Finding Network Motifs in Large Graphs using Compression as a Measure of Relevance

Peter Bloem  
Knowledge Representation and Reasoning Group  
VU University Amsterdam  
De Boelelaan 1105, 1081 HV Amsterdam, NL

Steven de Rooij  
Mathematical Institute  
University of Leiden  
Niels Bohrweg 1, 2333 CA Leiden, NL

Abstract

We introduce a new method for finding network motifs: interesting or informative subgraph patterns in a network. Current methods for finding motifs rely on the frequency of the motif: specifically, subgraphs are motifs when their frequency in the data is high compared to the expected frequency under a null model. To compute this expectation, the search for motifs 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 (MDL) literature to define a new measure of motif relevance, and a new algorithm for detecting motifs. Our method allows motif analysis to scale to networks with billions of links, while still resulting in informative motifs.

Keywords: graphs, minimum description length, unsupervised learning, pattern mining, network motifs

1. Introduction

Graphlets are small induced subgraphs a large network. Network motifs (Milo et al., 2002) are those graphlets that occur more frequently in the data than expected. Graphlets and network motifs are used for many purposes, for instance: as features in machine learning (Shervashidze et al., 2009; Wu et al., 2011), as candidates for “functional units” of the system represented by the graph (Milo et al., 2002), as a method to reduce the complexity of the graph (Koutra et al., 2015) or as a building block in community detection (Arenas et al., 2008). 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.

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. If the frequency is lower than expected, the subgraph is called an anti-motif.

The choice of null model is an important aspect of the analysis. Consider the case explored by Carstens (2013), where the data is directed and acyclic (for example, a citation graph). If the null model allows graph cycles, then any small graph containing a cycle will be an anti-motif: it is likely to occur in the data, but it does not occur at all. Such a result shows only that the data is acyclic, and obscures any deeper structure. A model for random acyclic graphs will fit the data better, and will allow us to explore deeper structure. This shows the role of the null model: the better we model the known structure in the data, the better we can expose the unknown structure.

However, there is usually no efficient way to compute the expected frequency of a subgraph under a null model. The most common approach is to generate a large number of random graphs, say 1000, from the null model and compare the frequencies of the subgraph in the graphs 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 1000 times to find out which subgraphs are motifs.

We introduce an alternative method that does not require us to repeat the motif search on samples from the null model. To this end, we use two probability distributions on the set of all graphs: the null model $p^{\text{null}}$, and a distribution $p^{\text{motif}}$ 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. Section 2 explains the principle and its theoretical justification.

To design $p^{\text{motif}}$, we use the Minimum Description Length (MDL) Principle (Rissanen, 1978; Grünwald, 2007). It can be shown that any description method $L$, a code, corresponds to a probability distribution $p^L$ in such a way that a graph $G$ with a short description under $L$ will have a high probability under $p^L$. This correspondence is detailed in the preliminaries. Thus, if we design a code that exploits recurring subgraphs to achieve a compressed description, we also get a probability distribution that assigns data with recurring subgraphs a high probability. In brief, our code accomplishes this by describing the motif only once, and referring back to this description wherever the motif occurs (analogous to compressing text by assigning common words a short codeword). Since we do not need to describe the motif explicitly for every occurrence, graphs with a high frequency of a certain motif will have a short description length, and thus a high probability. The code is described in Section 3.

We show the following:

- On commodity hardware, our method can be used to analyze graphs with millions of links in minutes. We can analyze graphs with billions of links in under 9 hours on a basic compute node, and 45 hours on hardware equivalent to a single laptop (Section 5.4).

1. Commonly, all possible graphlets of a given size are checked. Depending on the task, the full list of motif scores are used as feature vector representing the graph, or the top-$k$ motifs are used by a domain expert as candidates for further analysis. Since each motif is judged by a hypothesis test, the reader may wonder whether this approach constitutes multiple testing. As discussed in Section 6, however, we are using the hypothesis test merely as a pattern-mining heuristic, not as a statistical test.
• The method can be used for motifs with at least 10 nodes, although space and time complexity do increase greatly for larger motif sizes (Section 5.4).

• While our method uses a definition of motif that differs slightly from the most common definition, it can still retrieve motifs that have been injected into the data, even at relatively low quantities (Section 5.1).

• In real data, the judgments on whether a subgraph is a motif or not produced by our method are on a par with the traditional method, as judged by classification accuracy on a graph classification task (Section 5.3).

Contrast this with most data sets used for motif analysis in the literature, which contain in the order of 1000 to 10,000 links. Hočevar and Demšar (2014) report a complete subgraph count on a graph with 137,775 links, the largest we have found, but comparison to a null model is not performed.

The paper is structured as follows. The next two subsection discuss related work and preliminary information on graphs and MDL. Section 2 provides an in-depth explanation of the method we use to perform hypothesis testing through MDL. In Section 3, we detail our motif model. In Section 4 we detail the three null-models we will use to test its performance. In Section 5 we detail briefly how we search for motifs, and select candidates, and then report the results of four experiments to validate the claims made above.

All software used in this paper is available open source.  

1.1 Related work

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 graph \( G \).

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 \( p^{null}(F_M \leq f_M) \leq \alpha \) (usually with \( \alpha = 0.05 \)), we consider \( M \) a motif.

This was the approach proposed in Milo et al. (2002), the paper that coined the phrase “network motif.” 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 extreme scalability of the sampling approach.

We take an alternative route: instead of improving the 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.

The frequency of a motif is not the only statistic we can use. We might, for instance, compare the level of clustering among the instances with the value expected under the null model. This is relevant, because our compression method can be seen as a combination of several statistics. Specifically, properties that allow a motif to reject the null hypothesis are: a high frequency of non-overlapping instances, a large number of instances with few links to nodes outside the instance and a high number of internal links. While this deviates from the more common statistic of pure frequency, we believe these are all properties that fit the intuitive idea of a motif. In the graph mining literature, different notions of frequency of a subgraph are often called support measures (Wang and Ramon, 2012).

Picard et al. (2008) provide an analytical solution, which can perform the hypothesis test 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. The complexity of the hypothesis test is independent of the size of the graph, but it is is $O(k^2)$ in the size $k$ of the motif. While this method could 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.

The idea that compression can be used as a heuristic for subgraph discovery was also used in the SUBDUE algorithm by Cook and Holder (1994). Our approach uses a more refined compression method and we connect it explicitly to the framework of motif analysis and the use of hypothesis tests. We exploit the possiblity that the MDL approach offers, for a very scalable sampling algorithm, to replace the more restrictive beamsearch used in SUBDUE. In itemset mining, the use of MDL has a long and succesful history, as detailed by Aggarwal and Han (2014, Chapter 8).

4. Using non-overlapping instances is known as frequency concept 3 in (Schreiber and Schwobbermeyer, 2004).

5. 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); Genio et al. (2010). These are slower, but they provide a clearer bound on the accuracy of the estimate, as described in Section 4.1 and the supplementary materials.
The literature behind graph mining, graphlets and network motifs seems to have developed largely in parallel, repeating much of the same ground. While graph mining tends to focus on datasets consisting of many small graphs, rather than one large graph, there are efforts focusing on the latter. For a good overview of recent work, we refer the reader to Aggarwal and Han (2014, Chapter 13). Research on graphlets likewise tends to focus on datasets where each instance is represented by a separate graph, usually in a classification setting Shervashidze et al. (2009). However, recently, a great deal of machine learning research has emerged that deals with the case where the dataset consists of a single graph, and the instances to be classified are a subset of its nodes (Pham et al., 2016). Where graphlet approaches tend to focus on the raw frequencies of subgraphs in the data, network motif research focuses on the frequency relative to the expectation under a null model. This results in per-subgraph judgments that are more informative: where in graphlet analysis we can only look at the frequency vector for all subgraphs, to compare to different graphs, in network motif analysis, we can take the “motif score” of a single subgraph and expect it to contain information about the dataset.

1.2 Preliminaries: Graphs and Codes

**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, $V_G$ is always 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: i.e. no link connects a node to itself and the same link cannot occur more than once. Probability distributions on the set of simple graphs are usually the most complex, so that we can trust that a method that works for simple graphs is easily translated to other types.

Two graphs $G$ and $H$ are isomorphic if there exists a bijection $f : N_G \rightarrow N_H$ on the nodes of $G$ such that two nodes $a$ and $b$ are adjacent in $G$ if and only if $f(a)$ and $f(b)$ are adjacent in $H$. If two graphs $G$ and $H$ are isomorphic, we say that they belong to the same isomorphism class $[G]$.

**Codes and MDL**  The intuition behind MDL is that compressing data and learning from it are much the same process: in both cases we are analyzing data, and finding patterns that are characteristic for the data, or patterns that represent the data well. This is not just an intuition, there exists a very precise correspondence between optimizing probability and optimizing description length. We will detail the basic principle below, and give an intuitive example of how we apply the method in the next section. For more details, we refer the reader to Grünwald (2007).

