General Generator for Attributed Graphs with Community Structure

Seiji Maekawa[1], Jianpeng Zhang[2,3], George Fletcher[2], and Makoto Onizuka[1]

[1] Osaka University, 1–5 Yamadaoka, Suita, Osaka, Japan
{maekawa.seiji,onizuka}@ist.osaka-u.ac.jp
[2] Eindhoven University of Technology, the Netherlands g.h.l.fletcher@tue.nl
[3] National Digital Switching System Engineering & Technological R&D Center (NDSC), China zjp@ndsc.com.cn

Abstract. We propose acMark, a scalable and general generator for attributed graphs with cluster labels, which has the following advantages: (i) users can flexibly control the separability of the clusters from the viewpoints of both the topology and the attributes of graphs; (ii) users can precisely specify various distributions for node degrees, cluster sizes, and attribute values; and, (iii) graph generation scales linearly to the number of edges of generated graphs. Through extensive experiments, we demonstrate that acMark can generate large-scale graphs with controlled characteristics as needed by contemporary graph analytics researchers.

Keywords: graph generator · attributed graph · community structure · graph property · latent factors.

1 Introduction

Graph is a fundamental data structure consisting of nodes and their relationships. Graphs are ubiquitous in many application domains, such as web graph [7], social networks [8], protein complexes [6], traffic planning [9], computer vision [13], and gene expressions [3, 15]. Of the rich variety of graph analytics methods, graph clustering is one of the most widely used techniques in the machine learning and data mining fields [25].

When we consider real world applications, a node usually has multiple attributes. For example, in a social network a person node may have "age" and "language" as attributes. Contemporary graph databases support attributed graphs (also known as property graphs) [5] and there emerges many attributed graph techniques for clustering and classification tasks [1, 12, 23, 27] and representation learning [11, 31].

Researchers often need large-scale attributed graphs with various characteristics and flexible community structure in order to experimentally validate the effectiveness and efficiency of methods for such tasks. Although there are many real world attributed graphs available in repositories, most of them do not have cluster labels. Indeed, SNAP [19], which is the most widely known graph archive, does not include any attributed graphs with cluster labels. We can find attributed graphs with cluster labels in some graph research projects[4], however most of them are small-scale graphs. Turning to large-scale and synthetic graph generation, the state of the art methods for attributed graphs with clusters can not specify various types of distributions for node degree, cluster size, or attribute values [4, 17]. As an example, [17] does not allow the user to control the distribution of cluster size and the attribute values. It only permits to use a normal distribution.

To address this need in the field, we propose acMark for generating large-scale attributed graphs with controlled characteristics and flexible community structure. The major strength of acMark is that it can flexibly control the separability of the clusters and specify various types of distribution for node degree, cluster size, and attribute values. To the best of our knowledge, this is the first graph generator that can generate attributed graphs with flexible community structure. The idea of acMark is that, it first generates intermediate data structures, the latent factors of clusters, and then generates attributed graphs by using the latent factors. The latent factors are flexibly generated based on any distributions specified by the user. acMark generates nodes by following the proportions of the latent factors and then generates edges so as to satisfy the topology property. acMark also controls the distribution of the cluster sizes and attribute values. For each node attribute, acMark generates a value by following two rules: 1) if two nodes belong to the same cluster they should share similar values, and 2) the values should follow the distribution specified by the user. acMark is efficient in that the graph generation scales linearly with the edge size of the output graph. As a community resource, acMark is available as open source code.[5]

Our contributions in this paper include a systematic presentation of acMark (Sections 2 and 3) and a detailed experimental analysis of the scalability and quality of generated graphs, including an application to clarifying the characteristics of existing clustering methods (Section 4). We also discuss related state of the art methods (Section 5). We give concluding remarks in Section 6.

2 Preliminaries

An attributed graph is a triple $G = (N, E, X)$ where $N = \{1, 2, \ldots, n\}$ is a node set, $E = \{(i, j)\} \subseteq [n] \times [n]$ is an edge set, and $X : N \to dom_{X_1} \times \cdots \times dom_{X_d}$ is a function from a node to its d attribute values. The cluster assignment list obtained from the graph topology and the attributes is expressed by $\boldsymbol{C} \in \{1, ..., k_1\}^n$ and $\boldsymbol{C}^X \in \{1, ..., k_2\}^n$, respectively. Note that current graph generators only consider $\boldsymbol{C}$ without using $\boldsymbol{C}^X$.

2.1 Graph properties

We highlight two properties that real world graphs typically have: topology property and attribute property.

[4] For example, https://linqs.soe.ucsc.edu

[5] https://github.com/seijimaekawa/acMark

Topology property. Real world graphs have well-known topology properties [18, 22], which are desirable in synthetically generated graphs: 1) the node degrees follow a power-law distribution, and 2) the cluster sizes follow a power-law distribution. In addition, from the topology aspect, one of the difficulties in graph generation is that the output graph should satisfy the topology property while insuring all the edges have two adjacent nodes.

Attribute property. There are two types of attributes, categorical type (e.g. conference, university, city) and numerical type (e.g. price, timestamp). In real datasets, it is well known that most numerical attributes follow a normal distribution or power-law distribution. So, graph generators should control the distribution type of attribute values.

Comparisons. State of the art graph generators, such as LFR [16] and ANC [17], generate graphs by specifying distributions so that they satisfy the above properties, but only with limited types of distribution. Table 1 shows the comparisons of our acMark, LFR [16], and ANC [17].

Table 1: Comparisons of different graph generators. PL indicates a power law distribution and ND indicates a normal distribution. Our method supports arbitrary distributions for the node degrees, the cluster sizes, and attribute values. Actually, the distributions of power law, uniform, and normal are implemented. NA indicates that there is no support for the property.

