Large-scale network motif analysis using compression

Peter Bloem¹ · Steven de Rooij²

Received: 13 September 2019 / Accepted: 14 May 2020 / Published online: 23 June 2020
© The Author(s) 2020

Abstract

We introduce a new method for finding *network motifs*. Subgraphs are motifs when their frequency in the data is high compared to the expected frequency under a *null model*. To compute this expectation, a full or approximate count of the occurrences of a motif is normally repeated on as many as 1000 random graphs sampled from the null model; a prohibitively expensive step. We use ideas from the minimum description length literature to define a new measure of motif relevance. With our method, samples from the null model are not required. Instead we compute the probability of the data under the null model and compare this to the probability under a specially designed alternative model. With this new relevance test, we can search for motifs by random sampling, rather than requiring an accurate count of all instances of a motif. This allows motif analysis to scale to networks with billions of links.

Keywords Motifs · Network analysis · Minimum description length

1 Introduction

Graphlets are small, induced subgraphs in a large network. *Network motifs* (Milo et al. 2002) are those graphlets that occur more frequently in the data than expected. To be able to conclude that such frequent subgraphs really represent meaningful aspects of the data, we must first show that they are not simply a product of chance. That is, a subgraph may simply be a frequent subgraph in *any* random graph: a subgraph is only a *motif* if its frequency is higher *than expected*.

Responsible editor: Ira Assent, Carlotta Domeniconi, Aristides Gionis, Eyke Hüllermeier.

✉ Peter Bloem
vu@peterbloem.nl
Steven de Rooij
s.de.rooij@math.leidenuniv.nl

¹ Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

² Mathematisch Instituut, Leiden University, Leiden, The Netherlands

This expectation is defined in reference to a *null model*: a probability distribution over graphs. We determine what the expected frequency of the subgraph is under the null model, and if the observed frequency is substantially higher than this expectation, the subgraph is a motif.

Unfortunately, there is usually no efficient way to compute the expected frequency of a subgraph under the null model. The most common approach generates a large number of random graphs from the null model and compares the frequencies of the subgraph in this sample to its frequency in the data (Milo et al. 2002). This means that any resources invested in extracting the motifs from the data must be invested again for every graph in the sample to find out which subgraphs are motifs.

We introduce an alternative method that does not require such sampling from the null model. Instead, we use two probability distributions on graphs: the null model p^{null} , and a distribution p^{motif} under which graphs with one or more frequent subgraphs have high probability. If a subgraph M of a given graph G allows us to show that $p^{\text{motif}}(G)$ is larger than $p^{\text{null}}(G)$, then M is a motif.

To design p^{motif} , we use the minimum description length (MDL) principle (Rissanen 1978; Grünwald 2007). We design a description method for graphs, a *code*, which uses the frequent occurrence of a potential motif M to create a compressed description of the graph. Our approach is analogous to compressing text by giving a frequent word a brief codeword: we describe M once and refer back to this description wherever it occurs.

By a commonly used correspondence between codes and probability distributions, we derive p^{motif} from this code, a distribution that assigns high probability to graphs containing motifs.

MDL is often employed in pattern mining in situations where other heuristics provide too many positives to be useful (Vreeken et al. 2011; Aggarwal and Han 2014, Chapter 8). In the case of subgraph mining, motif analysis aims to solve the same problem by a null-hypothesis-test-based heuristic. Here, we show that MDL, when applied rigorously, can also be interpreted as a hypothesis test, thus unifying the two frameworks, and providing a hypothesis test with many computational advantages over the traditional motif analysis approach.

Our method can speed up motif analysis in two ways. First, we only need to compute $p^{\text{null}}(G)$ and $p^{\text{motif}}(G)$ instead of counting subgraphs in many random graphs. Second, it removes the need for accurate subgraph counts. For a potential motif M , we only need to find enough occurrences in G to achieve the required level of compression; we never need an exact count of the occurrences of M in G . We simply sample random subgraphs until we find subgraphs with enough occurrences to yield a positive compression.¹ We show the following:

1. Our method can be used to analyze graphs with millions of links in minutes. We can analyze graphs with billions of links in under 9h on a single compute node.
2. Our method can retrieve motifs that have been injected into random data, even at low quantities.

¹ Such “optional stopping” is normally not allowed in statistical testing, but is valid in many compression-based statistical tests. See the discussion in Sect. 6.2.

3. In real data, the motifs produced by our method are as informative in characterizing the graph as those returned by the traditional method.

Our exposition in this paper is relatively concise. We refer the reader to Bloem and de Rooij (2018) for a brief, intuitive tutorial on using MDL for graph pattern analysis. All software is available open-source.²

1.1 Preliminaries

1.1.1 Traditional motif analysis

Many different algorithms, techniques and tools have been proposed for the detection of motifs, all based on a common framework, consisting of three basic steps:

1. Obtain a count f_M of the frequency of subgraph M in G .
2. Obtain or approximate the probability distribution over the number of instances F_M given that G came from a particular null model p^{null} .
3. If $p^{\text{null}}(F_M \leq f_M) \leq 0.05$, consider M a motif.

This was the approach proposed by Milo et al. (2002), where the phrase *network motif* was coined. We will refer to this general approach as the *traditional* method of motif analysis throughout this paper.

One problem with this approach is that it is very expensive to perform naively. Step 1 requires a full *graph census*, and since the probability in step 3 cannot usually be computed analytically, we are required to perform the census again on thousands of graphs sampled from the null model in order to approximate it.

Most subsequent approaches have attempted to improve efficiency by focusing on step 1: either by designing algorithms to get exact counts more efficiently (Koskas et al. 2011; Li et al. 2012; Khakabimamaghani et al. 2013; Meira et al. 2014), or by approximating the exact count. The most extreme example of the latter is Kashtan et al. (2004), which simply counts randomly sampled subgraphs. The complexity of this algorithm is *independent of the size of the data*, suggesting an exceptionally scalable approach to motif detection. Unfortunately, while the resulting ranking of motifs by frequency is usually accurate, the estimate of their total frequency is not (Wernicke 2005), which makes it difficult to build on this approach in steps 2 and 3. Other algorithms provide more accurate and unbiased estimates (Wernicke 2005; Ribeiro and Silva 2010; Bhuiyan et al. 2012; Jha et al. 2015; Slota and Madduri 2014; Paredes and Ribeiro 2015), but they do not maintain the scalability of the sampling approach. See Sect. 5 for additional related work.

We take an alternative route: instead of improving sampling, we change the measure of motif relevance: we define a new hypothesis test as an alternative to steps 2 and 3, which does not require an accurate estimate of the number of instances of the motif. All that is required is a set of some instances; as many as can be found with the resources available. This means that the highly scalable sampling approach from Kashtan et al. (2004) can be maintained.

² See <https://github.com/pbloem/motive> and <https://github.com/pbloem/motive-cls>.

1.1.2 Graphs

A graph G of size n is a tuple (V_G, E_G) containing a set of *nodes* (or *vertices*) V_G and a set of *links* (or *edges*) E_G . For convenience in defining probability distributions on graphs, we take V_G to be the set of the first n natural numbers. E_G contains pairs of elements from V_G . For the dimensions of the graph, we use the functions $n(G) = |V_G|$ and $m(G) = |E_G|$. If a graph G is *directed*, the pairs in E_G are ordered, if it is *undirected*, they are unordered. A *multigraph* has the same definition as a graph, but with E_G a multiset, i.e. the same link can occur more than once.

There are many types of graphs and tailoring a method to each one is a laborious task. Here, we limit ourselves to datasets that are (directed or undirected) *simple graphs*: graphs with no multiple links and no self-links.

1.1.3 Codes and MDL

MDL is built on a very precise correspondence between maximizing for probability (learning) and minimizing for description length (compression). We will detail the basic principle below. For more information, see Grünwald (2007) and Bloem and de Rooij (2018).

Let $\mathbb{B}$ be the set of all finite-length binary strings. Let $|x|$ represent the length of $x \in \mathbb{B}$. A *code* for a set of objects $\mathcal{X}$ is an injective function $f : \mathcal{X} \rightarrow \mathbb{B}$, mapping objects to binary code words. All codes in this paper are *prefix-free*: no codeword is the prefix of another. We will denote a *codeword length function* with the letter L , i.e. $L(x) = |f(x)|$. It is common practice to compute L directly, without explicitly computing codewords and to refer to L itself as a code.

The correspondence mentioned above follows from the *Kraft inequality*: for any probability distribution p on $\mathcal{X}$, there exists a prefix-free code L such that for all $x \in \mathcal{X}$: $-\log p(x) \leq L(x) < -\log p(x) + 1$. Inversely, for every prefix-free code L for $\mathcal{X}$, there exists a probability distribution p such that for all $x \in \mathcal{X}$: $p(x) = 2^{-L(x)}$ (Grünwald 2007, Section 3.2.1; Cover and Thomas 2006, Theorem 5.2.1). All logarithms are base 2.

To explain the intuition, note that we can transform a code L into a sampling algorithm for p by feeding the decoding function random bits until it produces an output. For the reverse, an algorithm like arithmetic coding (Rissanen and Langdon 1979) can be used.

As explained by Grünwald (2007, p. 96), the fact that $-\log p(x)$ is real-valued and $L(x)$ is integer-valued can be safely ignored and we may *identify* codes with probability distributions, allowing codes to take non-integer values. In some cases, we allow codes with multiple codewords for a single x , optionally indicating the choice for a particular codeword by a parameter a as $L(x; a)$. We say that $p(x) = \sum_a 2^{-L(x; a)}$ where the sum is over all code words for x .

