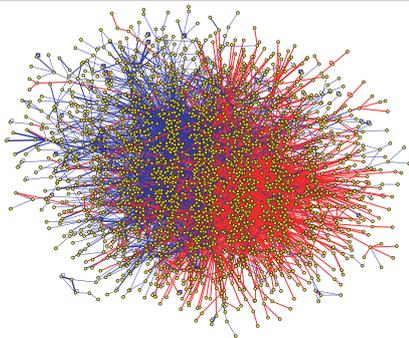


Large-scale NETWORK MOTIF ANALYSIS using compression

Peter Bloem, Steven de Rooij
 vub@peterbloem.nl



The title of our paper is “Large-scale network motif analysis using compression”. My name is Peter Bloem, and my co-author is Steven de Rooij.



source: Towards a proteome-scale map of the human protein-protein interaction network, Rual et al. 2005

Graphs, or networks, are a very powerful way of capturing and storing knowledge. However, the first view we get of our data is usually something like this: a big, tangled ball of wool. It looks impressive, but it tells us very little.

One solution, is to look for *building blocks*: small substructures that occur often, and where they occur play the same role. This is called motif analysis.

An analogy

structure in text versus structure in graphs



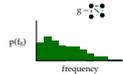
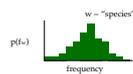
Most frequent patterns

the, be, to, of, and, a, ...

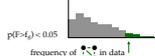
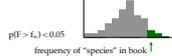
frequent subgraphs

Null model

a distribution on books/graphs



Expectedness of frequency with respect to null-model

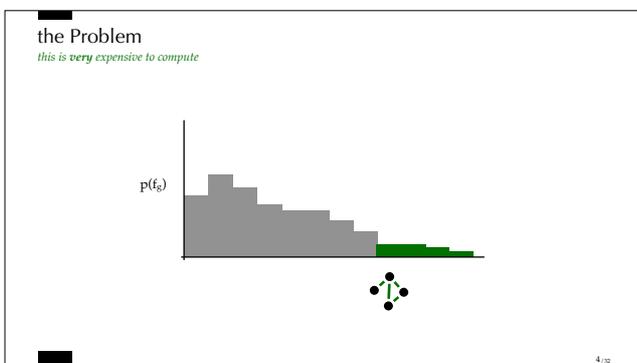


To explain by analogy, imagine that you are given a book, and you want to find its building blocks. Your first approach might be to simply look for *frequent* substructures. This would give you a list of words like the, be, to etc. This may be useful, but it isn't very specific to this particular book: you'd get the same result with any other English book.

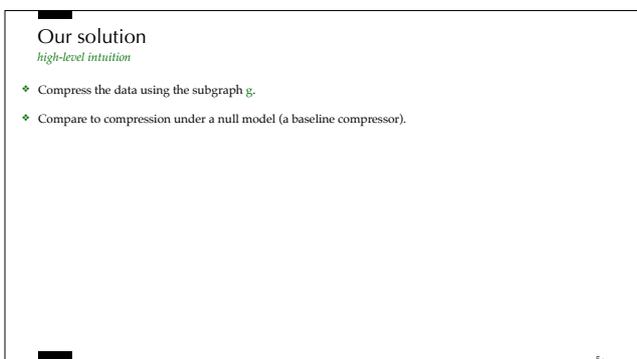
To find patterns that are particular to this book, we need a null model: a probability distribution that tells us, for a particular word, in this case “species” what frequencies we are likely to see in a book of this length.

We can then take the frequency of the word species in *our* book, and compute the probability of seeing the word “species” at least that often. If that probability is very low, we can say that we’ve seen the word unexpectedly often, and it is therefore a characteristic word for our book.

The same logic applies to graphs: we count subgraphs, but we evaluate their frequencies against those predicted by a null model.



The problem is that for a given distribution on graphs, the resulting distribution on frequencies is very expensive to compute. Usually, the best we can do is to sample a large number of graphs, and count the frequencies of the subgraph in each, building up a histogram approximation.



We propose the following approach; we compress the data using the fact that subgraph g occurs very often. Then, we compare the size of the compressed graph to the compression achieved by a baseline model.

If the first compressor does substantially better, we consider the subgraph a motif. This may seem like a radically different approach, but we can frame it in the same probabilistic terms as the traditional method.