Generator	Node degree	Cluster size	Attribute	Complexity
LFR [16]	PL	PL	NA	$O(m)$
ANC [17]	PL	NA	$\mathbb{R}$(ND)	$O(m)$
Our method	arbitrary	arbitrary	arbitrary	$O(m)$

3 acMark: attributed graph generator with flexible community structure

We next present acMark, our generator for attributed graphs with flexible community structure and real world graph properties. The idea of acMark is that, instead of generating attributed graphs directly, we first generate intermediate data structures, the latent factors of clusters, so that we can flexibly generate attributed graphs with cluster labels. The features of acMark are three-fold. First, acMark treats the topology clusters and the attribute clusters independently and the output graphs are generated by assuming there is a complex relationship between the topology clusters and the attribute clusters. For instance, the topology and the attributes usually have different number of clusters and, moreover, a topology cluster may be affected by multiple attribute clusters with different weights. Therefore, acMark controls the cluster separability by using topology cluster proportions and attribute cluster proportions and also takes into account such relationship by introducing cluster transfer proportions between the topology cluster proportions and the attribute cluster proportions. We call those three proportions the latent factors of clusters.

Second, acMark generates attributed graphs from the latent factors so that the generated graphs reflect the cluster proportions, and also the generated graphs should satisfy the graph properties as we described in Section 2.

Finally, we design acMark to scale linearly to the number of edges of generated graphs in terms of the time and space complexities.

3.1 Generating Model

We explain the detailed design of acMark by using Figure 1, with parameters given in Table 2. acMark generates output graphs with cluster labels by using latent factors: topology cluster proportions $\boldsymbol{U} \in \mathbb{R}_+^{n \times k_1}$, attribute cluster proportions $\boldsymbol{V} \in \mathbb{R}_+^{d \times k_2}$, and cluster transfer proportions $\boldsymbol{H} \in \mathbb{R}_+^{k_1 \times k_2}$ between $\boldsymbol{U}$ and $\boldsymbol{V}$. The output graph is expressed with topology cluster assignment list $\boldsymbol{C}$, adjacency matrix $\boldsymbol{S}$, and attribute matrix $\boldsymbol{X}$. Topology cluster assignment list $\boldsymbol{C}$ is obtained from topology cluster proportions $\boldsymbol{U}$ by choosing the largest proportion cluster for each node[6]. Intuitively, adjacency matrix $\boldsymbol{S}$ and attribute matrix $\boldsymbol{X}$ can be generated by $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively. But, we also use $\boldsymbol{U}$ and $\boldsymbol{H}$ for generating $\boldsymbol{X}$, so that we simulate the complex relationship between the topology clusters and the attribute clusters.

The adjacency matrix generation is not obvious, because the output graph should reflect the topology property while insuring all the edges have two adjacent nodes. Since deciding whether there exists a graph satisfying the give topology property is NP-complete [2], acMark takes a greedy approach for graph generation as follows. We sample the candidate of generating edges so as to satisfy the node degree constraint, and then, compute the probability of the edge existence from the latent factors. Finally, we use a Poisson distribution for confirming edges based on the probability. We also carefully design acMark so that runtime scales linearly with the number of edges.

acMark supports various input parameters as shown in Table 2. The most basic parameters are the number of nodes and edges to generate, and the distributions of the node degrees, the cluster sizes[7], and the attribute values (continuous or discrete)[8]. We normalize all attribute values to [0,1] without the loss of generality. The ratio of attributes whose values follow the distribution of Bernoulli, power law, uniform, or normal is controlled by att_{ber}, att_{pow}, att_{uni} and att_{nor}, respectively.[9]

[6] Similarly, attribute cluster assignment list $\boldsymbol{C}^X$ is obtained from $\boldsymbol{V}$.

[7] As for the exponents, we choose typical values of real networks: $2 \leq \phi_d \leq 3, 1 \leq \phi_c \leq 2$, where ϕ_d and ϕ_c are the parameters for node degree and cluster size, respectively.

[8] Categorical attribute values can be encoded with binary representation.

[9] As for the reminder part of the ratio, the attribute values are generated independently from the latent factors. In this case, the attribute values follow a randomly selected a distribution from bernoulli, power law, uniform, or normal distributions. The distribution parameters $\boldsymbol{\omega}$, which consist of $\phi_{att\_min}, \phi_{att\_max}, \delta_{att}, \sigma_{att\_min}, \sigma_{att\_max}$, are used for the reminder part.

Table 2: Description of the graph generator parameters. $\boldsymbol{\phi}$ consists of $\phi_d, \phi_c, \phi_{\boldsymbol{V}}, \phi_{\boldsymbol{H}}$, $\boldsymbol{\delta}$ consists of $\delta_d, \delta_c, \delta_{\boldsymbol{V}}, \delta_{\boldsymbol{H}}$, and $\boldsymbol{\sigma}$ consists of $\sigma_d, \sigma_c, \sigma_{\boldsymbol{V}}, \sigma_{\boldsymbol{H}}$. The subscripts of ϕ, δ, and σ indicate as follow: d represents node degree, c represents cluster size, $\boldsymbol{V}$ represents attribute cluster proportions and $\boldsymbol{H}$ represents cluster transfer proportions.