Often, a distribution will be *parametrized*: $p_a^{\text{dist}}(x)$, where a represents some set of numbers that determine the shape of the distribution. We can either choose a fixed value of a before seeing the data, or choose a value based on the data, and add it to the code. The latter option is called *two part coding*: we choose some prior probability distribution on a , use it to store the value of a and then store x using the parametrized

code. This gives us a total codelength of $L(x) = L^{\text{prior}}(a) + L_a^{\text{dist}}(x)$. Note the correspondence to a Bayesian posterior:

$$2^{-L(x)} = p_a^{\text{dist}}(x) p^{\text{prior}}(a).$$

Encoding integers and sequences As part of our larger codes for graphs, we will often need to encode single natural numbers, and sequences of objects from a finite set. For positive natural numbers, we will use the code corresponding to the probability distribution $p^{\mathbb{N}}(n) = 1/(n^2 + n)$, and denote it $L^{\mathbb{N}}(n)$.³

For a sequence S of l elements from a finite set $\Sigma = \{e_1, e_2, \dots\}$, we use the code corresponding to a *Dirichlet-Multinomial* (DM) distribution (sometimes referred to as the *prequential plug-in code*⁴). Conceptually, the DM distribution models the following sampling process: we sample a probability vector p on $[1, |\Sigma|]$ from a Dirichlet distribution with parameter vector α , and then sample l symbols iid from the categorical distribution represented by p . The resulting probability distribution over sequences can be expressed as

$$\begin{aligned} p_{\alpha l \Sigma}^{\text{DM}}(S) &= \prod_{i \in [1, l]} p_{\alpha l \Sigma}^{\text{DM}}(S_i \mid S_{1:i-1}) \\ p_{\alpha l \Sigma}^{\text{DM}}(S_i = e_j \mid S') &= \frac{f(e_j, S') + \alpha_j}{|S'| + \sum_j \alpha_j} \end{aligned} \quad (1)$$

where $f(x, X)$ denotes the frequency of x in X . The DM model encodes each element from S , using the smoothed relative frequency of S_i in the subsequence $S_{1:i-1}$ preceding it. Thus the probability of a given symbol changes at each point in the sequence, based on how often it has been observed up to that point. We fix $\alpha_j = 1/2$ for all j (rather than use two-part coding) and omit it from the subscript.

Note that this code is parametrized with l and Σ . If these cannot be deduced from information already stored, they need to be encoded separately. When encoding sequences of natural numbers, we will have $\Sigma = [0, n_{\max}]$, and we only need to encode $n_{\max}$.

A useful property of the DM code is that it is *exchangeable*: if we permute the elements of S , the codelength remains the same. We often use this property by assuming

³ To include zero, we store $n + 1$ instead of n . To include negative numbers we add one bit ($L^{\mathbb{N}}(n) + 1$) to encode the sign. With abuse of notation, we refer to all such codes as $L^{\mathbb{N}}(n)$ when it is clear from context which is being used. Note that $L^{\mathbb{N}}(n)$ is only used for storing single integers like graph dimensions, so the specifics of this code have minimal impact.

⁴ This is a very common approach to encoding sequences in MDL for data mining (Faas and van Leeuwen 2019; Budhathoki and Vreeken 2015). Technically, any sequential model that predicts/encodes the current token by using an estimator over the preceding tokens is a prequential plugin code (Grünwald 2007, Section 9.1). Assuming a finite Σ and fixed k , and choosing (1) as an estimator leads to the DM distribution over the sequence as a whole. Equation (1) is also known as the Krichevsky–Trofimov estimator (Krichevsky and Trofimov 1981).

that the sequence is stored in some canonical ordering but computing the codelength using a given ordering instead.

2 MDL motif analysis

The principle behind our method is simple: given two distributions p^{null} and p^{alt} we can show that for any data G

$$p^{\text{null}} \left[-\log p^{\text{null}}(G) + \log p^{\text{alt}}(G) \geq b \right] \leq 2^{-b}. \quad (2)$$

The significance of this inequality follows from the central principle of MDL: that any probability distribution $p(x)$ can be translated to a prefix-free code which assigns x a codelength of $L(x) = -\log p(x)$ bits. Thus, (2) states that $L^{\text{null}}(G) - L^{\text{alt}}(G)$, the extent by which an alternative code beats the null code, is exponentially unlikely to be high, if the data came from the null distribution.

This means that (2), known as the *no-hypercompression inequality* space (Grünwald 2007, p103), can be interpreted as a null hypothesis test comparing codelengths: if we compress our data by a code corresponding to a chosen null model, and by *any other code*, the probability that the alternative compresses better is exponentially small, as a function of the number of bits gained. In other words, if we see a compression gain of b bits, we can reject the null model with a confidence of $1 - 2^{-b}$.

The consequence is not *just* that we now know a threshold for null hypothesis testing. The more important message is the quantity $L^{\text{null}}(G) - L^{\text{alt}}(G)$ gives us a grounded heuristic for evaluating pattern relevance. The relation between the compression achieved and the rejection threshold is *exponential*. Every bit saved with respect to the null model corresponds to a *halving* of the p -value.

To use this principle to test whether a particular subgraph M is a motif we proceed as follows. First, we compute the codelength $-\log p^{\text{null}}(G)$ of the data under a chosen null model (see Sect. 3). We then compress the graph using an ad-hoc motif code $L^{\text{motif}}(G; M)$ (described below), designed to exploit the presence of a particular motif M and to be efficiently computable. If the latter compresses better than the former by a certain number of bits, we may reject the null model. In general, we will simply report $L^{\text{null}}(G) - L^{\text{alt}}(G)$, over a set of motifs, rather than explicitly choosing a rejection threshold.

If we ensure that L^{alt} exploits a particular pattern in the data (like a motif) and L^{null} does not (and they otherwise operate similarly), the pattern must be what causes the compression, and the compression indicates that L^{alt} better captures the characteristics of the source of the data than L^{null} .

In our experiments, we will use not just a single null-model, but a lower bound B^{null} on the codelength for all models in a particular set of models. If we achieve sufficient compression to beat the bound, we can reject all models in the set in a single test. For a more extensive explanation of this method, and the various subtleties, see Bloem and de Rooij (2018).

Fig. 1 An illustration of the motif code. We store M once, and remove its instances from G , replacing them with a single, special node. Storing H and M , together with some “rewiring” annotation, is enough to reconstruct G

2.1 Motif code

We will now define the code used to compress graphs using a particular motif. We will use a given motif, and a list of its occurrences in the data to try to find an efficient description of the data. If this description allows us to reject a null model, we consider the motif interesting.⁵

Let $G[S]$ refer to the induced subgraph of G , for a given sequence $S = \langle S_1, \dots, S_r \rangle$ of nodes from V_G .

Assume that we are given a graph G , a potential motif M , and a list $\mathcal{I}^{\text{raw}} = \langle I_1, \dots, I_k \rangle$ of instances of M in G . That is, each sequence $I \in \mathcal{I}^{\text{raw}}$ consists of nodes in V_G , such that the induced subgraph $G[I]$ is equal to M . Note that $\mathcal{I}^{\text{raw}}$ need not contain all instances of M in the data. Additionally, sequences in $\mathcal{I}^{\text{raw}}$ may overlap, i.e. two instances may share one or more nodes.

We are also provided with a generic graph code $L^{\text{base}}(G)$ on the simple graphs. We will define three such graph codes in Sect. 3, which will serve both as base codes within our motif code and as null models in the hypothesis test. For now we will use L^{base} as a placeholder for any such code.

The basic principle behind our code is illustrated in Fig. 1.

Removing overlaps The first thing we need is a subset $\mathcal{I}$ of $\mathcal{I}^{\text{raw}}$ such that the instances contained within it do not overlap: i.e. for each I_a and I_b in $\mathcal{I}$, we have $I_a \cap I_b = \emptyset$.

In our code, an important factor for compression is the number of links an instance has to nodes outside the instance. We call this the *exdegree*.⁶ We greedily remove all overlapping instances, always removing those with highest exdegree.

⁵ Note that we have not “found evidence” for the motif as a pattern in any sense. We only use the hypothesis test as a *heuristic*, as is the case in all motif analysis. For a more in-depth discussion, see Sect. 6.1.

⁶ Unlike the in- and outdegree, the exdegree is not a property of a node, but of a subgraph.

The motif code We can now define the full motif code. It stores the following elements. We use a prefix-free code for each, so we can simply concatenate the individual codewords to get a complete description of G . See Algorithm 1 for a pseudocode description of the whole procedure.

subgraph First, we store the subgraph M using $L^{\text{base}}(M)$ bits.

template We then create the *template graph* H by removing the nodes of each instance $I \in \mathcal{I}$, except for the first, which becomes a specially marked node, called an *instance node*. The internal links of I —those incident to two nodes both in I —are removed and links to a node outside of I are rewired to the instance node.

instance nodes L^{base} does not record which nodes of H are instance nodes, so we must record this separately. Once we have recorded how many instance nodes there are, there are $\binom{n(H)}{|\mathcal{I}|}$ possible placements, so we can encode this information in $L^{\mathbb{N}}(|\mathcal{I}|) + \log \binom{n(H)}{|\mathcal{I}|}$ bits.

rewiring For each side of a link in H incident to an instance node, we need to know which node in the motif it originally connected to. Given some canonical order, we only need to encode the sequence W of integers $w_i \in [1, \dots, n(M)]$, where w_i is the node inside the motif that the i -th link incident to an instance node should connect to (with links connecting two instance nodes occurring twice).

multiple edges Since L^{base} can only encode simple graphs, we remove all multiple edges from H and encode them separately. We assume a canonical ordering over the links and record for each link incident to an instance node, how many copies of it were removed. This gives us a sequence R of natural numbers $R_i \in [0, r_{\max}]$ which we store by first recording the maximum value in $L^{\mathbb{N}}(\max(R))$ bits, and then recording R with the DM model.

insertions Finally, while H and M together let us recover a graph isomorphic to G , we cannot yet reconstruct where each node of a motif instance belongs in the node ordering of G . Note that the first node in the instance became the instance node, so we only need to record where to insert the rest of the nodes of the motif. This means that we perform $|\mathcal{I}|(n(M) - 1)$ such insertions. Each insertion requires $\log(t + 1)$ bits to describe, where t is the size of the graph before the insertion. We require $\sum_{t=n(H)}^{n(G)-1} \log(t + 1) = \log(n(G)!) - \log(n(H)!) bits to store the correct insertions.$

Pruning the list of instances Since our code accepts any list of motif instances, we are free to take the list $\mathcal{I}$ and remove instances before passing it to the motif code, effectively discounting instances of the motif. This can often improve compression. We sort $\mathcal{I}$ by exdegree and search for the value c for which storing the graph with only the first c elements of $\mathcal{I}$ gives the lowest codelength.

