Detecting Network Motifs in Knowledge Graphs using Compression

July 10, 2017

Abstract

We introduce a method to detect *network motifs* in knowledge graphs. Network motifs are useful patterns or meaningful subunits of the graph that recur frequently. We introduce a scalable approach for detecting such motifs in large knowledge graphs, inspired by recent work for simple graphs, and show that the motifs returned reflect the basic structure of the graph. Specifically, we show that common motifs reflect graph patterns used in common queries, basic schematic units of the graph and meaningful functional subunits, for various knowledge graphs.

The Linked, Open Data cloud contains a wealth of knowledge graphs, and is growing quickly. At the time of writing the average knowledge graph contains over 65 000 edges, with many datasets containing more than a million.[1] For such large graphs, it can be difficult to see the forest for the trees: how is the graph structured at the lowest level? What kind of things can I ask of what types of objects? What are small, recurring patterns that might represent a novel insight into the data?

In the domain of plain (unlabelled) graphs, *network motifs* [**?**] were recently introduced as a tool to provide users of large graphs insight into their data. Network motifs are small subgraphs whose frequency is unexpected with respect to a null model. That is, for a given graph dataset g and small graph m, we count the frequency $F(m, g)$: how often m occurs in g as a subgraph. We define a probability distribution over graphs $p^{\text{null}}(g)$, and estimate the probability that a graph sampled from p^{null} contains more instances of m than observed in our data: $p^{\text{null}}(F(m, G) \geq F(m, g))$, where

[1] `http://stats.lod2.eu/stats`

G is a random variable representing a graph. If this probability is low (commonly, below 0.05), we consider m a motif. [2]

In [?], an alternative method is presented that uses compression as a heuristic for motif value: the better a motif compresses the data, the more likely it is to be meaningful. It is shown that this principle can be implemented in a similar sort of hypothesis test for a null model, allowing a comparable workflow to classical motif analysis.

In this paper, we extend the compression-based motif analysis to Knowledge Graphs. For the purposes of this research we define Knowledge graphs as labeled multigraphs. Nodes are labeled with entity names, and links are labeled with relations. We extend the definition of a motif to that of a basic graph pattern: that is, a motif is a small graph with some of its nodes and links labeled. The motif occurs in the graph where the graph structure matches, and the labels. The unlabeled nodes and links are free to contain whatever label when the motif is mapped to the data.

To maintain the connection to graph pattern queries, we do not require the motifs to be *induced* subgraphs: that is, when a motif is mapped to a graph, the graph can contain additional links that are not specified by the motif.

We perform several experiments to show that our method returns meaningful subgraphs. First, we look at several graphs for which schema information is provided. That is, we know the basic properties available for each entity. We show that the motifs found by our method correspond well to the schema, and compare performance against several basic benchmarks. Second, we run motif analysis on several datasets for which a list of real SPARQL queries entered by users is available. From these, we extract the basic graph patterns, and show that (a) these correspond to good motifs using our criterion and (b) our method can return such motifs with good precision and recall.

All code and benchmark datasets used in the paper are available, under open licenses.[3]

[2]This procedure has the structure of a hypothesis test, but it is important not to interpret it as statistical evidence for the meaningfulness of the motif. The only thing it proves (in a frequentist statistical sense) is that p^{null} is not the true source of the data. This is usually not a surprise: we are rarely able to model all aspects of a realistic data-generating process in a single distribution. The p-values used in motif analysis should be interpreted strictly as *heuristics*. See also [?].

[3]...

0.1 Related Work

0.2 Preliminaries

1 Method

We will first describe our method under the assumption that a good null model p^{null} is available, and that $-\log p^{\text{null}}(G)$ can be efficiently computed. In Section 2, we explain which null model we use for our experiments, and how it is implemented.

