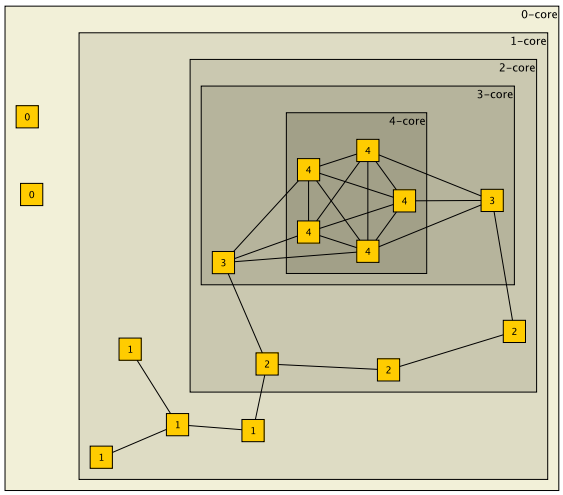


FIGURE 1. A k-core decomposition with 5 core shells.

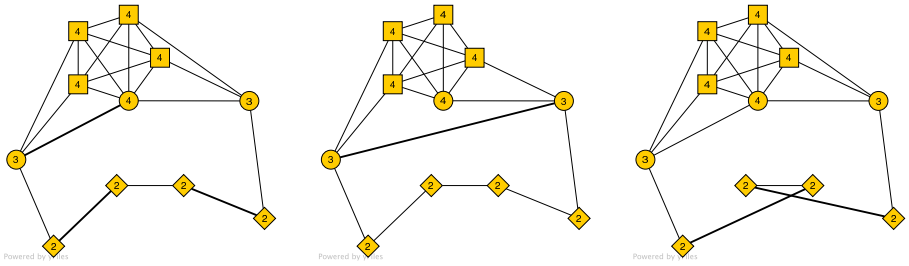
Informally speaking, the coreness of a node can be viewed as a robust version of the degree, i. e., a node of coreness i retains its coreness even after the removal of an arbitrary number of nodes of smaller coreness. In the following section we state some observations on core structures, that are crucial to our approach.

2.2. **Edges in a core hierarchy.** The following two lemmas summarize two facts about the relation of intra- and inter-shell edges. Note that Lemma 2.1 corrects a flaw present in a previous version of this paper [7]. We later exploit this interaction and interchangeability of edges in our network generation algorithm.

Lemma 2.1 (Rewiring)**.** *Let $G = (V, E)$ be a graph. Let $u, v \in V$ be two non-adjacent nodes with the same coreness and $\{u, w\}, \{v, w'\} \in E$ two edges such that* $\text{coreness}(u) < \min\{\text{coreness}(w), \text{coreness}(w')\}$*. Then $G' := (V, E')$ with $E' := E \setminus \{\{u, w\}, \{v, w'\}\} \cup \{u, v\}$ has the same core decomposition as G. Conversely, let $u, v \in V$ be two adjacent nodes with the same coreness k and with at most $k - 1$ neighbors in higher cores, and let $w, w' \in V$ be two nodes such that* $\text{coreness}(u) < \min\{\text{coreness}(w), \text{coreness}(w')\}$ *and $\{u, w\}, \{v, w'\} \notin E$. Then $G'' := (V, E'')$ with $E'' := E \setminus \{u, v\} \cup \{\{u, w\}, \{v, w'\}\}$ has the same core decomposition as G.*

Lemma 2.2 (Swapping)**.** *Let $G = (V, E)$ be a graph, $u, v, w, w' \in V$ be four nodes all having the same coreness, $\{u, v\}, \{w, w'\} \in E$ be two intra-shell edges, and $\{u, w\}, \{v, w'\} \notin E$. Then the graph $G' := (V, E')$ with $E' := E \setminus \{\{u, v\}, \{w, w'\}\} \cup \{\{u, w\}, \{v, w'\}\}$ has the same core decomposition as G.*

It is not hard to see that the correctness of both lemmas follows from the definition of cores. The cumbersome prerequisites can be understood more easily by the concept of a removal order that will be introduced later in Section 3.2. Informally speaking, Lemma 2.1 allows for most pairs of disconnected nodes of the same coreness to each remove one edge to some nodes of higher coreness and instead become



(a) original graph (b) after rewiring (c) after swapping

FIGURE 2. Rewiring and swapping edges in the left graph. The labels show the coreness of the nodes.

connected, and vice versa, without changing the decomposition. Furthermore, according to Lemma 2.2 we can swap the end-nodes of intra-shell edges if this does not interfere with existing connections. Figure 2 illustrates these two lemmas for an example graph. Using these statements, we can now establish (tight) bounds of the sizes of cores and shells.

Lemma 2.3 (Size of i-Cores)**.** *Let $G = (V, E)$ be a graph, $(V_0, \dots, V_k)$ its core decomposition and $G_i := (V_i, E_i) := G[V_i]$ the i-core. Then the size of every i-core is bounded as follows:*

$$i + 1 \leq |V_i| \quad \text{and} \quad \frac{(i+1)i}{2} \leq |E_i| \ . \tag{1}$$

Let $n_i := |V_i \setminus V_{i+1}|$ be the number of nodes with coreness i and $m_i := |E_i \setminus E_{i+1}|$ the number of all edges whose end-nodes with minimum coreness has coreness i for $0 \leq i \leq k$ (for convenience we define $V_{k+1} := \emptyset$ and $E_{k+1} := \emptyset$). Then the size of the i-shell is bounded as follows:

$$0 \leq \quad n_i \quad \leq |V| \tag{2}$$

$$\left.\begin{matrix} \left\lceil \frac{i \cdot |n_i|}{2} \right\rceil & \text{, if } n_i > i \\ \binom{n_i}{2} + n_i \cdot (i - n_i + 1) & \text{, if } n_i \leq i \end{matrix}\right\} \leq \quad m_i \quad \leq \begin{cases} i \cdot n_i & \text{, if } i < k \\ i \cdot n_i - \frac{i^2+i}{2} & \text{, if } i = k \end{cases} \tag{3}$$

Note that the bounds for the i-core (Eq. 1) are trivially obtained from the definition. The bounds for the i-shell (Eq. 2 and 3), however, use the above two lemmas, i. e., the shell has the minimum number of edges, if it has the maximum possible number of intra-shell edges, since each such edge contributes twice, and a minimum number of inter-shell edges. An analogous reasoning yields the upper bounds. We omit proofs for the bounds of this lemma except of the following.