Algorithm 1 The motif code $L^{\text{motif}}(G; M, \mathcal{I}, L^{\text{base}})$. Note that the nodes of the graph are integers.

given:

a graph G , a subgraph M ,
a list $\mathcal{I}$ of instances of M in G , a code L^{base} on the simple graphs.

```

1   $b_{\text{subgraph}} \leftarrow L^{\text{base}}(M)$  subgraph
2
3  # replace each instance with a single node
4   $H \leftarrow \text{copy}(G)$ ,  $W = []$  template
5
6  for each  $I = \{m_1, \dots, m_{n(M)}\}$  in  $\mathcal{I}$ :
7
8      # We use  $m_1$  (the  $m_1$ -th node in  $G$ ) as the instance node
9      for each link  $l$  between a node  $n_{\text{out}}$  not in  $I$  and a node  $m_j$  in  $I$ :
10         if  $j \neq 1$ : add a link between  $n_{\text{out}}$  and  $m_j$ 
11          $W.\text{append}(j)$ 
12
13     remove all nodes  $m_i$  except  $m_1$ , and all incident links
14
15      $b_{\text{rewiring}} \leftarrow L^{\text{DM}}_{|W|, n(M)}(W)$  rewiring
16
17     # remove multiple edges from  $H$  and record the duplicates in  $R$ 
18      $R, H' \leftarrow \text{simple}(H)$ 
19      $b_{\text{template}} \leftarrow L^{\text{base}}(H')$ 
20      $b_{\text{multi-edges}} \leftarrow L^{\mathbb{N}}(\max(R)) + L^{\text{DM}}_{|R|, \max(R)}(R)$  multiple edges
21
22      $b_{\text{instances}} \leftarrow L^{\mathbb{N}}(|\mathcal{I}|) \log \binom{n(H)}{|\mathcal{I}|}$  instance nodes
23      $b_{\text{insertions}} \leftarrow \log(n(G))! - \log(n(H))!$  insertions
24
25 return  $b_{\text{subgraph}} + b_{\text{template}} + b_{\text{rewiring}} + b_{\text{multi-edges}} + b_{\text{instances}} + b_{\text{insertions}}$ 

```

We observe that the codelength L^{motif} as a function of c is often roughly unimodal, so we use a *Fibonacci search* (Kiefer 1953) to find a good value of c while reducing the number of times we have to compute the full codelength.

Finding candidate motifs and their instances We search for motifs and their instances by sampling, based on the method described by Kashtan et al. (2004)). Since we do not require accurate frequency estimates we simplify the algorithm: start with a set N containing a single random node drawn uniformly. Add to N a random neighbour of a random member of N , and repeat until N has the required size. Extract and return $G[N]$.

The size $n(M)$ of the subgraph is chosen before each sample from $U(n_{\min}, n_{\max})$, where U is the uniform distribution over an integer interval. This distribution is biased towards small motifs: since there are fewer connected graphs for small sizes, small graphs are more likely to be sampled. However, the method still finds motifs with many nodes, so we opt for this simple, ad-hoc method.

We re-order the nodes of the extracted graph to a canonical ordering for its isomorphism class, using the Nauty algorithm (McKay et al. 1981). We maintain a map from

each subgraph in canonical form to a list of instances found for the subgraph. After sampling is completed, we end up with a set of potential motifs and a list of instances for each, to pass to the motif code.

3 Null models

We will define three null models. For each, we first describe a parametrized model (which is not a code for all graphs). We then use this to derive a bound so that we can reject a set of null models, and finally we describe how to turn the parametrized model into a complete model to store graphs within the motif code.

Specifically, let $L_\theta^{\text{name}}(G)$ be a parametrized model with parameter θ . Let $\hat{\theta}(G)$ be the value of θ that minimizes $L_\theta^{\text{name}}(G)$ (the maximum likelihood parameter). From this we derive a bound $B^{\text{name}}(G)$ —usually using $B^{\text{name}}(G) = L_{\hat{\theta}(G)}^{\text{name}}(G)$. Finally, we create the complete model by two-part coding: $L^{\text{name}}(G) = L^\theta(\hat{\theta}(G)) + L_{\hat{\theta}(G)}^{\text{name}}(G)$.

When performing the null-hypothesis test, we will use the bound in place of the null model and the complete two-part code within the motif code. This ensures a conservative test; we always use a lower bound on the codelength for the null model, and an upper bound on the optimal codelength for the alternative model.

3.1 The Erdős–Rényi model

The Erdős–Rényi (ER) model is probably the best known probability distribution on graphs (Rényi and Erdős 1959; Gilbert 1959). It takes a number of nodes n and a number of links m as parameters, and assigns equal probability to all graphs with these attributes, and zero probability to all others. This gives us

$$L_{n,m}^{\text{ER}}(G) = \log \binom{n^2 - n}{m} \quad L_{n,m}^{\text{ER}}(G) = \log \binom{(n^2 - n)/2}{m}$$

for directed and undirected graphs respectively. We use the bound $B^{\text{ER}}(G) = L_{n(G),m(G)}^{\text{ER}}(G)$.

For a complete code on simple graphs, we encode n with $L^{\mathbb{N}}$. For m we know that the value is at most $m_{\max} = (n^2 - n)/2$ in the undirected case, and at most $m_{\max} = n^2 - n$ in the directed case, and we can encode such a value in $\log(m_{\max} + 1)$ bits (+1 because 0 is also a possibility). This gives us: $L^{\text{ER}}(G) = L^{\mathbb{N}}(n(G)) + \log(m_{\max} + 1) + L_\theta^{\text{ER}}(G)$ with $\theta = (n(G), m(G))$.

3.2 The degree sequence model

The most common null model in motif analysis is the *degree-sequence model*, also known as the *configuration model* (Newman 2010). For undirected graphs, we define the degree sequence of graph G as the sequence $D(G)$ of length $n(G)$ such that D_i is the number of links incident to node i in G . For directed graphs, the degree sequence is

a pair of such sequences $D(G) = (D^{\text{in}}, D^{\text{out}})$, such that D_i^{in} is the number of incoming links of node i , and D_i^{out} is the number of outgoing links.

The parametrized model $L_D^{\text{DS}}(G)$ The degree-sequence model $L_D^{\text{DS}}(G)$ takes a degree sequence D as a parameter and assigns equal probability to all graphs with that degree sequence. Assuming that G matches the degree sequence, we have $L_D^{\text{DS}}(G) = \log |\mathcal{G}_D|$ where $\mathcal{G}_D$ is the set of simple graphs with degree sequence D . There is no known efficient way to compute this value for either directed or undirected graphs, but various estimation procedures exist. We use an importance sampling algorithm from Blitzstein and Diaconis (2011) and Genio et al. (2010)

The bound $B^{\text{DS}}(G)$ We make the assumption that the degrees are sampled independently from a single distribution $p^{\text{deg}}(n)$ on the the natural numbers. This corresponds to a code $\sum_{D_i \in D} L^{\text{deg}}(D_i)$ on the entire degree sequence. Let $f(s, D)$ be the frequency of symbol s in sequence D . It can be shown that $B^{\text{deg}}(D) = \sum_{D_i \in D} f(D_i, D)/|D|$ is a lower bound for any such code on the degree sequence. This gives us the bounds $B^{\text{DS}}(G) = B^{\text{deg}}(D(G)) + L_{D(G)}^{\text{DS}}(G)$ and $B^{\text{DS}}(G) = B^{\text{deg}}(D^{\text{in}}(G)) + B^{\text{deg}}(D^{\text{out}}(G)) + L_{D(G)}^{\text{DS}}(G)$.

The complete model $L^{\text{DS}}(G)$ For the alternative model we need a complete code. First, we store $n(G)$ with $L^{\mathbb{N}}$. We then store the maximum degree and encode the degree sequence with the DM model. For undirected graphs we get:

$$L^{\text{DS}}(G) = L^{\mathbb{N}}(n(G)) + L^{\mathbb{N}}(d) + L_{\theta}^{\text{DM}}(D) + L_{D(G)}^{\text{DS}}(G) \\ \text{with } \theta = (n(G), \max(D))$$

and for directed graphs

$$L^{\text{DS}}(G) = L^{\mathbb{N}}(n(G)) \\ + L^{\mathbb{N}}(\max(D^{\text{in}})) + L_{\theta}^{\text{DM}}(D^{\text{in}}) \\ + L^{\mathbb{N}}(\max(D^{\text{out}})) + L_{\phi}^{\text{DM}}(D^{\text{out}}) + L_{D(G)}^{\text{DS}}(G) \\ \text{with } \theta = (n(G), \max(D^{\text{in}})), \phi = (n(G), \max(D^{\text{out}}))$$

3.3 The edgelist model

While estimating $|\mathcal{G}_D|$ can be costly, we can compute an upper bound efficiently. Assume that we have a directed graph G with n nodes, m links and a pair of degree sequences $D = (D^{\text{in}}, D^{\text{out}})$. To describe G , we might write down the links as a pair of sequences (F, T) of nodes: with F_i the node from which link i originates, and T_i the node to which it points. Let $\mathcal{S}_d$ be the set of all pairs of such sequences

satisfying D . We have v possibilities for the first sequence, and $\binom{m}{D_1^{\text{out}}, \dots, D_n^{\text{out}}}$ for the second. This gives us $|\mathcal{S}_D| = \binom{m}{D_1^{\text{in}}, \dots, D_n^{\text{in}}} \binom{m}{D_1^{\text{out}}, \dots, D_n^{\text{out}}} = m!m! / \prod_{i=1}^n D_i^{\text{in}}! D_i^{\text{out}}!$. We have $|\mathcal{S}_D| > |\mathcal{G}_D|$ for two reasons. First, many of the graphs represented by such a sequence pair contain multiple links and self-loops, which means they are not in $\mathcal{G}_D$. Second, the link order is arbitrary: we can interchange any two different links, giving a different pair of sequences, representing the same graph. A graph with no multiple edges, is represented by $m!$ different sequence-pairs.

To refine this upper bound, let $\mathcal{S}'_D \subset \mathcal{S}_D$ be the set of sequence pairs representing simple graphs. Since all links in such graphs are distinct, we have $|\mathcal{G}_D| = |\mathcal{S}'_D|/m!$. Since $|\mathcal{S}'_D| \leq |\mathcal{S}_D|$, we have

$$|\mathcal{G}_D| \leq \frac{m!}{\prod_{i=1}^n D_i^{\text{in}}! D_i^{\text{out}}!}.$$

In the undirected case, we can imagine a single, long list of nodes of length $2m$. We construct a graph from this by connecting the node at index i in this list to the node at index $m + i$ for all $i \in [1, m]$. In this list, node a should occur D_a times. We define $\mathcal{S}_D$ as the set of all lists such that the resulting graph satisfies D . There are $\binom{(2m)!}{D_1, \dots, D_n}$ such lists. We now have an additional reason why $|\mathcal{S}_D| > |\mathcal{G}_D|$: each pair of nodes describing a link can be swapped around to give us the exact same graph. This gives us:

$$|\mathcal{G}_D| \leq |\mathcal{S}'_D|/(2^m m!) = \frac{(2m)!}{2^m m! \prod_{i=1}^n D_i!}.$$

This gives us the following parametrized code for directed graphs:

$$L_D^{\text{EL}}(G) = \log m! - \sum_{i=0}^n \log D_i^{\text{in}}! - \sum_{i=0}^n \log D_i^{\text{out}}! \quad (3)$$

where $(D^{\text{in}}, D^{\text{out}})$ are the degree sequences of G , and for the undirected case:

$$L_D^{\text{EL}}(G) = \log(2m)! - \log m! - m - \sum_{i=0}^n \log D_i!. \quad (4)$$

For the bound and the complete model, we follow the same approach we used for the degree-sequence model.