Input	Description
$n, m, d \in \mathbb{N}$	number of nodes, edges, attributes
$k_1, k_2 \in \mathbb{N}$	number of clusters for topology, attributes
$\alpha \in \mathbb{R}_+$	parameters for balancing inter-edges and intra-edges
$\beta \in \mathbb{R}_+$	parameters of separability for attribute cluster proportions
$\gamma \in \mathbb{R}_+$	parameters of separability for cluster transfer proportions
$\boldsymbol{\phi} \in \mathbb{R}^4$	parameters of exponent for power law distribution
$\boldsymbol{\delta} \in \mathbb{R}^4$	widths of uniform distribution
$\boldsymbol{\sigma} \in \mathbb{R}^4$	deviations of normal distribution
f_S, f_X	transformation for adjacency matrix, attribute matrix
$r \in \mathbb{N}$	number of iterations for edge construction
$att_{ber} \in \mathbb{R}$	ratio of attributes which takes discrete value
$att_{pow} \in \mathbb{R}$	ratio of attributes which follow power law distributions
$att_{uni} \in \mathbb{R}$	ratio of attributes which follow uniform distributions
$att_{nor} \in \mathbb{R}$	ratio of attributes which follow normal distributions
$\boldsymbol{\omega} \in \mathbb{R}^*$	parameters of random attributes

Algorithm Algorithm 1 describes the procedure of graph generation. It consists of two phases, latent factors generation phase and graph generation phase. In the latent factors generation phase, three latent factors are generated by latent_factor_generation function (line 1,3,4). We take a Bayesian approach for the latent factor generation described in Algorithm 2. We treat ξ as random variables and place a prior distribution over it. Let $\boldsymbol{W}$ be the output proportions of the function. A distribution $\boldsymbol{\chi}$ is specified by the user, either of power law, uniform, or normal (line 1). We generate a dirichlet distribution ξ based on $\tau\boldsymbol{\chi}$ where τ controls the effect of $\boldsymbol{W}$'s columns to the cluster proportions (line 2). That is, $\boldsymbol{W}$'s columns are related with only few clusters when τ is small. We use a multinomial distribution to produce the output proportion $\boldsymbol{W}_i$ for each row (line 3-5). We obtain the number of attributes for each distribution based on the ratio att_{ber}, att_{pow}, att_{uni} and att_{nor} given by the user (line 2 in Algorithm 1). These values are then used as input to decide the size of $\boldsymbol{V}$ (line 3). In the graph generation phase, we generate the adjacency matrix $\boldsymbol{S}$ (line 8), attribute matrix $\boldsymbol{X}$ (line 9), and cluster assignment list $\boldsymbol{C}$ (line 5-7). The cluster assignment is decided by the largest entry in the cluster proportion $\boldsymbol{U}$ for each node (line 6).

Algorithm 3 describes how the adjacency matrix $\boldsymbol{S}$ is generated. The idea is that we randomly generate candidate edges and then they are confirmed as real edges if they follow the expected degree proportion of their adjacent nodes. The detail is as follows. The expected node degree proportions $\boldsymbol{\theta}$ is derived from the

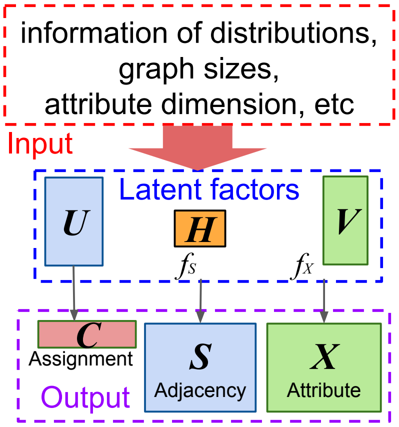


Fig. 1: Illustration of our proposal. Each row of $\boldsymbol{U}$ indicates a cluster proportion for each node. Each row of $\boldsymbol{V}$ indicates a cluster proportion for each attribute.

distribution specified by the user (line 1). $\boldsymbol{\theta}'$ is the actual node degrees of the generating graph. It is initialized as zeros (line 2) and its entries $\boldsymbol{\theta}'_i$ and $\boldsymbol{\theta}'_j$ are incremented when a new edge $(i,\ j)$ is confirmed to be real one (line 16). For each node i, edges are iteratively generated based on the node degree proportion $\boldsymbol{\theta}_i$ by *Candidate selection* step (line 7-10) and *Candidate confirmation* step (line 12-20). In *Candidate selection* step, a candidate edge is listed as a pair of i and randomly chosen node $j \in \boldsymbol{M}$ (line 9). In *Candidate confirmation* step, if the actual node degrees of the adjacent nodes (i and j) of a candidate edge do not reach to the expected ones (θ'_i and θ'_j, line 13), the edge existence is decided by using a Poisson distribution (line 14). Note there is a possibility that this loop does not stop, because the adjacency matrix generation is a type of combinatorial optimization problem. We take a greedy approach: we exit the loop at the user-specified r iterations (line 5)[10]. It is our future work to guarantee the theoretical quality bounds of the generated graphs.

Algorithm 4 describes how the attribute matrix $\boldsymbol{X}$ is generated. The output $\boldsymbol{X}$ is obtained by concatenating $\boldsymbol{X}^{UHV}$ and random matrix $\boldsymbol{X}^{ran}$ (line 8): we use $\boldsymbol{X}^{ran}$ for the purpose of generating values independent from clusters. $\boldsymbol{X}^{UHV}$ is generated by $f_X(\boldsymbol{U}, \boldsymbol{H}, \boldsymbol{V})$. That is, 1) we obtain base attribute vectors for nodes by computing the product of $\boldsymbol{U}, \boldsymbol{H}, \boldsymbol{V}$ so that two nodes in the same cluster (reflecting the effect of the topology and attributes) should share similar values, and then 2) we apply user-specified distribution to the base attribute vectors so that the attribute values should follow the distribution (line 1).

[10] Although we adjust the sum of all the node degrees in $\boldsymbol{\theta}$ to be the number of edges m, some candidate edges may not be generated when the node degrees of the adjacent nodes exceeds the expected ones, so the actual number of the generated edges tends to be smaller than the expected number of edges, m.