Our solution

in probabilistic terms

- ◆ Minimum Description Length (MDL): Every probability distribution is a compressor and vice versa.
 - Given p with probability $p(x)$, we have L with codelength $L(x) = 2^{p(x)}$
- ◆ **DON'T**: Compute the probability of g having frequency f under the null model.
- ◆ **DO**: Compute the probability that $L^{\text{motif}}(g) \ll L^{\text{null}}(g)$ under the null model.
 - If g comes from p^{null} , no compressor can substantially beat L^{null} .

6/12

Results

a preview

- ◆ Graphs with billions of edges can be usefully analyzed in ~9 hrs on a single compute node.
- ◆ Quality of motifs found is comparable to the traditional method.
 - We introduce a classification-based experiment for measuring this.

7/12

Here is a brief preview of the results; our methods allows very large graphs to be analyzed on modest hardware, and we show that the quality of the resulting motif is comparable to that of the traditional method.

Outline

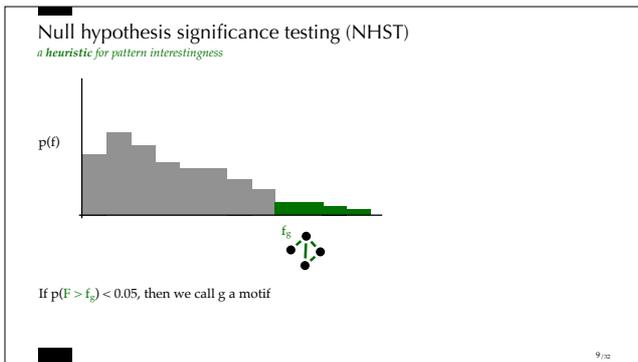
- ◆ MDL hypothesis testing for pattern analysis
 - preliminaries of MDL
 - NHST testing with MDL
- ◆ Network motif analysis with MDL
 - Motif code
 - Null models
- ◆ Results

8/12

Here is the outline for the rest of the talk.

Traditional motif analysis is based on the use of significance testing as a proxy for pattern mining. Using ideas from minimum description length theory, we can put our method in the same framework. We will discuss the basics of MDL and how we can perform significance testing with it.

We will then describe our motif code, and the null models we use, and finish up with some experimental results.

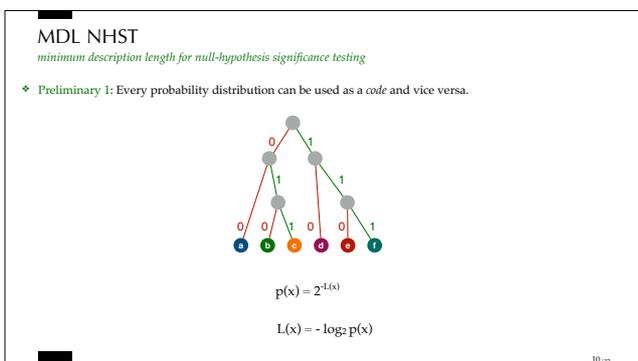


The traditional method is based on significance testing.

Our null hypothesis is that the data came from the null model. We then observe an event, a high frequency of subgraph g , that is so improbable under the null model, that we can reject the null hypothesis. The data couldn't have come from the null model.

The null model isn't usually very realistic, so this isn't that interesting in itself. However, we then use this as a *heuristic* for whether a subgraph is interesting. We say that any subgraph that allows us to reject the null hypothesis, is interesting.

To cast our method in the same framework, we need to look at the basics of the minimum description length principle.



The founding principle behind MDL is that codes, i.e. compressors, can be equated with probability distributions.

Imagine that we want to encode the six target objects **a** through **f**. The codes we use in MDL, so called prefix-free codes, have the property that they can be drawn as a binary tree with the target objects on the leaves. The codewords can be read off the path from the root to the target leaf.

We can easily turn this into a probability distribution: we start at the root, and flip a coin to decide whether to move left or

right. A leaf will be chosen with probability $2^{-\text{length}}$ minus the length of its codeword.

More surprising, is that the reverse is also possible. If we allow a one-bit margin of error, any probability distribution can be turned into a code, so that the length of the codeword for an object is the negative logarithm of its probability.

