A program generating homogeneous random graphs with given weights ☆

L. Bogacz [a], Z. Burda [b], W. Janke [a,*], B. Waclaw [b]

[a] *Institut für Theoretische Physik, Universität Leipzig, Augustusplatz 10/11, 04109 Leipzig, Germany*
[b] *Institute of Physics, Jagellonian University, Reymonta 4, 30-059 Krakow, Poland*

Abstract

We present a program package to generate homogeneous random graphs with probabilities prescribed by the user. The statistical weight of a labeled graph α is given in the form $W(\alpha) = \prod_{i=1}^{N} p(q_i)$, where $p(q)$ is an arbitrary user function and q_i are the degrees of the graph nodes. The program can be used to generate two types of graphs (simple graphs and pseudo-graphs) from three types of ensembles (micro-canonical, canonical and grand-canonical).

Program summary

Title of the program: GraphGen
Catalogue identifier: ADWL
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/ADWL
Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland
Computer for which the program is designed and others on which it has been tested: PC, Alpha workstation
Operating systems or monitors under which the program has been tested: Linux, Unix, MS Windows XP
Programing language used: C
Memory required to execute with typical data: 300 k words for a graph with 1000 nodes and up to 50 000 links
No. of bits in a word: 32
No. of processor used: 1
Has the code been vectorized or parallelized: No
No. of lines in distributed program, including test data, etc.: 2253
No. of bytes in distributed program, including test data, etc.: 14 330
Distribution format: tar.gz

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
  E-mail address: janke@itp.uni-leipzig.de (W. Janke).

1. Introduction

Complex networks can be easily found in the real world. If the world objects are represented by nodes, and the interactions between them by edges then phone calls, computer connections, disease spread diagrams and human contacts are only a few examples of such networks. The recent improvements of computer technology has made the data acquisition easier and in consequence has led to a development of large databases of topology of observed networks. It turned out that completely independent networks often share common features, such as small world effect, fat tail in node degree distribution or large clustering. These effects caused that random graph theory, being mainly studied by pure mathematics so far, has attracted the attention of physicists and other natural sciences (for reviews see [1–3]).

There are two natural approaches to simulate networks as random graphs: diachronic [1–3] and synchronic [4–11]. In the first the network evolution in time is being investigated. One simulates the process of growth and checks how different mechanisms influence the emerging final graphs. In the latter approach a statistical ensemble of graphs is constructed and methods of statistical mechanics are applied. Each graph has a weight determining the probability of its occurrence during random sampling. The emergence of real networks usually is a complex process and computer simulations require the application of both approaches together. For example, the Internet is still a growing network, but its older parts also evolve.

The program package we describe in this paper uses the synchronic approach, which is a natural extension of Erdős and Rényi ideas [12,13]. A statistical ensemble of graphs is built by assigning a weight to each labeled graph in the given set of graphs. The weights can be chosen arbitrary. As an illustration, in the program we chose them to depend on degrees of individual nodes. If more complicated weights are needed, slight modifications of the program source code are required.

The program package can be used to generate graphs which mimic scale-free networks, i.e., networks with power-law degree distribution [14] or, in general, any desired degree distribution. One crucial point must be explained here. For finite graph size all methods to generate networks with fat-tailed degree distributions introduce a cut-off effect, since it is a property of networks themselves [15–17]. The same is true for the program presented in this paper and there is no way to get rid of this effect unless one introduces self- and multiple-connections or internode correlations.

The rest of this paper is organized as follows. In Section 2 we present definitions of graphs, statistical ensembles of graphs and partition functions of the models presented in the paper. Section 3 contains the description of the method used to generate graphs. In the final section, we outline the program compilation and usage.

2. Definitions

Let us start with basic definitions. A graph is a set of N nodes (vertices) connected by L edges (links) (see the example in Fig. 1). The edges can be directed or undirected, but in this paper we constrain ourselves only to graphs with undirected edges. Graphs without multiple connected or self-connected nodes will be called simple graphs (or graphs). Graphs containing multiple-connected or self-connected nodes will be denoted pseudographs or degenerated graphs. A graph does not need to be connected.

Both simple graphs and pseudographs can be represented by an adjacency matrix. For graphs with N nodes this is an $N \times N$ matrix with elements A_{ij} equal to the number of edges connecting node i and node j (for $i \neq j$). Diagonal elements A_{ii} count twice the number of self-connecting edges attached to node i, because we count each endpoint of link once. For example, the adjacency matrix of the graph shown in Fig. 1 has the following form:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 1 & 2 & 0 \end{pmatrix}. \tag{1}$$

The adjacency matrix is symmetric for any type of graph with undirected edges. Additionally, for simple graphs all diagonal elements are zero, and all other elements are zero or one. The sum of elements in the ith row (or in the ith column) gives the degree (the order) of the vertex i, i.e., the number of edges connected to that vertex.