4 Experiments

In all experiments, we report the *log-factor*:

$$B^{\text{null}}(G) - L^{\text{motif}}(G; M, \mathcal{I}, L^{\text{null}}).$$

This is a value in bits, indicating how much better the motif code compresses than the lower bound on the null model. If the log-factor is larger than 10 bits, we can interpret it as a successful hypothesis test at $\alpha = 0.001$.⁷

All experiments were executed on a single machine with a 2.60 GHz Intel Xeon processor (E5-2650 v2) containing 8 physical cores and 64 GB of memory. The memory and cores available to the program differ per experiment and are reported where relevant.

4.1 Recovering motifs from generated data

In our first experiment we will test some of the basic expected behaviours from our method: (1) a graph sampled randomly from a simple null model should contain many frequent subgraphs, but no motifs. (2) If a subgraph is manually inserted a number of times, we should be able to detect this graph as a motif. To test this, we sample a graph from a null model and inject k instances of a specific motif, in a way that corresponds broadly to the motif code.

We use the following procedure to sample an undirected graph with 5000 nodes and 10,000 links, containing k injected instances of a particular motif M with n' nodes and m' links. Let M be given (in our experiment M is always the graph indicated in red in Fig. 2).

1. Let $n = 5000 - (n' - 1)k$ and $m = 10000 - m'k$ and sample a graph H from the uniform distribution over all graphs with n nodes and m links.
2. Label k random nodes, with degree 5 or less, as instance nodes.
3. Let p^{cat} be a categorical distribution on $\{1, \dots, 5\}$, chosen randomly from the uniform distribution over all such distributions.
4. Label every connection between an instance node and a link with a random value from p^{cat} . Links incident to two instance nodes, will thus get *two* values.
5. Reconstruct the graph G from M and H .

On this sampled graph, we run our motif analysis. We run the experiment multiple times, with $k = 0$, $k = 10$ and $k = 100$, using the same subgraph M over all runs, but sampling a different H each time. For each value of k , we repeat the experiment 10 times. Per run, we sample only 5000 subgraphs. We use the ER null model, as that corresponds to the sampling procedure.

Figure 2 shows the results for the 21 possible connected simple graphs of size 5. This result shows that, on such generated data, the method behaves as expected in the following ways:

- If no motifs are injected, no subgraphs are motifs.
- Even for very low k , the correct motif is given a positive log-factor. Other subgraphs are shown to have very high frequencies, but a negative log factor.
- If the number of motifs is high ($k = 100$), the resulting log-factor increases.

⁷ A negative log-factor means that we do not have sufficient evidence to reject the null model, but a different experiment might yet achieve a positive log-factor.

Fig. 2 Motif analysis on generated data with k inserted motifs for all 21 simple connected graphs with 5 nodes. The middle row shows the number of non-overlapping instances found by the sampling algorithm for each potential motif. The bars show the average value over 10 randomly sampled graphs, with the same subgraph (shown in red) injected each time. Error bars represent the *range* (Colour figure online)

We can also see that once we insert 100 instances of the motif, two other subgraphs “become motifs”: in both cases, these share a part of the inserted motif (a rectangle and a triangle). This effect is not unique to our method, but occurs in all motif analysis.

The relative magnitude of the log factors provides a ranking within those subgraphs marked as motifs. In traditional motif methods, computing these relative magnitudes accurately requires very large samples of random graphs.

4.2 Motifs from real-world data

Next, we show how our approach operates on a selection of data sets across domains. Our main aim with this experiment is to show how the three null models influence the results. Specifically, to ascertain whether the edgelist model provides a reasonable approximation for the degree-sequence model (since the latter is expensive to compute). The following datasets are used.

kingjames (undirected, $n = 1773$, $m = 9131$) Co-occurrences of nouns in the text of the King James Bible (KONECT 2014; Römhild and Harrison 2007).

yeast (undirected, $n = 1528$, $m = 2844$) A network of the protein interactions in yeast, based on a literature review (Reguly et al. 2006).

physicians (directed, $n = 241$, $m = 1098$) Nodes are physicians in Illinois (KONECT 2015; Coleman et al. 1957).

citations (directed, $n = 1769$, $m = 4222$) The arXiv citation network in the category of theoretical astrophysics, as created for the 2003 KDD Cup (Gehrke et al. 2003).⁸

All data sets are simple (no multiple edges, no self-loops). In each case we take $5 \cdot 10^6$ samples with $n_{\min} = 3$ and $n_{\max} = 6$. We test the 100 motifs with the highest

⁸ We follow the procedure outlined in Carstens (2013): we include only papers before 1994, remove forward citations, and select the largest connected component.

Fig. 3 The results of the motif extraction on the 2 undirected networks (Colour figure online)

number of instances (after overlap removal), and report the log-factor for each null model. For the edgelist and ER models we use a Fibonacci search at full depth, for the degree-sequence model we restrict the search depth to 3. For the degree-sequence estimator, we use 40 samples and $\alpha = 0.05$ to determine our confidence interval. We use the same set of instances for each null model.

The results are shown in Figs. 4 and 3.

The negative log-factors in some of the results may appear surprising at first, since the motif code should never do worse than the null model by more than a few bits (as it can simply store everything with the base code). The reason it does worse is due to the use of a *bound*. The motif code must store the degree sequence explicitly, while the null model uses a lower bound for that part of the code.

Our first observation is that for the physician data set, there are no motifs under the degree-sequence null model. This is likely because the physicians network is too small: the use of a bound for the null model means that the alternative model requires a certain amount of data before the differences become significant. Note, however, that if we were to compare against a complete model (instead of the bound), a constant term would be added to all compression lengths under the null model. In other words, the ordering of the motifs by relevance would remain the same.

Fig. 4 The results of the motif extraction on the 2 directed networks (Colour figure online)

In both the kingjames and the yeast graphs, many motifs contain cliques, or otherwise highly connected subgraphs. This suggests the data contains communities of highly interconnected nodes which the null model cannot explain.

For the experiments in this section, the maximum Java heap space was set to 2 Gigabytes. The computation of the log-factor of each motif was done in parallel, as was the sampling for the degree sequence model, with at most 16 threads running concurrently, taking advantage of the 8 available (physical) cores.

These experiments took relatively long to run (ranging from 31 min to nearly 24 h). The bottleneck is the computation of the degree-sequence model. If we eliminate that, as we do in Sect. 4.4, we see that we can run the same analysis in minutes on graphs that are many orders of magnitude larger than these. Moreover, the plots show a reasonable degree of agreement between the EL model and the DS model, suggesting that the former might make an acceptable proxy. The next section tests whether the resulting motifs are still, in some sense, informative.

Fig. 5 A schematic illustration of the classification experiment. **a** We start with a classification task on simple, undirected graphs. **b** These graphs are reduced to 29-dimensional binary feature vectors. Each feature corresponds to an undirected, connected mini-graph of size 3, 4, or 5. **c** We apply a linear SVM to perform the classification (Colour figure online)

4.3 Evaluation of motif quality

In this section we evaluate on real-world data whether the subgraphs identified as motifs found by our method can be reasonably thought of as motifs. We provide an operational definition of a motif, which naturally suggests a classification experiment, and show that our method is significantly better than chance, and competitive with baseline methods.

The definition of what constitutes a motif is exceedingly vague: papers variously describe a motif as a “functional unit”, a “characteristic pattern” or a “statistically significant subgraph.” To operationalise this to something that we can test empirically, we will define a network motif as a subgraph that is *characteristic* for the full graph. That is, in some manner, the information that “ M is a motif for G ” should *characterize* G : it distinguishes G from the graphs for which M is not a motif, and that distinction should be meaningful in the domain of the data.

We operationalize “making a meaningful distinction in the domain of the data” as *graph classification*. If motif judgments (as binary features) can be used to beat a majority-class baseline by a significant amount, we can be sure that they make a meaningful distinction in the domain of the data.

We start with a set of undirected simple graphs, with associated classes. We then translate each graph into a binary feature vector using *only the motif judgments* of the algorithm under evaluation. We test all connected subgraphs of size 3, 4, and 5, giving us 29 binary features. If a simple classifier (in our case a linear SVM) can classify the graphs purely on the basis of these 29 motif judgments, the algorithm has succeeded in characterizing the graph. Figure 5 illustrates the method.

For those algorithms that succeed, the classification accuracy can be used to measure relative performance, although we should not expect high performance in a task as challenging as graph classification from just 29 binary features.

This approach—quantifying unsupervised pattern extraction through classification—was also used by van Leeuwen et al. (2006).

Our main aim is to establish that the resulting classifier performs better than chance. Our secondary aim is to show that we do not perform much worse than the traditional method.

For our purposes, we require classification tasks in a narrow range of sizes: the graphs should be small enough that we can use the traditional method without approximation, but large enough that our method has enough data to confidently reject a null bound.

In order to tune the graph classification tasks to our needs, we adapt them from three standard classification tasks on knowledge graphs, as described by Ristoski et al. (2016). The AIFB data describes a several research groups including their employees and publications, with the task of predicting which group an employee is affiliated to. The AM data (de Boer et al. 2013) describes the collection of the Amsterdam Museum with the task of predicting the category of artifacts. The BGS data is derived from the British Geological Survey, with the task of predicting rock lithogenesis.

In all these, the data is a single labeled, directed multigraph, and the task is to predict classes for a specific subset of nodes (the *instance nodes*). We translate the graph to an unlabeled simple graph by using the same nodes (ignoring their labels) and connecting them with a single undirected edge only if there are one or more directed edges between them in the original knowledge graph.