Algorithm 1 Graph generation

```
Require: $n, m, d, k_1, k_2, \alpha, \beta, \gamma, \boldsymbol{\phi}, \boldsymbol{\delta}, \boldsymbol{\sigma}, f_S, f_X, r, att_{ber}, att_{pow}, att_{uni}, att_{nor}, \boldsymbol{\omega}$
Ensure: adjacency matrix $\boldsymbol{S}$, attribute matrix $\boldsymbol{X}$, cluster assignment list $\boldsymbol{C}$
  # Latent factors generation phase
  $\boldsymbol{U}$ = latent_factor_generation($c, n, \alpha, \boldsymbol{\phi}, \boldsymbol{\delta}, \boldsymbol{\sigma}$)
  $d_{ber}, d_{pow}, d_{uni}, d_{nor} \leftarrow int(d \times att_{ber}), int(d \times att_{pow}), int(d \times att_{uni}), int(d \times att_{nor})$
  $\boldsymbol{V}$ = latent_factor_generation($V$,$d_{ber} + d_{pow} + d_{uni} + d_{nor}$,$\beta, \boldsymbol{\phi}, \boldsymbol{\delta}, \boldsymbol{\sigma}$)
  $\boldsymbol{H}$ = latent_factor_generation($H, k_1, \gamma, \boldsymbol{\phi}, \boldsymbol{\delta}, \boldsymbol{\sigma}$)
  # Graph generation phase
  for $i = 0$ to $n$ do
    $\boldsymbol{C}_i$ = argmax($\boldsymbol{U}_i$) # cluster assignment
  end for
  $\boldsymbol{S}$ = adjacency_matrix_generation($\boldsymbol{U}, \boldsymbol{H}, \boldsymbol{V}, r, f_S$)
  $\boldsymbol{X}$ = attribute_matrix_generation($\boldsymbol{U}, \boldsymbol{H}, \boldsymbol{V}$,$d - (d_{ber} + d_{pow} + d_{uni} + d_{nor}), \boldsymbol{\omega}$)
```

Algorithm 2 latent_factor_generation($type, size, \tau, \boldsymbol{\phi}, \boldsymbol{\delta}, \boldsymbol{\sigma}$)

```
$\boldsymbol{\chi} = chooseFrom(power\,law(\phi_{type}), uniform(\delta_{type}), normal(\sigma_{type}))$
$\xi = Dirichlet(\tau\boldsymbol{\chi}/\sum\boldsymbol{\chi})$
for $i = 1$ to $size$ do
  $\boldsymbol{W}_i = mutinomial(\xi)$
end for
return $\boldsymbol{W}$
```

Algorithm 3 adjacency_matrix_generation($\boldsymbol{U}, \boldsymbol{H}, \boldsymbol{V}, r, f_S$)

```
$\boldsymbol{\theta} = chooseFrom(power\,law(\phi_d), uniform(\delta_d), normal(\sigma_d))$
$\boldsymbol{\theta}' = [0]^n$
for $i = 1$ to $n$ do
  $counter = 0$
  while $counter < r$ and $\theta'_i < \theta_i$ do
    # Candidate selection
    $\boldsymbol{M} = \{\}$
    for $j = 1$ to $\theta_i$ do
      $\boldsymbol{M} = \boldsymbol{M} \bigcup Rand_{int}(1, n)$
    end for
    # Candidate confirmation: ($i$,$j$) is a candidate edge
    for $j \in \boldsymbol{M}$ do
      if $S_{ij} == 0$ and $\theta'_i < \theta_i$ and $\theta'_j < \theta_j$ then
        $S_{ij} = f_S(\boldsymbol{U})$ #e.g. $Poisson(<\boldsymbol{U}_i, \boldsymbol{U}_j>)$)
        if $S_{ij} > 0$ then
          $S_{ji} = S_{ij}$ # since undirected graph
          $(\theta'_i, \theta'_j) = (\theta'_i + 1, \theta'_j + 1)$
        end if
      end if
    end for
    $counter = counter + 1$
  end while
end for
return $\boldsymbol{S}$
```

Algorithm 4 attribute_matrix_generation($\boldsymbol{U}, \boldsymbol{H}, \boldsymbol{V}, d_{ran}, \boldsymbol{\omega}$)

```
$\boldsymbol{X}^{UHV} = f_X(\boldsymbol{U}, \boldsymbol{H}, \boldsymbol{V})$ (e.g. $(power\ law, uniform, normal)(\boldsymbol{UHV}^\top)$ for numerical values, and $Bernoulli(\boldsymbol{UHV}^\top)$ for categorical values)
for $j = 0$ to $d_{ran}$ do
    $dist = RAND(Bernoulli, powerlaw, uniform, normal)$
    for $i = 0$ to $n$ do
        $\boldsymbol{X}^{ran}_{ij} = dist(\boldsymbol{\omega})$
    end for
end for
$\boldsymbol{X}$ = concatenation($\boldsymbol{X}^{UHV}, \boldsymbol{X}^{ran}$)
return $\boldsymbol{X}$
```

Complexity We discuss the space/time complexities of acMark. As is typical in network analytics, we focus on sparse graphs [22] as real-world graphs are often sparse. On the sparse condition, the mean of the node degree can be treated as a constant, $m \propto n$, so m is considered to be a much smaller value than n^2.

Space complexity. The largest concern is the adjacency matrix $\boldsymbol{S}$ since the size of $\boldsymbol{S}$ could be as large as n^2. Hence, we use sparse representation for $\boldsymbol{S}$, such as an adjacency list, with size $O(m)$. The size of an attribute matrix $\boldsymbol{X}$ is nd. We can omit the size of the latent factors because $k_1, k_2 \ll n, d$. Therefore, the space complexity is $O(m + nd)$,

Time complexity. We analyze the time complexity of the latent factor generation, the adjacency matrix generation, and the attribute matrix generation, respectively. First, the complexity for generating the topology cluster proportions, the attribute cluster proportions, and cluster transfer proportions is $O(nk_1 + dk_2 + k_1k_2)$ based on their matrix sizes. Second, the adjacency matrix generation consists of the *candidate selection* step and the *candidate confirmation* step. In the *candidate selection* step, candidates are chosen based on the node degree. In *candidate confirmation* step, we calculate the edge probability whose complexity is $O(k_1)$. These two steps are executed for each node so they require $O(nk_1 r\theta_{Avg})$, where r is the number of iterations for edge generation. Further, $r = c \times k_1$ where c is constant and $m = n\theta_{Avg}$, that is, the complexity becomes $O(mk_1^2)$. Finally, in the attribute matrix generation, the complexity is $O(ndk)$ where $k = min(k_1, k_2)$. Therefore, the total time complexity is $O(mk_1^2 + ndk)$.