A statistical ensemble of graphs is defined by ascribing a statistical weight to every graph in a given set. Among many possible choices we have defined and implemented in the program three sets of graphs [11]:

1. The *canonical* ensemble consists of all labeled graphs with fixed number of nodes N and edges L. It is a generalization of well known Erdős and Rényi graphs, where one connects N nodes by L edges chosen at random from all possibilities.
2. The *grand-canonical* ensemble is the ensemble of all labeled graphs with fixed number of nodes N. The number of edges L is varying. This is a generalization of the so-called binomial model, also introduced by Erdős and Rényi [12].
3. The *micro-canonical* ensemble consists of all labeled graphs with fixed degree of nodes given by a set of numbers $q_1, \ldots, q_N$.

Each of those ensembles can exist in two versions, consisting of only simple graphs or also pseudographs. By fully labeled (called for simplicity labeled) graph we mean a graph with labeled nodes and edges. Each edge has two labels, attached to two endpoints (see Fig. 1). Graphs which are identical in the sense of shape are not necessarily
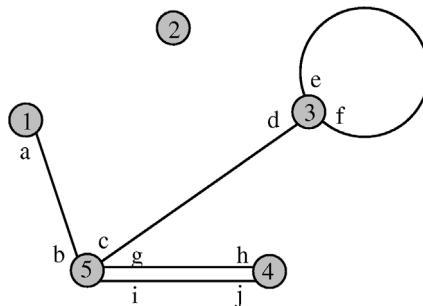

Fig. 1. An example of a graph with $N = 5$ nodes and $L = 5$ links. Positions of vertices in the picture are meaningless. The only information which matters is connectivity.
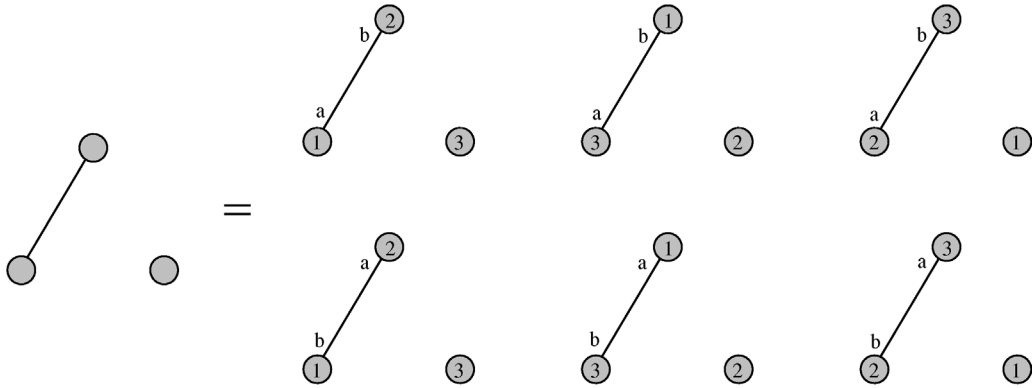
Fig. 2. The unlabeled graph on the left corresponds to the six possible labeled graphs on the right.

identically labeled graphs. Consider for example the graph shown in Fig. 2. This unlabeled graph has 6 different realizations as labeled graph, shown on the right hand side of Fig. 2.

The weight of each graph in the ensemble is defined in two steps. First we introduce a uniform configurational weight $1/(N!(2L)!)$ for each labeled graph. This weight is to compensate the number of permutations of indices. However, we will see below that for graphs possessing special symmetries the number of distinct labeled graphs is smaller than $N!(2L)!$ and therefore some residual factor remains. The partition function for the canonical ensemble of graphs with uniform measure is defined as:

$$Z_u(N, L) = \sum_{\alpha' \in flg(N,L)} \frac{1}{N!(2L)!} = \sum_{\alpha \in g(N,L)} w(\alpha), \tag{2}$$

where $flg(N, L)$ is the set of all fully labeled graphs with given number of nodes N and edges L and $g(N, L)$ denotes the set of all unlabeled graphs of size N and L. The weight function $w(\alpha)$ is defined in such a way that $w(\alpha)N!(2L)!$ is the total number of fully labeled graphs corresponding to the unlabeled graph α. If one considers only simple graphs the edge labeling can be abandoned. In this case the edge position is uniquely determined by two nodes at its endpoints. The $1/(2L)!$ factor cancels all possible edge relabelings, so exactly the same model can be defined when one replaces the uniform measure $1/(N!(2L)!)$ by $1/N!$ and does not introduce edge labels. The $Z_u(N, L)$ function defined above is just the partition function of the Erdős–Rényi model [12].