Let $\mathbb{B}$ be the set of all finite-length binary strings. We use $|b|$ to represent the length of $b \in \mathbb{B}$. Let $\log(x) = \log_2(x)$. A code for a set of graphs $\mathcal{G}$ is an injective function $f : \mathcal{G} \rightarrow \mathbb{B}$. All codes in this paper are prefix-free: no code word is the prefix of another. We will denote a codelength function with the letter $L$, i.e. $L(G) = |f(G)|$. It is common practice
to compute $L$ directly, without explicitly computing the codewords. In fact, we will adopt
the convention of referring to $L$ itself as a code.

A well known result in information theory is the association between codes and probability
distributions, implied by the Kraft inequality: for each probability distribution $p^*$ on
$G$, there exists a self-delimiting code $L^*$ such that for all $G \in G$: $-\log p^*(G) \leq L^*(G) <
-\log p^*(G) + 1$. Inversely, for each self-delimiting code $L^*$ for $G$, there exists a probability
distribution $p^*$ such that for all $G \in G$: $p^*(G) = 2^{-L^*(G)}$. For proofs, see (Grünwald, 2007,
Section 3.2.1) or (Cover and Thomas, 2006, Theorem 5.2.1). To explain the intuition, note
that we can easily transform a code $L^*$ into a sampling algorithm for $p^*$ by feeding the
decoding function random bits until it produces an output. To transform a probability
distribution to a code, techniques like arithmetic coding (Rissanen and Langdon, 1979) can be used.

As explained in Grünwald (2007, page 96), the fact that $-\log p^*(G)$ is real-valued and
$L^*(G)$ is integer-valued can be safely ignored and we may identify codes with probability
distributions, allowing codes to take non-integer values.

When we need to encode a single choice from a finite set $S$ of options, we will often use
the code with length $\log |S|$, corresponding to a uniform distribution on $S$.

2. Model Rejection by Codelength

In this section we provide an intuitive discussion of the principle behind our method. To
simplify the explanation, we will illustrate the idea on a simpler problem than motif analysis:
determining whether a single large clique in a graph is an unusual feature, or to be expected
under a particular null model. For certain null models, such as the Erdős-Renyi (ER) model,
the expected size of the largest clique is well understood. For other models, less so. Without
resorting to sampling a large number of models, how can we determine that a particular
size of clique is unusually large under a given null model?

Consider an undirected graph $G$ containing a large clique: all nodes in some subset
$V_C \subseteq V_G$ are directly connected to one another. We can describe the graph by first describing
the set $V_C$, and then describing $G$ in a canonical manner. Since every node in $V_C$ is
connected to every other node in the clique, we can omit these links from the second part
of our description, shortening the total description length, if $V_C$ is large enough. This gives
us a code $L^{\text{clique}}$ with short codelengths for graphs containing a single large clique, and—by
the correspondence mentioned in the preliminaries—also a probability distribution $p^{\text{clique}}$ with
high probabilities for graphs with a large clique. The principle is illustrated in Figure 1.

Of course, there is no guarantee that of all the distributions with a bias towards large
cliques, $p^{\text{clique}}$ is the precise distribution from which we sampled our data. Luckily, it does
not need to be: as long as we can show that our clique-based model encodes the data more
efficiently than the null model, the presence of the clique disproves the hypothesis that the
data came from the null model. We do not know where it came from, but it did not come from
the null model. Under the hypothesis that the null model was the source of the data,
we can show that the probability that any other model compresses the data better by $k$ bits
or more, decays exponentially in $k$. This is known as the no-hypercompression inequality
(Grünwald, 2007, p103). More precisely, let $p^{\text{null}}(x)$ be any probability distribution, with
Finding Network Motifs in Large Graphs

\[ L^{\text{null}}(x) = -\log p^{\text{null}}(x) \text{ and let } L(x) \text{ be any code, then we have } \]

\[ p^{\text{null}}(L^{\text{null}}(x) - L(x) \geq k) \leq 2^{-k}. \]

Thus, under the null model, the probability that \( L^{\text{clique}} \) will compress the data better than the null model by 10 bits or more is less than one in one-thousand. For twenty bits, we get one in a million, for thirty bits, one in a billion, and so on. So, while a low codelength under \( L^{\text{clique}} \) does not prove that the clique-code is the true model, it does allow us to comfortably reject the null model.

We can interpret this procedure as a significance test: the difference in compression \( D \) between the null model and the alternative model is a statistic (Grünwald, 2007, Example 14.2). The no-hypercompression inequality gives us a bound on the probability \( p^{\text{null}}(D \geq k) \). To reject the null model with significance level \( \alpha \), we must find some code on the set of all graphs and show that it compresses the data better than the null model by \( k \) bits, with \( 2^{-k} \leq \alpha \). Any code will do, as long as it was chosen before seeing the data.

Note that \( D \) is also the logarithm of the likelihood ratio between the null model and \( L \), so we can see this as a likelihood ratio test. We can also interpret the difference in codelength between two models \( p_a \) and \( p_b \) as the logarithm of the Bayes factor \( p_a(x)/p_b(x) \) (Grünwald, 2007, Section 14.2.3).

Now, while our test rejects the null model, it does not necessarily confirm anything about the pattern we used to compress the data. We would like to make it as likely as possible that it was the pattern we are interested in (e.g., the clique) that allowed us to reject the null model, and not some other aspect of the alternative model. This is a complex problem and, it is difficult to provide any guarantees. If the alternative model stores, say, the linkset of a graph more efficiently than the null, it may well be that the alternative model always beats the null, regardless of the pattern used. To rule out most such artifacts, we must re-use the same model we used for the null model within the alternative model to store every part of the graph except the pattern. There is usually an intuitive way to do this: for instance, in the alternative model for the clique, we store the graph in two parts: first the nodes belonging to the clique, and then the full graph, minus any links in the clique. If we use, say, the ER model as our null, we also use the ER model to store the graph in this second part of the code. In this way, we know that any compression achieved must be due to the clique. \(^6\)

To ensure this, we aim to have the alternative model exploit only the pattern, and nothing else: we reuse the same model we used as the null model to store all of the graph, save for the pattern. For instance, in the example above, the clique model must store the graph minus the links of the clique. If we use the null model for this, we know that the only change between the null model and the alternative is the use of the clique, so that must be what made the difference.

\(^6\) The lack of guarantees may be a cause for skepticism. Consider, however, that the hypothesis test by itself is never proof that the pattern found is meaningful. The only thing it proves is that the null hypothesis is incorrect. If it is reasonable to assume that the null hypothesis might be exactly true, we can translate a rejection of the null hypothesis into a judgement on the value of the statistic: if we can disprove the null hypothesis that a coin is fair, we have proved that the coin is unfair. In graph analysis, however, our models are almost never very realistic models for our data. Thus, we must consider the use of hypothesis testing nothing more than a heuristic, and we must not deceive ourselves into thinking that we have proved anything we did not already know.
Figure 1: A simple example of graph analysis by MDL: finding cliques. (a) The data. (b) A representation of the data that exploits the existence of a large clique. We first store a set indicating which nodes belong to the clique. We then store the graph, omitting those links that belong to the clique. (c) An illustration of the principle of a bound. For a complete code on graphs, using $L_{\text{uniform}}$, we need to first store the size of the graph using some arbitrary code on the integers $L_N$. To make sure that it does not matter which code we choose for $L_N$, we require our code ($L_{\text{alt}}$) to beat the bound $B(G)$ (by at least $k$ bits).

2.1 Rejecting multiple null-models

A final benefit of this method is that we can reject multiple null models with a single test. In many situations we will have a function $B(G)$ that lowerbounds any code in some set $\mathcal{L}$. If our alternative model provides a codelength below $B(G) - k_\alpha$ with $k_\alpha$ the number of bits required for our chosen $\alpha$, we can reject all of $\mathcal{L}$. As an example, Let $\mathcal{G}_n$ be the set of all undirected graphs of size $n$. We can define a uniform code on such graphs: $L_n^{\text{uniform}} = \log |\mathcal{G}_n|$. This code captures the idea that the size of the graph is the only informative statistic: given the size, all graphs are equally likely. This is a good null model to test the assumption that the graph contains no significant structure, save for its size. However, it is parametrized. It is currently not a code on all graphs, just those of size $n$. To turn it into a code that can represent all graphs, we need to encode the parameter $n$ as well, with some code $L_N$ over the natural numbers

$$L_{\text{complete}}(G) = L_N(n(G)) + L_{n(G)}^{\text{uniform}}(G).$$

This is called two-part coding, we encode the parameters of a model first, and then the data given the parameters. For some parametrized model $L_\theta$, we can choose any code for $\theta$ to make it complete. We will call the set of all such complete codes the two-part codes on $L_\theta$. Note that we can simply concatenate the two codewords, since all codes are prefix-free.