We will start with some formal definitions of our basic ingredients. To comfortably define probabilities and codes on graphs, we will slightly deviate from conventional definitions. Let V be a countably infinite set containing all possible entities, and let R be a countably infinite set containing all possible relations between them. Let $v : V \to \mathbb{N}$ and $r : R \to \mathbb{N}$ be arbitrary mappings of these sets to the natural numbers.[4]

A *knowledge graph* G, hereafter known simply as a *graph* is a, is tuple $G = (V_G, E_G)$ with finite sets $V_G \subset V$ and $E_G \subset V \times R \times V$. We define the tripleset of G as $G^T = \{(s, p, o) \in \mathbb{N}^3 : (v(s), r(p), v(o)) \in E_G\}$. In this paper we will *identify* the graph with the triple list. That is when we speak of encoding the graph, we mean encoding the triple list, and when we speak of a probability distribution over graphs, we mean a probability distribution over triple lists.

This means that we are purposely *not* modelling certain aspects of the data. There might, for instance bet a similarity between two node labels $v(a)$ and $v(b)$ (such as a shared IRI prefix). We are not interested in exploiting such structure, so we eliminate it from our notation. Additionally, is we use a simple code to store the integers in the triple-list we will end up using fewer bits for those nodes and triples with labels who were arbitrarily assigned short integers. Equivalently, when sampling from a simple distribution over the integers, small numbers will be more likely to be generated. In practice, however, we will always learn a distribution over the integers from the data, so that the choice of e and r will not affect the code.

As shown in Figure **??**, the motifs that we aim to find are partially labeled: some nodes and edges are labeled with the marker "?" indicating that when we fit the motif to the graph, those edges can contain any label. In other words, the motif is a *basic graph pattern*, and the edges and nodes marked with "?" are variables. To represent this, we add a single variable

[4]For instance, let V and E be the set of finite strings, and let v and e represent length-lexicographical order.

symbol ?. Thus, a motif M is defined by the triple set $M \subset (? \cup \mathbb{N})^3$, with integer values in triples interpreted as in complete knowledge graphs.

1.1 Motif search

Given a graph G, our aim is to find motifs M that represent meaningful building blocks, or that are otherwise characteristic for G. In the design of our method, we will constantly aim to find a trade-off between completeness and efficiency that allows the method to scale to very large graphs. Specifically, when we economize, we will only do so in a way that makes the hypothesis test described in Section 1.2 *more conservative.* For instance, we will use a very fast search for candidate motifs, which returns only a limit number of instances. A more expensive search might yield more motifs, but never less. That is, for those motifs that we find, we can be sure that they would also be found by a more expensive method. Since pattern mining of this kind often results in too many potential patterns rather than too few, it seems efficient to start with the motifs that can be found with fast, simple methods, and only extend the search once these have been exhausted.

Specifically, we generate candidate motifs by *sampling* them from the data. This method, inspired by [**?**] is based on the simple insight that when we sample a pattern from the data, we are more likely to sample a frequently occurring pattern than a infrequent one. In we sample K times, and rank the results by relative frequency within the sample, we are likely toe end up with a good indication of the most frequent patterns in the data. For unlabeled graphs, as little as 500 samples can be sufficient to find the most frequent subgraph. Theoretically, this allows us to generate promising motifs with time complexity indepenedent of the size of the graph. Our sampling algorithm is described in Figure **??**

1.2 Relevance test

1.3 Motif code

2 Null model

The most common null model for classical motif analysis is the degree-sequence model (also known a the configuration model []): a uniform distribution over all graphs that share the same in and out degrees of the data for every node. We extend this to knowledge graphs by also including the frequency with which each relation occurs. Let a *degree sequence* D of length n be a triple of three integer sequences: $(D^{\text{in}}, D^{\text{rel}}, D^{\text{out}})$. If D is the degree sequence of a graph, then node i has D_i^{in} incoming links, D_i^{out} outgoing links and for each relation r, there are D_r^{rel} links.