On the basis of the canonical partition function $Z_u(N, L)$, we define the partition function for the grand-canonical ensemble [11]:

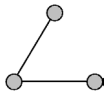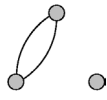$$Z_u(N, \mu) = \sum_L \exp(-\mu L) Z_u(N, L) = \sum_L \exp(-\mu L) \sum_{\alpha \in g(N,L)} w(\alpha), \tag{3}$$
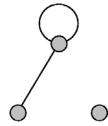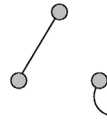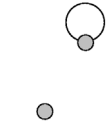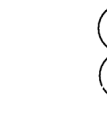
where μ can be interpreted as chemical potential for edges. Defining p as $\frac{p}{1-p} \equiv \exp(-\mu)$ one realizes that $Z_u(N, \mu)$ is the partition function for the binomial model. The partition function for the micro-canonical ensemble with given node degree sequence $q_1, q_2, \ldots, q_N$ can be defined as:

$$Z_u\big(N, \{q_i\}\big) = \sum_{\alpha \in g(N,L)} \left(\prod_{i=1}^{N} \delta\big(q_i(\alpha) - q_i\big)\right) w(\alpha), \tag{4}$$

where $\delta(x) = 1$ if $x = 0$ and zero otherwise. The $q_i(\alpha)$ gives the degree of the ith vertex of graph α. Consider as an example the canonical ensemble of graphs with $N = 3$ nodes and $L = 2$ edges. There are 6 possible unlabeled graphs, shown in Table 1. For each graph the number of corresponding labeled graphs, the uniform weight and the normalized occurrence probabilities $p(\alpha) = w(\alpha)/\sum_\beta w(\beta)$ are also shown.

Table 1
Number of possible labelings of graphs with $N = 3$, $L = 2$, from the canonical ensemble, their weights $w(\alpha)$ and normalized probabilities $p(\alpha)$ for graphs occurrence: $p(\alpha) = w(\alpha)/\sum_\beta w(\beta)$

graph α						
#labelings	72	36	72	36	18	9
$w(\alpha)$	1/2	1/4	1/2	1/4	1/8	1/16
$p(\alpha)$	0.2963	0.1482	0.2963	0.1482	0.0741	0.0370

The uniform weight ($w(\alpha) = 1$, $\forall \alpha$) leads to networks with Poissonian degree distribution. In the real world one rather observes networks with fat tails. Therefore we introduce an additional functional weight $W(\alpha)$, which is defined as

$$W(\alpha) = \prod_{i=1}^{N} p(q_i), \tag{5}$$

where the $p(q_i)$ function depends on the degree q_i of ith graph's vertex. The $p(q)$ can be chosen to obtain desired properties of the statistical ensemble. For example, one can show (see, e.g., [6]) that for the canonical ensemble of graphs the choice $p(q) = q!\pi(q)$ leads to the average degree distribution $\pi(q)$ in the limit $N \to \infty$. Therefore, taking $\pi(q) \propto q^{-\gamma}$ we obtain scale-free networks.

The partition functions for canonical, grand-canonical and micro-canonical ensembles with additional weight $W(\alpha)$ are:

$$Z(N, L) = \sum_{\alpha' \in flg(N,L)} \frac{W(\alpha')}{N!(2L)!} = \sum_{\alpha \in g(N,L)} w(\alpha)W(\alpha), \tag{6}$$

$$Z(N, \mu) = \sum_{L} \exp(-\mu L) \sum_{\alpha \in g(N,L)} w(\alpha)W(\alpha), \tag{7}$$

$$Z(N, \{q_i\}) = \sum_{\alpha \in g(N,L)} \left(\prod_{i=1}^{N} \delta\big(q_i(\alpha) - q_i\big) \right) w(\alpha)W(\alpha). \tag{8}$$

Because of the chosen form, the functional weight $W(\alpha)$ has the same value for each graph taken from the micro-canonical ensemble. Thus it factorizes and has no influence on properties of the micro-canonical ensemble. However, in the general case when one defines a more complicated function $W(\alpha)$, for example dependent on the number of certain motives present in the graph, it will modify the relative weights of graphs also in the micro-canonical ensemble. To introduce such a function, modifications of the program code are required.

3. Methods

The purpose of the presented program is to generate graphs with probabilities proportional to their statistical weights. Unfortunately there is no efficient algorithm which would be able to pick up an element from a large set with given probability. The naive algorithm which would pick up a random element and then accept or reject it with probability proportional to its weight would be very inefficient because of low acceptance rates. Therefore we use instead a Markov chain Monte Carlo technique, known from simulations of physical systems [18,19].