Proof of Upper Bound 3. By definition, there exists an removal order σ that iteratively removes a node v from V_k with $\deg(v) \leq k$, such that eventually all nodes in V_k are removed. We now count the maximum number of edges that still allow such an order of removal $\sigma(v)$, $v \in V_k$ by adding up the number of edges the removed nodes in such an removal order can maximally be incident to. For the first $n_k - (k+1)$ nodes (which can be zero), the removal order σ implies that the current node v can have a maximum degree of k. For the last $k+1$ nodes (minimum number of nodes for a k-shell) however, the number of incident edges during the removal

order is even less, resulting in a $(k+1)$-clique supported by $(k^2+k)/2$ edges. Thus we arrive at

$$\underbrace{(n_k - (k+1)) \cdot k}_{\text{by nodes beyond } k+1} + \underbrace{\frac{(k+1) \cdot k}{2}}_{\text{by clique of last } k+1 \text{ nodes}} = k \cdot n_k - \frac{k^2+k}{2} \tag{4}$$

edges in total, which proves the bound. It is easy to see that this bound is sharp, since our arguments induce an immediate construction.

Note, that this bound also applies to lower shells when excluding edges to higher shells. □

2.3. **Random models and preferential attachment.** A plethora of models for random graphs have been proposed in the past. The most prominent and fundamental include the *Erdős-Rényi* model [13], also known as $\mathcal{G}(n,m)$, Gilbert's model $\mathcal{G}(n,p)$ [15] and the Watts and Strogatz model [26], which is also known as the *small-world*-model. However, in a number of real-world graphs some properties have been identified that are unlikely to emerge in these models, most notably a distribution of node degrees that roughly obeys a *power-law*, a fact that has been identified by Faloutsos et al. [14]. More precisely, the number of nodes with degree d is proportional to $d^{-\gamma}$ for some constant γ. Graphs with this property are commonly referred to as *scale-free*. Barabási and Albert describe a growth process coined *preferential attachment* [1] that generates graphs with such a degree distribution. Starting out with an empty graph, this process iteratively adds a new node that is adjacent to a fixed number of already existing nodes. The choice of a specific neighbor is made with probability proportional to the current degree of the nodes. We closely adhere to the particularly efficient implementation of preferential attachment proposed by Batagelj and Brandes [4].

3. Core generator.

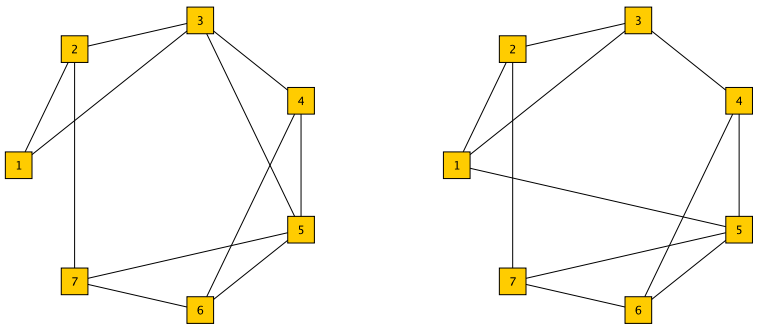
In this section, we first introduce a set of relevant parameters for the construction of core structures and discuss which combinations of these lead to feasible instances, i. e., are capable of realizing a graph with a predefined core structure. Then we describe our basic algorithm that generates such graphs, and point out several variations.

As the 0-shell only contains isolated nodes and in order to reduce technical peculiarities, we restrict ourselves to generating graphs with an empty 0-shell.

3.1. **Input parameters.** There are several possibilities to specify core structures. Of the quantitative approaches, the most obvious is to give the number of nodes per shell, the number of intra-shell edges, and the number of inter-shell edges (for each pair of shells). This can be coded as a vector $N \in \mathbb{N}_0^k$ where n_i is the number of nodes in the i-shell and a symmetric matrix $M \in \mathbb{N}_0^{k \times k}$, where $m_{i,j}$ contains the number of edges connecting the i-shell with the j-shell. We call this the core fingerprint. For example, the graph (omitting isolated nodes) given in Figure 1 has the following fingerprint:

$$N := (4,3,2,5) \quad \text{and} \quad M := \begin{pmatrix} 3 & 1 & 0 & 0 \\ 1 & 2 & 2 & 0 \\ 0 & 2 & 0 & 6 \\ 0 & 0 & 6 & 10 \end{pmatrix}$$

Clearly, the implied sizes of the shells have to respect the bounds established in Lemma 2.3. This kind of specification of core structures provides the maximum



(a) before rewiring (b) after rewiring

FIGURE 3. Example of rewiring. The fingerprint $N = (0, 0, 7)$ and $m_{3,3} = 11$ resulted in the left hand graph. Clearly, node 1 has insufficient degree. In the rewiring phase we can choose either node 3 or 5 as the RICH node. For the right hand graph we selected node 3 and node 5 as the RICH node and the pivot node, respectively. Thus we arrive at $E = E \setminus \{\{3, 5\}\} \cup \{\{1, 5\}\}$.

degree of freedom, i. e., the user can configure the size distribution of each shell and is only limited by constraints ensuring consistency.

One can easily relax the requirement of absolute values in the input by replacing them by parameters that correspond to the ratio of edges with respect to the tight bounds established in Equation 3. To further simplify the structure of the input, these ratios could be replaced by a density function. Such a function could, e.g., follow a simple power-law.

3.2. **Algorithmic approach.** Our generator builds a graph by iteratively adding new shells beginning at the maximum core. When adding a new shell, we create nodes and edges according to the given core fingerprint and take care to not change the coreness of nodes in previously built higher shells. The detailed pseudo code is given in Algorithm 1. We omit in-depth explanations of supplementary operations such as APPENDALL or REMOVEALLOCCURENCES, since these methods have one-to-one equivalences in most high-level programming languages. Non-computer scientists can assume that these methods match their intuitive nomenclature.

In order to guarantee that the coreness of nodes in the i-shell will not exceed i, we define an order σ_i which will be maintained as a valid removal order for this shell (line 4). It is of vital importance to ensure that for every node in V_i the sum of the number of neighbors in the shell i with a higher value of σ_i and the number of neighbors in higher shells does not exceed i. To model this, newly created edges are directed such that inter-shell edges point from the lower shell to the higher shell and intra-shell edges are directed in accordance to our predefined order σ_i and each node in V_i is restricted to a maximum out-degree of i (line 18). We are left to guarantee that the coreness is exactly i and not less. An example where this not yet satisfied is given in Figure 3(a).