Which code we choose for the parameter is arbitrary. We may be able to reject the uniform code for one choice of $L_N$, or several, but how can we prove that $L_{\text{complete}}$ will be rejected whatever $L_N$ we choose? Instead of choosing an arbitrary code for the size, we can use the bound $B(G) = L_{n(G)}^{\text{uniform}}(G)$ as our null model. This is not a code, but it is a
lower bound for any two-part code on $L_n^{\text{uniform}}$. If $L^{\text{clique}}(G)$ is shorter than $B(G)$, it is also shorter than $L^{\text{complete}}(G)$ whatever the choice of $L^n,7$

Note that when we store the rest of the graph within $L^{\text{clique}}$ we cannot use $B(G)$ in place of $L^{\text{complete}}(G)$. We want a conservative hypothesis test: the probability of incorrectly rejecting a null model may be lower than $\alpha$ but never higher. By this principle, bounds chosen in place of either model should always decrease $D$. The code corresponding to the null model must always be lowerbounded, and the optimal code for the alternative model must always be upperbounded.

3. Encoding with Motifs

We will now use the principles described above to define the alternative model for the detection of a motif. The idea is that 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. As noted above, this is a “best effort” approach: there is no need to choose a model that corresponds to the true source of the data, or to choose one that compresses optimally, but the better we design our code, the more motifs we will find.

Let $S = \langle S_1, \ldots, S_k \rangle$ be a sequence of nodes from $V_G$. The induced subgraph $G[S]$ is a graph $G'$ with $k$ nodes, containing a link $(i, j)$ if and only if $G$ has a link $(S_i, S_j)$. That is, the induced subgraph extracts all links existing between members of $S$.

Assume that we are given a graph $G$, a potential motif $G'$, and a list $M^{\text{raw}} = \langle M_1, \ldots, M_k \rangle$ of instances of $G'$ in $G$. That is, each sequence $M \in M^{\text{raw}}$ consists of nodes in $N_G$, such that the induced subgraph $G[M]$ is equal to $G'$. Note that that $M^{\text{raw}}$ need not contain all instances of $G'$ in the data. Sequences in $M^{\text{raw}}$ 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.

The basic principle behind our code is illustrated in Figure 2: we want to store the motif only once, remove as many instances of the motif from the data as we can, and replace them with references to the stored motif. The two graphs combined contain enough information to recover the data, but we have only had to describe the motif once. Algorithm 1 describes the exact process.

Removing overlaps The first thing we need is a subset $M$ of $M^{\text{raw}}$ such that the instances contained within it do not overlap: i.e. for each $M_a$ and $M_b$ in $M$, we have $M_a \cap M_b = \emptyset$. To maximize the scalability of the algorithm, we will use a greedy approach to find a reasonable subset, rather than attempting to approximate an optimal solution.

As we will see later, the most important factor for compression is the number of links an instance has to nodes outside the instance. We call this the exdegree. Unlike the in- and outdegree, the exdegree is not a property of a node, but of a subgraph.

---

7. In probabilistic terms, the code on the parameter corresponds to a prior on the parameter. The two-part codes correspond to maximum likelihood posterior probabilities: $p(\theta)p(x \mid \theta)$. Our bound corresponds to the maximal likelihood of the data: $p(x \mid \hat{\theta})$. This shows us that the bound applies not only to the two-part codes, but also to the full Bayesian mixture: $\sum_\theta p(x \mid \theta)p(\theta) \leq \sum_\theta p(x \mid \theta)p(\theta) = p(x \mid \hat{\theta})$

8. Unlike the in- and outdegree, the exdegree is not a property of a node, but of a subgraph.
Figure 2: An illustration of the motif code. We store $G'$ once, and remove its instances from $G$, replacing them with a single, special node. The links to special nodes are annotated with ‘rewiring’ information, which tells us how to rewire the subgraph back into $H$. Storing only $H$ and $G'$ is enough to reconstruct the data.

Encoding integers  
In the following, we will often need to encode single natural numbers, or a sequence of natural numbers from a finite range. For single numbers, we will use the code corresponding to the probability distribution $p^N(n) = 1/(n(n+1))$, and denote it $L^N(n)$.

For sequences of elements from a finite set, we use the code corresponding to a Dirichlet-Multinomial (DM) distribution. Let $S$ be a sequence of length $k$ of elements from some alphabet $\Sigma$. Conceptually, the DM distribution models the following sampling process: we sample a probability vector $p$ on $[0,|\Sigma|]$ from a Dirichlet distribution with parameter vector $\alpha$, and then sample $k$ symbols from the categorical distribution represented by $p$. The probability mass function corresponding to this process can be expressed as

$$p^\text{DirM}_\alpha(S \mid k, \Sigma) = \prod_{i \in [1,k]} \text{DirM}_\alpha(S_i \mid S_{1:i-1}, k, \Sigma)$$

$$\text{DirM}_\alpha(S_i \mid S', k, \Sigma) = \frac{f(S_i, S') + \alpha_i}{|S'| + \sum_i \alpha_i}$$

where $f(x, X)$ denotes the frequency of $x$ in $X$. We use $\alpha_i = 1/2$ for all $i$. Let $L^\text{DirM}_{k, \Sigma}(S) = -\log p^\text{DirM}(S \mid k, \Sigma)$. The DM model can be seen as encoding each element from $S_i$, 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.

Note that this code is parametrized with $k$ and $\Sigma$. If these cannot be deduced from information already stored, they need to be encoded separately. When encoding natural numbers, we will have $\Sigma = [0, n_{\text{max}}]$, and we only need to encode $n_{\text{max}}$. A useful property of the DM code is that it is exchangeable: if we re-arrange the elements of $S$, the codelength remains the same.
Note that, since we use $L^N(n)$ and $L^{\text{DirM}}(n)$ only in the motif code, there is no need for them to be optimal. The better they compress, the more motifs we will find, but we do not require optimal results for the algorithm to be valid.

The motif code We can now define the full motif code. It stores the following elements. Since each corresponds to a prefix-free code, we can simply concatenate their codewords for a prefix-free codeword for the data.

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

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

**instance nodes** $L^\text{base}$ 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 $n(H)^{\vert \mathcal{M} \vert}$ possible placements, so we can encode this information in $L^N(\vert \mathcal{M} \vert) + \log(n(H)^{\vert \mathcal{M} \vert})$ 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. Let there be some agreed-upon order in which to enumerate the links of any given graph. Given this order, we only need to encode the sequence $W$ of integers $w_i \in [1, \ldots, n(G')]$. We do so using the DM model described above. The maximum symbol and length of $W$ can be deduced from parts already encoded. Note that this code is invariant to the ordering of $W$, so the particulars of the canonical node ordering do not need to be specified.

**multiple edges** Since $L^\text{base}$ can only encode simple graphs, we cannot use it to store $H$ directly, since collapsing the instances into single nodes may have created multiple edges. We remove all multiple edges 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 $R_i \in [0, r_{\text{max}}]$ which we store by first recording the maximum value in $L^N(\text{max}(R))$ bits, and then recording $R$ with the DM model.

**insertions** Finally, while $H$ and $G'$ give us enough information to 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 $\vert \mathcal{M} \vert(n(G') - 1)$ such insertions. Each insertion requires $\log(t + 1)$ bits to describe, where $t$ is the size of the graph before the insertion. Let $H$ be the template graph and $G$ the complete graph, then we require $\sum_{t=n(H)}^{n(G)-1} \log(t + 1) = \log(n(G)!) - \log(n(H)!)$ bits to record the correct insertions.
Algorithm 1 The motif code $L^\text{motif}(G; G', M, L^\text{base})$. Note that the nodes of the graph are integers.

Given:
- a graph $G$, a subgraph $G'$,
- a list $M$ of instances of $G'$ in $G$, a code $L^\text{base}$ on the simple graphs.

$$b_{\text{subgraph}} \leftarrow L^\text{base}(G')$$

# replace each instance with a single node
$$H \leftarrow \text{copy}(G), W = []$$

for each $M = \{m_1, \ldots, m_n(G')\}$ in $M$:
  # We use $m_1$ (the $m_1$-th node in $G$) as the instance node
  for each link $l$ between a node $n_{\text{out}}$ not in $M$ and a node $m_j$ in $M$:
    if $j \neq 1$: add a link between $n_{\text{out}}$ and $m_j$
    $W$.append($j$)
  remove all nodes $m_i$ except $m_1$, and all incident links
$$b_{\text{rewiring}} \leftarrow L^{|W|_n(G')} (W)$$

# Remove multiple edges from $H$ and record the duplicates in $R$
$$R, H' \leftarrow \text{simple}(H)$$
$$b_{\text{template}} \leftarrow L^\text{base}(H')$$
$$b_{\text{multi-edges}} \leftarrow L^{N(\max(R))} (R) + L_{|R|,\max(R)} (R)$$

$$b_{\text{instances}} \leftarrow L^{N(|M|)} \log (\frac{n(H)}{|M|})$$
$$b_{\text{insertions}} \leftarrow \log(n(G))! - \log(n(H))!$$

return $b_{\text{subgraph}} + b_{\text{template}} + b_{\text{rewiring}} + b_{\text{multi-edges}} + b_{\text{instances}} + b_{\text{insertions}}$

Pruning the list of instances Since our code accepts any list of motif instances, we are free to take the list $M$ and remove instances before passing it to the motif code, effectively discounting instances of the motif. This can often improve compression, as storing the rewiring information for instances with high exdegrees may cost more than we gain from removing them from the graph. We sort $M$ by exdegree and search for the value $c$ for which compressing the graph with only the first $c$ elements of $M$ gives the lowest codelength.