We construct a guided random walk in the configuration space of graphs. In each step, the program recursively generates a new graph α_{t+1} by modification of the current one α_t. In this way we obtain a Markov chain of configurations $\alpha_0 \to \alpha_1 \to \alpha_2 \to \cdots$. The chain is determined by the transition probabilities matrix $P(\alpha \to \beta)$ between any pair of graphs α, β in the ensemble, and the initial configuration. If the process is ergodic (which roughly means that all configurations are accessible) and if the probabilities fulfill the detailed balance condition:

$$W(\alpha)P(\alpha \to \beta) = W(\beta)P(\beta \to \alpha), \tag{9}$$

where $W(\alpha)$ is the weight of graph α, then the frequencies of graph occurrence approach the distribution $W(\alpha)/Z$ as the number of steps goes to infinity. In the program presented here, $P(\alpha \to \beta)$ is chosen as:

$$P(\alpha \to \beta) = \min\left\{1, \frac{W(\beta)}{W(\alpha)}\right\}, \tag{10}$$

which is known as Metropolis algorithm [20]. Depending on the considered graph ensemble we propose as elementary move one of the three transformations described below.

The first graph transformation called "T-move" is used to modify graphs belonging to the canonical ensemble. First, one node j and one edge $i \to k$ are chosen at random. Then we rewire the edge to $i \to j$ which means that the edge is detached from its endpoint k and attached to j (see Fig. 3). The total number of edges L is thus conserved but the degrees of the vertices k and j are changed: $q_k \to q_k - 1$, $q_j \to q_j + 1$. The probability for accepting the transformation is given by formula (10) as

$$P_a(\alpha \to \beta) = \min\left\{1, \frac{p(q_k - 1)p(q_j + 1)}{p(q_k)p(q_j)}\right\}, \tag{11}$$

where we explicitly used the form of the functional weight given by (5).

The second graph transformation which we consider is used to modify graphs belonging to the grand-canonical ensemble. For this ensemble we introduce two reciprocal transformations—addition and deletion of a link. Both of them preserve the number of nodes in the graph but change the number of edges (see Fig. 3). The decision which of those two is used in each elementary step is taken at random with probability $1/2$.
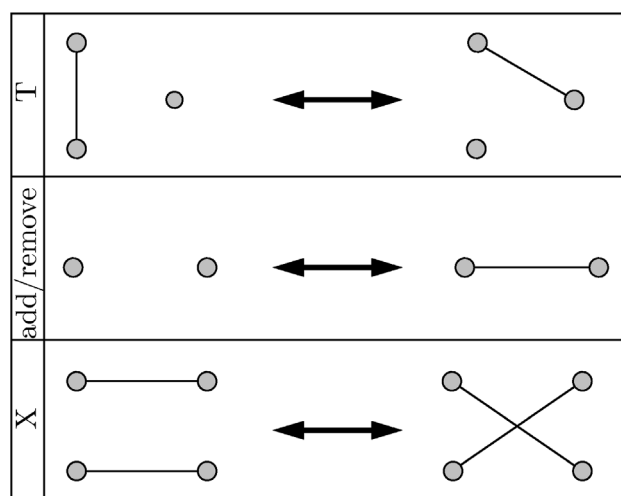

Fig. 3. Three types of graph modification used for generating graphs from canonical (T-move), grand-canonical (add/remove) and micro-canonical (X-move) ensembles.

As it was shown in [11], the probabilities of accepting addition and removal of a link are respectively:

$$P_a(\alpha \to \beta) = \min\left\{1, \exp(-\mu)\frac{N^2}{2(L(\alpha)+1)}\frac{W(\beta)}{W(\alpha)}\right\}, \tag{12}$$

and

$$P_a(\beta \to \alpha) = \min\left\{1, \exp(+\mu)\frac{2L(\beta)}{N^2}\frac{W(\alpha)}{W(\beta)}\right\}. \tag{13}$$

The last transformation called "X-move" is used to modify graphs from the micro-canonical ensemble [21]. First, two links l_1, l_2 are chosen randomly from all existing edges. Assume that l_1 connects vertices i_a, i_b and l_2 connects j_a, j_b. Next we exchange their endpoints so that l_1, l_2 point onto j_b, i_b, respectively. The degrees of all four nodes remain unchanged (see Fig. 3). The probability of accepting the move is equal to one, because the weights of all labeled graphs in the micro-canonical ensemble are identical.

If we want to generate only simple graphs, additional constraints must be introduced: we reject all moves leading to self- or multiple-connections. This does not change the probabilities of graph occurrences but only restricts the configuration space to what we need.