While lines 3 to 25, called the *element generation phase*, avoid erroneously high values of coreness, as further detailed below, the *rewiring phase* in lines 27 to 38 solves the problem of erroneously low values of coreness by a sophisticated movement

Algorithm 1: Core Generator

Input: integer k, vector $N \in \mathbb{N}_0^k$, valid symmetric matrix $M \in \mathbb{N}_0^{k\times k}$
Output: graph $G = (V, E)$

```
1  V ← ∅; E ← ∅; targetNodes ← ∅;
2  for i ← k to 1 do                                          // introduce next shell
3      list V_i ← {n_i new nodes} ;
4      σ_i : V_i → {1, ..., n_i} defined by σ_i^{-1}(ℓ) = V_i[ℓ] ;   // removal order
5      u ← V_i[n_i] ;                                  // last node in removal order
6      list sourceNodes ← V_i \ {u} ;                  // u cannot source intra-edges
7      list targetNodes[i] ← {u} ;                                  // u into PA-list
8      list unconnectable ← {u} ;                                   // see line 21
9      for j ← i to k do                                  // select target shell
10         for m ← 1 to m_{i,j} do                        // introduce m_ij edges
11             s ← sourceNodes[random] ;                   // source of new edge
12             C ← targetNodes[j] ;                    // target candidates list
13             C.removeAllOccurences(neighbors(s) ∪ {s});
14             if j = i then                          // check removal order σ
15                 C.removeAllOccurences({ℓ ∈ V_i | σ(ℓ) < σ(s)});
16             t ← C[random] ;                            // target of new edge
17             E ← E ∪ (s, t) ;
18             if outdeg(s) = i then                        // source saturated
19                 sourceNodes.remove(s);
20             else if j = i and outdeg(s) ≥ n_i − σ_i(s) then
21                 sourceNodes.remove(s) ;          // no more intra-targets
22                 unconnectable.append(s) ;  // store for inter-targets
23             targetNodes[i].append(s, t);
24         if j = i then
25             sourceNodes.appendAll(unconnectable) ;          // restore
26     remove direction of edges;
27     list poorNodes ← {v ∈ V_i | deg(v) < i} ;
28     list richNodes ← {v ∈ V_i | deg(v) > i} ;
29     while poorNodes ≠ ∅ do                  // rewire unsaturated nodes
30         v ← poorNodes[random];
31         w ← richNodes[random];
32         C ← neighbors(w) \ neighbors(v) ;              // pivot candidates
33         c ← C[random];
34         E ← E \ {{w, c}} ∪ {{v, c}};
35         if deg(v) = i then                                  // v saturated
36             poorNodes.remove(v);
37         if deg(w) = i then                              // w no longer RICH
38             richNodes.remove(w);
39     V ← V ∪ V_i ;                                       // shell i completed
40 return graph G = (V, E);
```

of edges. We choose a node v with insufficient degree (line 30) and a node w with degree greater than i (line 31). Then we select a neighbor $c \in \text{NEIGHBORS}(w)$ which is not yet adjacent to v (lines 32 and 33) and replace this adjacency $\{w, c\}$ by a new edge $\{v, c\}$ (line 34).

Before we revisit the element generation phase in detail, we recapitulate the mechanism of preferential attachment. The network is grown from an arbitrary, small seed such as a single node or a triangle. Iteratively nodes are added and connected to a fixed number of neighbors. These neighbors are randomly selected from existing nodes with probability proportional to their degree. This behavior can be modeled by maintaining a list of nodes to which both end-nodes of each newly inserted edge are appended. Thus, this list contains each node with multiplicity equal to its current degree. Drawing uniformly at random from this list is a legitimate and efficient realization of preferential attachment, as described in detail by Batagelj and Brandes [4].

Shells are created iteratively, starting with the maximum core. First the predefined number n_i of nodes are created (line 3), together with an arbitrary removal order σ_i on them (line 4). In the element generation phase, some subtlety has been put into the choice of incident nodes of new edges. Since we only predefine the connectivity between shells, there is no fixed number of neighbors newly inserted nodes can be connected to. Instead, we maintain a list of sourceNodes which initially contains each node of shell i (line 6) exactly once, and an initially empty list targetNodes into which we insert the end-nodes of each new edge following the approach of Batagelj and Brandes [4].

We now iterate over each shell j that has already been created, starting with the very shell that has just been created ($j = i$), and create m_{ij} edges from shell i to shell j (loop starting at line 10). Each time an edge is created, we draw its source uniformly at random from sourceNodes (line 11) and check (line 18) whether it now has the maximum outdegree for belonging to shell i, in which case we remove it from sourceNodes. Further, in the case $j = i$, if there are no more feasible targets for this source, i.e., it is already connected to all nodes with a higher value of σ_i, we remove it from sourceNodes (line 21). However, such a node is not yet saturated and therefore stored in the list unconnectable (line 22) for later use in the case $j \neq i$ (line 25). Note that as a consequence, the highest ranking node $u = \text{argmax}_{v \in V_i} \sigma_i(v)$ in the current shell i is removed before the loop from sourceNodes (lines 5 and 6) and instantly added to the list unconnectable.

Since edges can be directed towards any higher shell, we maintain the list of targetNodes$[i]$ for each shell i throughout the algorithm. As mentioned above, these lists are the key for realizing preferential attachment. We initialize targetNodes$[i]$ with u (line 7), since u is the only feasible target for all $v \in V_i$. For each choice of s in line 11, a list of feasible target nodes C is created (line 12). To this end, we prune list C of illegal choices, which are the source itself and its neighbors (line 13), and, in the case of $j = i$, nodes $v \in V_i$ with a lower value of $\sigma(v)$ (line 15). Concluding the creation of a new edge, we append its source and target to the list of targetNodes (line 23).

3.3. **Analysis of the algorithm.** Based on the observations in the previous section, we prove the correctness of Algorithm 1 and analyze its running time in the following.