The takeaway is that we can equate probability distributions and codes. A distribution that assigns x a high probability with assign it a short

MDL NHST
minimum description length for null-hypothesis significance testing

- ◆ Preliminary 2: the no-hypercompression inequality.
 - If x is sampled from p^{null} , then no code L^* can compress x substantially better than L^{null} .
 - More precisely:

$$p^{\text{null}}(L^{\text{null}}(x) - L^*(x) \geq k) \leq 2^{-k}$$

11/12

The second principle we will use is the no hypercompression inequality. It states that if we sample x from a distribution, no other distribution will give us a better compression than the code corresponding to the source of x .

Specifically the probability that any other code does better by k bits decays exponentially. Note that this means that seen an improvement of even 30 bits is a one-in-a-billion event.

MDL NHST
minimum description length for null-hypothesis significance testing

- ◆ Choose alternative code L^* before seeing the data.
- ◆ Compute $L^{\text{null}}(x)$, $L^*(x)$

- ◆ We can reject p^{null} as the source of the data with p-value 2^{-k}

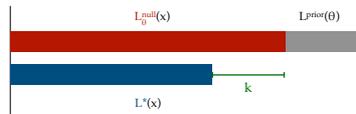
12/12

Using the no-hypercompression inequality, we can define an MDL-based hypothesis test: we define an alternative code L^* , before seeing the data, and if it compresses x better than the code corresponding to the **null model**, we can reject the **null model** as the source of the data with p-value 2^{-k} .

MDL NHST with parameter θ

minimum description length for null-hypothesis significance testing

- ◆ For example $p_{\text{ER}}^{\text{null}}(g)$: equal probability for all graphs with dimensions n, m (zero for all others).
- ◆ Two part coding $L_{\text{ER}}^{\text{null}}(g) = L_{\text{prior}}(\theta) + L_{\text{code}}^{\text{null}}(g)$



- ◆ We can reject p^{null} , for all priors, as the source of the data with confidence at least 2^{-k}

13/32

If our null model has a parameter, we need to deal with that. For instance, the Erdos-Renyi model assigns equal probability to all graphs with particular dimensions (n, m) . This is not a code for all graphs. To make it one, we can use two-part coding: we first use a different code to store (n, m) and then use that information to encode the graph using the Erdos-Renyi code. We call this first code the prior.

This raises the question how much the choice of prior influences the outcome. To eliminate the choice entirely, we can compare to the null code minus the prior, which is always a lowerbound on the total codelength. This makes things harder for L^* , but if we still manage to reject the null model, we can reject it for *all possible priors*.

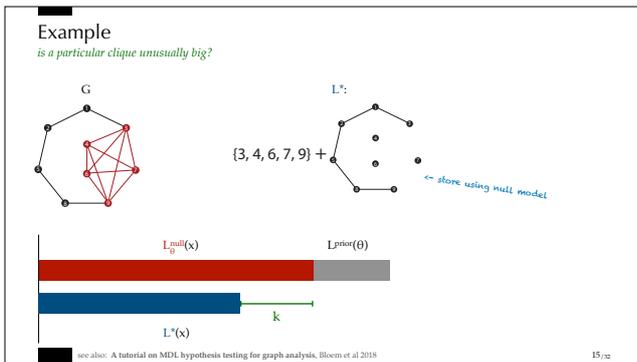
Example

is a particular clique unusually big?

- ◆ Let G be a large graph with clique C
 - Is C unusually big?
- ◆ Traditional approach: measure clique sizes in a large sample from p^{null} , and compare.
- ◆ MDL: Design a code L^* that uses the fact that C is a clique to compress.
 - L^* : record a list of nodes in C , remove all edges in C , store the remainder with L^{null} .

see also: A tutorial on MDL hypothesis testing for graph analysis, Bloem et al 2018

14/32



Here's an example for how to use an MDL hypothesis test to show that a clique in unexpectedly large. We first design a code that uses the fact that there is a large clique to compress the graph: we store the nodes of the clique, remove all its edges from the graph, and store the remained. This information is enough to reconstruct the graph.

When we store the rest of the graph, we do so using the null model, so that the only advantage the alternative code has is that it knows the clique.

If this code beats the null model code by k bits, discounting the prior, we can reject the null model with confidence 2^{-k} and take this as a heuristic for the meaning of the clique.

Caveats and reminders

- ◆ We prove (statistically) that p^{null} is not the source of the data. We do **not** prove (in any sense) that the pattern we used is interesting or characteristic. The NHST is a *heuristic* for pattern mining, that the alternative model is the source of the data.
- ◆ In fact p^{null} is usually chosen as a trade-off between scalability and power, and so is p^* .
- ◆ All of this is true for both the MDL and the traditional approach.