This gives us a classification task on the nodes of a single, undirected simple graph. We turn this into a classification task on *separate* graphs by extracting the 3-neighborhood around each instance node. To control the size of the extracted neighborhoods, we remove the h nodes with the highest degrees from the data before extracting the neighborhoods. h was chosen by trial-and-error, before seeing the classification performance, to achieve neighborhoods with around 1000–2000 nodes.

We now have a graph classification task from which we can create feature vectors as described above. For our method, we sample 100,000 subgraphs, with size 3, 4, 5 having equal probability and test the compression levels under the edgelist model. We judge a subgraph to be a motif if it beats the EL bound by more than $-\log \alpha$ bits with $\alpha = 0.05$.

Many methods for motif analysis have been published, but most are approximations or more efficient counting algorithms. Therefore, a single algorithm based on exact counts can act as a baseline, representing most existing approaches: we perform exact subgraph counts on both the data and 1000 samples from the DS model. The samples from the null model are taken using the Curveball algorithm (Strona et al. 2014; Carstens et al. 2016). We estimated the mixing time to be around 10,000 steps, and set the run-in accordingly. The subgraph counts were performed using the ORCA method.⁹ We mark a subgraph as a motif if fewer than 5% of the graphs generated from the DS model have more instances of the subgraph than the data does.¹⁰

For performance reasons (we are at the limits of what the traditional method allows), we use only 100 randomly chosen instances from the classification task. On these 100 instances, we perform fivefold cross-validation. To achieve good estimates, we then repeat the complete experiment, from sampling instances to cross-validation, 10 times. The classifier is a linear SVM ($C = 1$). For tasks with more than 2 classes, the one-against-one approach (Knerr et al. 1990) is used.

⁹ We created a Java implementation, available at <https://github.com/pbloem/orca>.

¹⁰ Note that the commonly used z-score method is seriously flawed, as discussed in Picard et al. (2008), so we do not use it here.

Fig. 6 The results of the classification experiment for the traditional method (*counting*) and ours (*motive*). The bars show the mean accuracy of ten runs (with cross-validation within each run). Error bars show a 95% confidence interval. The baseline is a majority-class classifier which ignores the features. The table shows the size of the data, the average size and number of links ($\bar{n}$, $\bar{m}$) of the instance graphs, the number of classes K and the number h of hubs removed (Colour figure online)

The results are shown in Fig. 6. For one data set, our method is significantly better, for another, the traditional approach is significantly better, and for one, the difference is not significant. While the performance of neither method is stellar as a classifier, the fact that both beat the baseline significantly, shows that at the very least, the motifs contain *some* information about the class labels of the instance represented by the graph from which the motifs were taken.

4.4 Large-scale motif extraction

Section 4.3 showed that our method can, in principle, return characteristic motifs, even when used with the edgelist null-model. Since the codelength under the EL model can be computed very efficiently, this configuration should be extremely scalable. To test its limits, we run several experiments on large data sets ranging from a million to a billion links.

In all experiments, we sample 1,000,000 subgraphs in total. We take the 100 most frequent subgraphs in this sample and compute their log-factors under the ER and EL models. We report the number of significant motifs found under the EL model.

Table 1 shows the results. The largest data set that we can analyse stored in-memory with commodity hardware is the wiki-nl data set. For larger data sets, we store the graph on disk. Details are provided below.

This experiment shows we can perform motif analysis on data in the scale of billions of edges with modest hardware. The sampling phase is ‘embarrassingly parallel’, and indeed, a good speedup is achieved for multithreaded execution. We also observe that the amount of motifs found can vary wildly between data sets. The twitter and friendster data sets are from similar domains, and yet for twitter, no motifs are found,

Table 1 The results of various runs of the algorithm on large data sets

Data	Disk	n/m	s	Mem.	t	Preload	Search	m
wiki-nl ^a		1 M/13 M	3–6	16 GB	16		7 m	8
			3–6	5 GB	16		13 m	8
			3–6	2 GB	1		25 m	8
			10	11 GB	1		2 h 41 m	0
wiki-en ^b	✓		3–6	1 GB	1	8 m	1 h 30 m	8
	✓	12 M/ 378 M	3–6	2 GB	1	4 h 58 m	6 h 6 m	10
	✓		8	8 GB	1		6 h 5 m	23
twitter ^c	✓	53 M/ 1963 M	3–6	6 GB	1	17 h 12 m	33 h 19 m	0
	✓		7	8 GB	1		54 h 26 m	0
friendster ^d	✓	68 M/2586 M	3–6	6 GB	1	42 h 37 m	45 h 2 m	68
	✓		3–6	56 GB	9		8 h 38 m	68
	✓		10	7 GB	1		35 h 7 m	57

The s column indicates the sizes of motifs that were sampled. The t column shows the number of threads allowed to run concurrently. The memory column indicates the maximum heap space allowed for the Java Virtual Machine. The m column indicates how many significant motifs were returned (under the EL model)

^a KONECT (2016d), Preusse et al. (2013), multiple links were removed

^b KONECT (2016c), Auer et al. (2008), self-loops were removed

^c KONECT (2016b), Kwak et al. (2010)

^d KONECT (2016a)

by a wide margin,¹¹ whereas for friendster the majority of the subgraphs are motifs. What exactly causes the difference in these data sets is a matter of future research. The bottom row of Fig. 8 shows numbers of motifs found for 30 more graphs in various domains.

With large data, using the full parallelism available is not always the best option. There is a trade-off between maximizing concurrency and avoiding garbage collection. The second line for the friendster data shows the fastest runtime (using the maximum stable heap space) which used 9 concurrently running threads (with 16 logical cores available).

We also show that our method can scale to larger motifs, often with a modest increase in resources. This is highly dependent on the data, however. On the twitter data, sampling motifs larger than 7 did not finish within 48 hours. This may be due to an incomplete implementation of the Nauty algorithm: the data may contain subgraphs that take a long time to convert to their canonical ordering. A more efficient canonization algorithm (like the complete Nauty) could improve performance. However, as the results show, some data allows for fast analysis on larger motifs.

Disk-based graphs For very large graphs, in-memory storage is insufficient, and we must store the data on disk. We take the following approach. The graph is stored in two lists, as it is in the in-memory version. The first, the *forward list*, contains at index i a

¹¹ The EL model compressed better than the motif model by millions of bits in all cases.

Fig. 7 A scatterplot of previously published motif analyses. Note the logarithmic scale. Results are taken from references (Bhuiyan et al. 2012; Hočevár and Demšar 2014; Kashtan et al. 2004; Milo et al. 2004; Picard et al. 2008; Ribeiro and Silva 2010; Carstens 2013; Milo et al. 2002; Wang et al. 2014; Li et al. 2012; Meira et al. 2014; Paredes and Ribeiro 2015; Slota and Madduri 2013; Wang and Ramon 2012; Wernicke 2005). **Full motif** analyses are those where the number of motifs is counted exactly on the data, and on a random ensemble. **Approximate motif** analyses are those where the count is approximated by sampling. A census performs the count, but no hypothesis test (which would require a count on many samples from the null model). We place our method somewhere between the full motif analysis and the approximate motif analysis: we make certain changes to the notion of a motif to achieve scalability, but our hypothesis test is fully correct (i.e. a conservative test, rather than an approximation) (Colour figure online)

sorted list of integers j for all links (n_i, n_j) that exist: i.e. a list of outgoing neighbors of n_i . The second, the *backward list*, contains lists of incoming neighbors for each node. The data is stored on disk in a way that allows efficient random access (using the MapDB database engine).¹²

For large graphs, converting a file from the common edgelist encoding (a file with a line for each link, encoded as a pair of integers) to this format can take considerable time, but this needs to be done only once, so we show the preloading and analysis times separately. Preloading the graph is done by performing a disk-based sort of the edgelist-encoded file, on the first element of each pair, loading the forward list, sorting again by the second element, and loading the backward list. This minimizes random disk access as both lists can be filled sequentially in one pass.

We only require one pass over the whole data to compute the model parameters (eg. the degree sequence). For the sampling and the computation of the log factors only relatively small amounts of random access are required.

For disk-based experiments, we limit the total number of rewritten links in the template graph to 500,000, to limit memory use.

¹² <http://www.mapdb.org/>.

Comparison to alternative methods These results cannot be compared one-on-one to results from the literature: the protocol we follow differs in key places from the standard protocol envisioned for motif analysis. First, we do not check *all* motifs of a given size, we use sampling frequency to make a preselection of candidates. Second, we focus on different statistics to determine what constitutes a motif. To still provide a broad sense of scale, we plot the sizes of graphs subjected to motif analysis in existing papers, together with our own in Fig. 7. The collected data is available.¹³

Note that there is a marked difference between the motif experiments and the census experiments. We suggest that this is not solely due to the extra cost of repeating the census on the random ensemble, but also due to the cost of just sampling the random ensemble. Such sampling is usually done through an MCMC method like the switching or the curveball algorithm. Such methods not only require a full copy of the data to be made for each sample, they also require a run-in of random transitions, until a proper mix is achieved. This mixing time increases with graph size, which means that even if the approximate census can be performed in constant time, producing the random ensemble becomes a bottleneck.

4.5 Computational complexity

Motif code computation The computational complexity of Algorithm 1 as described here, depends on intricate manipulation of a copy of the graph G . This makes the computational complexity at least $O(m)$, since a copy of the graph is required.

However, by creating implementations that are specific to the base code used, we can make this operation more efficient: the computation of $L_{\text{base}}(H)$ is usually the dominant operation, and for the codes described here, we do not need the whole graph H' to compute this codelength, only certain descriptive statistics (the dimensions in the case of the ER model and the degree sequence in the case of the DS and EL models). We will illustrate the case for the EL model.

Proposition 1 *Let $\text{inc}(I)$ be the set of all links incident to only one node in subgraph I , and let $\text{inc}(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} \text{inc}(I)$. If the EL code is used as a base, the motif code for a graph G and a motif M with instances $\mathcal{I}$ can be computed in time $O(n(G) + |\mathcal{I}| \cdot m(M) + |\text{inc}(\mathcal{I})|)$.*

Proof Assume that the graph is stored as a forward and backward list as described in Sect. 4.4 (or one such list in the case of an undirected graph). We first require an array representation of the degree sequence in memory.

If the graph is in memory, this can be achieved in $O(1)$ by creating an object that refers to the node lists dynamically. If the graph is stored on disk, this will take $O(n(G))$. Call the resultant object D . If multiple motifs are tested for one graph, this part of the code can be amortized.