Because of the chosen graph generation method, each simulation should start from a "thermalization" sequence. Graphs generated during this sequence are not saved and no measurements are made. This is necessary for the graph occurrence probabilities to approach the proper distribution resulting from the weight function since we usually start from a graph which does not need to be "typical" in the given ensemble. The length of the "thermalization" sequence depends on the chosen ensemble, graph size and weight function. To estimate this length one may look at one particular property of a graph like degree distribution and check how many steps are needed to obtain the expected shape, using χ^2 function calculated for theoretical and measured degree distribution. Starting from one particular configuration, e.g., a Poissonian random graph, one has to wait until $\chi^2 \approx 1$. One can use the *degdist* program, included in the package, to generate node degree distributions for different lengths of the "thermalization" sequence. Comparing those with theoretical distributions and calculating χ^2 one may find an appropriate length of "thermalization" sequence.

The graphs generated by the program are correlated. The autocorrelation time depends on program parameters but also on the measured observable. As an example we report the autocorrelation time for the average clustering coefficient and for the total number of triangles in the graphs generated from the canonical ensemble. The autocorrelation time for graphs with unit weight, with $N = 100$ nodes and $L = 1000$ links, when a sweep contains 100 graph modification trials (see the SWEEP definition in the next section) is $t_{ac} \approx 3.9$ for the clustering coefficient and $t_{ac} \approx 4.9$ for the number of triangles. The correlation length grows approximately linearly with the number of graph links. To reduce this autocorrelation time simply increase the SWEEP parameter value.

4. Program description

4.1. Source code

We provide two programs for the generation of the described graph ensembles. Both of them are written in the "C" language. The first, *graphgen* is designed for generating graphs and saving them to a file. The user can make desired operations on the generated and saved sample. The second program called *degdist* demonstrates how to write a simple program calculating some quantities like the average degree distribution without saving the intermediate results to a file. Both programs use the same procedures, collected in a few separate files. The complete set of source files is presented below:

(1) `init.c`—set of functions used to build (initialize) a new graph. The initial graph is constructed by adding some links between randomly chosen nodes.

(2) `links.c`—functions used to perform operations on graphs. These are for example inserting or removing a link from a graph, choosing links or edges at random etc.

(3) `sweep.c`—functions performing three types of graph modification (T-move, addition/removal of links, X-move) used to modify the graphs from all ensembles.

(4) `save_load.c`—functions used for loading the initial graph from a file and saving generated graphs.

(5) `graphgen.c`—main function of program *graphgen*, responsible for reading parameters from the command line and management of the graph generation process.

(6) `degdist.c`—the program *degdist* that generates the histogram of degree distribution for a given ensemble of graphs.

First we describe the program *graphgen*. The source code has been divided into eight files: three header files (`def.h, functions.h, variables.h`) and five source code files (the above (1)–(5)). The `def.h` file should be edited before compilation. Constants defined therein determine the ensemble type used for the simulation, the weight function, the save and load file format and limits for the maximal number of nodes and edges. The complete list of options will be described in detail in Section 4.2. The execution and description of output data file is given in Sections 4.3 and 4.4.

The program *degdist* is described in Sections 4.5 and 4.6. The first one is devoted to compilation while the last one gives some informations about execution and output format.

4.2. Compilation of graphgen

To increase program efficiency, the decision which type of ensemble is going to be simulated is made before program compilation. Therefore before program compilation one should check and modify the definitions in the `def.h` file if necessary. The structure of the file corresponds to the definitions of macro constants in the "C" language. Each line has the following form:

#define NAME value

where NAME and value can be any pair from the list:

- ENSEMBLE [1, 2, or 3]: This value determines what type of ensemble the program uses to generate graphs. Use 1 for micro-canonical, 2 for canonical, and 3 for grand-canonical ensemble.
- GRAPH_TYPE [1, 2, or 3]: This determines if self- and multiple-connections are allowed. Use 1 to generate simple graphs only, 2 to generate multi-graphs with multiple-connections but without self-connections, and 3 to generate pseudographs with self- and multiple-connections.
- SAVE_FORMAT [1, 2, or 3]: This constant sets the default format for saving and loading a graph. Use 1 for full adjacency matrix format, 2 for short adjacency matrix format, and 3 for node order format (for a detailed description, see Section 4.4).
- WEIGHT_FUNCTION $p(q)$: The function $p(q)$ determines the contribution from one of the nodes to the total graph weight (5). Here q is an integer number equal to the node degree. The function $p(q)$ can be defined in any format consistent with the "C" language (for example $1.0/q$). It is used only if canonical or grand-canonical ensembles are chosen and the parameter RATIO_WEIGHT_FUNCTION is not defined.
- RATIO_WEIGHT_FUNCTION $p(q + 1)/p(q)$: In the calculation of transition probabilities (11), (12), (13) only the ratio $p(q + 1)/p(q)$ is used. Therefore it is better to define this ratio instead of the function $p(q)$. This reduces round-off errors and increases efficiency of the program (for example, use $q + 1$, when $p(q) = q!$, which avoids calculating the factorial). If the RATIO_WEIGHT_FUNCTION is defined then the WEIGHT_FUNCTION is ignored. The ratio can be defined in any format consistent with the "C" language.