16/32

It's important to emphasize that we are *proving* only that the null model was not the source of the data. We use the fact that a pattern allows us to do this, as a heuristic for the pattern being meaningful, but we haven't proved anything about the pattern.

In fact, because the null model is usually chosen as a tradeoff between scalability and power, the fact that the null model can be rejected is rarely surprising.

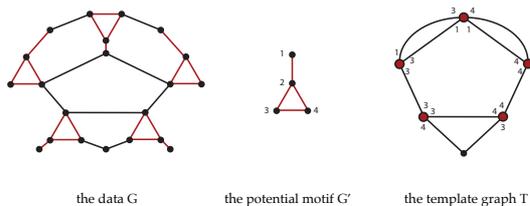
Outline

- ◆ MDL hypothesis testing for pattern analysis
 - preliminaries of MDL
 - NHST testing with MDL
- ◆ Network motif analysis with MDL
 - Motif code
 - Null models
- ◆ Results

17/32

Now that have our framework in place, we just need to define a motif code, and some null models.

Motif code



18/32

Here is the basic principle behind our motif code: we remove the motif from the graph, replacing each instance by a special motif node. These nodes are annotated to indicate which node inside the motif the edge connects to.

We store the motif once, together with the template graph T. This information is sufficient to reconstruct the graph. Note that we've skipped a lot of details here for the sake of time.

Motif code

```

Given:
  a graph G, a subgraph G',
  a list M of instances of G' in G, a code L^mot on the simple graphs.
  null model
b_subgraph ← L^mot(G')                                subgraph
# replace each instance with a single node
H ← copy(G), W = []                                    template
for each M = {m1, ..., m|M|} in M:
  # We use m1 (the m1-th node in G) as the instance node
  for each link l between a node mout not in M and a node mj in M:
    if j ≠ 1: add a link between mout and mj
    W.append(j)
  remove all nodes m1, except m1, and all incident links
  b_wiring ← L^Dir(G')(W)                               rewiring
# Remove multiple edges from H and record the duplicates in R
R, H' ← simple(H)
b_template ← L^mot(H')
b_multi-edges ← L^N(maxc(R)) + L^DirM_maxc(R)           multiple edges
b_instances ← L^N(|M|) log^N(|M|)                    instance nodes
b_insertions ← log(n(G))! - log(n(H))!                insertions
return b_subgraph + b_template + b_wiring + b_multi-edges + b_instances + b_insertions
    
```

19/32

Note that we only compute the codelength directly. We are never computing the actual codewords.

Searching for subgraphs and instances

quick and dirty

- ◆ Take a short random walk, extract induced subgraph, repeat (1M iterations for all experiments).
- ◆ Keep a dictionary from (canonicalized) subgraphs to a list of known instances.
- ◆ This algorithm is
 - $O(1)$ in the size of the graph,
 - very biased.

20/12

The motif code itself doesn't tell us which subgraphs to try. For that, we need to generate candidate subgraphs, and their instances in the graph.

Probably the fastest way of doing this is to sample subgraphs by doing short random walks. After a random walk, we extract the induced subgraph of the encountered nodes and keep a dictionary mapping subgraphs to their instances in the graph.

The running time of this algorithm is independent of the size of the graph, so very scalable. But because the *frequencies* it returns are very biased, it has never been reliable for traditional motif analysis. In our method, this bias doesn't affect the correctness of the hypothesis test.

Null models

defining null models in the context of MDL

- ◆ Erdős-Renyi (ER) model: uniform distribution over all graphs with the same dimensions.
- ◆ Degree sequence (DS) model: uniform distribution over all graphs with degree sequence d .
- ◆ Edgelist (EL) model: approximation to the DS model.
 - Strictly worse, but much cheaper to compute.

21/12

We've seen the Erdos-Renyi model already, but a more common distribution in motif analysis is the **degree sequence distribution**. This distribution which assign equal probability to all graphs with the same degree sequence.

The DS model is a bit expensive to compute, so we also introduce a cheaper alternative, the EL model. This model is strictly worse, but it can be computed in linear time in the length of the degree sequence.

In our experiments we will pay particular attention to whether the EL model is an

acceptable proxy for the DS model, since the use of the EL model is crucial to making our method scalable.