Let $\mathcal{G}_D$ be the set of all graphs with degree sequence D. Then the configuration model can be expressed simply as

$$p(G) = \frac{1}{|\mathcal{G}_D|}$$

for any G that satisfies D and $p(G) = 0$ otherwise. Unfortunately, there is no efficient way to compute $|\mathcal{G}_D|$ and even approximations tend to be costly for large graphs. Following the approach in [**?**], we define an approximation. We can define a knowledge graph by three length-m integer sequences: S, P, O, with $\left(S_j, P_j, O_j\right)_j$ the graph's tripleset. If the graph satisfies degree sequence D, then we know that S should contain node j D_j^{out} times, P should contain relation r D_r^{rel} times and O should contain node j D_j^{in} times. Let $\mathcal{S}_D$ be the set of all such triples of integer sequences satisfying D. We have

$$|\mathcal{S}_D| = \binom{m}{D_1^{\text{out}}, \ldots, D_n^{\text{out}}}\binom{m}{D_1^{\text{rel}}, \ldots, D_n^{\text{rel}}}\binom{m}{D_1^{\text{in}}, \ldots, D_n^{\text{in}}}.$$

While every member of $\mathcal{S}_D$ contains a valid graph satisfying D, many graphs are represented multiple times. Firstly, many elements of S_D contain the same link multiple times, which means they are not is $\mathcal{G}_D$. We call the set without these elements $\mathcal{S'}_D \subset \mathcal{S}_D$. Secondly the links of the graph are listed in arbitrary order; if we apply the same permutation to all three lists S, P and O, we get an encoding of the same graph. Since we know that any element in $\mathcal{S}'_D$ contains only unique triples, we know that each graph is present exactly $m!$ times. Thus, we have

$$|\mathcal{G}_D| = |\mathcal{S}'_D|\frac{1}{m!} \leq |\mathcal{S}_D|\frac{1}{m!}.$$

We can thus use

$$p_D^{\text{EL}}(G) = \frac{m!}{\binom{m}{D_1^{\text{out}}, \ldots, D_n^{\text{out}}}\binom{m}{D_1^{\text{rel}}, \ldots, D_n^{\text{rel}}}\binom{m}{D_1^{\text{in}}, \ldots, D_n^{\text{in}}}}$$

as an approximation for the DS model. We call this the edge-list (EL) model. It always lower-bounds the true degree sequence model, since it affords some probability mass to graphs that cannot exist. [5]Experiments in the classical motif setting have shown that the ES model is an acceptable proxy [**?**], especially considering the extra scalability it affords.

[5]Note that we cannot simply think of p^{ES} as a uniform model for graphs containing multiple triples, since in we can only divide by $m!$ is we know that all triples are unique.

Encoding D In order to encode a graph with L^{EL}_{D}, we must first encode D. [6] For each of the three sequences in D we use the following model:

$$p(D) = \prod_i p^{\mathbb{N}}(D_i) L(D) = -\sum_i \log p^{\mathbb{N}}(D_i)$$

where $p^{\mathbb{N}}$ is a distribution on the natural numbers. This is an optimal encoding for D assuming that its members are independently drawn from $p^{\mathbb{N}}$. When we use p^{DS} as the null model, we use the data distribution for $p^{\mathbb{N}}$ to ensure that we have a lower bound to the optimal code-length. When we use p^{DS} as part of the motif code, we must use a fair encoding, so we use a Dirichlet-Multinomial model to store D.

3 Experiments

3.1 Schema reconstruction

3.2 Query induction

3.3 Scale

To show the scale of our method, we show its performance on some very large graphs, culminating with a dump of the complete LOD cloud, containing 38 billion triples, from [].

3.4 Various

In this section, we show some experiments that are not intended to substantiate any claims about our method, but rather to illustrate the variety of uses for succesfull motif detection.

3.4.1 Analogical reasoning

3.4.2 Visualization

3.4.3 Feature extraction

3.4.4 Node embedding

[6] Or, equivalently, to make p^{EL} a complete distribution on all graphs, we must provide it with a prior on D.