Observation 1. *Algorithm 1 generates valid core structures for the maximum number of intra-shell edges, i. e., $m_{ii} = i \cdot n_i - (i^2 + i)/2$ for $1 \leq i \leq k$.*

Proof. Let $m = i \cdot n_i - (i^2 + i)/2$. A node is removed from sourceNodes if either its out-degree is equal to i or it is connected to all nodes with a higher value of σ_i. If sourceNodes is empty we have inserted $(n_i - (i+1)) \cdot i + (i+1) \cdot i/2 = m$ edges (see Equation 4). □

Based on this observation, Lemmas 3.1 and 3.2 prove the correctness of Algorithm 1 inductively.

Lemma 3.1. *Given a matrix M belonging to a valid core fingerprint and a valid subgraph $G[V_k \cup \cdots \cup V_{i+1}]$, the element generation phase constructs the subgraph $G[V_k \cup \cdots \cup V_i]$ such that M is obeyed and all nodes $u \in V_\ell$ have* coreness $(u) \leq \ell$, *for all $i \leq \ell \leq k$.*

Proof. Let $j = i$. Lines 15 and 18 guarantee that σ_i is a valid removal order. Thus all nodes $v \in V_i$ have coreness $(v) \leq i$ and the coreness of all other nodes remains unchanged. Due to Observation 1 the upper bounds in Lemma 2.3 can be attained, thus any valid m_{ii} can be realized.

Now let $j > i$. Analogously, requiring $\text{outdeg}(v) \leq i$ preserves the removal order and thus a coreness of i or less for nodes in V_i. Again, the coreness of all other nodes remains unchanged, and the upper bound in Lemma 2.3 can be attained. □

The above lemma shows that the element generation phase fits in all nodes and edges required by the fingerprint and grants to each node a coreness equal to or less than the required value. We are left to prove that the rewiring phase refines the edge set such that equality holds.

Lemma 3.2. *Given a matrix M belonging to a valid core fingerprint and a valid subgraph $G[V_k \cup \cdots \cup V_{i+1}]$. If* coreness $(v) \leq i$ *holds for all $v \in V_i$, then the rewiring phase moves edges such that the subgraph $G[V_k \cup \cdots \cup V_i]$ is valid, i. e., M is obeyed, and all nodes $u \in V_\ell$ have* coreness $(u) = \ell$, *for all $i \leq \ell \leq k$.*

Proof. We have to proof that the list poorNodes defined in line 27 is empty when the algorithm terminates. Suppose there exists at least one node $v \in$ poorNodes. Since $\deg v < i$, clearly $\text{coreness}(v) < i$. Then, the list richNodes is not empty since otherwise all nodes $u \in V_i$ have $\deg u \leq i$ contradicting Lemma 2.3. Let $w \in$ richNodes, i. e., $w \in V_i$ and $\deg(w) > i$. Since $\deg(w) > \deg(v)$, the set of pivot candidates $C = \text{neighbors}(w) \setminus \text{neighbors}(v)$ is not empty. Choosing $c \in C$, the new set of edges $E' = E \setminus \{\{w, c\}\} \cup \{\{v, c\}\}$ still obeys M, decrements $\deg(w)$ and increments $\deg(v)$, increasing $\text{coreness}(v)$ by at most one.

Thus, the rewiring phase maintains the invariant. Furthermore, due to the strict increase and decrease of $\deg(v)$ and $\deg(w)$, respectively, $|$poorNodes$|$ strictly decreases to 0, which terminates the algorithm. □

By induction, Lemmas 3.1 and 3.2 yield that Algorithm 1 constructs a graph in accordance with M and V_i, $0 \leq i \leq k$, since the base case, i. e., the empty graph, is trivial.

In terms of running time the crucial parts of the algorithm are the updates and random accesses of the lists sourceNodes, targetNodes, poorNodes, and richNodes, and the creation of the target candidate and pivot candidate lists (lines 12–15 and 32). We use array-backed lists to guarantee constant-time access

to random elements. When we remove an element e we fill its position with the last element of the list, avoiding moving all successive elements of e. Since we only have random access to the lists, preserving their orders is not required.

Lemma 3.3. *The asymptotic running time of Algorithm 1 is bounded by* $O((m^2 + n^2k)\log(n))$.

Proof. The runtime of the element generation phase is dominated by the assembling of target candidates in lines 12–15. Building a decision tree for the nodes to be remove in $O(n \log n)$ time, based on the ordering σ, we can prune list C in time $O(m \log n + n \log n)$ per edge, which dominates lines 3 to 25.

The running time of the rewiring phase is dominated by determining the list of pivot candidates in line 32 using $O(n \log n)$ time per rewiring. The total number of rewirings is bounded by $n \cdot k$. This dominates lines 27 to 38 as well as the element generation phase and all peripheral steps. Assuming the graph is connected, in total, both phases sum up to a running time of $O((m^2 + n^2k)\log(n))$. □

Since real-world networks seldom exhibit pathologic characteristics, we replaced the eager computation of the candidate list in lines 12–15 by a *lazy* selection from TARGETNODES[i] that is repeated until a valid t has been drawn. Clearly this does not improve worst-case running time but works faster for virtually all applications.

We performed our experiments on a recent standard PC, running SUSE Linux 10.2 with an implementation in Java. Absolute running times ranged between 100 and 500 milliseconds for the AS network which is comparable to BRITE. The running time of Inet is in the order of minutes. See Section 4.1 for the description of these generators.

3.4. **Refinements.** Although the core fingerprint is the prime characteristic we focus on in this work, together with the inclusion of a preferential attachment mechanism, a number of potentially describing features of a network exist. In this section, we briefly discuss other relevant features, that can easily be integrated in our generator.

Connectivity is a very basic characteristic of a network, boiling down to the number of connected components. Building upon the core decomposition, this can be refined to the number of connected components *per shell*. While the whole graph or even the i-core can be connected, the i-shell can still have several disconnected components. If this is not desired, the user can specify the number and the sizes of connected components. The generator will then first create a spanning forest, where each tree is the seed of a component, and mark these edges as not rewirable. Note that requiring a specific set of connected components restricts the set of valid shell-connectivity matrices. However, this can be resolved by allowing the number of edges or the number and sizes of connected components to slightly deviate from the predefined values, depending on the user's interests.