We can then move through the motif's instances $\mathcal{I}$ and work out how the removal of each instance from the graph, replacing it by an instance node, transforms the degree sequence of the graph. For links entirely within an instance, the degrees of

¹³ <https://github.com/pbloem/motive/blob/master/src/main/resources/data/motif-experiments.csv>.

Fig. 8 Top: Motif analysis runtimes for 30 graphs from the KONECT repository (Kunegis 2013). Dotted lines show a linear fit in log space. The legend shows the slope. Middle: Sampling runtimes in the same experiment. Bottom: Number of motifs in the 100 candidate subgraphs. Error bars in the top and middle rows show the 95% confidence interval over 10 repeats. For the bottom row, they show the range of the data (Colour figure online)

both incident nodes are decremented. For a link with one side inside the instance, the degree of the inside node is decremented and the degree of the first node of the instance (which becomes the instance node) is incremented. If this particular link has been created already, the node degree is not incremented, and the extra link is stored separately, in dictionary R . That is, the modified degree sequence is that of H' .

Since D can take up a lot of memory for large graphs, we do not copy it, but instead create a dynamic view based on the increments and decrements required. This process takes $O(|\mathcal{I}| \cdot m(G) + \text{inc}(\mathcal{I}))$ when implemented as a simple loop over all links in all instances.

Given the resulting degree sequence D' of graph H' , we can compute $L_{\text{EL}}(H')$ in $O(m(G))$ by a simple loop over the degrees.

The rest of the code can be computed straightforwardly, as described in Algorithm 1: the rewiring bits can be computed in $O(|\mathcal{I}| \times m(M))$, by a simple loop over all links in all instances. The code for the multiple edges can be computed with a loop over all elements in R , which in the worst case has size $O(\text{inc}(\mathcal{I}))$. The codelength for the instance nodes and insertions can both be computed in $O(1)$.

Summing up these terms, and ignoring constant factors, we get a time complexity of $O(n(G) + |\mathcal{I}| \cdot m(M) + \text{inc}(\mathcal{I}))$. $\square$

This implementation was designed under the assumption that the graph is large, and relatively few instances are found, so that the second and third terms become negligible. If this assumption is invalid (for example if the instances cover most of the graph), then other implementations may be preferable.

Sampling For the sampling process (finding motifs and their instances), the computational complexity is more difficult to establish. Random walks on the graph can be performed in constant time (assuming constant-time random access), suggesting that, in theory, the sampling phase is independent of the number of nodes and links. The Nauty algorithm, however, has worst-case exponential time complexity (in the size of the subgraph), with efficient behavior for various subclasses of graphs. This means that the *type* of subgraphs present in the data are likely the main factor determining the complexity.

Empirical validation To provide some insight into whether these conjectures hold, we ran the full experiment on 30 medium-sized graphs from the KONECT repository (Kunegis 2013). For each, we sampled 1,000,000 instances, and performed a motif test on the top 100 candidates, using the EL model. We separate the runtime into the *sampling phase*, and the *motif testing phase*.

Figure 8 shows the result. The pattern is noisy, but the motif testing phase admits a linear fit in $n(G)$ as expected. We fit a line to the logarithms of the values. The slope is close to one, suggesting that it is not unreasonable to expect linear scaling in both n and m .

The sampling, as expected, has a high variance between datasets, and low correlation with the size of the data. The four graphs for which sampling took the longest are large graphs, but these are also from the same domain (hyperlink graphs). It is not clear what causes this increase in runtime, since there are graphs of similar size for which sampling is much faster.

5 Related work

Subjective interestingness Motif analysis is a specific instance of a well established notion in data mining: that the subjective interestingness of a pattern can be quantified as unexpectedness given the user's beliefs, with the latter represented by a "background distribution" on the data. This principle is described in general terms by Kontonasis et al. (2012) and in an information theoretic framework by De Bie (2011). This framework has been applied to various types of graph pattern (though not recurring subgraphs). Specifically, it was used by van Leeuwen et al. (2016) for the case of dense subgraphs and by Adriaens et al. (2019) for the case of connecting trees.

Scaling motif analysis While most motif analysis focuses purely on the frequency on the subgraph, our statistic of compressibility under the motif code can be seen as a combination of several attributes. Specifically, properties that allow a motif to reject the null hypothesis are: a high frequency of *non-overlapping*¹⁴ instances, a large number of instances with few links to nodes outside the instance and a high number of internal links. In the graph mining literature, different notions of frequency of a subgraph are often called *support measures* (Wang and Ramon 2012). Using non-overlapping instances is known as *frequency concept 3* in Schreiber and Schwobbermeyer (2004).

Picard et al. (2008) provide an analytical solution to step 2 in Section 1.1.1, which allows the test to be performed without generating a random sample of networks. The constraint here is that the null model should be *exchangeable*. For the commonly used degree-sequence model, a sampling based-approximation is used.¹⁴

The complexity of the hypothesis test is independent of the size of the graph, but it is $O(k!^2)$ in the size k of the motif. While this method could, in principle, be used for very scalable motif detection (for small k , with exchangeable null models), we are not aware of any practical efforts to this effect.

MDL-based subgraph discovery The idea that compression can be used as a heuristic for subgraph discovery was also used in the SUBDUE algorithm (Cook and Holder 1994). Our approach uses a more refined compression method and we connect it explicitly to the framework of motif analysis. We also exploit the possibility that the MDL approach offers for a very scalable sampling algorithm, to replace the more restrictive beamsearch used in SUBDUE.

The principle of using MDL to find relevant subgraphs was also exploited by Koutra et al. (2015) and by Navlakha et al. (2008) to help practitioners interpret the structure of large graphs. TimeCrunch (Shah et al. 2015) provides an MDL based solution for the same problem in the dynamic graph setting. The SlashBurn algorithm (Lim et al. 2014) achieves strong graph compression through identification of a hierarchical community structure in graphs. The MDL framework is not explicitly used, but a similar effect is achieved: compressing the data leads to insight into its structure. Rosvall and Bergstrom (2007) used MDL principles to perform community detection.¹⁵

Motifs, subgraphs and graphlets The literature behind subgraph mining, graphlets and network motifs seems to have developed largely in parallel, independently working towards different goals. Subgraph mining tends to focus on datasets consisting of many small graphs, rather than one large graph. As noted in Aggarwal and Han (2014):

Defining the support of a subgraph in a set of graphs is straightforward, which is the number of graphs in the database that contain the subgraph. However, it is much more difficult to find an appropriate support definition in a single large graph [...].

There are some efforts in the subgraph mining literature to define new support measures, and other ways of efficiently counting frequent subgraphs. For the purposes of

¹⁴ The degree sequence model always seems to require approximate solutions. FANMOD (Wernicke 2005) requires the computation of the number of graphs with a given degree sequence and a specific subgraph at a particular location. They use the approximation from Bender and Canfield (1978), which is asymptotically correct, but provides no bounds for the case of a finite graph. For our algorithm, we require a similar estimate (the number of graphs with a particular degree sequence). We use the sampling-based methods from Blitzstein and Diaconis (2011) and Genio et al. (2010). These are slower, but they provide a clearer bound on the accuracy of the estimate, as described in Section 3.

¹⁵ Note that while community detection, graph summarization and dense subgraph mining are all about identifying interesting subgraphs, there is no requirement that the same subgraph structure recurs in different places in the graph, as there is in motif analysis.

this paper, we distinguish motif analysis from the broader field of subgraph mining as follows: *motif analysis refers to methods which aim to extract subgraphs characteristic for a single large graph, and which use the framework of hypothesis testing as a heuristic for this purpose*. Note that it is much harder to evaluate whether a method successfully returns *characteristic* subgraphs than it is to evaluate whether it returns *frequent* subgraphs. We provide one way to operationalise this definition in Section 4.3.

A related problem that is well-studied for the single-graph case is that of dense subgraph mining (Gionis and Tsourakakis 2015; Tsourakakis et al. 2013). In this setting, the aim is to find node clusters that are unusually densely connected.

For a general overview of recent work in graph pattern mining, we refer the reader to Aggarwal and Han (2014, Chapter 13).

6 Discussion

Several aspects of our methods deserve closer scrutiny. We will discuss these in detail in this section.

6.1 Significance testing as a heuristic

The defining principle behind motif analysis is that a null-hypothesis significance test (NHST) is used to determine whether a pattern is a motif or not. We have followed this philosophy to bring our work in line with existing methods, and to facilitate comparisons, but there is some confusion about the use of NHSTs in this way, which we would like to address here in detail. This discussion applies to all methods of motif analysis based on NHSTs.

The null hypothesis in question always refers to a probability distribution p^{null} over graphs, not over patterns. Therefore, the result of a successful null hypothesis rejection, is that we have proved (in a statistical sense) that p^{null} did not generate the data. We have not proved in any way that the motif that allowed for a successful NHST constitutes a characteristic or interesting pattern. For the latter purpose, the NHST functions purely as a pattern mining *heuristic*.

This may lead the reader to wonder why we insist on still framing our method NHST-based terms and referring to significance in the outcome of our experiments, if all this is purely a heuristic. To make things precise, let us define the motif-analysis heuristic as follows:

Given a strong null model p^{null} , a subgraph pattern that allows us to successfully reject p^{null} by null-hypothesis significance test is likely to be a meaningful or characteristic subgraph for the data as a whole.

The key here is that while we are computing a heuristic, not any kind of evidence, we are defining and motivating that heuristic in terms of an NSHT. We can thus reasonably expect to strengthen the heuristic by strengthening the power and the correctness of the NHST.

This shows a subtle advantage of our method over the traditional method. The latter is, for large graphs, always an *approximation* to a correct NHST, since the subgraph census is approximated. This means that for a given rejection threshold, the probability of a Type I error (a null hypothesis falsely rejected) may be larger or smaller than the chosen rejection threshold.

In our method, by contrast, the probability of a type I error is always strictly lower than the given p -value (our NHST is *conservative*). This provides a much safer approach to pattern mining, since adding more computation (e.g. sampling more subgraphs) to make the motif code stronger will only increase the set of identified motifs, never shrink it.¹⁶

6.2 Multiple testing and optional stopping

In most of our experiments, we presented log-factors for up to a hundred subgraphs. Should the reader take into account multiple testing when interpreting these results? Does the probability that one or more of these motifs is a “false positive” increase with the number of subgraphs checked?

