Karlsruhe Reports in Informatics 2012,17
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

An Efficient Generator for Clustered Dynamic Random Networks

Robert Görke, Roland Kluge, Andrea Schumm,
Christian Staudt, and Dorothea Wagner

2012

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

An Efficient Generator for Clustered Dynamic Random Networks*

Robert Görke, Roland Kluge, Andrea Schumm, Christian Staudt, and Dorothea Wagner

Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany

Abstract. A *planted partition graph* is an Erdős-Rényi type random graph, where, based on a given partition of the vertex set, vertices in the same part are linked with a higher probability than vertices in different parts. Graphs of this type are frequently used to evaluate *graph clustering* algorithms, i.e., algorithms that seek to partition the vertex set of a graph into densely connected clusters. This is motivated by the possibility to compare the computed clustering to the planted partition or *ground-truth*, which can be used to assess clusterings independently of any particular optimization function. We propose a self-evident modification of this model to generate sequences of random graphs that are obtained by *atomic updates*, i.e., the deletion or insertion of an edge or vertex. The random process follows a dynamically changing ground-truth clustering that can be used to evaluate dynamic graph clustering algorithms. We give a theoretical justification of our model and show how the corresponding random process can be implemented efficiently.

1 Introduction

The broad variety of network structures we encounter in many fields of science and everyday life can mostly be modelled as *graphs*, where entities are mapped to vertices and the observed relationships to edges. It is not so clear how to subdivide the vertices of these graphs into clusters. In fact, no single answer to this question exists but a broad variety of approaches has been proposed in the past [1,2]. What all these measures agree on, is that clusters are characterized as densely connected subgraphs by some means or other. Frequently, the networks under consideration evolve over time, as vertices and links between them may appear or disappear. This can be either caused by random fluctuations or the split or merge of communities.

Even though real-world dynamic clustered instances with a reference clustering exist – that is, we know the clustering of the data in advance – there are several reasons why we are interested in generating artificial test data: First, we would like to produce graphs with a set of predefined properties such as the distribution of vertex degrees or size of the clusters, enabling us to examine the behavior of dynamic clustering algorithms under almost arbitrary circumstances. Second, real-world instances are not numerous and, most often, subject to confidentiality agreements.

Numerous results on random graphs exist [3]. One of the oldest and most fundamental models is the one introduced by Gilbert [4], in which each possible edge is present with uniform probability. These model can be altered in a straightforward way to incorporate a *planted partition*, i.e., a partition of the vertex set such that vertices in the same part are connected with a higher probability than vertices in different parts [5,6,7]. These graphs are frequently used to evaluate static graph clustering algorithms [1]; the big advantage of this approach is that the clustering obtained by an algorithm can be compared to the planted partition or *ground-truth clustering* used by the generator, which yields a possibility to assess clusterings independently of any particular quality measure. Well-known examples of random graphs based on this concept are the GN benchmark introduced by Girvan and Newman [8] and the relaxed cavemen graphs [9].

Prominent and fundamental models of social networks include small world networks [10] and the Barabási-Albert model [11]. The latter can be seen as a dynamic model for graph growth according to a preferential attachment process. Numerous variations thereof exist, most of which are targeted in capturing more accurately properties observed in real world social

* This work was partially supported by the DFG under grant WA 654/19-1 and WA 654/15-1

networks [12,13]. These models typically only simulate network growth and do not incorporate a known reference clustering. An exception is the model of Bagrow [14], where, starting from a graph generated according to Barabási-Albert, edges are randomly rewired to incorporate a given planted partition. The well-known LFR benchmark introduced by Lancichinetti and Fortunato is generated in a similar way [15]. While these approaches combine a reference partitioning with a more realistic degree distribution, the inherently dynamic process is lost. Other modifications of the planted partition model include the generalization to weighted [16] and bipartite [17] graphs, as well as hierarchical [18] and overlapping [19] reference clusterings. Aldecoa and Marín [20] propose to interpolate between two graphs with a strong clustering structure by rewiring edges at random. This process does not keep track of an explicit reference clustering over time, however, the assumption is that intermediate clusterings should have low distance to both the initial and the resulting clustering. Brandes and Mader [21] use as data for their experiments dynamic random graphs that are obtained by using exponential-family random graph models [22] for the generation of the initial graph and stochastic actor-oriented models [23] to model the evolution between two networks. Both steps rely on properties of real-world dynamic networks that are given as input.

Other models for dynamic graphs based on random evolution according to a given Markov chain include *edge-Markovian Dynamic Graphs* [24,25]. This model does not incorporate a reference clustering but uses two fixed parameters p and q that represent the *edge birth-rate* and *edge death rate* of each possible edge. In contrast to the model we consider, an arbitrary number of edge deletions and insertion can take place in each time step.

Our Contribution In this work, we augment the planted partition model by allowing *dynamic events*; edge and vertex events add or delete an edge or vertex, whereas cluster events split or merge clusters. More formally, we generate a time series of random graphs $G_0, \ldots, G_n$, where G_t emerges from G_{t-1} via *atomic updates*, i.e., the insertion or deletion of an edge or vertex. Over the whole generation process, the generator keeps track of a (dynamic) ground truth clustering. The probability of atomic events is chosen in a way that adheres to this clustering, without losing randomness. Graph growth/shrinkage and cluster dynamics can be simulated, steered by input parameters. Together with the benefit of the reference clustering, this can be used to thoroughly evaluate dynamic graph clustering algorithms, i.e., algorithms that incrementally update the calculated clustering as new node/edge events occur. A preliminary version of our generator is documented in our technical report [26] and the dissertation of one of the authors [27], and has been used in [28]. The new generator documented here differs fundamentally in the data structures used, which allows for faster practical and worst case running time, as well as linear space complexity. As the random model and parameters used are taken from the old generator, their description closely adheres to the technical report. Our generator is free for use and can be downloaded as Java software from our project page [1].

Notation Let $G = (V, E)$ be an undirected, unweighted, and simple graph, i.e., G is loopless and has no parallel edges[2]. If not otherwise stated, n and m will always denote the cardinality of the sets V and E, respectively. The *degree* $\deg v$ of a vertex v is the number of its adjacent vertices. The set of all possible undirected edges in G is denoted with $\binom{V}{2}$. For a given graph G its *complement graph* $\overline{G}$ is defined as $\overline{G} = (\overline{V} = V, \overline{E} = \binom{V}{2} \setminus E)$. The pairs of vertices in $\overline{E}$ are called *non-edges* of G and $\overline{m} := |\overline{E}|$.