- NV [integer number]: This sets the upper limit for the number of graph vertices and restricts the size of the graph to be generated or loaded. The larger the limit is, the more memory is required to run the program.
- NL [integer number]: As NV but for graph edges.
- SWEEP [integer number]: To obtain a new graph from the previous one, the program modifies the graph by a sequence of elementary transformations described in Section 3. The parameter SWEEP denotes the number of attempts of such elementary transformations.
- THERM [integer number]: This value determines the number of sweeps to be made at the beginning of a simulation without saving the generated graphs. Such starting sequence is necessary to "thermalize" the system.
- GRAPHS [integer number]: Determines how many graphs should be generated (saved or printed). After the starting sequence, the generated graphs are saved after every sweep.
- INITIAL_N_NODES [integer number]: Determines the default number of nodes in the initial graph.
- INITIAL_N_LINKS [integer number]: Determines the default number of links in the initial graph.
- NO_DRAND48: Add this definition if the pseudo-random number function drand48() is not defined on a computer where the program is going to be compiled. In that case the corresponding built-in function generating pseudo-random numbers will be used.

An example of the `def.h` file which can be used to generate 100 simple graphs from the canonical ensemble with weight function $p(q) = 1/(q + 1)$ is:

```
#define ENSEMBLE 2
#define GRAPH_TYPE 1
#define SAVE_FORMAT 3
#define WEIGHT_FUNCTION 1.0/(q + 1.0)
#define NV 3000
#define NL 3000
#define SWEEP 5000
#define THERM 100
#define GRAPHS 100
#define INITIAL_N_NODES 100
#define INITIAL_N_LINKS 100
```

The choice of ensemble, graphs type, limits for maximal number of nodes and edges as well as the weight function cannot be changed without program re-compilation. The other parameters like input/output format, simulation length etc. can be treated as defaults, since they can be overridden from the command line while starting the program. To make program compilation as easy as possible a `Makefile` is attached. Therefore if one has *make* installed, the compilation can be started by issuing the *make* command. The resulting executable is called `graphgen.exe`. Every time the file `def.h` is modified, a re-compilation is required before changes take effect.

4.3. Execution

To execute the program, type in the command line:

graphgen.exe [options]

where [options] can be one or more from the following list:

- *-h*: Help, i.e., print the list of all possible command line options.
- *-n* [integer number]: Number of nodes in the initial graph. This number is read from the input file if given.
- *-l* [integer number]: Number of links in the initial graph. This number is read from the input file if given.

- *-i* [inputfile]: The name of the file with the initial graph. If there is more than one graph saved in the file, only the first is used. If no input file is specified a random graph is generated as the initial graph.
- *-if* [1, 2, or 3]: Input file format. Use 1 for full adjacency matrix format, 2 for short adjacency matrix format, and 3 for node degrees format (the details are given below).
- *-o* outputfile: Name of the file to which generated graphs are saved. If no file is specified, the program uses standard output.
- *-of* [1, 2, or 3]: Output file format (the numbers have the same meaning as for the load format).
- *-r* [any long integer number]: Number used to initialize the pseudo-random number generator.
- *-g* GRAPHS: Number of graphs to be generated.
- *-s* SWEEP: Length of elementary sweep (i.e., number of elementary transformation attempts, see description in Section 4.2).
- *-t* THERM: Number of initial "thermalization" sweeps (see description in Section 4.2).

For example to generate 100 graphs and save their adjacency matrices to file `graphs.dat` type:

graphgen.exe -g 100 *-of* 1 *-o graphs.dat*

4.4. Output data file

The result of a single program run is the list of generated graphs printed or saved to a file (in turn without empty lines in between). The graphs can be saved in one of three possible formats. In each format the first two lines contain information about the actual number of nodes "nv" and the number of links "nl" in the graph. After these two lines the proper information about the graph structure is saved.