Returning our focus to the degree distribution, the approach described in Section 3.2, depending on not a single parameter, can clearly be further elaborated. We tested two variants of our implementation of preferential attachment. In the first variant, we require the degree distributions of each shell as an input. Based on these we then prefill the array TARGETNODES[i] in line 7 with the nodes in V_i, using the exact multiplicities as given by the degree distribution and an ordering analogous to σ. This approach clearly biases the preferential attachment process towards the desired degree distribution (see Figure 4). Alternatively, we can solely

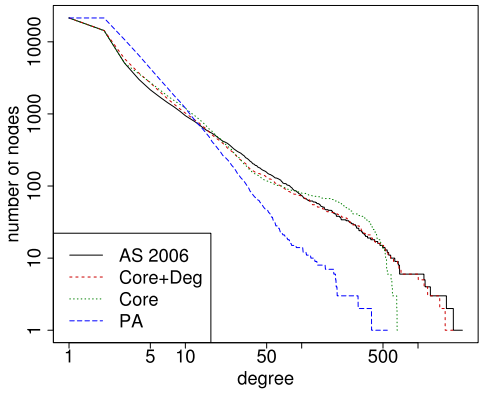


FIGURE 4. The number of nodes with degree at least d for the AS network, the original, and the refined Core generator for January 2006. A graph generated by preferential attachment of approximately the same size is shown for comparison.

	AS 2002-01	AS 2006-01	AS 2007-07
Number of Nodes	12,485	21,419	25,787
Number of Edges	25,980	45,638	53,014

TABLE 1. Sizes of the AS network snapshots.

rely on a post-processing step. In this case we can completely abandon preferential attachment and simply apply a sequence of rewirings (Lemma 2.1) and swappings (Lemma 2.2) in order to approach a given degree distribution. Although both these techniques yielded very good results, we exclude them from further evaluation, due to their requiring rather specific parameters in addition to the core fingerprint.

4. **Modeling the AS network.** An important application of a core-aware network generator is the simulation of the Internet at the AS level. In this section we compare networks generated by our method and established topology generators with three exemplary snapshot of the real AS network at the router level taken by the Oregon Routeviews project [23] at midnight on January 1, 2002 (oix-full-snapshot-2002-01-01-0000), on January 1, 2006 (oix-full-snapshot-2006-01-01-0000), and on July 1, 2007 (oix-full-snapshot-2007-07-01-0000). Table 1 shows the sizes of these graphs.

4.1. **Topology generators.** The first methods to generate networks with Internet-like structure date back to the 1990s and a multitude of techniques has been proposed since then. Among the most popular and widely used tools we have chosen Inet-3.0 [17] and BRITE [20] for our comparison since these are commonly included in other studies which cover a broader range of existing models [19, 17]. Although nem [18] also seems promising we do not take it into account because of its limitation to networks not greater than 4000 nodes.

The *Internet topology generator Inet* [17] generates an AS-level representation of the Internet. Its developers claim that *"it generates random networks with characteristics similar to those of the Internet from November 1997 to February 2002, and beyond"*. Basically, Inet generates networks with a degree distribution which fits

to one of the power laws originally found by Faloutsos et al. [14], namely that the frequency of nodes with degree d is proportional to d raised to a power of a constant α: $f(d) \propto d^{\alpha}$. Since this law does not cover all nodes and in order to match other relevant properties as well, optimizations for various specific conditions were added to the original procedure over time. The complete generation method is explained in [17]. Since the procedures of Inet are already customized to AS networks, only a small number of input parameters can be specified: the total number of nodes, the fraction of degree-one nodes, and the size of the square used for node placement.

The *Boston university Representative Internet Topology gEnerator BRITE* [20] can generate networks for different levels of the Internet topology. Beside this, it offers various other options to customize the generation procedure.

Drawing area. The nodes of the generated topology are distributed in a square of a certain size.

Node distribution. In the drawing area, nodes are either distributed uniformly at random or Pareto.

Outgoing links. New nodes are connected with a specific number of outgoing links to other, already existing nodes.

Connectivity. The neighborhood of a node is selected based on certain guidelines such as geometric locality, preferential attachment, or a combination of both.

Procedure. Nodes can either be placed before the addition of edges or in an incremental fashion. In the latter case each new node introduces a number of new edges that can only connect to already existing nodes.

4.2. **Characteristics.** In [17], an extensive collection of characteristics is evaluated that judge the fitness of a generated graph with respect to its real world counterpart. We repeated this evaluation for a representative selection of these properties with a focus on the assessment of the core generator. In the following, we summarize the properties we employed in our analysis.

General statistics. To see how well the generated networks fit to the most obvious characteristics we computed some basic properties: the number of edges, the minimum and the maximum degree. Note that all models strictly meet the given number of nodes, so the number of edges corresponds to density and average degree.

Cores. The core decomposition is a significant structural property of an AS network. We compare not only the core number but the extensive core fingerprint.

Clustering coefficient. The clustering coefficient is a measure for the local density around a node. It counts how many of a node's pairs of neighbors are themselves adjacent. These values are averaged to get a single measure for the network. Closely related characteristics are the numbers of triangles and triples and the transitivity [24].

Path length. We compare two properties based on path length: *characteristic path length*, which is the average of the distances of all node pairs and average *eccentricity*. The eccentricity of a node is its maximum distance to all other nodes. Average eccentricity then is the average of all nodes eccentricities.

Frequency versus degree. One of the classic power laws found by Faloutsos et al. [14] is $f(d) \propto d^{\alpha}$, that is, the frequency of nodes with degree d, is proportional to d raised to a power of a constant α. Since this power law does not hold for nearly

	AS 2002-01	Core	BRITE	Inet
Number of Nodes	12,485	12,485	12,485	12,485
Number of Edges	25,980	25,980	24,967	27,494
Minimum Degree	1	1	2	1
Maximum Degree	2,538	644	302	2,154
Core Number	20	20	2	9
Number of Triples	7,258,817	3,140,777	347,443	6,821,628
Number of Triangles	22,832	17,272	157	11,144
Transitivity	0.009	0.016	0.001	0.005
Clustering Coeff.	0.45	0.24	0.00	0.29
Avg. Path Length	3.63	3.69	5.09	3.29
Avg. Eccentricity	8.74	9.71	8.35	6.85

TABLE 2. Characteristics of the AS network of January 2002 and the three generators.