A *clustering* $\mathcal{C} = \{C_1, \ldots, C_k\}$ is a partition of V where each of the C_i is non-empty. If not defined otherwise, the variable k will always refer to the number of clusters in $\mathcal{C}$. $\mathcal{C}(v)$ denotes the cluster in $\mathcal{C}$ that contains vertex v. For a cluster C the graph $G(C) = (C, E(C))$ is the vertex induced subgraph of C, where $E(C)$ are called *intracluster edges* of C. We identify a cluster C with the set of vertices it constitutes and with its vertex-induced subgraph of G. $m(C)$ is the

[1] http://i11www.iti.uni-karlsruhe.de/en/projects/spp1307/dyngen
[2] Throughout this work, we will only consider graphs with this property

number of edges in $G(C)$ and $\overline{m}(C) = \left|\binom{V(C)}{2} \setminus E(C)\right|$ is the number of *intracluster non-edges* of C. Edges having endpoints in two distinct clusters are called *intercluster edges*; the number of intercluster edges will be denoted with m_{inter}.

A *Markov chain* is a pair $M = (S, P)$, where S is a finite set of *states* and P a row stochastic matrix containing transition probabilities between the states. $C \subseteq S$ is *closed* if for all $i \in C$ and $j \in S \setminus C$ the transition probability from i to j is 0. M is *irreducible*, if there is no proper closed subset of S. We call a distribution vector w *stationary* if w is a left eigenvector of P.

2 Static Model

Gilbert's model on the generation of random graphs with uniform edge probability [4] can be easily modified to incorporate a planted partition [6,7]. The idea behind this random model, which we will call $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$, is that vertices in the same cluster should be linked with high probability p_{in}, whereas intercluster edges should be present with a lower probability p_{out}, i.e., we always assume $p_{\text{in}} > p_{\text{out}}$.

The parameter n denotes the number of vertices. Edges are added randomly according to the following process. The generation is based on a fixed *ground truth* clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ of the vertices. We chose to set p_{out} to a single value, whereas p_{in} is a list of length k: $p_{\text{in}} = \big(p_{\text{in}}(C_1), \ldots, p_{\text{in}}(C_k)\big)$. For two vertices u and v the probability for edge $e = \{u, v\}$ to exist in a graph created with $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ is called *edge probability* $p(e) = p(u, v)$, where

$$p(u,v) = \begin{cases} p_{\text{in}}(C_i) & u, v \in C_i \\ p_{\text{out}} & \text{otherwise} \end{cases}$$

The probability of a graph G according to this model is thus

$$p(G) = \prod_{e \in E} p(e) \cdot \prod_{e \in \overline{E}} (1 - p(e))$$

Our dynamic generator is strongly based on this concept, and the first graph in the generated sequence is a $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ graph. The number of clusters as well as a list of intracluster probabilities and the uniform intercluster probability p_{out} are input parameters. Cluster sizes can either be set manually or determined automatically by the generator. In the latter case, we choose the cluster of a vertex uniformly at random which entails a binomial distribution of the cluster sizes with mean $\frac{n}{k}$.

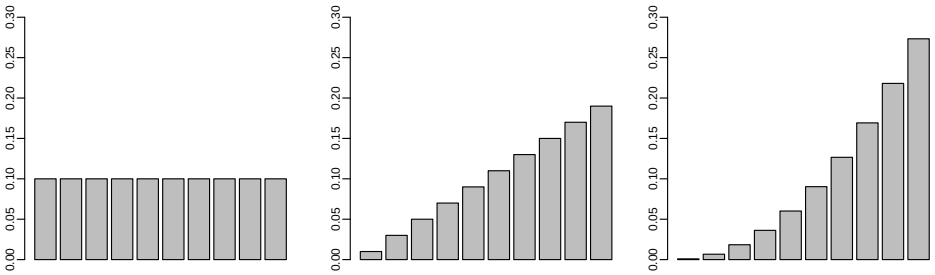
Furthermore we introduce a coefficient β which *skews* the binomial distribution as follows ($\beta = 1$ in the unskewed case): Each cluster C_i is assigned to a subinterval $\left[\frac{i-1}{k}, \frac{i}{k}\right)$ of $[0, 1)$. When searching for a cluster to add a new vertex to, we draw an integer $i \in [0, k-1]$. Now, we add the vertex to the cluster which is assigned to the surrounding interval of $\left(\frac{i}{k}\right)^{\beta}$. Examples for cluster size distributions with different values of β can be found in Figure 1.

3 Edge Dynamics

Neglecting cluster dynamics for the moment, we describe the random process we use for edge dynamics, along with some theoretical properties of the random sequence generated. We further give details on how this process is implemented in our generator, together with a worst case guarantee on the running time of edge events.

3.1 Associated Markov Chain and Distribution

At first glance, we would like to have a random process that triggers an edge operation, i.e., insertion or deletion, in each time step such that the relative frequency of a graph G in this



(a) $k = 10$, $\beta = 1.0$ (b) $k = 10$, $\beta = 0.5$ (c) $k = 10$, $\beta = 0.33$

Fig. 1: Expected fractions of $|V|$ in each cluster for $k = 10$, using different values of $\beta \leq 1.0$ for biased selection.

sequence follows its probability $p(G)$ in the $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ model. Unfortunately, such a random process does not have to exist in general: Consider for example the simple case that we have two vertices and the probability of the edge between these equals 0.1. Under these assumptions, the probability of the empty (complete) graph on two vertices equals 0.9 (0.1), respectively. On the other hand, there is only one possible edge operation in each state. Hence, each random sequence alternates between the states, which can never lead to the assumed probabilities.

There is however a simple random process that follows prescribed edge probabilities if we allow for repeated occurrences of a graph in the sequence. This process chooses in each step a pair of vertices u and v uniformly at random, deletes the edge $\{u, v\}$ if it exists and (re)inserts it with probability $p(u, v)$. There is a natural correspondence between this procedure and a Markov chain M' whose states represent all possible (labeled) graphs on n vertices. It is not hard to see that $G(n, p_{\text{in}}, p_{\text{out}})$ is the unique stationary distribution of this chain. Thus, if we choose our initial state according to this distribution, the expected relative frequency of a graph generated by this Markov chain follows $G(n, p_{\text{in}}, p_{\text{out}})$.