Using the first format, the graph structure is written as an adjacency matrix. Each line contains one row of the matrix. Matrix elements are separated by spaces. For example the output file for the graph in Fig. 1 has the form:

```
#nv = 5
#nl = 4
 0  0  0  0  1
 0  0  0  0  0
 0  0  2  0  1
 0  0  0  0  2
 1  0  1  2  0
```

In the second format, only non-zero adjacency matrix elements are saved. Each line in the output file contains information about position (row and column) and value of one non-zero matrix element. Because of the symmetry, it is enough to save information about the upper triangle of the matrix (column $\geqslant$ row). Thus the graph in Fig. 1 would be saved as:

```
#nv = 5
#nl = 4
 0  4  1
 2  2  2
 2  4  1
 3  4  2
```

If one uses the third format, only nodes degrees are saved. Usually this does not preserve the whole information required to reconstruct the graph but it may be useful, e.g., to construct histograms giving the degree distribution $\pi(q)$. Each line of the output file contains the order of one graph vertex. For the graph in Fig. 1 it is:

```
#nv = 5
#nl = 4
0
1
3
4
2
```

The same formats are used by the program to load the initial graph from a file.

4.5. Compilation of degdist

We now come to *degdist*. This is an independent program, which makes use of some functions defined in source files `init.c`, `links.c` and `sweep.c` described in previous sections. These files are included during the compilation by means of the #include directive. Thus the program can be compiled as a single file, without any special arrangements. One can also use attached `Makefile` and issue the command *make degdist*, which will generate the `degdist.exe` executable file.

Constants used in the program have the same meaning as it was already described. As a default all constants are defined in `degdist.c`, but for convenience there is an option to use the definition from the `def.h` file, exactly as it was in the *graphgen* program. The only one additional constant:

- HIST "name"

defines the name of the output file into which the histogram of the measured degree distribution is saved.

An example of constants definition is given below:

```
#define ENSEMBLE 2
#define GRAPH_TYPE 3
#define RATIO_WEIGHT_FUNCTION (q < 1)?1e + 20 : (q * (q + 1.)/(q + 3.))
#define SWEEP 500
#define THERM 10000
#define GRAPHS 100000
#define INITIAL_N_NODES 100
#define INITIAL_N_LINKS 100
#define NV 30000
#define NL 30000
#define HIST "test.dat"
#define NO_DRAND48
```

This allows to generate 10^5 pseudographs from the canonical ensemble with $N = 100$, $L = 100$ and Barabási–Albert degree distribution [14]:

$$\pi(q) = \frac{4}{q(q + 1)(q + 2)} \tag{14}$$

which leads to $p(q) = 4q!/(q(q + 1)(q + 2))$ and $p(q + 1)/p(q)$ as given by RATIO_WEIGHT_FUNCTION. Each graph is generated from the previous one after 500 attempted rewirings. The measured histogram of degree distribution averaged over the generated sample of the canonical ensemble is saved into `test.dat` file. One can check that this agrees well with the theoretical distribution $\pi(q)$ up to finite-size corrections (cut-off).

4.6. Execution and output data format of degdist

After compilation the program *degdist* can be executed simply from the command line without any arguments. For parameters given above, the running time is less than one minute on a modern PC. The result of a single run is one data file. Each line consists of three columns separated by tabulators: $q, \pi(q), \Delta\pi(q)$. Here $\pi(q)$ is estimated from measurements of the averaged degree distribution while $\Delta\pi(q)$ gives a rough estimation of the statistical error for this quantity and a given degree q. A typical set of data is presented below:

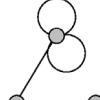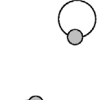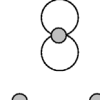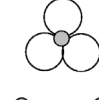|		
1	0.65392	0.00026
2	0.167699	0.00013
3	0.0686797	8.3e−005
4	0.0352373	5.9e−005
5	0.0205495	4.5e−005
6	0.0131624	3.6e−005
7	0.0089971	3e−005
8	0.006363	2.5e−005

. . .
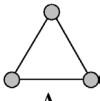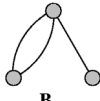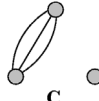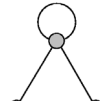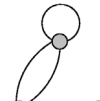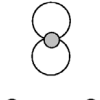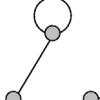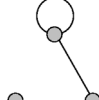
where . . . stands for the rest of the file. The $\pi(q)$ given in the second column are normalized such that $\sum_q \pi(q) = 1$. The program can also be compiled with constant **GRAPHS** set to **1** which means that only one graph is generated and $\pi(q)$ is the degree distribution for this particular graph.