4 Experiments

We next describe an experimental study of acMark. Our goal in the experiments is to answer the following questions:

Q1 Does acMark support users to control the distributions of graph properties?
Q2 How well does acMark scale?
Q3 Does acMark support users to flexibly control the separability of clusters?

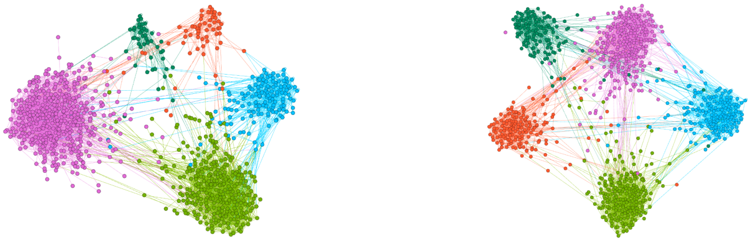
Q4 Can we use acMark to clarify the characteristics of contemporary clustering methods?

We use a power law distribution ($\phi_d = 3, \phi_c = 2$) for node degrees and cluster sizes as a default setting. acMark is implemented in Python3, and, as noted in Section 1, is made available as open source. The experiments are operated on 64 Intel(R) Xeon(R) CPU E5-4650 0 @ 2.70GHz with 32GB memory.

4.1 Q1: Visualization with controlled graph properties

To confirm that acMark supports the control of the distributions of graph properties, we visualize the distributions of the generated graphs.

Graphs with various topology properties. We visualize the generated graphs with various cluster sizes (Figure 2) and with various node degrees (Figure 3a, 3c)[11]. Figure 3b, 3d depict the histograms of the node degree distributions. These figures indicate that the generated graphs follow the graph properties specified by the user.



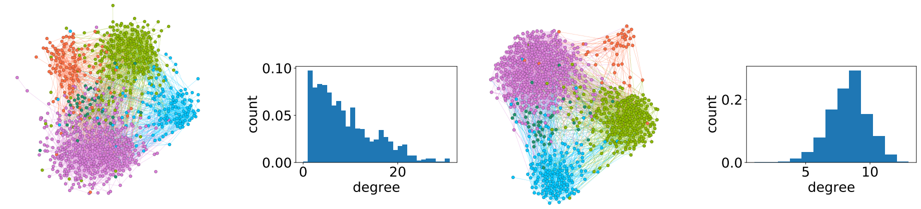
(a) The cluster sizes follow a power law distribution ($\phi_c = 2$).

(b) The cluster sizes follow a normal distribution ($\sigma_c = 0.1$).

Fig. 2: Visualization of graphs with two different cluster size distributions generated by acMark. The node colors represent clusters. These graphs have 1000 nodes and about 4000 edges. The node degrees follow a power law distribution ($\phi_d = 3$).

Graphs with various attribute properties. We analyze the attribute values in the generated graphs. Figure 4 depicts the histograms of a single attribute and 2-D plots of the values of two attributes. Figure 4a, 4b depict the statistics of attribute values that are generated without specifying distributions. Figure 4a indicates that the values are clearly separated into clusters and we clearly observe base attribute vectors in Figure 4b. In contrast, Figure 4c, 4d depict the results with specifying a normal distribution to the attribute values. We observe that the values actually follow a normal distribution.

[11] We utilize Gephi for the visualization. Note: Gephi is limited to place the vertices based only on the graph topology and ignores the effect of the attributes.



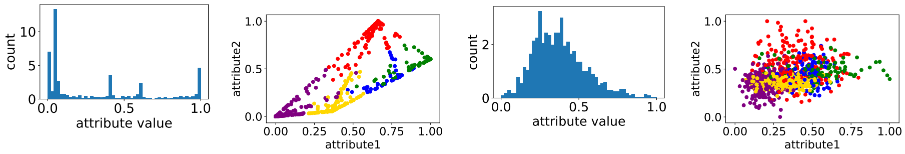
(a) The node degrees follow a power law distribution ($\phi_d = 3$).

(b) The node degree distribution of (a).

(c) The node degrees follow a normal distribution ($\sigma_d = 0.1$).

(d) The node degree distribution of (c).

Fig. 3: Visualization of graphs with two node degree distributions generated by acMark. The node colors represent clusters. These graphs have 1000 nodes and about 4000 edges. The cluster sizes follow a power law distribution ($\phi_c = 2$).



(a) Histogram of attribute values generated without specifying distributions.

(b) Attribute values of (a) plotted in 2-D space.

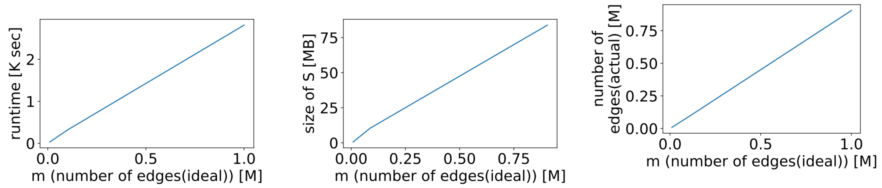
(c) Histogram of attribute values generated by a normal distribution.

(d) Attribute values of (c) plotted in 2-D space.