However, in the context of evaluating dynamic algorithms, we are typically interested in sequences of graphs such that consecutive graphs follow from each other by atomic changes. We therefore slightly modify M' such that each time step that does not change the graph is discarded and call the resulting Markov chain M. Let $P_E := \sum_{e \in E} \big(1 - p(e)\big)$ and $P_{\overline{E}} := \sum_{e \in \overline{E}} p(e)$. Then, the probability $p_{\text{del}}(u, v)$ that one step of M deletes an existing edge $\{u, v\}$ is

$$p_{\text{del}}(u, v) = \frac{\binom{n}{2} \cdot \big(1 - p(u, v)\big)}{\sum_{e \in E} \left[\binom{n}{2} \cdot \big(1 - p(e)\big)\right] + \sum_{e \in \overline{E}} \left[\binom{n}{2} \cdot p(e)\right]} = \frac{1 - p(u, v)}{P_E + P_{\overline{E}}}$$

and the probability $p_{\text{ins}}(u, v)$ of inserting a non-existing edge is

$$p_{\text{ins}}(u, v) = \frac{\binom{n}{2} \cdot p(u, v)}{\sum_{e \in E} \left[\binom{n}{2} \cdot \big(1 - p(e)\big)\right] + \sum_{e \in \overline{E}} \left[\binom{n}{2} \cdot p(e)\right]} = \frac{p(u, v)}{P_E + P_{\overline{E}}}.$$

Intuitively, we expect the relative frequency of unlikely states in the sequence generated by M to be slightly larger than in the sequence generated by M' and vice versa, as they are less often discarded. The following theorem makes this intuition precise.

Theorem 1. *If we choose the initial graph G_0 according to $\mathcal{G}(n, p_{in}, p_{out})$, the expected relative frequency of a graph $G = (V, E)$ in a sequence R generated by M is*

$$p(G) \cdot \left[\frac{\sum_{e \in E} \big(1 - p(e)\big) + \sum_{e \in \overline{E}} p(e)}{2 \cdot \sum_{e \in \binom{V}{2}} p(e) \big(1 - p(e)\big)} \right]$$

Proof. We use that M is derived from M' and look at the random sequence $R' = G'_1, \ldots, G'_{T'}$ with $T' \geq T$ that M' generates. We call an occurrence of a graph at time step $t \geq 1$ *valid*, if $G'_t \neq G'_{t-1}$. The probability that G has a valid occurrence at a certain time step t is then

$$p(G'_t{=}G) \cdot p(G'_{t-1}{\neq}G \mid G'_t{=}G) = p(G'_t{=}G) \cdot [1 - p(G'_{t-1}{=}G \mid G'_t{=}G)] \tag{1}$$

If we use that the probability of G at any time step equals $p(G)$, the probability that no edge in G is changed from $t-1$ to t equals

$$p(G'_{t-1}{=}G \mid G'_t = G) = \frac{p(G'_{t-1}{=}G)}{p(G'_t{=}G)} \cdot p(G'_t{=}G \mid G'_{t-1} = G) = \sum_{e \in E} \frac{p(e)}{\binom{n}{2}} + \sum_{e \in \overline{E}} \frac{1-p(e)}{\binom{n}{2}}$$

Hence, the right hand side of Equation 1 is equal to

$$p(G) \cdot \left[1 - \sum_{e \in E} \frac{p(e)}{\binom{n}{2}} - \sum_{e \in \overline{E}} \frac{1-p(e)}{\binom{n}{2}} \right] = p(G) \cdot \frac{1}{\binom{n}{2}} \cdot \left[\sum_{e \in E} \big(1 - p(e)\big) + \sum_{e \in \overline{E}} p(e) \right] \tag{2}$$

The probability that a time step contains a valid occurrence of an arbitrary graph is the sum over all pairs of vertices that the corresponding edge is flipped in this time step, which equals $2 \cdot \frac{1}{\binom{n}{2}} \cdot \sum_{e \in \binom{V}{2}} p(e)(1-p(e))$. Dividing the probability of a valid occurrence of G at time step t (Equation 2) by this probability yields the claim. □

Until now, we have assumed that the initial graph is chosen according to $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ and that the following time steps are obtained by mere edge dynamics. As motivated in the introduction, our generator also incorporates vertex and cluster dynamics, which we will introduce later on. The latter two types of dynamics disturb the probability distribution in a way that is difficult to analyze. However, as it is always possible to reach any graph from any other graph in a finite number of steps with a positive probability, M is irreducible. The expected relative frequency in Theorem 1 is therefore the only stationary distribution of M and we can expect the relative frequency of graphs to get close to this distribution after a sufficiently large number of edge operations following vertex or cluster events [29].

3.2 Data Structures and Implementation

After a brief description of how vertex pairs can be enumerated continuously, we introduce the dynamic data structures our generator builds upon. In the second part, we show how these data structures can be used to efficiently implement edge dynamics.

Enumerating Vertex Pairs To simplify the process of drawing random edges, we will use a bijection between pairs of vertices and integers between 0 and $\binom{|V|}{2}$ proposed by Batagelj and Brandes [30] to enumerate potential edges. Figure 2 intuitively illustrates this bijection by using the adjacency matrix of the graph. As we only consider undirected graphs, potential edges correspond to the entries below the diagonal. These entries can be enumerated by traversing this sub matrix likewise. For given vertices u and v, the index $e(u,v)$ can be obtained by

$$e(u,v) = \sum_{k=0}^{u-1} k + v = \frac{1}{2}(u-1)u + v$$

Vice versa, given the edge index $e(u,v)$, the corresponding vertices u and v can be found as follows:

$$u = 1 + \left\lfloor -\frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot e(u,v)} \right\rfloor$$