5. Test run

The program *graphgen* has been tested for a number of systems. As an example we show in Table 2 the results of simulations of a canonical ensemble of pseudographs with $N = 3$ nodes, $L = 3$ links and the weight function

Table 2
Comparison of theoretically calculated frequencies of graph occurrences with those generated by the program, for the canonical ensemble with $N = 3$, $L = 3$. The weight function is $p(q) = q!(q + 1)^{-0.5}$. During the simulation 10^7 graphs were generated (with THERM = 100 and SWEEP = 50)

graph	A	B	C	D	E	F	G
theor.	$\frac{p(2)^3}{6} \to$ 0.0142	$\frac{p(1)p(2)p(3)}{2} \to$ 0.0675	$\frac{p(0)p(3)^2}{12} \to$ 0.0414	$\frac{p(1)^2 p(4)}{4} \to$ 0.0740	$\frac{p(1)p(2)p(3)}{2} \to$ 0.0675	$\frac{p(0)p(2)p(4)}{4} \to$ 0.1708	$\frac{p(2)^3}{8} \to$ 0.0106
simul.	0.0142(1)	0.0676(1)	0.0414(1)	0.0739(1)	0.0675(1)	0.1709(1)	0.0106(1)

graph	H	I	J	K	L	M	N
theor.	$\frac{p(1)^2 p(4)}{16} \to$ 0.0185	$\frac{p(1)p(2)p(3)}{4} \to$ 0.0338	$\frac{p(0)p(3)^2}{8} \to$ 0.0620	$\frac{p(0)p(1)p(5)}{8} \to$ 0.2388	$\frac{p(2)^3}{48} \to$ 0.0018	$\frac{p(0)p(2)p(4)}{16} \to$ 0.0427	$\frac{p(0)^2 p(6)}{96} \to$ 0.1563
simul.	0.0184(1)	0.0338(1)	0.0621(1)	0.2389(1)	0.0018(1)	0.0427(1)	0.1562(1)

$p(q) = q!(q + 1)^{-0.5}$. The number of 10^7 graphs have been generated (with THERM = 100 and SWEEP = 50). The comparison of graph frequencies calculated theoretically with those generated by the program shows perfect agreement.

The program package contains the example input file `in_graph.dat` and the example of output file `o_graph.dat`. In the input file a graph with $N = 10$, $L = 50$ is saved in the adjacency matrix format. The output file consists of a list of 20 graphs, saved in the short adjacency matrix format, generated by the following command:

> *graphgen.exe -g* 20 *-i in_graph.dat -if* 1 *-o o_graph.dat -of* 2

The program *degdist* has also been tested carefully. The file `test.dat` contains the degree distribution generated for the set-up given in Section 4.5 as an example. This was done by compiling and executing *degdist.exe* from command line without any arguments.

Acknowledgements

References

[1] R. Albert, A.-L. Barabási, Rev. Modern Phys. 74 (2002) 47.
[2] S.N. Dorogovtsev, J.F.F. Mendes, Adv. Phys. 51 (2002) 1079.
[3] M.E.J. Newman, SIAM Rev. 45 (2003) 167.
[4] Z. Burda, J.D. Correia, A. Krzywicki, Phys. Rev. E 64 (2001) 046118.
[5] J. Berg, M. Lässig, Phys. Rev. Lett. 89 (2002) 228701.
[6] S.N. Dorogovtsev, J.F.F. Mendes, A.N. Samukhin, Nucl. Phys. B 666 (2003) 396.
[7] Z. Burda, J. Jurkiewicz, A. Krzywicki, Physica A 344 (2004) 56.
[8] I. Farkas, I. Derenyi, G. Palla, T. Vicsek, in: Lecture Notes in Phys., vol. 650, Springer, Berlin, 2004, p. 163.
[9] J. Park, M.E.J. Newman, Phys. Rev. E 70 (2004) 066117.
[10] D.-S. Lee, K.-I. Goh, B. Kahng, D. Kim, Nucl. Phys. B 696 (2004) 351.
[11] L. Bogacz, Z. Burda, B. Waclaw, cond-mat/0502124.
[12] P. Erdős, A. Rényi, Publ. Math. Debrecen 6 (1959) 290; Publ. Math. Inst. Hung. Acad. Sci. 5 (1960) 17.
[13] B. Bollobás, Random Graphs, Academic Press, New York, 1985.
[14] R. Albert, A.-L. Barabási, Science 286 (1999) 509.
[15] Z. Burda, A. Krzywicki, Phys. Rev. E 67 (2003) 046118.
[16] M. Boguñá, R. Pastor-Satorras, A. Vespignani, Eur. Phys. J. B 38 (2004) 205.
[17] S.N. Dorogovtsev, J.F.F. Mendes, A.M. Povolotsky, A.N. Samukhin, cond-mat/0505193.
[18] D.P. Landau, K. Binder, Monte Carlo Simulations in Statistical Physics, Cambridge University Press, Cambridge, 2000.
[19] B.A. Berg, Markov Chain Monte Carlo Simulations and Their Statistical Analysis, World Scientific, Singapore, 2004.
[20] N.D. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, J. Chem. Phys. 21 (1953) 1087.
[21] S. Maslov, K. Sneppen, A. Zaliznyak, cond-mat/0205379.