The codelength $L^\text{motif}$ as a function of $c$ is roughly unimodal, which means that a ternary search should give us a good value of $c$ while reducing the number of times we have to compute the full codelength. We use a Fibonacci search (Kiefer, 1953), an elegant variation on ternary search requiring only one sample per recursion.

Implementation The template part of the code can be time and memory intensive to compute for large graphs, as it involves creating a copy of the data. For any given $L^\text{base}$, we can create a specific implementation which computes the codelength required for storing the template graph without constructing $H$ explicitly. This will speed up the computation
of the code at the expense of creating a new implementation for each new null model. We use such specific implementations for our three null models.

More precisely, since we would like to operate efficiently on very large graphs, with a relatively low number of motif instances, we would like to avoid, where possible, operations requiring a full pass over the entire graph. We do this by storing the parameters of the null model (either the degree sequence or the size and number of links) for the complete graph, and computing how they change when the template graph is created. We can do this with a loop over the motif instances, computing directly how the model parameters change. Since the parameters determine the code length, this is all we need to determine the number of bits required to store the template graph. In this computation we must make sure also to compute which links of the template graph become multiple links, and how many external links connect to each node within the motif.

4. Null Models

We will define three null models. For each model we follow the same pattern: we first describe a parametrized model (which does not represent a code on all graphs). We then use this to derive a bound as described in the second section, 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 \( \theta \). Let \( \hat{\theta}(G) \) be the value of \( \theta \) that minimizes \( L_\theta^{\text{name}}(G) \) (the maximum likelihood parameter). From this we derive a bound \( B^{\text{name}}(G) \) — usually using \( B^{\text{name}}(G) = L_\hat{\theta}(G)^{\text{name}} \) — which we will use in place of the null model. Finally, we create the complete model by two-part coding:

\[
L^{\text{name}}(G) = L^{\theta}(\hat{\theta}(G)) + L_\hat{\theta}(G).
\]

4.1 The Erdős-Renyi Model

The Erdős-Renyi (ER) model is probably the best known probability distribution on graphs (Renyi 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

\[
L_{n,m}^{\text{ER}}(G) = \log \left( \frac{(n^2 - n)/2}{m} \right)
\]

for undirected graphs, and

\[
L_{n,m}^{\text{ER}}(G) = \log \left( \frac{n^2 - n}{m} \right)
\]

for directed graphs. We use the bound \( B^{\text{ER}}(G) = L^{\text{ER}}_{n(G),m(G)}(G) \).

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

\[
L^{\text{ER}}(G) = L^{\text{N}}(n(G)) + \log(m_{\text{max}} + 1) + L^{\text{ER}}_{n(G),m(G)}(G).
\]
4.2 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 $D(G) = (D^{in}, D^{out})$, such that $D^{in}_i$ is the number of incoming links of node $i$, and $D^{out}_i$ is the number of outgoing links.

The parametrized model $L^{DS}_D(G)$ The degree-sequence model $L^{DS}_D(G)$ 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 $L^{DS}_D(G) = \log |G_D|$ where $G_D$ 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 discovered independently by Blitzstein and Diaconis (2011) and Genio et al. (2010). This algorithm is guaranteed to produce any graph matching $D$ with some nonzero probability. Crucially, the algorithm does not backtrack or reject candidates, which means that if we multiply the probability of each random choice made in sampling, we get the probability of the sample under our sampling procedure. That is, the algorithm produces, along with a sample $G \in G_D$, the probability $q^{DS}_D(G)$ of the algorithm producing $G$. While the samples are not uniform, we do have

$$E \left[ \frac{1}{q^{DS}_D(G)} \right] = |G_D|$$

(1)

where $G$ is a random variable representing a sample from the algorithm. Thus, we can sample a number of graphs and take the mean of their inverse probability under $q^{DS}_D$ to estimate $p^{DS}_D(G)$. This is a form of importance sampling.

Unfortunately, even with the highly optimized implementations described in Genio et al. (2010) and Kim et al. (2012) sampling can be slow for large graphs. Luckily, we are only interested in an estimate of the codelength accurate to around the level of single bits, which means that we only need to sample until we have a rough estimate of the order of magnitude of $|G_D|$. For instance, if we accept a margin of error of only 15 bits (of the potentially $10^6$ bits required to store the graph), we can underestimate the number of graphs by 4 orders of magnitude and still end up within the margin. All we need is a reliable confidence interval for our estimate, so that we can choose a suitably conservative bound. Our method of obtaining such a confidence interval is described in the Appendix. In all cases, we use a one-sided confidence interval: when computing the codelength under the null model, we use a lower bound for the true value, and when computing the codelength for the motif code, we use an upper bound. Thus, the difference in codelength is a lower bound for the true value.

The bound $B^{DS}(G)$ To get a bound for all two-part codes on $L^{DS}_D$, we could use $B^{DS}(G) = L^{DS}_{D(G)}(G)$. Beating such a bound would tell us that no property of the degree sequence could

---

9. Specifically, our implementation uses the algorithms described in Genio et al. (2010) and Kim et al. (2012). However the non-uniform sampling from the candidate set, discussed in (Blitzstein and Diaconis 2011, p10, step 5) is crucial to achieving a low variance in the sampling distribution, and thus a fast convergence.
explain the motif we had found. Unfortunately, the degree sequence forms a large part of
the code, and a lot of evidence is required to compress better than $B^{DS}(G)$ with a complete
code.

Instead, 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 bound $B^{DS}(G) = B^{\text{deg}}(D(G)) + L^D_{D(G)}(G)$. For directed graphs, we use $B^{DS}(G) = B^{\text{deg}}(D^{\text{in}}(G)) + B^{\text{deg}}(D^{\text{out}}(G)) + L^D_{D(G)}(G)$.

Note that the scale-free property of many complex graphs is captured by this model.
Compressing better than $B^{DS}(G)$ with a certain motif $M$, indicates that the occurrences of
$M$ are not explained by the scale-freeness of the graph.

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

$$L^{DS}(G) = L^N(n(G)) + L^N(\max(D)) + L^{\text{DirM}}_{n(G), \max(D)}(D) + L^D_{D(G)}(G)$$

and for directed graphs

$$L^{DS}(G) = L^N(n(G)) + L^N(\max(D^{\text{in}})) + L^{\text{DirM}}_{n(G), \max(D^{\text{in}})}(D^{\text{in}}) + L^N(\max(D^{\text{out}})) + L^{\text{DirM}}_{n(G), \max(D^{\text{out}})}(D^{\text{out}}) + L^D_{D(G)}(G).$$

Note that in the computation of $L^{\text{m}otif}$ with $L^{DS}$ as a base model, we estimate $|\mathcal{G}_D|$ for
both the template graph and the motif. It is important to combine the confidence intervals
over these two estimates carefully, so that we end up with a correct confidence interval over
the total codelength. This is discussed in the supporting materials. For $L^{\text{m}otif}$, we compute
a one-sided confidence interval to get an upper bound, so that with 95% confidence we are
overestimating the size of the motif code.

4.3 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 $S_d$ be the set of all pairs of such sequences satisfying $D$. We have
$(D^{\text{in}}_1, \ldots, D^{\text{in}}_m)$ possibilities for the first sequence, and $(D^{\text{out}}_1, \ldots, D^{\text{out}}_m)$ for the second. This gives us $|S_D| = (D^{\text{in}}_1, \ldots, D^{\text{in}}_m)(D^{\text{out}}_1, \ldots, D^{\text{out}}_m) = m! m! / \prod_{i=1}^n D^{\text{in}}_i D^{\text{out}}_i$. We have $|S_D| > |\mathcal{G}_D|$ 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, and we would get a different pair of sequences, representing the same graph, so that for a graph with no multiple edges, there are $m!$ different sequence-pairs to represent them.
To refine this upper bound, let $S'_D \subset S_D$ be the set of sequence pairs representing simple graphs. Since all links in such graphs are distinct, we have $|\mathcal{G}_D| = |S'_D|/m!$. Since $|S'_D| \leq |S_D|$, we have

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

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 $S_D$ as the set of all lists such that the resulting graph satisfies $D$. There are $(2m)!$ such lists. We now have an additional reason why $|S_D| > |\mathcal{G}_D|$: each pair of nodes describing a link can be swapped around to give us the exact same graph. This gives us:

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

In both cases, the fact that we have an upperbound gives us a code: while the code as described assigns some probability mass to non-simple graphs, we can easily assume that this is assigned instead to some null-element, since we are only interested in the codelengths and probabilities of simple graphs. This gives us the following parametrized code for directed graphs:

$$L^\text{EL}(G) = \log m! - \sum_{i=0}^{n} \log D^\text{in}_i! - \sum_{i=0}^{n} \log D^\text{out}_i!$$

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

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

For the bound and the complete model, we follow the same strategy we used for the degree-sequence model: $B^\text{EL}(G) = B^\text{deg}(G) + L^\text{EL}_{D(G)}(G)$ and,

$$L^\text{EL}(G) = L^\text{N}(n(G)) + L^\text{N}(\max(D)) + L^\text{DirM}_{n(G),\max(D)}(D) + L^\text{EL}_{D(G)}(G)$$

for undirected graphs and

$$L^\text{EL}(G) = L^\text{N}(n(G)) + L^\text{N}(\max(D^{\text{in}})) + L^\text{DirM}_{n(G),\max(D^{\text{in}})}(D^{\text{in}}) + L^\text{N}(\max(D^{\text{out}})) + L^\text{DirM}_{n(G),\max(D^{\text{out}})}(D^{\text{out}}) + L^\text{EL}_{D(G)}(G)$$

for directed graphs.