$$v = e(u,v) - \frac{1}{2}u(u-1)$$

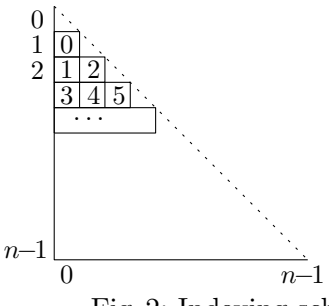


Fig. 2: Indexing scheme

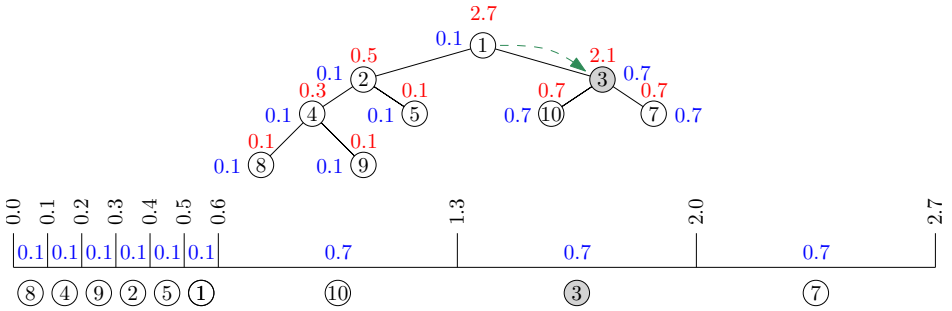


Fig. 3: Example of a random binary selection tree and its associated array. Blue numbers are weights of individual elements, whereas red number correspond to the accumulated weight of subtrees. Random number 1.08 in $[0, 2.7)$ guides the selection process through the tree.

Random Binary Selection Tree In order to choose the next edge to be added or deleted, we need a data structure that allows us to efficiently draw an element from a weighted set $O = \{o_1, \ldots, o_n\}$ such that the probability to choose a certain element is proportional to its weight. Given such a data structure, a naïve generator could simply store each potential edge $\{u, v\}$ with weight $p(u, v)$ and each non-edge with weight $1 - p(u, v)$ and iteratively draw edges to add or delete. Later on, we will show that one entry for each cluster rather than for each edge is sufficient.

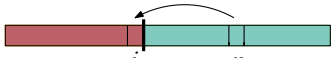
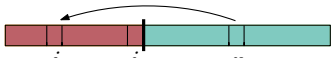
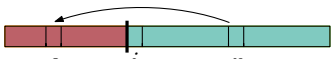
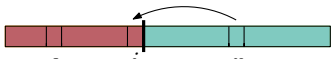
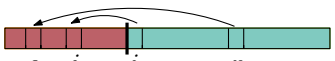
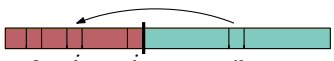
A simple solution for such a data structure is an array A storing prefix sums of the weights, i.e., $A[k] = \sum_{i=1}^{k-1} w(o_i)$, $1 \leq k \leq n + 1$. Let W be the sum of all weights. Then we can draw a uniformly distributed random number x in the interval $[0, W)$ and use binary search to find the index k such that $A[k] \leq x < A[k + 1]$. It is easy to see that this process selects each element o_k with probability $w(o_k)/W$. For static sets this approach works well, however, we will need to update weights frequently, and to add and delete elements occasionally. Updating the prefix sums has linear worst-case complexity, which we would like to avoid.

We therefore use a complete binary tree to store the elements. We define each vertex of this tree to be a tuple $q_i = (o_i, w_i, l_i, r_i)$, where o_i is an element, w_i is its associated weight $w(o_i)$, l_i is the sum of the weights in the left subtree and r_i is the sum of weights in its right subtree. The weight l_m and r_m of a leaf q_m are simply 0. Contrary to the prefix sum array, inserting and deleting elements as well as weight updates can be done in logarithmic time. To keep the tree balanced, new elements are inserted as leaves with minimum distance to the root and deleted elements are replaced by a leaf element with maximum distance to the root. Afterwards, weight changes have to be propagated on the path(s) to the root.

The procedure for the selection of an element starts at the root s by drawing a random number x from the interval $[0, l_s + w_s + r_s)$. Now there are three possible ranges for x: if $l_s \leq x < l_s + w_s$, the element is returned; if $x < l_s$, the carryover x is sent to the left subtree; and if $l_s + w_s \leq x$, the carryover $x - w_s - l_s$ is sent to the right subtree. The procedure continues recursively from there until an element is returned after at most $O(\log n)$ steps. The correctness of the selection process can be seen by constructing an array with prefix sums of the weights such that elements in the array are ordered according to an inorder traversal of the tree. Selecting an element in the tree is equivalent to a binary search in this array. An example for a random binary selection tree and the corresponding array can be found in Figure 3.

Virtual Fisher Yates Shuffle As edges between pairs of vertices in the same cluster are equiprobable, using a binary tree to store all intracluster vertex pairs is quite inefficient. Instead, we use a modified *Fisher-Yates shuffle* [31] for this task. A Fisher-Yates shuffle is a simple method to uniformly sample without replacement from a given set of n elements. The elements are stored in an array of size n with indices from 0 to $n - 1$ and the *border index* i of the

Table 1: Illustrative figures for *select*. Index $r \in \{i, \dots, n-1\}$ is drawn uniformly at random.

	Initial state	Final state
Case 1		
Case 2		
Case 3		
Case 4		

shuffle is initially set to 0. In each step, a random number r between i and $n-1$ is drawn, the corresponding element at index r is marked as selected and swapped with the element at index i. Then, the border index i is increased by 1. It is easy to see that each element can only be chosen once and the probability to choose each subset of size k is the same.

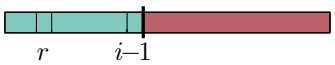
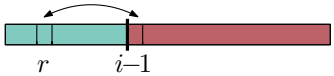
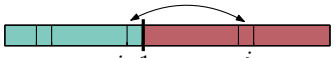
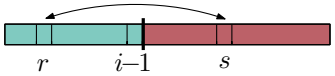
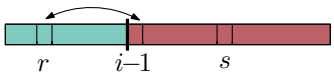
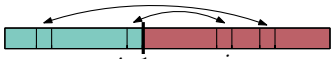
A drawback of this approach is that we have to enumerate and store each element explicitly. For elements that can be easily enumerated it is more efficient to store an implicit representation of this array one of which is the *virtual Fisher-Yates shuffle* introduced by Batagelj and Brandes [30].

Let i be the number of elements drawn so far, L be the set of indices smaller than i that have not yet been drawn and H the set of indices at least i that have been drawn. In our view, the indices in L and H are "exceptions from the rule that small indices are selected and large indices unselected". The crucial observation is that the cardinality of L and H is equal. Hence, we can define a bijection from H to L and store it in a map `replace`. Similarly to the original Fisher-Yates shuffle, we can now iteratively draw a random number r between i and $n-1$. If $\texttt{replace}(r) = \perp$, i.e., if there is no entry for r in the map, index r has not yet been selected and we select the corresponding element. If $\texttt{replace}(r) = s$, we choose index s instead. This process guarantees that we draw in each step an unselected element with uniform probability.