2% of the highest degree nodes, we use a modified version [8, 10]:

$$F(d) = \sum_{i>d} f(i) \propto d^{\alpha} \quad .$$

Size of k-neighborhood. Another power law identified in [14] is $\mathcal{N}(k) \propto k^{\beta}$, where $\mathcal{N}(k)$ is the sum over all nodes of their neighborhood sizes within distance k, i. e., $\mathcal{N}(k) = \sum_{u \in V} \sum_{v \in V} \mathrm{dist}_k(u, v)$, where

$$\mathrm{dist}_k(u, v) = \begin{cases} 1 & \text{, if } \mathrm{dist}(u, v) \leq k \\ 0 & \text{, otherwise.} \end{cases}$$

Note that this characteristic can also be measured as an average over all nodes, and it is also known as the *number of pairs within k hops*.

4.3. **Evaluation.** In the following, we detail the findings of our systematic evaluation. We gathered results on the three generators as described in Sections 3 and 4.1 and on the real AS network for all the properties listed in Section 4.2.

Based on the previous studies we set appropriate parameters for the generators Inet and BRITE. For Inet we have chosen the default input parameters except for the number of nodes and the random seed. As the results in [21] suggest, we have used preferential attachment and incremental growth for BRITE. Furthermore, we add two edges for each new node to fit the average degree of AS networks.

By construction, the numbers of nodes match the reference AS network, however, the numbers of edges already differ heavily. While the number of edges is only slightly lower for graphs generated by BRITE, and exactly fits the reference for core generator (called *Core* in the following), the edge set created by Inet is larger by one third.

The well-known phenomenon of highly connected hubs in the AS network accompanied by the power-law degree distribution is regarded as one of the most significant properties of the Internet. Inet reproduces these quite well, but overstates the maximum degree. In contrast, the degree distribution of Core oscillates around the reference but fails to produce high-degree nodes due to its lack of preferential attachment and the degree distribution of BRITE suggests that the preference of

	AS 2006-01	Core	BRITE	Inet
Number of Nodes	21,419	21,419	21,419	21,419
Number of Edges	45,638	45,638	42,835	58,069
Minimum Degree	1	1	2	1
Maximum Degree	2,408	662	411	3,572
Core Number	26	26	2	19
Number of Triples	12,161,105	5,631,122	637,716	30,643,658
Number of Triangles	46,256	36,052	177	75,770
Transitivity	0.011	0.019	0.001	0.007
Clustering Coeff.	0.38	0.17	0.00	0.53
Avg. Path Length	3.81	3.84	5.31	3.07
Avg. Eccentricity	8.52	10.36	8.63	6.45

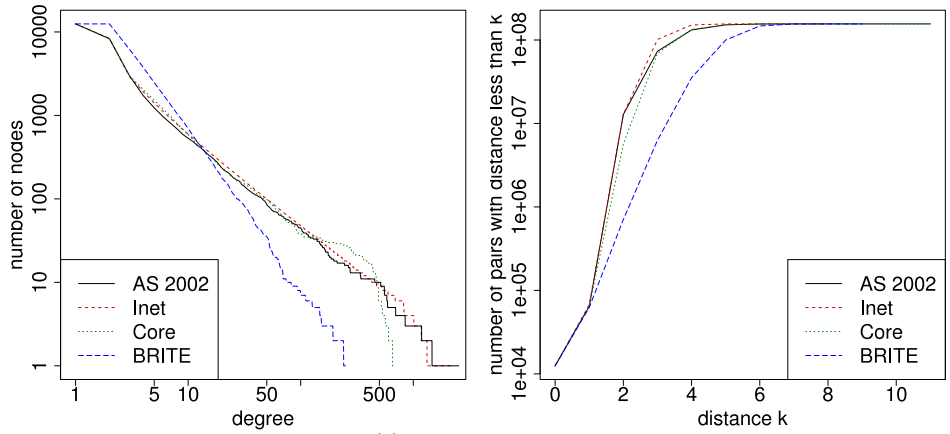
TABLE 3. Characteristics of the AS network of January 2006 and the three generators.

	AS 2007-07	Core	BRITE	Inet
Number of Nodes	25,787	25,787	25,787	25,787
Number of Edges	53,014	53,014	51,571	76,467
Minimum Degree	1	1	2	1
Maximum Degree	2,391	838	393	5,168
Core Number	22	22	2	26
Number of Triples	13,889,150	6,759,443	757,653	56,514,215
Number of Triangles	39,646	29,612	174	162,889
Transitivity	0.009	0.013	0.001	0.009
Clustering Coeff.	0.33	0.15	0.00	0.65
Avg. Path Length	3.89	3.92	5.39	2.99
Avg. Eccentricity	10.24	10.64	8.72	6.52

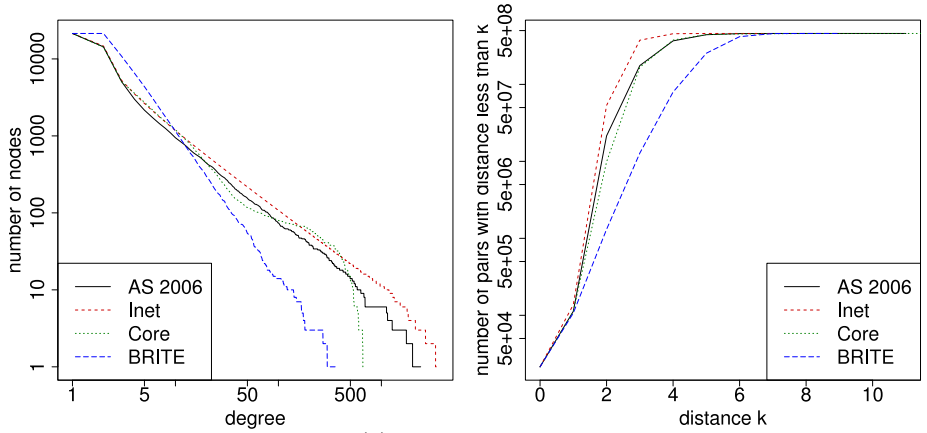
TABLE 4. Characteristics of the AS network of July 2007 and the three generators.

new nodes to connect to existing hubs is not strong enough either. These facts can be observed in Figure 5.