10. This value was previously used in Bezáková et al. (2006) as a precise value for the number of graphs with multiple edges. This is incorrect, as we can only divide by $m!$ if we know that no graphs have multiple edges.
5. Experiments

To validate and illustrate our method, we will perform four experiments. First, we will construct a graph by injecting instances of a single motif into a random network. The method should then recover only this motif as significant. Second, we will run the method on data sets from four different domains, and show the results for the most frequent subgraphs, using the three null models we have described. Third, we use the motifs identified by our method, and by the traditional approach, as features in a classification task, showing comparable performance, well above chance. This tells us that, at least for the purposes of classification, the motifs returned by our method are no less informative than those found by the traditional method. Finally, to show the scalability of the method with fast null models, we will run the analysis on several large graphs, first using purely memory based methods, and then with a disk-backed datastructure to store the graph.

In all experiments we search for motifs by sampling, based on the method described by Kashtan et al. (2004). Note that we have no particular need for a sampling algorithm which provides an accurate approximation of the actual frequencies present in the graph, so long as it can provide us with a large selection of non-overlapping instances with low exdegree. For this reason, we adapt the algorithm to improve its speed: we start with a set \( N' \) containing a single random node drawn uniformly. We then add to \( N' \) a random neighbour of a random member of \( N' \), and repeat until \( N' \) has the required size. We extract and return \( G[N'] \). In the case of a directed graph, nodes reachable by incoming and outgoing links are both considered neighbours.

The size \( n(G') \) of the subgraph is chosen before each sample from a uniform distribution over the interval \( [n_{\text{min}}, n_{\text{max}}] \). This distribution over sizes is biased towards small motifs: since there are fewer connected graphs for small sizes, small graphs are more likely to be sampled. As the results show, 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 described in Section 3.

In all experiments, we report the log-factor: \( B_{\text{null}}(G) - L_{\text{motif}}(G; G', \mathcal{M}, L_{\text{null}}) \). That is, we use the bound in place of the null model, and the complete code of the same null model is used in the motif code (to store the template graph and the motif). If the log-factor is larger than 10 bits, we can interpret it, as described in Section 2, as a successful significance test, allowing us to reject the null model at \( \alpha = 0.001 \). In all cases, 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. This could be achieved by sampling more subgraphs, using a different algorithm to find motif instances or taking more samples from the degree-sequence estimator.

All experiments in this paper were run on a single machine with a 2.60 Ghz Intel Xeon processor (E5-2650 v2) with 64 Gigabytes of memory and 8 physical cores. The memory and cores available to the program differ per experiment and are reported where relevant.
5.1 Recovering Motifs from Generated Data

We use the following procedure to sample an undirected graph with 5000 nodes and 10000 links, containing \( n^i \) injected instances of a particular motif \( G' \) with \( n' \) nodes and \( m' \) links:

1. Let \( n = 5000 - (n' - 1)n^i \) and \( m = 10000 - m'n^i \) and sample a graph \( H \) from the uniform distribution over all graphs with \( n \) nodes and \( m \) links.

2. Label \( n^i \) random nodes, with degree 5 or less, as instance nodes.

3. Let \( p^{\text{cat}} \) be a categorical distribution on \( \{1, \ldots, 5\} \), 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 \( p^{\text{cat}} \). Links incident to two instance nodes, will thus get \textit{two} values.

5. Reconstruct the graph \( G \) from \( G' \) and \( H \).

This is roughly similar to sampling from our motif code. In this graph, \( G' \) should be the only significant motif, with the exception of motifs that can be explained from the prevalence of \( G' \), ie. subgraphs and supergraphs of \( G' \), or graphs that contain part of \( G' \). However, these should have a markedly lower log-factor than \( G' \). For our experiment, we will only extract subgraphs of size 5, to rule out the first two cases.

On this sampled graph, we run our motif analysis. We run the experiment multiple times, with \( n^i = 0, n^i = 10 \) and \( n^i = 100 \), using the same subgraph \( G' \) over all runs, but sampling a different \( H \) each time. For each value of \( n^i \), we repeat the experiment 10 times. Per run we sample 5000 motifs. This value is chosen to show that even a very low sample size is sufficient to recover the motif. The null model in all cases is the ER model, as that corresponds to the source of the data.

Figure 3 shows the results for the 21 possible connected simple graphs of size 5. As expected, when we insert no subgraphs, the motif model cannot compress the graph better than the null model, for any motifs, since the source of the data is the null model. There are motifs with very high frequencies (shown on the right), much higher than the frequencies of our motif, but these can be explained as a consequence of the null model and have a negative log-factor. 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 is an important lesson for motif analysis: not every significant subgraph represents a meaningful result, some may be a byproduct of other motifs.

5.2 Various Data Sets and null models

Next, we show how our approach operates on a selection of data sets across domains. We use the following data sets:

**kingjames (undirected, \( n = 1773, m = 9131 \)**) Co-occurrences of nouns in the text of the King James Bible (KONECT, 2014; Römhild and Harrison, 2007). Nodes represent nouns (places and names) and links represent whether these occur together in one or more verses. The full motif analysis took 23 hours and 16 minutes.
Figure 3: The results of the experiment on generated data. The bottom row shows all 21 simple connected graphs with 5 nodes (up to isomorphism). The middle row shows the number of non-overlapping instances found by the sampling algorithm for \( n^i = 0, n^i = 10 \) and \( n^i = 100 \) from left to right, for each motif. The bars show the average value over 10 randomly sampled graphs, with the same subgraph (shown in red) injected each time. The top row shows the difference between the code length under the null model (the ER model) and under the motif code. The error bars represent the range, i.e., they are drawn from the smallest to the largest observation.

**yeast (undirected, \( n = 1528, m = 2844 \))** A network of the protein interactions in yeast, based on a literature review (Reguly et al., 2006). Nodes are proteins, and links are reported interactions between proteins. We removed 81 self-loops. The full motif analysis took 3 hours and 19 minutes.

**physicians (directed, \( n = 241, m = 1098 \))** Nodes are physicians in Illinois (KONECT, 2015b; Coleman et al., 1957). Links indicate that one physician turns to the other for advice. The full motif analysis took 31 minutes.

**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). To create a workable graph, we follow the procedure outlined in Carstens (2013): we include only papers published before 1994, remove citations to papers published after the citing paper, and select the largest connected component. The full motif analysis took 8 hours and 53 minutes.

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

Our first observation is that for the physician data set, there are no motifs under the degree-sequence null model. This 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.

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

We also observe a degree of agreement between the degree sequence model and the edgelist model, suggesting that the edgelist model may be an acceptable proxy for the degree sequence model. In the next section we put this idea to the test, evaluating our method with the edgelist model against the traditional method.

Finally, we would like to emphasize the scale of the null-hypothesis rejections. In the kingjames test, the most significant motif under the DS model has a log-factor of over 3500 bits. This corresponds to a rejection of the null model at a significance at \(\alpha = 2^{-3500} \approx 10^{-1050}\). Note that accurate estimates of low significances are a common problem in the traditional approach, since this requires very high numbers of samples from the null model (Picard et al., 2008).

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.

Now, clearly, these analyses took relatively long, for data sets that are of medium size. The bottleneck here is the computation of the degree-sequence model. If we eliminate that, as we do in Section 5.4, 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. But are the motifs returned still, in some sense, informative? We investigate this in the next section, comparing the traditional method of motif detection to our method, using the EL model.