After we have selected the element, we have to update the map such that it is still guaranteed that each element in L is assigned to a corresponding element in H. Depending on the previous state of the shuffle, we have to consider the four cases shown in Table 1. Entries of the form $\texttt{replace}(x) = y$ are depicted as arrows from x to y and the thick line marks the border between elements with index less than i and larger i. It is easy to see that in each case a constant number of lookup, delete and insert operations in `replace` suffices. If we assume that the data structure we use for the map `replace` is a binary search tree, these operations take logarithmic time. Hence, the time complexity of choosing the next element is in $O(\log n)$.

Unlike the generation of static random graphs, we also need to delete edges over time. To this end, we have to modify the virtual Fisher-Yates shuffle slightly to deselect already selected numbers, i.e., putting them once again in the set of selectable elements. This can be done by making the pointers bidirectional, i.e., for each entry of the form $\texttt{replace}(j) = i$, we add a corresponding entry $\texttt{replace}(i) = j$. Deselection now works analogously to selection: First, we draw a random number r in the interval $[0, i-1]$. If there is no entry for r in `replace`, r is a currently selected index. In this case, we undo the selection by setting $\texttt{replace}(i-1) = r$ and vice versa and move the border one index to the left. If r is an unselected element, we undo the selection of $\texttt{replace}(r)$ instead. Special care has to be taken if the element at index $i-1$ is also unselected, i.e., if $\texttt{replace}(i-1) \neq \perp$. Figure 2 depicts all possible cases and the corresponding

Table 2: Illustrative figures for *deselect*. Index $r \in \{0, \dots, i-1\}$ is drawn uniformly at random.

	Initial state	Final state
Case 1		
Case 2		
Case 3		
Case 4		

updates of the map. Deselection is therefore just as efficient as selection. For each selected element, we have to store at most two entries in the map, hence the space requirement is linear in the number i of currently selected elements.

Overview of Selection Procedure In this section we explain how we use a combination between a random binary selection tree and virtual Fisher-Yates shuffles to efficiently implement edge dynamics according to the random model described in Section 3.1. For the sake of simplicity, we first assume that we only want to draw intracluster edges, i.e., $p_{\text{out}} = 0$. In this case, the following procedure can be used.

Each cluster has an associated shuffle that stores all intracluster edges in the cluster. This shuffle is used to uniformly select edges in the cluster to delete or to insert. To be able to use the enumeration scheme described above, each vertex v receives two ids, a global id that unambiguously identifies the vertex during the whole generation process and a local id in the range $[0, |\mathcal{C}(v)|-1]$. The local id is used to enumerate intracluster edges in the individual shuffles.

On top of that, we store two randomized binary selection trees Γ_{ins} and Γ_{del} that each contain one entry for every cluster. The weight of each cluster C in Γ_{ins} corresponds to the sum of the probability weight of edge insertions within the cluster, $\overline{m}(C) \cdot p_{\text{in}}(C)$. Similarly, its weight in Γ_{del} is defined as $m(C) \cdot \big(1 - p_{\text{in}}(C)\big)$. The overall process of selecting the next edge operation is now divided into three steps:

1. As introduced in Section 3.1, let $P_E = \sum_{\{u,v\} \in E} \big(1 - p(u,v)\big)$ and $P_{\overline{E}} = \sum_{\{u,v\} \notin E} p(u,v)$. With probability $P_E/(P_E + P_{\overline{E}})$ we decide to delete and with probability $P_{\overline{E}}/(P_E + P_{\overline{E}})$ to insert an edge.
2. For edge deletions, we choose a cluster C in Γ_{del} according to the stored weights. Similarly, we choose a cluster in Γ_{ins} if we have decided to insert an edge.
3. Depending on the choice in the first step, we insert or delete an edge in the virtual Fisher-Yates shuffle associated with C.

Finally, the weight of C in Γ_{ins} and Γ_{del} has to be updated. Later on, we will show that this process inserts an intracluster edge between two unconnected vertices u and v with probabilities according to the random process described in Section 3.1. Before we detail this, we describe how this procedure can be altered to be able to deal with intercluster edges.

Dealing with Intercluster Edges In principle, it would be possible to handle intercluster edges analogously and just introduce a virtual Fisher-Yates shuffle containing all pairs of vertices in different clusters. As all these vertices exist with the same probability, this would be

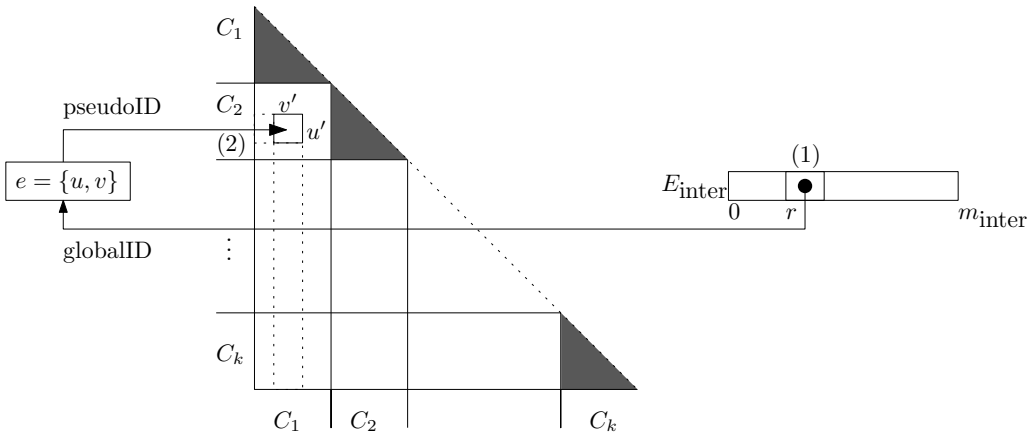


Fig. 4: Example for a *deselect* operation in the pseudocluster: r is the integer drawn uniformly at random in the range of $0, \ldots, m_{\text{inter}} - 1$ and u', v', resp. u, v are the local, resp. global vertex indices of the endpoints of the selected edge. The *white* area consists of the intercluster edges and *dark gray* areas contain all (forbidden) intracluster edges.