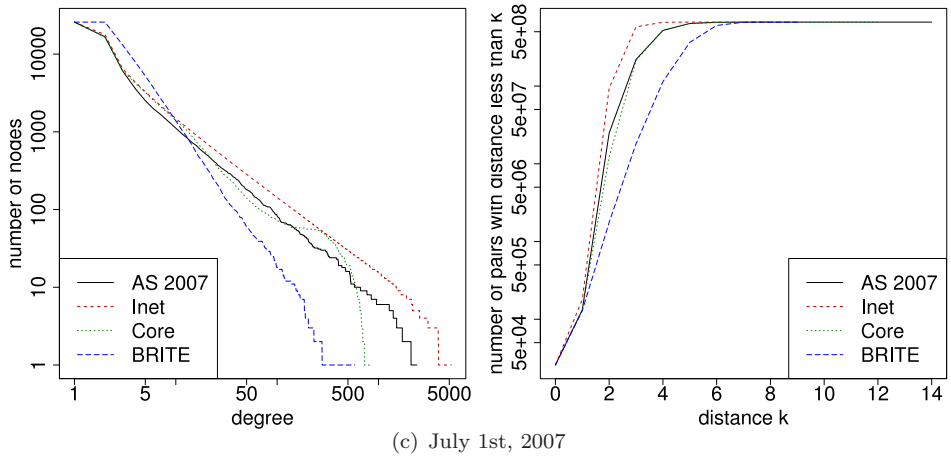
At a first glance, BRITE clearly fails to build up any kind of deep core structure (the core number is 2). The reason for this becomes evident from the incremental generation process of BRITE: the iterative addition of nodes incident to two new edges can simply be reversed, resulting in a valid removal sequence for the 2-core that ultimately yields an empty 3-core. Figure 6 plots both the number of nodes and the number of edges per k-core exemplary for January 2006. Inet builds up a decent core hierarchy but fails to attain a sufficient depth for earlier snapshots, obviously resulting in larger mid-level shells, in terms of both nodes and edges. However, as Inet seems to systematically overestimate the number of edges, for later snapshots, the core hierarchy becomes too deep. By construction, Core perfectly matches the reference. The plots in Figure 7 show the numbers of nodes and edges per k-shell, again exemplary for January 2006. They confirm the above observations and additionally grant an insight into the absolute numbers of elements per shell.



(a) January 1st, 2002



(b) January 1st, 2006



(c) July 1st, 2007

FIGURE 5. The number of nodes with a degree at least d (left) and the k-neighborhood for distances $k \in [0, 10]$ (right) for the AS network and the generated graphs for 2002, 2006, and July 2007.

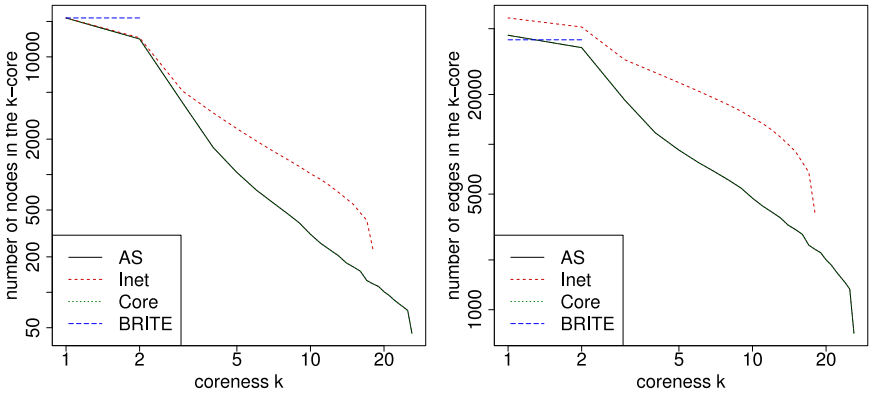


FIGURE 6. The numbers of nodes (left figure) and of edges (right figure) per k-core. Note that BRITE generates only nodes in the 2-core and that the lines of the AS 2006 and Core perfectly match by construction.

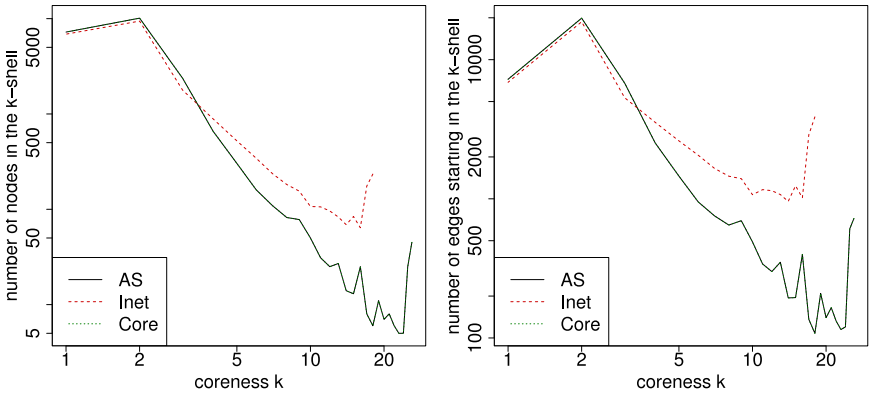


FIGURE 7. The numbers of nodes (left figure) and of edges (right figure) per k-shell (BRITE omitted). An edge is considered to belong to the k-shell if its endnode with smallest coreness has coreness k. Note that the lines of the AS 2006 and Core perfectly match by construction.

The shallow core structure created by BRITE is accompanied by a very low transitivity alongside a negligible number of triangles and a tiny clustering coefficient, suggesting that the BRITE graph is primarily composed of a set of paths of length two. The high average path length further corroborates this conjecture, since by virtue of preferential attachment hubs of high degree evolve, which, however, are interconnected via paths of length two by construction.

The absolute numbers of triples and triangles as well as the transitivity and the clustering coefficient are acceptable for both Core and Inet. The discrepancy of the latter generator from the reference can quite generally be explained by the increased number of edges. The behavior of Core with respect to these values is largely due to the absence of high-degree nodes, since, intuitively speaking, star-shaped structures

yield a high number of triples. The relatively high number of triangles thus yields an increased transitivity. The low clustering coefficient, however, suggests, that there is large number of nodes with a sparse direct neighborhood. Since, at the same time, Core exhibits a high number of triangles, the majority of these triangles is incident to nodes with higher degree.

Figure 5 depicts the size of the neighborhood within k hops (sum over all nodes). Note that the high average path length of BRITE mentioned earlier comes along with the slow growth of the neighborhood size. The low average path length and the low average eccentricity exhibited by Inet are, again, due to the large edge set. With respect to these values, Core excels. Both the average path length and the k-neighborhood practically match the reference.