### 5.3 Comparison with the traditional method

Pattern mining on graph data is a very difficult task to evaluate. Not only is it unsupervised, we also have no ground truth data for what people might consider interesting subgraphs. Often, we do not even know what such a ground truth would look like: even a domain expert may not recognize a “correct” motif at first sight. Nevertheless, we have rather stretched the definition of a motif—choosing a different statistic, and a faster null model—so we need some assurance that after all this, the motifs found by our method are not completely meaningless.
Figure 4: The results of the motif extraction on the 2 undirected networks.
Figure 5: The results of the motif extraction on the 2 directed networks.
Figure 6: 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, connect mini-graph of size 3, 4, or 5. If the motif algorithm considers the mini-graph a motif for the current instance, the feature is 1, otherwise it is 0. This reduces the entire graph to just 29 bits of information, using only the judgements of the motif algorithm to represent the graph. If these judgements do indeed contain some information that characterises the graph, these feature vectors should be sufficient to perform better than a majority class classifier. (c) We apply a simple, linear SVM in the 29-dimensional feature space to perform the classification task on just the motif judgments.

The definition of what constitutes a good motif is exceedingly vague: papers variously describe a motif as a “functional unit”, a “characteristic pattern” or a “statistically significant subgraph”. To be able to operationalize the definition into 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.

Using this definition, we can test how effective a motif detection method is, using a graph classification task. 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 judgements of the algorithm under evaluation. We test all connected subgraphs of size 3, 4, and 5, giving us 29 binary features. As we established in the previous paragraph, a good motif characterizes a graph in a way that is relevant to its domain. In the case of a graph classification task we are provided with a domain-relevant distinction in the form of the classes provided. 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. This approach—quantifying unsupervised pattern extraction through classification—was also used in van Leeuwen et al. (2006).

Our main aim is to establish that the resulting classifier performs better than chance. For this purpose we compare it to a majority class baseline. Our secondary aim is to show that we do not perform much worse than the traditional method. Note that our method
allows motif analysis to scale up by at least four orders of magnitude, so we consider a small reduction in classification quality acceptable, so long as the method still significantly outperforms the majority class baseline.

Finding a graph classification task that fits our requirement is not easy: the graphs must be large enough to provide our method with enough data, but small enough that the traditional method works without approximation. Moreover, the graphs must be undirected, so that the feature vectors stay relatively small, and we can use the ORCA method (Hočevar and Demšar, 2014) to perform fast subgraph counting for the traditional method. In order to tune the graph classification tasks to our needs, we adapt them from classification tasks on knowledge graphs, described in Ristoski et al. (2016). In these tasks, the data consists of 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 if there are one or more directed edges between them in the original knowledge graph.

This gives us a classification task on a subset of 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 \(-\log_2 \alpha\) bits with \(\alpha = 0.05\).

For the traditional method, we use complete subgraph counts, both on the data and on 1 000 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.\(^{12}\) We mark a graph as a motif if fewer than 5% of the graphs generated from the DS model have more occurrences than the data does.\(^{13}\)

For performance reasons, we use only 100 randomly chosen instances from the classification task. On these 100 instances, we perform five-fold cross-validation. We repeat the complete experiment, from sampling instances to five fold 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.

Note that the accuracy should not be compared to that achieved by others on these benchmarks, as we have thrown away almost all information in the original knowledge graph. We are merely using classification accuracy as a proxy for the quality of motif analysis.

\(^{11}\) For smaller graphs our method can be adapted by testing against a single model, instead of using a bound to reject multiple null models as described in Section ??, However, for this test, we would like to use the method as we expect it to be used: on large graphs.

\(^{12}\) We created a Java implementation, available at https://github.com/pbloem/orca

\(^{13}\) Note that the commonly used z-score method is seriously flawed, as discussed by Picard et al. (2008), so we do not use it here.
The results are shown in Figure 7. 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, 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.
5.4 Large-Scale Motif Extraction

| data    | disk | n   | m   | $|\mathcal{M}|$ | mem. | t    | preload | search | motifs |
|---------|------|-----|-----|----------|------|------|---------|--------|--------|
| wiki-nla |      | 1 M | 13 M| 3–6      | 16 Gb| 16   | 7m      | 8      |
|         | 3–6  | 5 Gb| 16  |          | 13 m | 8    |         |        |
|         | 2 Gb | 1   | 25 m|          | 8    |      |         |        |
|         | 11 Gb| 1   | 2h  | 14 m     | 0    |      |         |        |
| ✓       | 3–6  | 1 Gb| 1   | 8m       | 1h 30m| 8    |         |        |
| wiki-enb| ✓    | 12 M| 378 M| 3–6    | 2 Gb | 1    | 4h 58m | 6h 6m | 10     |
| ✓       | 8    | 8 Gb| 1   |          | 6h 5m | 23   |         |        |
| twitterc| ✓    | 53 M| 1 963 M| 3–6 | 6 Gb | 1    | 17h 12m| 33h 19m| 0      |
| ✓       | 8    | 8 Gb| 1   |          | 54h 26m| 0    |         |        |
| friendsterd| ✓ | 68 M| 2 586 M| 3–6 | 6 Gb | 1    | 42h 37m| 45h 2m | 68     |
| ✓       | 3–6  | 56 Gb| 9   |          | 8h 38m| 68   |         |        |
| ✓       | 10   | 7 Gb | 1   |          | 35h 7m| 57   |         |        |

a KONECT (2016c); Preusse et al. (2013), multiple links were removed.
b KONECT (2016a); Auer et al. (2008), self-loops were removed.
c KONECT (2016d); Kwak et al. (2010)
d KONECT (2016b)

Table 1: The results of various runs of the algorithm on large data sets. Sizes are rounded to the nearest million. The second column indicates whether the graph was stored on disk, or in memory. The ‘size’ column indicates the sizes of motifs that were sampled. The $t$ column shows the number of threads allowed to run concurrently. The last column indicates how many significant motifs were returned (under the EL model). Only the 100 subgraphs with the highest number of instances after sampling were tested. All runtimes are rounded to the nearest minute. The memory column indicates the maximum heapspace allowed for the Java Virtual Machine. Preloading was always done with the lowest amount of memory indicated for that data set, and can be sped up if more memory is available.

Section 5.3 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 $10^6$ motifs in total (if multiple motif sizes are used, each size is equally likely to be sampled). We take the 100 most frequent motifs 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. All experiments were executed on a single, dedicated machine, with 64 Gb of memory, and a single 2.60 Ghz Intel Xeon processor (E5-2650 v2).

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 datasets, we store the graph on disk. 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 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 so that random access can be performed efficiently (using the MapDB database engine\textsuperscript{14}).

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 only needs to be done once, so we show the preloading and analysis times separately. Loading 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 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 (e.g. the degree sequence). For the sampling and the computation of the log factors only relatively small amounts of random access are required. Since a graph can, in principle, be compressed with only a very small number of occurrences of a given motif, this gives us a very scalable method to find motifs in large data.

For disk-based experiments, we also limit the total number of rewritten links in the template graph to 500 000, to limit memory use. If the motif with a given list of instances results in more rewritten links, we do not consider it. Note that, since we search for a good pruning of the instance list, the motif will still be considered with a more heavily pruned instance list. A large number of rewritten links suggest that there are many instances with high ex-degree, so we likely do not lose much by this heuristic.

The main conclusion from this experiment is that we can perform motif analysis on data in the scale of billions of edges with very modest hardware. The problem is ‘embarrassingly parallel’, and indeed, a decent 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 (large social networks), and yet for twitter, no significant motifs are found, by a wide margin,\textsuperscript{15} whereas for the friendster data the majority of the 100 most frequent motifs in the sample are significant. What exactly causes the difference in these data sets is a matter of future research; most likely requiring domain experts to investigate the motifs and their occurrences.

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 heapspace) 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.

\textsuperscript{14} http://www.mapdb.org/

\textsuperscript{15} The EL model compressed better than the motif model by millions of bits in all cases.
Preloading can be prohibitively expensive, in the same order as the analysis itself. However, the graph in database format does not take up considerably more space than it does in raw edgelist-encoding. Preloading times could therefore be eliminated by distributing graph data in a suitable indexed binary format. For example, in the domain of knowledge graphs the HDT format by Fernández et al. (2013) provides both compression, and indexing over the links of a graph.

6. Conclusion

We have introduced a new method of testing motif relevance, which allows motif analysis to be scaled up to graphs with billions of nodes, even on commodity hardware.

Our method has several advantages:

- The search for motifs only needs to be run once: on the data \(G\). Where the traditional approach requires the search to be repeated on samples from the null model, we only need to compute \(p^{null}(G)\) and compare it to \(p^{motif}(G)\).

- The search does not need to find all instances of a motif; we do not even require an accurate estimate of the total number of instances. 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. This is explained in Section 2.

- The resulting measure of relevance (the compression achieved by using the motif, in bits) can be used to compare the significance of motifs of different sizes (within the same dataset) in a meaningful way.

One observation from our experiments deserves further mention: in the first experiment, we saw that injection of one subgraph into a network caused other subgraphs to become motifs, i.e. their frequencies became statistically significant. This tells us that even if some motifs represent functional units of a network, as is both claimed (Milo et al., 2002) and contested (Konagurthu and Lesk, 2008) in the literature, the fact that a subgraph occurs with statistically significant regularity cannot be taken as proof that it is a functional unit. Hypothesis testing allows one to make a decision on the basis of noisy data, but that decision is always about the null model. A low \(p\)-value should not be interpreted as evidence for the meaning of the subgraph. At this level of abstraction, the best any method can do is to offer sound candidates for functional units.