perfectly feasible. The problem with this approach is that it is not easy to consistently enumerate intercluster vertex pairs, as, due to vertex and cluster dynamics, the number of vertices in each cluster changes. For this reason, we introduce a shuffle for a *pseudocluster* containing all vertices in the graph. This pseudocluster gets an entry in Γ_{ins} with weight $\overline{m} \cdot p_{\text{out}}$. As the pseudocluster contains all vertices, the associated shuffle also contains intracluster vertex pairs. Hence, it is possible to draw intracluster edges either in the shuffle of the corresponding cluster or in the shuffle of the pseudocluster, which overestimates the probability of choosing such edges. To correct this, we exploit our assumption that $p_{\text{in}}(C) > p_{\text{out}}$ for each cluster C and decrease the weight associated with C in Γ_{ins} to $\overline{m}(C) \cdot (p_{\text{in}}(C) - p_{\text{out}}) \geq 0$.

For edge deletions, this trick cannot be used as $1 - p_{\text{out}}$ is larger than $1 - p_{\text{in}}(C)$. This is why we introduce an additional array storing the global id of all intercluster edges and draw a random edge in this array in case we decide to delete an intercluster edge. The respective edge is then both deleted in the array and in the shuffle of the pseudocluster. This procedure guarantees that we do not erroneously delete intracluster edges by choosing from the pseudocluster, which is why the weight of the true clusters in Γ_{del} remains unchanged. The weight of the pseudocluster in Γ_{del} corresponds to $m_{\text{inter}}(1 - p_{\text{out}})$. Whenever an intracluster edge is inserted or deleted, either in the pseudocluster or in its own cluster, the corresponding entry in the other shuffle has to be updated. Asymptotically, this does not increase running times and space requirements. Figure 4 illustrates the data structure used for the pseudocluster and the process of deleting an intercluster edge.

Correctness and Time Complexity In the following, we show that the selection process for edge dynamics described up to now follows the random model introduced in Section 3.1.

Proposition 1. *The described process inserts an edge between two unconnected vertices u and v with probability*

$$p(u, v)/(P_E + P_{\overline{E}}).$$

Proof. The probability of an edge insertion is $P_{\overline{E}}/(P_E + P_{\overline{E}})$. We first consider the case that u and v are both contained in a cluster C. The probability of choosing C in Γ_{ins} is $\overline{m}(C)/P_{\overline{E}} \cdot (p_{\text{in}}(C) - p_{\text{out}})$ and the probability of choosing the pseudocluster in Γ_{ins} is $\overline{m}/P_{\overline{E}} \cdot p_{\text{out}}$. Similarly, the probability of choosing $\{u, v\}$ in the associated shuffles is $1/\overline{m}(C)$ and $1/\overline{m}$, respectively. Hence, these probabilities sum up to

$$\frac{P_{\overline{E}}}{P_E + P_{\overline{E}}} \cdot \Big[\frac{\overline{m}(C)}{P_{\overline{E}}} \cdot (p_{\text{in}}(C) - p_{\text{out}}) \cdot \frac{1}{\overline{m}(C)} + \frac{\overline{m}}{P_{\overline{E}}} \cdot p_{\text{out}} \cdot \frac{1}{\overline{m}}\Big] = \frac{p(u, v)}{P_E + P_{\overline{E}}}$$

If $\{u, v\}$ is an intercluster pair, it can only be chosen via the pseudocluster. Hence, the corresponding probability is

$$\frac{P_{\overline{E}}}{P_E + P_{\overline{E}}} \cdot \frac{\overline{m}}{P_{\overline{E}}} \cdot p_{\text{out}} \cdot \frac{1}{\overline{m}} = \frac{p(u,v)}{P_E + P_{\overline{E}}} \qquad \square$$

A very similar proof yields the following proposition stating that deletions also follow the specified random model.

Proposition 2. *The described process deletes an existing edge $\{u, v\}$ with probability*

$$\big(1 - p(u,v)\big)/(P_E + P_{\overline{E}})$$

Deciding whether to insert or delete an edge is done in constant time. For both operations, choosing the cluster in the respective cluster tree takes time $O(\log k)$, where k denotes the current number of clusters. Similarly, selection or deselection in the corresponding virtual Fisher-Yates shuffle takes time logarithmic in the size of the shuffle. Hence, the expected time complexity for both operations is in $O(\log n)$. As the total size of the shuffles is asymptotically upper bounded by the number of edges of the current graph, it is easy to verify that the total space requirement is linear in the size of the graph.

4 Vertex and Cluster Dynamics

Assessing dynamic clustering algorithms usually involves the question, whether the algorithm is able to follow changes in the ground truth clustering. This is why we also included the possibility to incorporate vertex and cluster dynamics, both of which are not covered by $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$.

Vertex Dynamics Vertex dynamics are steered by a parameter p_χ that specifies the probability that instead of an edge operation, we delete or insert a vertex. If a vertex operation is to be performed, another parameter p_ν determines the probability that this operation is a vertex insertion. Choosing p_ν to be smaller or larger than 0.5 gives the opportunity to simulate graph growth or shrinkage. For deletion, a vertex is chosen uniformly at random and all incident edges are deleted. To adhere to the initial cluster size distribution, new vertices are assigned to clusters according to expected cluster sizes, similar to the generation of the initial clustering. To dampen the effect of additional edge deletions in the course of vertex deletions and to stay closer to the $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ model, new vertices are immediately connected to other vertices according to the prescribed edge probabilities. Naïvely, this takes $O(n)$ time, however, it is possible to use the *geometric method* introduced by Fan et al. [32] and used by Batagelj and Brandes [30] to reduce the running time to $O(\deg v)$, where $\deg v$ is the resulting degree of the new vertex v.

It remains to explain what has to be done to update the data structures. Updating the affected entry in the cluster trees takes $O(\log k)$ time. If a new vertex is inserted or deleted, the index space of the shuffle of its cluster and of the pseudocluster has to be adapted. For insertion, it suffices to assign the highest vertex id in the cluster to the new vertex and increase the index space of the shuffle accordingly. If a vertex is deleted, we have to guarantee that the index space is still continuous. We do this by relabeling the vertex v_f with the previously largest local id in the shuffle of the deleted vertex u by the id of u. The same procedure has to be performed for the vertex w_f with the largest local id in the pseudocluster. Figure 5 illustrates this case.