This question highlights again the point raised in the previous subsection: the question we are trying to answer—is M a motif?—is not the question we are testing for. While we test multiple motifs, we use them to test only a *single* null-hypothesis. To perform a valid, single null-hypothesis test, we can test many motifs and select the best compressing one to represent the graph. The log-factor of that motif represents the significance with which we reject the hypothesis that the null model generated the data. This is a valid code: we can check as many motifs as we like before choosing the one we will use to compress the data with. Since it is a valid code, it is a valid, single hypothesis test, and there is no need to correct for multiple testing.¹⁷

To make this more precise, assume that we have some data G . The null model assigns codelength $L^{\text{null}}(G)$ and the motif code assigns $L^{\text{motif}}(G; M, \mathcal{I})$ for a particular motif M with instances $\mathcal{I}$. The motif code gives us many different ways to describe this graph (one code word for each motif M and subset $\mathcal{I}$ of its instances). To remove this inefficiency, we can define an idealized code $L^{\text{motif}*}(G)$ that sums the probability mass of all these codewords, so that it has only a single codeword per graph:

$$\begin{aligned} p^{\text{motif}}(G; M, \mathcal{I}) &= \exp_2 \left(-L^{\text{motif}}(G; M, \mathcal{I}) \right) \\ p^{\text{motif}*}(G) &= \sum_{M, \mathcal{I}} p^{\text{motif}}(G; M, \mathcal{I}) \\ L^{\text{motif}*}(G) &= -\log p^{\text{motif}*}(G) \end{aligned}$$

¹⁶ Note that use of the degree-sequence model requires an approximation for the codelength, which may cause the test to become not strictly conservative. Similarly, if we take the edgelist model as an approximation to the degree-sequence model, such errors may be introduced (but not if we see the test as a rejection of the edgelist model itself).

¹⁷ In some experiments (such as those in Sect. 4.2) we test multiple *null models* on a single dataset. In these cases, multiple testing *should* be taken in to account, but only for the multiple null models, not for the multiple motifs. A Bonferroni correction for using m models raises the acceptance threshold by $\log_2 m$ bits, which does not make a big difference for the numbers observed here.

The codeword that L^{motif^*} provides for G is smaller than any codeword provided by L^{motif} .

We can now use L^{motif^*} to perform a *single* test: if L^{motif^*} compresses better than L^{null} , then by the no-hypercompression inequality (2), we can reject the null model. We cannot compute L^{motif^*} , for any reasonable size of graph, but we *can* upperbound it. If we can find only *one* codeword that allows L^{motif} to compress better than L^{null} by the required margin, we have proved that L^{motif^*} does so as well.¹⁸ This allows us to test as many motifs as we like, for as many different instance-sets $\mathcal{I}$ as we like, and we may use exhaustive search or sampling to find these: the real test is whether L^{motif^*} compresses the data better than the null model.

Of course, in general practice, we are not actually interested in whether or not the null model is true. What we are investigating is whether a given *motif* is “true”. Does the general intuition of multiple hypothesis testing still apply? Does testing for more than one motif raise the probability of incorrectly identifying subgraphs as motifs? As shown in Sect. 4.1, even without multiple testing, we should expect false positives, since many subgraphs will overlap with the true motif. This can be expected in all motif analysis.

However, increasing the number of motifs checked, does not raise the amount of such false positives. This is because we analyze multiple motifs on a *single dataset*. The process generating the data is stochastic, but once the data is known, the log-factors of all motifs (under L^{motif^*}) are determined. Checking more subgraphs will provide a better lowerbound to the true number of motifs, but it will not put us in danger of an increased number of false positives.

A related problem is that of *optional stopping*. In normal NHST settings, sampling until the desired p -value is observed and then stopping, invalidates the test. In our setting, this is not the case: we may check whether we have compressed enough, and keep sampling subgraphs if not. We may also (as we do in our code) reject samples that do not contribute to the compression. The literature on optional stopping in Bayesian statistics and MDL (van der Pas et al. 2018) is technical, but in our case, the argument made above suffices: just as we may test different motifs, we may test different subsets of their instances.

6.3 The choice of null model

Theoretically, the choice of null model is a way to express the user’s subjective beliefs about the data. Practically, the choice of null model is more often a pragmatic one, based on computational trade-offs. In general, most motif analysis is performed with some approximation of the degree sequence model.

In both our method and the traditional framework there are strong computational requirements on the choice of a null model that limit the available options. In our case, the null model must provide a single codeword per graph, and be exactly computable. This rules out many of the tricks we used to make computation of the motif code more efficient.

¹⁸ As usual, if we do not find such a codeword, we do not know whether or not the null hypothesis can be rejected.

While the use of the degree-sequence model is largely a practical choice based on computational properties, we would like to note that this model subsumes many others. For instance, any model that is built on an assumption about the degree distribution of the graph (such as models for scale-free graphs) can be framed as a prior on the degree-sequence model. Most of these assume some iid distribution on the node degrees (like a power law) and so are still included in the family captured by the bound defined in Sect. 3.2.

In exploring alternative null models, we must note that the constraints on the null model are stronger than those on the alternative model. The null model code is bounded from below, and the alternative model is bounded from above. In our motif code, we use the length of one of many codewords, rather than the true probability (which would require a sum over all codewords). In the null model, such approximations are *not* appropriate, and each graph must have a single codeword. For the DS model, this makes the codelength so expensive to compute that computation becomes as costly as with the traditional method. Only when we use the edgelist model as a proxy, do we get better scalability.

Finally, we should note that in the MDL framework, null-models are not entirely subjective. Some models are fundamentally stronger than others at the expense of greater computational complexity (with the top of the hierarchy occupied by the model corresponding to the Kolmogorov complexity). Of course, if the null model subsumes the motif code in this way (that is, the null model can exploit motif occurrences as well as the motif code), nothing will register as a motif. Thus, the ideal null model is the strongest model that is unable to exploit the presence of motifs. In this sense the degree sequence and edgelist models provide good options for middle- and larger-scale graphs respectively.

6.4 Limitations and extensions

Unlike the traditional method, our current model does not allow for the detection of *anti-motifs*, subgraphs that occur significantly less than expected. For that purpose, another model would be required; one which exploits the property that a subgraph has a *lower* frequency than expected to compress the data. In theory, this is certainly possible: any such non-randomness can be exploited for the purposes of compression. We leave this as a matter for future research.

While our method allows fast motif search with the ER and EL null models, for other models, it can be as slow as the traditional method. To be scalable, our method requires a null model for which the probability of a graph under the null model can be quickly computed or approximated. The traditional method in contrast, requires a null model that allows efficient *sampling*.

It is still a complicated question whether graph motifs, from this method or any other, represent a useful insight into the structure of the data. We have provided some empirical and theoretical arguments for why our method should return relevant patterns. Ultimately, however, the proof of the pudding is in the eating, and even for domain experts, the relevance of a motif may be hard to evaluate.

Our aim is to extend the method to knowledge graphs. Hopefully, in this setting, the resulting motifs will be easier for domain experts to evaluate.

7 Conclusion

We have presented a novel method for finding network motifs. Our method has several advantages over the traditional approach, that make it suitable for large graphs.

- While the traditional approach requires a search for instances of a motif over many graphs sampled from the null model, ours requires such a search only on the data.
- The search does not need to find all instances of a motif, or make an accurate approximation of their frequency. We only require as many instances as can be found with the resources available. For large graphs, a relatively small number of instances may suffice to prove some motifs significant.
- This also allows us to retain a list of exactly those instances that made the subgraph a relevant motif. These can then be inspected by a domain-expert to establish whether the motif truly represents a “functional unit”.
- Given sufficiently strong evidence, a single test can be used to eliminate multiple null models.
- The resulting relevance can be used to compare the significance of motifs of different sizes in a meaningful way.
- If a motif allows for good compression, extremely low p -values can be reliably computed. Note that the traditional method requires more samples for lower p -values.

Finally, we hope that our approach is illustrative of the general benefit of MDL techniques in the analysis of complex graphs. In conventional graph analysis a researcher often starts with a structural property that is observed in a graph, and then attempts to construct a process which generates graphs with that structural property. A case in point is the property of scale-freeness and the preferential attachment algorithm that was introduced to explain it (Albert and Barabási 2002).

The correspondence between codes and probability distributions allows us instead to build models based on a description method for graphs. The trick then becomes to find a code that efficiently describes those graphs possessing the relevant property, instead of finding a process that is likely to generate such graphs. For many properties, such as cliques, communities, motifs or specific degree sequences, such a code readily suggests itself.

Acknowledgements We are grateful to the anonymous reviewers for their helpful comments. We thank Pieter Adriaans for valuable discussions. This publication was supported by the Dutch national program COMMIT, by the Netherlands eScience center, and by the Amsterdam Academic Alliance Data Science (AAA-DS) Program Award to the UvA and VU Universities.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If