Fig. 4: Histograms and 2-D plots of attribute values. These graphs have 1000 nodes and about 4000 edges. The cluster sizes follow a power law distribution ($\phi_c = 2$). In Figure 4c, 4d, *mean* $= 1$ and *deviation* is randomly chosen from the range of $0.1 - 0.3$.

4.2 Q2: Scalability of graph generation

To investigate the scalability of acMark, we demonstrate how the runtime and memory consumption vary for various values of parameters. The default setting of the parameters is as follows: $d = 100, k_1 = 10, k_2 = 10, \alpha = 1/k_1, \sigma_d = 0.1, \sigma_c = 0.1, r = 10 * k_1, att_{ber} = 0.0, att_{pow} = 0.4, att_{uni} = 0.1, att_{nor} = 0.4$. m is chosen from $\{10^4, 10^5, 10^6, 10^7\}$ and $n = m/10$ for keeping the sparse condition of generating graphs. Figure 5a indicates that the runtime actually scales linearly to the number of edges. The memory size of $\boldsymbol{S}$ also scales linearly to the number of edges in Figure 5b. Figure 5c indicates that the number of the actual edges is smaller than the input parameter m: this result follows the discussion we made in footnote 8. As a whole, these experiments demonstrate that both time and space complexities are linear to m, the number of edges.



(a) Runtime.

(b) Size of $\boldsymbol{S}$.

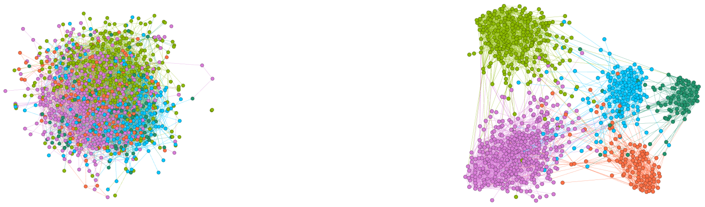
(c) Comparison with m and the actual number of edges.

Fig. 5: Scalability experiments against the number of edges. The other parameters are set as follow: $n = m/10$, $d = 100$, $k_1 = 10$, $k_2 = 10$.

4.3 Q3: Separability of clusters

We next study the extent to which acMark supports flexible control of the separability of clusters in terms of the topology and the attributes. First, we visualize the topology structure with various values of α and the attribute values with various combinations of β and γ. Second, we investigate structural measures with various graph sizes and with various values of α.

Visualization. Figure 6 shows that the ratio of intra-edges is large when α is small. This is one of the features of dirichlet distribution: clusters are more separated if α is set at a smaller value. We also conduct experiments for β and γ. Fig 7 shows how those parameters can control the separability of attributes. The clusters are more overlapped when β and γ are set at larger values (See Fig 7(a)) by following the feature of dirichlet distribution.



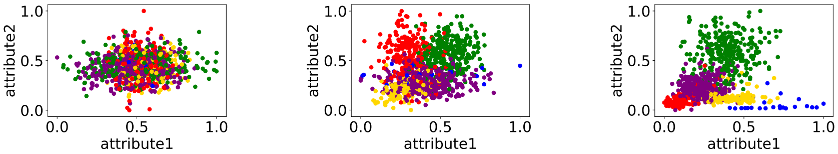
(a) $\alpha = 1.0$.

(b) $\alpha = 0.1$

Fig. 6: Separability of clusters w.r.t. the topology when $n = 1000, m = 4000, k_1 = 5$. The node colors represent the clusters.

Structural measures. We investigate how modularity [21], average cluster coefficient, and graph diameter[12] vary with various values of α. These measures are often used to clarify the characteristics of graphs. Modularity is a measure

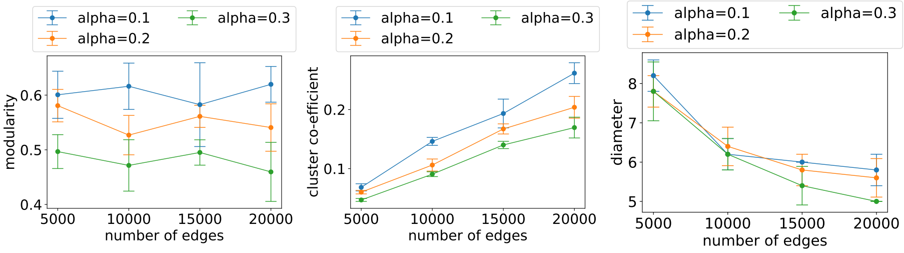
[12] Diameters can not be calculated in unconnected graphs so we use the largest diameter among connected components as the diameter of the whole graph.



(a) $\beta = 100, \gamma = 100$. (b) $\beta = 10, \gamma = 10$. (c) $\beta = 1, \gamma = 1$

Fig. 7: Separability of clusters w.r.t. attributes when $n = 1000, k_1 = 5$. The node colors represent the clusters.

for topology: the score is high when clusters are separated with each other. Figure 8 shows the average and deviation of the measures, where m is chosen from $\{5000, 10000, 15000, 20000\}$, n is fixed at 1000, and α is chosen from $\{0.1, 0.2, 0.3\}$. For all measures, the scores tend to be large when α is small, that is, separability is high. Graphs with high separability have obviously separated clusters so modularity should be high. The separated clusters consist of densely connected edges and, in this case, the average cluster co-efficient is high. As for the diameter, the small value of α causes fewer number of inter-edges, so the diameter becomes large when α is small. Overall, these results validate that acMark flexibly controls the separability of clusters.



(a) Modularity (b) Cluster co-efficient (c) Diameter

Fig. 8: Modularity, average cluster co-efficient, and diameter with various numbers of edges and various values of α. The number of nodes is fixed at 1000. The bars around the points (average) represent the standard deviations of five trials.