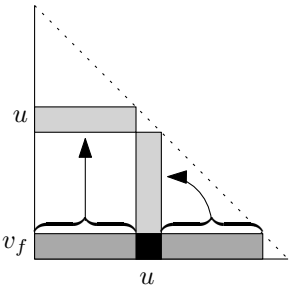


Fig. 5: Update if vertex with local id u is deleted.

For the relabeling step, we first delete all edges incident to v_f or w_f in the shuffle and then reinsert them using the new edge ids. Hence, the overall expected time complexity of the deletion of vertex v is $O\big((\deg v + \deg v_f + \deg w_f) \cdot \log n\big)$, whereas for vertex insertion we get $O\big(\deg v \cdot \log n\big)$.

Cluster Dynamics Cluster dynamics is independent of vertex and edge dynamics in the sense that in each time step, additionally to the vertex or edge operation performed, a cluster operation can take place. The probability of a cluster operation is determined by the input parameter p_ω and the probability that this cluster operation merges two clusters is determined by the parameter p_μ. With probability $1 - p_\mu$, one of the clusters is split by assigning each of its vertices to one of the new clusters with uniform probability.

To calculate new values for p_{in}, an obvious possibility is to just use the old value(s). For a cluster split, this means that the new clusters inherit p_{in} from the old cluster, whereas for a cluster merge, the new cluster is assigned the average of the intracluster edge probabilities of the participating clusters. This process leads to increasingly uniform p_{in} values over the course of time. For this reason, depending on a user parameter, new p_{in} values are generated randomly according to a Gaussian distribution estimated from the initially given list of intracluster probabilities. A more detailed description can be found in our technical report [26]

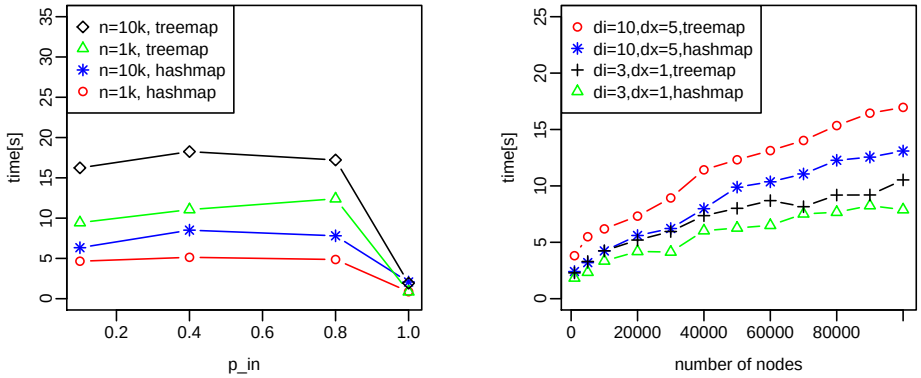
One of the main motivations for using $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ graphs for the evaluation of clustering algorithms is the knowledge of a ground truth clustering the result of the algorithm can be compared to. However, for cluster dynamics, it can easily be imagined that immediately after a split or merge operation, the clustering algorithm has no chance to detect the current ground truth clustering, as the change is not yet reflected in the edge structure. For this reason, the generator keeps track of an additional *reference clustering* that follows the current ground-truth clustering with some delay, roughly speaking, as soon as the involved subgraph becomes similar enough to the ground truth's expectation. A detailed description of the behavior of the reference clustering can be found in our technical report [26]. To prevent the interleaving of concurrent cluster events, as long as the change in the ground-truth is not propagated to the reference clustering, the participating clusters are not available for further cluster operations. If, for this reason, in some time step a cluster event is triggered but no available clusters are found, the cluster event does not take place.

To keep the data structures up to date, we delete the involved old shuffle(s) and create one or two new shuffles from scratch, depending on the kind of cluster operation. Furthermore, the additional array associated with the pseudocluster that stores all intercluster edges has to be updated. The reason for this is that for a split operation, new intercluster edges between the two parts arise, whereas cluster merges turn some intercluster edges into intracluster edges.v If pointers are used to link the occurrences of an edge in different data structures, all these operations take $O\big((n(C) + m(C)) \cdot \log n\big)$ time, where C is either the cluster that is about to be split or the new cluster after a merge. Removing the old cluster(s) and inserting the new cluster(s) in the cluster trees takes logarithmic time. Hence, this does not increase the (asymptotic) running time.

5 Experiments

We give a brief impression of the absolute running times of different sample configurations. All values are averaged over 15 runs and do not include the time to write the graph to hard disc.

The first experiments in Figure 6a evaluate the running times for one million steps of the generator, while only the intracluster density varies. The planted partition contains 15 clusters and the cluster size distribution is unskewed ($\beta = 1$). The number of vertices as well as the intercluster edge probability $p_{\text{out}} = 0.1$ are constant. As the number of edges is in $\Theta(n^2)$, the time to generate the initial graph sometimes dominates the time needed for edge dynamics and is therefore not included in the plot. Similarly, as the expected degree of each vertex is in $\Theta(n)$, we didn't include vertex and cluster dynamics and set the corresponding probabilities to 0. As expected, the running time is almost independent of p_{in}. The low running times for the experiments with $p_{\text{in}} = 1$ can be explained by the fact that intracluster edges are never deleted or inserted and all dynamics involve intercluster vertex pairs. To obtain logarithmic worst case



(a) Edge dynamics, constant number of vertices, variable p_{in}

(b) Full dynamics, constant expected intracluster (intercluster) degree `di` (`dx`), variable n

Fig. 6: Running times for some sample configurations.

running time for edge operations, the map `replace` used for the virtual Fisher-Yates shuffles can be stored in a binary search tree. For comparison, we repeated the experiments with hash maps instead of these trees. It can be seen that for graphs of high density, hash maps yield better practical running times[3].

Figure 6b illustrates the running time for less artificial parameter settings. Here, the number of clusters equals $\sqrt{n}$ and the size distribution is skewed ($\beta = 0.5$). The probability of a vertex event instead of an edge event is set to 0.1 and in half of the cases a vertex is added (deleted). The probability of a cluster event is 0.01 and in half of the cases a cluster is split (two clusters are merged). The expected degree of a vertex is constant, which yields very sparse graphs. To give a more realistic impression of the total running time, we included the time to generate the initial graph, followed by 100 000 dynamic updates. As above, the running times obtained by using hash maps are better than for the tree based variant.