The proof that a particular motif actually corresponds to a meaningful unit can only be achieved in context: that is, a domain expert should evaluate the list of instances found for a particular motif, to see whether a large subset of them perform the same role in the network, or if not, what other reason can be found for the prevalence of the motif. In other
words, motif analysis is necessarily an exploratory technique, and while a significance test provides a good heuristic to separate trivially frequent subgraphs from subgraphs which may represent important properties, it is ultimately just a heuristic. The only thing it proves is the incorrectness of the null model.

As noted in the introduction, since we are testing multiple motifs on the same data, we should consider the problem of multiple testing. It is however, not a traditional multiple testing scenario, because we are testing the same hypothesis multiple times. This means, for instance, that the different tests are highly dependent, and something like the Bonferroni correction is not appropriate. Moreover, when the use-case for motif detection is to generate a set of candidates for inspection by a domain expert, we are only interested in the most likely motif. Some motifs might no longer be motifs after an appropriate correction is applied, but the ranking of all subgraphs remains the same. For this reason, such corrections are not common in motif analysis. It should be up to the user to decide, on the basis of the use case, if and how to correct for multiple testing.

Our current model does not allow the detection of anti-motifs. 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.

Motif detection is hampered by the difficulty of properly evaluating the motifs found. Even showing the results to a domain expert might not suffice, as a given motif might not be good or bad at first sight. Depending on the domain, confirming a motif as meaningful might require considerable further experimentation. If the method were extended to knowledge graphs (labeled graphs expressing relational data), the resulting motifs will likely be easier to evaluate. Such an extension, however, is not trivial, and we leave this too as a matter for future research.

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 describes such graphs with the desired property efficiently, instead of finding a process that is likely to generate such graphs. For many properties, such as cliques, motifs or specific degree sequences, such a code readily suggests itself.

Acknowledgments

Acknowledgements 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.
7. Appendix

Confidence Intervals for the Degree-Sequence Model In Blitzstein and Diaconis (2011), the sample mean $1/n \sum_i 1/q_i^{DS}(G_i)$ was used as an estimator for the expectation in line (1) in the main paper. However, as shown in Genio et al. (2010), the distribution of $1/q_i^{DS}(G)$ tends to be very close to log-normal. This means that the sample mean will converge very slowly to the correct value. Specifically, the standard deviation of this estimate after $n$ samples is $\frac{1}{\sqrt{n}} \sqrt{(e^{\sigma^2} - 1)e^{2\mu + \sigma^2}}$, which for a distribution with $\mu = 200$ and $\sigma = 10$, leads to a standard deviation of approximately $\frac{1}{\sqrt{n}} e^{300}$.

For this reason, we use the maximum-likelihood estimator for the log-normal distribution instead. Let $Q_i = 1/q_i^{DS}(G_i)$. We assume $Q_i$ is log-normally distributed, so that $Y_i = \log Q_i$ is normally distributed. Let $\bar{Y} = \frac{1}{n} \sum_i Y_i$ and $S_Y = 1/n \sum_i (Y_i - \bar{Y})^2$; then the maximum-likelihood estimator of $E Q$ is $\exp (\bar{Y} + \frac{1}{2} S_Y)$. Thus, the codelength under the degree sequence model can be estimated as $(\bar{Y} + \frac{1}{2} S_Y) \log_2(e)$.

As mentioned in the body of the text, even with the highly optimized implementations described in Genio et al. (2010) and Kim et al. (2012) sampling can be slow for large graphs. In our implementation, a modern day laptop can take several minutes to produce a single sample for a random graph with $10^4$ nodes and $10^5$ links. However, we are not interested in precision beyond several orders of magnitude, so if we have a reliable method for determining confidence intervals, we can use those to provide us with safe bounds.

Since we are dealing with a log-normal source, we cannot simply use twice the standard error of the mean to approximate our error bars. We will use the parametric bootstrap procedure provided in Angus (1994); Zhou and Gao (1997). To substantiate this method, we test the coverage of the two-sided symmetric confidence interval on three data sets. We proceed as follows: first we estimate the true value of $|G_D|$ with the ML estimator, using $10^6$ samples. Call this value $g$. We use this as our gold standard. We then sample a small number (5, 10, 20) of graphs and their associated probabilities. Using the bootstrap method we construct a two-sided symmetric confidence interval with $\alpha = 0.05$ on this sample. We repeat the procedure of sampling data and constructing an interval 5000 times and report the proportion of times $g$ was inside the confidence interval. If the bootstrap method is accurate, the resulting value should be close to 0.95. We use the following data sets:

- **cattle**: Observed dominance behaviors between cows. A directed graph with 28 nodes, and 217 links. Schein and Fohrman (1955); kon (2015)

- **revolution**: Affiliations of 136 people to 5 organizations encoded as a bipartite graph. 141 nodes and 160 links. KONECT (2015a)

- **random**: A simple undirected random graph of 50 nodes, with each pair of distinct nodes connected with probability 0.5.

As Table 2 shows, this method becomes relatively reliable at around 10 samples, although the intervals are quite large at that sample size.

Now, when we use $L^{DS}$ as a base model in $L^{motif}$, the intractable value $|G_D|$ occurs in two places: the encoding of the subgraph, and the encoding of the template graph. Since we use our estimator for both, we must be careful to end up with a correct confidence interval for the resulting motif code. Let $D'$ be the degree sequence of the subgraph,
and $D$ be the degree sequence of the template. Then, we can split the total code length into three components: $\log |G_D'|$, $\log |G_D|$ and $R$. $R$ is the sum of all parts of the code that we can compute exactly, including the sizes and sequences of the motif and template graph (i.e. everything but $\log |G_D'|$ and $\log |G_D|$). The total code length is described by $\log |G_D'| + \log |G_D| + R$, where the first two terms require the use of the estimator. Let $Q_m$ and $Q_h$ be random variables representing the inverse probability of graphs sampled from the importance sampling algorithm, for the degree sequence of the motif and the template graph respectively. In other words, for the degree sequence of the motif and the template graph respectively. In other words, the true code length for the motif code is

$$\log EQ_m + \log EQ_h + R$$

$$= \log EQ_mEQ_h + R$$

$$= \log E[Q_mQ_h] + R$$

where the last line follows from the fact $Q_m$ and $Q_h$ are independent. So to get a correct confidence interval, we can take the same number of samples of both $Q_m$ and $Q_h$, multiply their probabilities, and perform the bootstrap analysis on the list of these multiplied probabilities (since we are summing the logarithms of log-normally distributed variables, the result is log-normally distributed as well).

### References

Cattle network dataset – KONECT, January 2015. URL http://konect.uni-koblenz.de/networks/moreno_cattle

Charu C Aggarwal and Jiawei Han. *Frequent pattern mining*. Springer, 2014.

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

John E Angus. Bootstrap one-sided confidence intervals for the log-normal mean. *The Statistician*, pages 395–401, 1994.

Alex Arenas, Alberto Fernandez, Santo Fortunato, and Sergio Gómez. Motif-based communities in complex networks. *Journal of Physics A: Mathematical and Theoretical*, 41 (22):224001, 2008.
Sren Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a web of open data. In Proc. Int. Semantic Web Conf., pages 722–735, 2008.

Edward A Bender and E Rodney Canfield. The asymptotic number of labeled graphs with given degree sequences. Journal of Combinatorial Theory, Series A, 24(3):296–307, 1978.

Ivona Bezáková, Adam Kalai, and Rahul Santhanam. Graph model selection using maximum likelihood. In William W. Cohen and Andrew Moore, editors, Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006, volume 148 of ACM International Conference Proceeding Series, pages 105–112. ACM, 2006. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143858. URL http://doi.acm.org/10.1145/1143844.1143858.

Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In 2012 IEEE 12th International Conference on Data Mining, pages 91–100. IEEE, 2012.

Joseph K. Blitzstein and Persi Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. Internet Mathematics, 6(4):489–522, 2011. doi: 10.1080/15427951.2010.557277. URL http://dx.doi.org/10.1080/15427951.2010.557277

C. J. Carstens. Motifs in directed acyclic networks. In International Conference on Signal-Image Technology & Internet-Based Systems, SITIS 2013, Kyoto, Japan, December 2-5, 2013, pages 605–611. IEEE, 2013. doi: 10.1109/SITIS.2013.99. URL http://dx.doi.org/10.1109/SITIS.2013.99

Corrie Jacobien Carstens, Annabell Berger, and Giovanni Strona. Curveball: a new generation of sampling algorithms for graphs with fixed degree sequence. arXiv preprint arXiv:1609.05137, 2016.

James Coleman, Elihu Katz, and Herbert Menzel. The diffusion of an innovation among physicians. Sociometry, pages 253–270, 1957.

Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. CoRR, cs.AI/9402102, 1994. URL http://arxiv.org/abs/cs.AI/9402102

Thomas M. Cover and Joy A. Thomas. Elements of information theory (2. ed.). Wiley, 2006. ISBN 978-0-471-24195-9.

Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (ldt). Web Semantics: Science, Services and Agents on the World Wide Web, 19:22–41, 2013.

Johannes Gehrke, Paul Ginsparg, and Jon Kleinberg. Overview of the 2003 kdd cup. ACM SIGKDD Explorations Newsletter, 5(2):149–151, 2003.
Finding Network Motifs in Large Graphs

Charo I. Del Genio, Hyunju Kim, Zoltán Toroczkai, and Kevin E. Bassler. Efficient and exact sampling of simple graphs with given arbitrary degree sequence. *CoRR*, abs/1002.2975, 2010. URL http://arxiv.org/abs/1002.2975.

Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, pages 1141–1144, 1959.

P.D. Grünwald. *The minimum description length principle*. The MIT Press, 2007.

Tomaž Hočevar and Janez Demšar. A combinatorial approach to graphlet counting. *Bioinformatics*, 30(4):559–565, 2014.

Madhav Jha, C Seshadhri, and Ali Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th International Conference on World Wide Web*, pages 495–505. ACM, 2015.

Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004. doi: 10.1093/bioinformatics/bth163. URL http://dx.doi.org/10.1093/bioinformatics/bth163.

Sahand Khakabimamaghani, Iman Sharafuddin, Norbert Dichter, Ina Koch, and Ali Masoudi-Nejad. Quatexelero: an accelerated exact network motif detection algorithm. *PloS one*, 8(7):e68073, 2013.

Jack Kiefer. Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, 4(3):502–506, 1953.

Hyunju Kim, Charo I Del Genio, Kevin E Bassler, and Zoltán Toroczkai. Constructing and sampling directed graphs with given degree sequences. *New Journal of Physics*, 14(2):023012, 2012.

Stefan Knerr, Léon Personnaz, and Gérard Dreyfus. Single-layer learning revisited: a step-wise procedure for building and training a neural network. In *Neurocomputing*, pages 41–50. Springer, 1990.

Arun S Konagurthu and Arthur M Lesk. On the origin of distribution patterns of motifs in biological networks. *BMC Systems Biology*, 2(1):73, 2008.

KONECT. King james network dataset – KONECT, October 2014. URL http://konect.uni-koblenz.de/networks/moreno_names

KONECT. American revolution network dataset – KONECT, April 2015a. URL http://konect.uni-koblenz.de/networks/brunson_revolution.

KONECT. Physicians network dataset – KONECT, April 2015b. URL http://konect.uni-koblenz.de/networks/moreno_innovation.

KONECT. Wikipedia, english network dataset – KONECT, October 2016a. URL http://konect.uni-koblenz.de/networks/dbpedia-link.
KONECT. Friendster network dataset – KONECT, October 2016b. URL http://konect.uni-koblenz.de/networks/friendster.

KONECT. Wikipedia, nl (dynamic) network dataset – KONECT, October 2016c. URL http://konect.uni-koblenz.de/networks/link-dynamic-nlwiki.

KONECT. Twitter (www) network dataset – KONECT, October 2016d. URL http://konect.uni-koblenz.de/networks/twitter.

Michel Koskas, Gilles Grasseau, Etienne Birmelé, Sophie Schbath, and Stéphane Robin. Nemo: Fast count of network motifs. Book of Abstracts for Journées Ouvertes Biologie Informatique Mathématiques (JOBIM), pages 53–60, 2011.

Danai Koutra, U. Kang, Jilles Vreeken, and Christos Faloutsos. Summarizing and understanding large graphs. Statistical Analysis and Data Mining, 8(3):183–202, 2015. doi: 10.1002/sam.11267. URL http://dx.doi.org/10.1002/sam.11267.

Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In Proc. Int. World Wide Web Conf., pages 591–600, 2010.

Xin Li, Douglas S Stones, Haidong Wang, Hualiang Deng, Xiaoguang Liu, and Gang Wang. Netmode: Network motif detection without nauty. PloS one, 7(12):e50093, 2012.

Brendan D McKay et al. Practical graph isomorphism. Department of Computer Science, Vanderbilt University Tennessee, US, 1981.

Luis AA Meira, Vinícius R Máximo, Álvaro L Fazenda, and Arlindo F Da Conceição. Accmotif: accelerated network motif detection. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 11(5):853–862, 2014.

Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. Science, 298(5594):824–827, 2002.

Mark Newman. Networks: an introduction. Oxford University Press, 2010.

Pedro Paredes and Pedro Ribeiro. Rand-fase: fast approximate subgraph census. Social Network Analysis and Mining, 5(1):1–18, 2015.

Trang Pham, Truyen Tran, Dinh Phung, and Svetla Venkatesh. Column networks for collective classification. arXiv preprint arXiv:1609.04508, 2016.

Franck Picard, J-J Daudin, Michel Koskas, Sophie Schbath, and Stephane Robin. Assessing the exceptionality of network motifs. Journal of Computational Biology, 15(1):1–20, 2008.

Julia Preusse, Jérôme Kunegis, Matthias Thimm, Thomas Gottron, and Steffen Staab. Structural dynamics of knowledge networks. In Proc. Int. Conf. on Weblogs and Social Media, 2013.
Finding Network Motifs in Large Graphs

Teresa Reguly, Ashton Breitkreutz, Lorrie Boucher, Bobby-Joe Breitkreutz, Gary C Hon, Chad L Myers, Ainslie Parsons, Helena Friesen, Rose Oughtred, Amy Tong, et al. Comprehensive curation and analysis of global interaction networks in saccharomyces cerevisiae. Journal of biology, 5(4):11, 2006.

A Renyi and P Erdős. On random graphs. Publicationes Mathematicae, 6(290-297):5, 1959.

Pedro Ribeiro and Fernando Silva. G-tries: an efficient data structure for discovering network motifs. In Proceedings of the 2010 ACM Symposium on Applied Computing, pages 1559–1566. ACM, 2010.

Jorma Rissanen. Modeling by shortest data description. Automatica, 14(5):465–471, 1978.

Jorma Rissanen and Glen G Langdon. Arithmetic coding. IBM Journal of research and development, 23(2):149–162, 1979.

Petar Ristoski, Gerben Klaas Dirk de Vries, and Heiko Paulheim. A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In International Semantic Web Conference, pages 186–194. Springer, 2016.

Christoph Römhild and Chris Harrison. http://chrisharrison.net/projects/bibleviz/index.html, 2007. Accessed: 2014-08-22.

Martin W. Schein and Milton H. Fohrman. Social dominance relationships in a herd of dairy cattle. The British J. of Animal Behaviour, 3(2):45–55, 1955.

Falk Schreiber and Henning Schwobbermeyer. Towards motif detection in networks: frequency concepts and flexible search. Proc. Intl. Wsh. Network Tools and Applications in Biology (NETTAB04), pages 91–102, 2004.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M Borgwardt. Efficient graphlet kernels for large graph comparison. In AISTATS, volume 5, pages 488–495, 2009.

George M Slota and Kamesh Madduri. Complex network analysis using parallel approximate motif counting. In Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, pages 405–414. IEEE, 2014.

Giovanni Strona, Domenico Nappo, Francesco Boccacci, Simone Fattorini, and Jesus San-Miguel-Ayanz. A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. Nature communications, 5, 2014.

Matthijs van Leeuwen, Jilles Vreeken, and Arno Siebes. Compression picks item sets that matter. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Berlin, Germany, September 18-22, 2006, Proceedings, volume 4213 of Lecture Notes in Computer Science, pages 585–592. Springer, 2006. ISBN 3-540-45374-1. doi: 10.1007/11871637_59. URL http://dx.doi.org/10.1007/11871637_59.
Yuii Wang and Jan Ramon. An efficiently computable support measure for frequent subgraph pattern mining. *Machine Learning and Knowledge Discovery in Databases*, pages 362–377, 2012.

Sebastian Wernicke. A faster algorithm for detecting network motifs. In Rita Casadio and Gene Myers, editors, *Algorithms in Bioinformatics, 5th International Workshop, WABI 2005, Mallorca, Spain, October 3-6, 2005, Proceedings*, volume 3692 of *Lecture Notes in Computer Science*, pages 165–177. Springer, 2005. ISBN 3-540-29008-7. doi: 10.1007/11557067_14. URL http://dx.doi.org/10.1007/11557067_14.

Guangyu Wu, Martin Harrigan, and Pádraig Cunningham. Characterizing wikipedia pages using edit network motif profiles. In *Proceedings of the 3rd international workshop on Search and mining user-generated contents*, pages 45–52. ACM, 2011.

X.H. Zhou and S. Gao. Confidence intervals for the log-normal mean. *Statistics in medicine*, 16(7):783–790, 1997.