4.4 Q4: Applying clustering methods to generated graphs

Finally, we show that acMark can clarify the characteristics of clustering methods. We apply three types of clustering methods to generated graphs, an attributed graph clustering method which captures both effects of the topology and attributes (NAGC [20]), a topology-based graph clustering method (METIS [14]), and an attribute-based clustering method (kmeans).

We adopt several measures for evaluating clustering methods, modularity, entropy, NMI (Normalized Mutual Information), and ARI (Adjusted Rand Index) [29]. Entropy is a well-known measure for the diversity of attribute values.

NMI and ARI are popular measures for evaluating clustering quality with cluster labels.

The input parameters of the datasets we generated are described in Table 3. The results of the clustering methods may depend on initial values so we conduct five restarts for each experiment. We report the results of the average and the standard deviations.

Table 3: Generated datasets and their input parameters for the clustering experiments

ID	n	m	d	k_1	k_2	α	β	γ	att_{pow}	att_{uni}	att_{nor}
acMark1	1000	4000	10	5	5	0.2	10	1	0.0	0.0	0.9
acMark2	1000	4000	10	5	5	0.2	10	1	0.1	0.1	0.1
acMark3	1000	4000	10	5	5	0.1	10	1	0.1	0.1	0.1

Table 4 shows the clustering quality of the methods. The true clusters of acMark1 should highly depend on the attributes since most of the attributes follow normal distributions (See high value in att_{nor} column). The results show that NAGC and kmeans perform better than METIS in terms of NMI and ARI, so acMark clarifies these methods take the effect of the attribute values into community structure. Many attributes are independent from the community structure in acMark2 and acMark3 since 70% (= 1 - att_{pow} - att_{uni} - att_{nor}) of those attributes are randomly generated. Therefore, on acMark2, the attributes are less dependent on the community structure than on acMark1. The result of NAGC is kept high in terms of NMI and ARI while the result of kmeans largely descends. This implies that NAGC well captures both effects of the topology and the attributes. On acMark3, the clusters are more separated than acMark2 because the separability parameter α is smaller. Actually, the measurements of all methods are better than those on acMark2.

According to these results, we confirm that acMark controls the characteristics of generated graphs and it is useful to clarify the characteristics of clustering methods.

5 Related work

There is a rich literature on graph generation (e.g., [2, 10, 18]). In this section we review the most closely related methods to our proposal.

Generator for graphs with community structure. The GN-benchmark [10] is the first graph generator for evaluating community mining algorithms. It only supports 128 nodes and four groups. The LFR-benchmark [16] is the expansion of the GN-benchmark and it is a scalable graph generator. It assumes that the distributions of node degrees and community sizes should follow power-law

Table 4: The average and standard deviation (in parenthesis) of clustering quality. Oracle represents the evaluation by using the ground truth.

Dataset	Method	NMI	ARI	Modularity	Entropy
acMark1	Oracle	1.000(0.000)	1.000(0.000)	0.571(0.030)	−0.787(0.045)
$m = 4000$	NAGC	0.751(0.000)	0.805(0.002)	0.579(0.000)	−0.622(0.000)
$\alpha = 0.2$	METIS	0.553(0.043)	0.512(0.060)	0.534(0.033)	−0.588(0.068)
$att_{nor} = 0.9$	kmeans	0.620(0.028)	0.547(0.041)	0.348(0.048)	−0.768(0.062)
acMark2	Oracle	1.000(0.000)	1.000(0.000)	0.495(0.094)	−0.450(0.147)
$m = 4000$	NAGC	0.662(0.008)	0.741(0.001)	0.524(0.002)	−0.320(0.006)
$\alpha = 0.2$	METIS	0.475(0.080)	0.393(0.135)	0.497(0.046)	−0.268(0.085)
$att_{nor} = 0.1$	kmeans	0.321(0.063)	0.209(0.056)	0.167(0.026)	−0.394(0.047)
acMark3	Oracle	1.000(0.000)	1.000(0.000)	0.653(0.021)	−0.412(0.048)
$m = 4000$	NAGC	0.834(0.014)	0.890(0.010)	0.599(0.005)	−0.321(0.008)
$\alpha = 0.1$	METIS	0.669(0.044)	0.606(0.053)	0.611(0.030)	−0.290(0.046)
$att_{nor} = 0.1$	kmeans	0.406(0.099)	0.278(0.088)	0.182(0.052)	−0.346(0.063)

distributions. It is extended to generate synthetic graphs with overlapping communities [26] and hierarchical communities [28]. However, they do not support node attributes.

Generator for graphs with community structure and node attributes. There are few generators which take both community structure and node attributes into account. ANC [17] is a generator for attributed graphs with community structure. However, the community structure of generated graphs only depends on the attributes, so the user can not explicitly control community sizes and the effect of the topology to community structure. ANC is extended to generate dynamic graphs [4], it generates dynamic attributed networks with community structure, which follow the properties of real-world networks.

Recent graph generators. Although many graph generation methods have been developed, they are not capable of preserving important properties in real-world graphs. Also, they are not suitable for generating large-scale graphs due to their large space and time complexities. In order to preserve multiple properties of real-world graphs, there are two recent work as follows.

GraphRNN. Jiaxuan et al. [30] propose GraphRNN that captures the non-unique nature and the non-local dependencies in real world graphs. GraphRNN uses a scalable generative model for generating graphs. That is, GraphRNN learns how to generate graphs by training a representative set of graphs by leveraging RNN and generates graphs. The drawback is that it does not generate arbitrary graphs by controlling the types of distributions and also it does not directly treat community structure.

EvoGraph. Himchan et al. [24] presents EvoGraph, which is capable of upscaling given graphs, while preserving the major properties of the original graphs. The rationale of EvoGraph is that new edges are generated and attached to the original graph based on the preferential attachment mechanism in an effective and efficient way. However, an original graph is required as an input, so it is hard to control the graph properties as the user needs.