In summary, the experiments show that hash maps yield better practical performance and that dynamics can be added to the planted partition model without causing much overhead.

Implementation Notes We conducted all experiments on a Dual-Core AMD Opteron(tm) Processor clocked at 2.6 GHz, using Java version 1.6.0_22. The machine has 32GB of RAM and 2×1 MB of L2 cache. The implementation uses no external libraries. As hash map, we used `java.util.HashMap`, whereas the tree-based implementation uses `java.util.TreeMap`, which is based on a red-black tree.

6 Conclusion and Outlook

We proposed a dynamic generalization of the planted partition model that can be used to evaluate dynamic graph clustering algorithms, with the additional benefit of a known reference clustering. Furthermore, we described how large dynamic random graphs according to this model can be efficiently generated and showed the practicability of this approach on selected example configurations. In order to make this model more realistic, modifications similar to the static model are conceivable. Possible changes in the random model include vertex movements, less uniform degree distribution, higher clustering coefficient as well as the generalization to hierarchical reference clusterings.

[3] Note that entries in the hash map are not distributed evenly over all possible indices, which is why we don't have expected constant time for all parameter settings

References

1. Fortunato, S.: Community detection in graphs. Physics Reports **486**(3–5) (2010) 75–174
2. Schaeffer, S.E.: Graph Clustering. Computer Science Review **1**(1) (August 2007) 27–64
3. Bollobás, B.: Random Graphs. Cambridge University Press (2001)
4. Gilbert, H.: Random Graphs. The Annals of Mathematical Statistics **30**(4) (1959) 1141–1144
5. Condon, A., Karp, R.M.: Algorithms for Graph Partitioning on the Planted Partition Model. Randoms Structures and Algorithms **18**(2) (2001) 116–140
6. Brandes, U., Gaertler, M., Wagner, D.: Experiments on Graph Clustering Algorithms. In: Proceedings of the 11th Annual European Symposium on Algorithms (ESA'03). Volume 2832 of Lecture Notes in Computer Science., Springer (2003) 568–579
7. Gaertler, M., Görke, R., Wagner, D.: Significance-Driven Graph Clustering. In: Proceedings of the 3rd International Conference on Algorithmic Aspects in Information and Management (AAIM'07). Lecture Notes in Computer Science, Springer (June 2007) 11–26
8. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proceedings of the National Academy of Science of the United States of America **99**(12) (2002) 7821–7826
9. Watts, D.J.: Small worlds: The dynamics of networks between order and randomness. Princeton University Press (1999)
10. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393**(6684) (June 1998) 440–442
11. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286** (1999) 509–512
12. Leskovec, J., Kleinberg, J.M., Faloutsos, C.: Graphs Over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Press (2005) 177–187
13. Vázquez, A.: Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations. Physical Review E **67** (May 2003) 056104
14. Bagrow, J.: Evaluating local community methods in networks. Journal of Statistical Mechanics: Theory and Experiment (2008) P05001 Doi:10.1088/1742-5468/2008/05/P05001.
15. Lancichinetti, A., Fortunato, S.: Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Physical Review E **80**(1) (2009) 016118
16. Fan, Y., Li, M., Zhang, P., Wu, J., Di, Z.: Accuracy and precision of methods for community identification in weighted networks. Physica A **377**(1) (2007) 363–372
17. Guimerà, R., Sales-Pardo, M., Amaral, L.A.N.: Module identification in bipartite and directed networks. Physical Review E **76** (September 2007) 036102
18. Zhou, H.: Network landscape from a Brownian particle's perspective. Physical Review E **67** (2003) 041908
19. Sawardecker, E.N., Sales-Pardo, M., Amaral, L.A.N.: Detection of node group membership in networks with group overlap. The European Physical Journal B **67** (2009) 277–284
20. Aldecoa, R., Marín, I.: Closed benchmarks for network community structure characterization. Physical Review E **85** (February 2012) 026109
21. Brandes, U., Mader, M.: A Quantitative Comparison of Stress-Minimization Approaches for Offline Dynamic Graph Drawing. In: Proceedings of the 19th International Symposium on Graph Drawing (GD'11). Lecture Notes in Computer Science, Springer (2012) 99–110
22. Robins, G., Pattison, P., Kalish, Y., Lusher, D.: An introduction to exponential random graph (p*) models for social networks. Social Networks **29**(2) (2007) 173–191
23. Snijders, T.A.: The Statistical Evaluation of Social Network Dynamics. Sociological Methodology **31**(1) (2001) 361–395
24. Clementi, A.E.F., Macci, C., Monti, A., Pasquale, F., Silvestri, R.: Flooding time in edge-Markovian dynamic graphs. SIAM Journal on Discrete Mathematics **24**(4) (2010) 1694–1712
25. Baumann, H., Crescenzi, P., Fraigniaud, P.: Parsimonious flooding in dynamic graphs. In: Proceedings of the 28th ACM symposium on Principles of distributed computing, ACM Press (2009) 260–269
26. Görke, R., Staudt, C.: A Generator for Dynamic Clustered Random Graphs. Technical report, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH) (2009) Informatik, Uni Karlsruhe, TR 2009-7.
27. Görke, R.: An Algorithmic Walk from Static to Dynamic Graph Clustering. PhD thesis, Fakultät für Informatik (February 2010)
28. Görke, R., Maillard, P., Staudt, C., Wagner, D.: Modularity-Driven Clustering of Dynamic Graphs. In Festa, P., ed.: Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10). Volume 6049 of Lecture Notes in Computer Science., Springer (May 2010) 436–448
29. Behrends, E.: Introduction to Markov Chains With Special Emphasis on Rapid Mixing. Friedrick Vieweg & Son (October 2002)
30. Batagelj, V., Brandes, U.: Efficient Generation of Large Random Networks. Physical Review E (036113) (2005) 036113
31. Fisher, R.A., Yates, F.: Statistical Tables for Biological, Agricultural and Medical Research. Oliver and Boyd, London (1948)
32. Fan, C.T., Muller, M.E., Rezucha, I.: Development of Sampling Plans by Using Sequential (Item by Item) Selection Techniques and Digital-Computers. Journal of the American Statistical Association **57**(298) (1962) 387–402