5. Conclusion.

In the recent past, the core decomposition has been found to be a crucial characteristic of real world complex systems. In this paper we presented a novel algorithm for the generation of graphs that brings together the well-know concepts of k-cores and preferential attachment.

After scrutinize and clarifying how to specify the core fingerprint of a network by examining the inter-connectivity of each pair of shells. We employ this core fingerprint to introduce a simple and efficient algorithm for the generation of random graphs based on the core decomposition.

We exemplify the feasibility of our technique in a case study using the AS network of the Internet, comparing our generator to the established topology generators BRITE [20] and Inet [17]. Our results yield that our generator is highly suitable for the simulation of AS topologies, confirming the importance of the core decomposition. Moreover we show that BRITE largely fails to capture significant characteristics of the AS network, including its core structure, and that Inet roughly matches the reference except for its general tendency to be too densely connected. While our core generator and BRITE create a topology within seconds, a major drawback of Inet is its generation time of several minutes.

The high customizability of our rather generic core generator suggests several adaptations that can further increase the fitness to the specific peculiarities of the AS network. Such adaptations to special networks can be realized by employing a number of structural modifications such as swapping and rewiring without interfering with the core decomposition.

Acknowledgements. We would like to thank Jorge Busch for pointing out a problem in the previous version of Lemma 2.1, and for valuable comments and discussion on its resolution. Moreover, we would like to thank the referees very much for their valuable comments and suggestions.

REFERENCES

[1] Réka Albert and Albert-László Barabási, *Statistical mechanics of complex networks*, Reviews of Modern Physics, **74** (2002), 47–97.

[2] José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat and Alessandro Vespignani, *k-Core decomposition: A tool for the analysis of large scale internet graphs,* Electronically published at http://arxiv.org/abs/cs.NI/0511007, November 2005.

[3] José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat and Alessandro Vespignani, *Large scale networks fingerprinting and visualization using the k-core decomposition*, In "Advances in Neural Information Processing Systems 18," pages 41–50, MIT Press, 2006.

[4] Vladimir Batagelj and Ulrik Brandes, *Efficient generation of large random networks*, Physical Review E, (036113), 2005.

[5] Vladimir Batagelj and Matjaž Zaveršnik, *An $\mathcal{O}(m)$ algorithm for cores decomposition of networks*, Technical Report 798, IMFM Ljublana, Ljubljana, 2002.
[6] Vladimir Batagelj and Matjaž Zaveršnik, *Generalized cores*, Preprint 799, IMFM Ljublana, Ljubljana, 2002.
[7] Michael Baur, Marco Gaertler, Robert Görke, Marcus Krug and Dorothea Wagner, *Generating graphs with predefined k-core structure*, In "Proceedings of the European Conference of Complex Systems (ECCS'07)," October 2007.
[8] Tian Bu and Don Towsley, *On distinguishing between internet power law topology generators*, In "INFOCOM'02" [16].
[9] Shai Carmi, Shlomo Havlin, Scott Kirkpatrick, Yuval Shavitt and Eran Shir, *A model of internet topology using k-shell decomposition*, in "Proceedings of the National Academy of Science of the United States of America," **104** (2007), 11150–11154.
[10] Qian Chen, Hyunseok Chang, Ramesh Govindan and Sugih Jamin, *The origin of power laws in internet topologies revisited*, In INFOCOM'02 [16], 608–617.
[11] Sergey N. Dorogovtsev, Andrew V. Goldberg and Jose Ferreira F. Mendes, k-Core Organization of Complex Networks, *Physical Review Letters*, **96** (2006), 1–4 .
[12] Nicolas Ducheneaut, Nicholas Yee, Eric Nickell and Robert J. Moore, *Alone together?: Exploring the social dynamics of massively multiplayer online games*, In "Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)," 407–416, ACM Press, 2006.
[13] Paul Erdős and Alfred Rényi, *On random graphs I*, Publicationes Mathematicae Debrecen, **6** (1959), 290–297.
[14] Michalis Faloutsos, Petros Faloutsos and Christos Faloutsos, *On power-law relationships of the internet topology*, In "SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication," 251–262, ACM Press, 1999.
[15] Horst Gilbert, *Random graphs*, The Annals of Mathematical Statistics, **30** (1959), 1141–1144.
[16] "Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)," volume 1. IEEE Computer Society Press, 2002.
[17] Cheng Jin, Qian Chen and Sugih Jamin, "Inet Topology Generator," Technical Report CSE-TR-433, EECS Department, University of Michigan, 2000.
[18] Damien Magoni, *Nem: A software for network topology analysis and modeling*, In "Proceedings of the 10th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems," IEEE Computer Society, 2002.
[19] Damien Magoni and Jean Jacques Pansiot, *Analysis and comparison of internet topology generators*, In "Proceedings of the 2nd International IFIP-TC6 Networking Conference," 364–375. Springer, 2002.
[20] Alberto Medina, Anukool Lakhina, Ibrahim Matta and John Byers, *BRITE: An approach to universal topology generation*, In "Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems," 2001.
[21] Alberto Medina, Ibrahim Matta and John Byers, *On the origin of power laws in internet topologies,* Computer Communication Review, **30** (2000).
[22] Romualdo Pastor-Satorras and Alessandro Vespignani, "Evolution and Structure of the Internet: A Statistical Physics Approach," Cambridge University Press, 2004.
[23] University of Oregon Routeviews Project, http://www.routeviews.org/.
[24] Thomas Schank and Dorothea Wagner, *Approximating Clustering Coefficient and Transitivity,* Journal of Graph Algorithms and Applications, **9** (2005), 265–275.
[25] Stephen B. Seidman, Network Structure and Minimum Degree, *Social Networks*, **5** (1983), 269–287.
[26] Duncan J. Watts and Steven H. Strogatz, *Collective dynamics of "small-world" networks*, Nature, **393** (1998), 440–442.
[27] Stefan Wuchty and Eivind Almaas, *Peeling the yeast protein network,* Proteomics, **5** (2005), 444–449.

Received August 2007; revised March 2008.

E-mail address: baur@informatik.uni-karlsruhe.de;wagner@informatik.uni-karlsruhe.de
E-mail address: rgoerke@informatik.uni-karlsruhe.de;krug@informatik.uni-karlsruhe.de
E-mail address: gaertler@informatik.uni-karlsruhe.de