6 Concluding remarks

We proposed acMark, a scalable and general generator for attributed graphs with flexible community structure. The novel features of our method are threefold: 1) acMark flexibly controls the cluster separability between the topology and the attributes, 2) acMark supports various distributions for node degrees, cluster sizes and attribute values, and 3) acMark is a scalable generator: it scales linearly to the number of edges. We conducted extensive experiments which confirm these features. Throughout the experiments, we validated that acMark can generate massive graphs with various characteristics as the user wants.

Interesting directions for future work include extending acMark to generate directed graphs, overlapping communities, and hierarchical communities. Also it is an interesting topic to consider the graph generation problem as an inverse of graph clustering problem. We can design a single framework that treats both directions of the graph generation and graph clustering.

References

1. Akoglu, L., Tong, H., Meeder, B., Faloutsos, C.: Pics: Parameter-free identification of cohesive subgroups in large attributed graphs. In: Proceedings of ICDM. pp. 439–450 (2012)
2. Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G., Lemay, A., Advokaat, N.: gMark: Schema-driven generation of graphs and queries. IEEE Trans. Knowl. Data Eng. **29**(4), 856–869 (2017)
3. Ben-Dor, A., Shamir, R., Yakhini, Z.: Clustering gene expression patterns. Journal of computational biology **6**(3-4), 281–297 (1999)
4. Benyahia, O., Largeron, C., Jeudy, B., Zaïane, O.R.: Dancer: Dynamic attributed network with community structure generator. In: Proceedings of ECML PKDD. pp. 41–44 (2016)
5. Bonifati, A., Fletcher, G., Voigt, H., Yakovets, N.: Querying graphs. Synthesis Lectures on Data Management **10**(3), 1–184 (2018)
6. Brohee, S., Van Helden, J.: Evaluation of clustering algorithms for protein-protein interaction networks. BMC bioinformatics **7**(1), 488 (2006)
7. Flake, G.W., Lawrence, S., Giles, C.L., Coetzee, F.M.: Self-organization and identification of web communities. Computer **35**(3), 66–71 (2002)
8. Fortunato, S.: Community detection in graphs. Physics reports **486**(3-5), 75–174 (2010)
9. George, B., Kim, S., Shekhar, S.: Spatio-temporal network databases and routing algorithms: A summary of results. In: Proceedings of SSTD. pp. 460–477 (2007)
10. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proceedings of the national academy of sciences **99**(12), 7821–7826 (2002)
11. Huang, X., Li, J., Hu, X.: Accelerated attributed network embedding. In: Proceedings of SIAM SDM. pp. 633–641 (2017)

12. Huang, Z., Ye, Y., Li, X., Liu, F., Chen, H.: Joint weighted nonnegative matrix factorization for mining attributed graphs. In: Proceedings of PAKDD. pp. 368–380 (2017)
13. Jain, A., Zamir, A.R., Savarese, S., Saxena, A.: Structural-rnn: Deep learning on spatio-temporal graphs. In: Proceedings of CVPR. pp. 5308–5317 (2016)
14. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on scientific Computing **20**(1), 359–392 (1998)
15. Kulis, B., Basu, S., Dhillon, I., Mooney, R.: Semi-supervised graph clustering: a kernel approach. Machine learning **74**(1), 1–22 (2009)
16. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Physical review E **78**(4), 046110 (2008)
17. Largeron, C., Mougel, P.N., Rabbany, R., Zaïane, O.R.: Generating attributed networks with communities. PloS one **10**(4), e0122777 (2015)
18. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: An approach to modeling networks. Journal of machine learning research **11**(Feb), 985–1042 (2010)
19. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014)
20. Maekawa, S., Takeuch, K., Onizuka, M.: Non-linear Attributed Graph Clustering by Symmetric NMF with PU Learning. arXiv preprint arXiv:1810.00946 (2018)
21. Newman, M.E.: Modularity and community structure in networks. Proceedings of the national academy of sciences **103**(23), 8577–8582 (2006)
22. Newman, M.E.: Networks: An Introduction. Oxford University Press (2010)
23. Parimala, M., Lopez, D.: Graph clustering based on structural attribute neighborhood similarity (sans). In: Proceedings of IEEE ICECCT. pp. 1–4 (2015)
24. Park, H., Kim, M.S.: EvoGraph: an effective and efficient graph upscaling method for preserving graph properties. In: Proceedings of KDD. pp. 2051–2059 (2018)
25. Sahu, S., Mhedhbi, A., Salihoglu, S., Lin, J., Özsu, M.T.: The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. PVLDB (2017)
26. Sengupta, N., Hamann, M., Wagner, D.: Benchmark generator for dynamic overlapping communities in networks. In: Proceedings of ICDM. pp. 415–424 (2017)
27. Xu, Z., Ke, Y., Wang, Y., Cheng, H., Cheng, J.: A model-based approach to attributed graph clustering. In: Proceedings of SIGMOD. pp. 505–516 (2012)
28. Yang, Z., Perotti, J.I., Tessone, C.J.: Hierarchical benchmark graphs for testing community detection algorithms. Physical review E **96**(5), 052311 (2017)
29. Yeung, K.Y., Ruzzo, W.L.: Details of the adjusted rand index and clustering algorithms, supplement to the paper an empirical study on principal component analysis for clustering gene expression data. Bioinformatics **17**(9), 763–774 (2001)
30. You, J., Ying, R., Ren, X., Hamilton, W.L., Leskovec, J.: GraphRNN: a deep generative model for graphs. arXiv preprint arXiv:1802.08773 (2018)
31. Zhang, Z., Yang, H., Bu, J., Zhou, S., Yu, P., Zhang, J., Ester, M., Wang, C.: ANRL: Attributed network representation learning via deep neural networks. In: Proceedings of IJCAI. pp. 3155–3161 (2018)