material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Adriaens F, Lijffijt J, De Bie T (2019) Subjectively interesting connecting trees and forests. *Data Min Knowl Discov* 33(4):1088–1124
- Aggarwal CC, Han J (2014) *Frequent pattern mining*. Springer, Berlin
- Albert R, Barabási AL (2002) Statistical mechanics of complex networks. *Rev Mod Phys* 74(1):47
- Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives Z (2008) DBpedia: a nucleus for a web of open data. In: *Proceedings of international semantic web conference*, pp 722–735
- Bender EA, Canfield ER (1978) The asymptotic number of labeled graphs with given degree sequences. *J Combin Theory Ser A* 24(3):296–307
- Bhuiyan MA, Rahman M, Rahman M, Al Hasan M (2012) Guise: uniform sampling of graphlets for large graph analysis. In: *2012 IEEE 12th international conference on data mining*. IEEE, pp 91–100
- Blitzstein JK, Diaconis P (2011) A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Math* 6(4):489–522. <https://doi.org/10.1080/15427951.2010.557277>
- Bloem P, de Rooij S (2018) A tutorial on MDL hypothesis testing for graph analysis. *arXiv preprint arXiv:1810.13163*
- Budhathoki K, Vreeken J (2015) The difference and the norm—characterising similarities and differences between databases. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer, pp 206–223
- Carstens CJ (2013) Motifs in directed acyclic networks. In: *International conference on signal-image technology & internet-based systems, SITIS 2013, Kyoto, Japan, 2–5 Dec 2013*. IEEE, pp 605–611. <https://doi.org/10.1109/SITIS.2013.99>
- Carstens CJ, Berger A, Strona G (2016) Curveball: a new generation of sampling algorithms for graphs with fixed degree sequence. *arXiv preprint arXiv:1609.05137*
- Coleman J, Katz E, Menzel H (1957) The diffusion of an innovation among physicians. *Sociometry* 20:253–270
- Cook DJ, Holder LB (1994) Substructure discovery using minimum description length and background knowledge. *CoRR*, [arXiv:cs.AI/9402102](https://arxiv.org/abs/cs.AI/9402102)
- Cover TM, Thomas JA (2006) *Elements of information theory*, 2nd edn. Wiley, Hoboken
- De Bie T (2011) An information theoretic framework for data mining. In: *Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, pp 564–572
- de Boer V, Wielemaker J, van Gent J, Oosterbroek M, Hildebrand M, Isaac A, van Ossenbruggen J, Schreiber G (2013) Amsterdam museum linked open data. *Semant Web* 4(3):237–243
- Faas M, van Leeuwen M (2019) Vouw: geometric pattern mining using the mdl principle. *arXiv preprint arXiv:1911.09587*
- Gehrke J, Ginsparg P, Kleinberg J (2003) Overview of the 2003 KDD cup. *ACM SIGKDD Explor Newsl* 5(2):149–151
- Genio CID, Kim H, Toroczkai Z, Bassler KE (2010) Efficient and exact sampling of simple graphs with given arbitrary degree sequence. *CoRR*, [arXiv:1002.2975](https://arxiv.org/abs/1002.2975)
- Gilbert EN (1959) Random graphs. *Ann Math Stat* 30:1141–1144
- Gionis A, Tsourakakis CE (2015) Dense subgraph discovery: KDD 2015 tutorial. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, pp 2313–2314
- Grünwald P (2007) *The minimum description length principle*. The MIT Press, Cambridge
- Hočevár T, Demšar J (2014) A combinatorial approach to graphlet counting. *Bioinformatics* 30(4):559–565
- Jha M, Seshadhri C, Pinar A (2015) Path sampling: a fast and provable method for estimating 4-vertex subgraph counts. In: *Proceedings of the 24th international conference on world wide web*, ACM, pp 495–505
- Kashtan N, Itzkovitz S, Milo R, Alon U (2004) Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics* 20(11):1746–1758. <https://doi.org/10.1093/bioinformatics/bth163>

- Khakabimamaghani S, Sharafuddin I, Dichter N, Koch I, Masoudi-Nejad A (2013) Quatexelero: an accelerated exact network motif detection algorithm. *PLoS ONE* 8(7):e68073
- Kiefer J (1953) Sequential minimax search for a maximum. *Proc Am Math Soc* 4(3):502–506
- Knerr S, Personnaz L, Dreyfus G (1990) Single-layer learning revisited: a stepwise procedure for building and training a neural network. In: *Neurocomputing*. Springer, pp 41–50
- KONECT (2014) King James network dataset—KONECT. http://konect.uni-koblenz.de/networks/moreno_names
- KONECT (2015) Physicians network dataset—KONECT. http://konect.uni-koblenz.de/networks/moreno_innovation
- KONECT (2016a) Friendster network dataset—KONECT. <http://konect.uni-koblenz.de/networks/friendster>
- KONECT (2016b) Twitter (www) network dataset—KONECT. <http://konect.uni-koblenz.de/networks/twitter>
- KONECT (2016c) Wikipedia, english network dataset—KONECT. <http://konect.uni-koblenz.de/networks/dbpedia-link>
- KONECT (2016d) Wikipedia, nl (dynamic) network dataset—KONECT. <http://konect.uni-koblenz.de/networks/link-dynamic-nlwiki>
- Kontonasios KN, Spyropoulou E, De Bie T (2012) Knowledge discovery interestingness measures based on unexpectedness. *Wiley Interdiscip Rev Data Min Knowl Discov* 2(5):386–399
- Koskas M, Grasseau G, Birmelé E, Schbath S, Robin S (2011) Nemo: fast count of network motifs. *Book of abstracts for Journées Ouvertes Biologie Informatique Mathématiques (JOBIM)*, pp 53–60
- Koutra D, Kang U, Vreeken J, Faloutsos C (2015) Summarizing and understanding large graphs. *Stat Anal Data Min* 8(3):183–202. <https://doi.org/10.1002/sam.11267>
- Krichevsky RE, Trofimov VK (1981) The performance of universal encoding. *IEEE Trans Inf Theory* 27(2):199–207
- Kunegis J (2013) KONECT—The Koblenz network collection. In: *Proceedings international conference on world wide web companion*, pp 1343–1350. <http://userpages.uni-koblenz.de/~kunegis/paper/kunegis-koblenz-network-collection.pdf>
- Kwak H, Lee C, Park H, Moon S (2010) What is Twitter, a social network or a news media? In: *Proceedings International World Wide Web Conference*, pp 591–600
- Li X, Stones DS, Wang H, Deng H, Liu X, Wang G (2012) Netmode: network motif detection without nauty. *PLoS ONE* 7(12):e50093
- Lim Y, Kang U, Faloutsos C (2014) Slashburn: graph compression and mining beyond Caveman communities. *IEEE Trans Knowl Data Eng* 26(12):3077–3089
- McKay BD et al (1981) Practical graph isomorphism. Vanderbilt University Tennessee, US Department of Computer Science
- Meira LA, Máximo VR, Fazenda ÁL, Da Conceição AF (2014) Acc-motif: accelerated network motif detection. *IEEE/ACM Trans Comput Biol Bioinform (TCBB)* 11(5):853–862
- Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U (2002) Network motifs: simple building blocks of complex networks. *Science* 298(5594):824–827
- Milo R, Itzkovitz S, Kashtan N, Levitt R, Shen-Orr S, Ayzenshtat I, Sheffer M, Alon U (2004) Superfamilies of evolved and designed networks. *Science* 303(5663):1538–1542
- Navlakha S, Rastogi R, Shrivastava N (2008) Graph summarization with bounded error. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, pp 419–432
- Newman M (2010) *Networks: an introduction*. Oxford University Press, Oxford
- Paredes P, Ribeiro P (2015) Rand-fase: fast approximate subgraph census. *Soc Netw Anal Min* 5(1):1–18
- Picard F, Daudin JJ, Koskas M, Schbath S, Robin S (2008) Assessing the exceptionality of network motifs. *J Comput Biol* 15(1):1–20
- Preusse J, Kunegis J, Thimm M, Gottorn T, Staab S (2013) Structural dynamics of knowledge networks. In: *Proceedings international conference on weblogs and social media*
- Reguly T, Breitkreutz A, Boucher L, Breitkreutz BJ, Hon GC, Myers CL, Parsons A, Friesen H, Oughtred R, Tong A et al (2006) Comprehensive curation and analysis of global interaction networks in *Saccharomyces cerevisiae*. *J Biol* 5(4):11
- Renyi A, Erdős P (1959) On random graphs. *Publ Math* 6(290–297):5
- Ribeiro P, Silva F (2010) G-tries: an efficient data structure for discovering network motifs. In: *Proceedings of the 2010 ACM symposium on applied computing*. ACM, pp 1559–1566
- Rissanen J (1978) Modeling by shortest data description. *Automatica* 14(5):465–471

- Rissanen J, Langdon GG (1979) Arithmetic coding. *IBM J Res Dev* 23(2):149–162
- Ristoski P, de Vries GK, Paulheim H (2016) A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In: *International semantic web conference*. Springer, pp 186–194
- Römhild C, Harrison C (2007). <http://chrisharrison.net/projects/bibleviz/index.html>. Accessed 22 Sept 2014
- Rosvall M, Bergstrom CT (2007) An information-theoretic framework for resolving community structure in complex networks. *Proc Natl Acad Sci* 104(18):7327–7331
- Schreiber F, Schwobbermeyer H (2004) Towards motif detection in networks: frequency concepts and flexible search. In: *Proceedings International WSH network tools and applications in biology (NET-TAB'04)*, pp 91–102
- Shah N, Koutra D, Zou T, Gallagher B, Faloutsos C (2015) Timecrunch: Interpretable dynamic graph summarization. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, pp 1055–1064
- Slota GM, Madduri K (2013) Fast approximate subgraph counting and enumeration. In: *2013 42nd International conference on parallel processing (ICPP)*. IEEE, pp 210–219
- Slota GM, Madduri K (2014) Complex network analysis using parallel approximate motif counting. In: *Parallel and distributed processing symposium, 2014 IEEE 28th international*. IEEE, pp 405–414
- Strona G, Nappo D, Boccacci F, Fattorini S, San-Miguel-Ayanz J (2014) A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. *Nat Commun* 5:1–9
- Tsourakakis C, Bonchi F, Gionis A, Gullo F, Tsiarli M (2013) Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: *Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, pp 104–112
- van Leeuwen M, Vreeken J, Siebes A (2006) Compression picks item sets that matter. In: Fürnkranz J, Scheffer T, Spiliopoulou M (eds) *Knowledge discovery in databases: PKDD 2006, 10th European conference on principles and practice of knowledge discovery in databases*, Berlin, 18–22 Sept 2006, *Proceedings*, Springer, *Lecture notes in computer science*, vol 4213, pp 585–592. https://doi.org/10.1007/11871637_59
- van Leeuwen M, De Bie T, Spyropoulou E, Mesnage C (2016) Subjective interestingness of subgraph patterns. *Mach Learn* 105(1):41–75
- van der Pas S, Grünwald P et al (2018) Almost the best of three worlds: risk, consistency and optional stopping for the switch criterion in nested model selection. *Stat Sin* 28:25
- Vreeken J, Van Leeuwen M, Siebes A (2011) Krimp: mining itemsets that compress. *Data Min Knowl Discov* 23(1):169–214
- Wang Y, Ramon J (2012) An efficiently computable support measure for frequent subgraph pattern mining. In: *Machine learning and knowledge discovery in databases*, pp 362–377
- Wang P, Lui J, Ribeiro B, Towsley D, Zhao J, Guan X (2014) Efficiently estimating motif statistics of large networks. *ACM Trans Knowl Discov Data (TKDD)* 9(2):8
- Wernicke S (2005) A faster algorithm for detecting network motifs. In: Casadio R, Myers G (eds) *Algorithms in bioinformatics, 5th international workshop, WABI 2005, Mallorca, Spain, 3–6 Oct 2005, Proceedings*, Springer, *Lecture notes in computer science*, vol 3692, pp 165–177. https://doi.org/10.1007/11557067_14