Null models
degree sequence model

\mathcal{G}^d : all graphs with degree sequence d

$$P_d^{\text{DS}}(G) = \frac{1}{|\mathcal{G}^d|}$$

$$L_d^{\text{DS}}(G) = \log |\mathcal{G}^d|$$

$$L^{\text{DS}}(G) = L^{\text{prior}}(\#d) + L_{\text{seq}}^{\text{seq}}(d) + L_d^{\text{DS}}(G)$$

22/22

We've seen the Erdos-Renyi model already, but a more common distribution in motif analysis is the **degree sequence distribution**, This distribution which assign equal probability to all graphs with the same degree sequence.

If we discount the prior on the whole sequence entirely, the lowerbound becomes much too strict to detect any motifs. Instead we use a kind of three-part coding approach. We first store the frequencies of the degrees of d using an arbitrary prior. Then we store the sequence itself using a Dirichlet-multinomial model, and then store the

For a directed graph the degree sequence is a sequence of pair of integers (the in- and outdegree).

Edge list model

- Computing $|\mathcal{G}^d|$ is expensive.

$$|\mathcal{G}^d| < |\mathcal{S}^d|$$

contains some duplicates, but $|\mathcal{S}^d|$ is very easy to compute

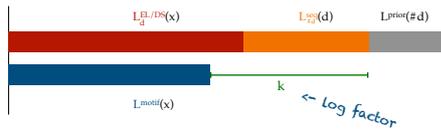
- $L_d^{\text{EL}}(G) = \log |\mathcal{S}^d|$

23/32

The size of \mathcal{G}^d is, unfortunately, difficult to compute. As an alternative, we can look at \mathcal{S}^d : the set of all lists of edges representing simple graphs with degree sequence d . This represents the same set of graphs as \mathcal{G}^d , but some will be counted double. We get the EL code by replacing \mathcal{G}^d with \mathcal{S}^d .

This means our null model becomes strictly less good, but much faster to compute. For smaller graphs, we can compute both, to see if the EL model makes an acceptable proxy for the DS model.

Summary



24/32

Outline

- MDL hypothesis testing for pattern analysis
 - preliminaries of MDL
 - NHST testing with MDL
- Network motif analysis with MDL
 - Motif code
 - Null models
- Results

25/32

Results

- ◆ Synthetic data
 - Sanity checks: are inserted motifs detected, are non-motifs left undetected?
- ◆ Medium-sized real world data
 - Is the EL model an acceptable proxy for the DS model?
- ◆ Classification
 - Are the subgraphs found *characteristic* for the data?
- ◆ Scaling up
 - What can we do on a single compute node (8 cores, 64Gb) in 1-2 days?

26/32

We perform four experiments.

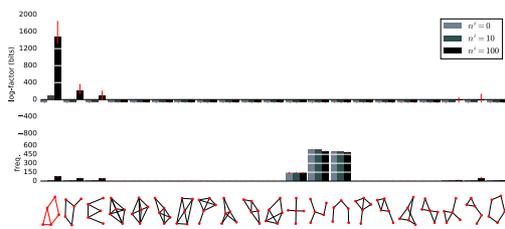
One on **synthetic data**, to see whether the basic properties we expect of a motif analysis hold.

On **real world** datasets to see whether the degree sequence model and edgelist model provide similar results.

On a **graph classification dataset** to see whether the results of our analysis are good at *characterizing* graphs.

And finally, we see how far we can **scale** our method using a single compute

Synthetic data



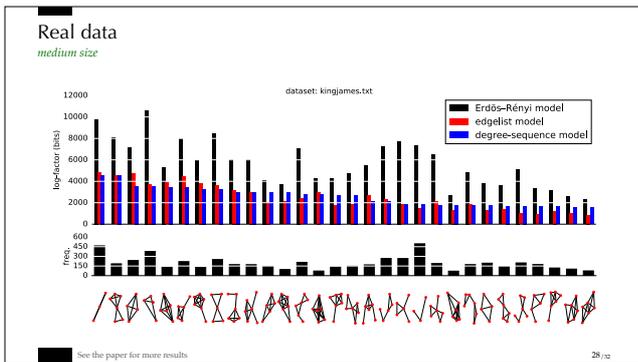
27/32

For this experiment, we sampled a medium sized graph and inserted a number of instances of a specific motif. The motif is the 'house' shape bottom left, and the number of inserted motifs was either 0, 10 or 100.

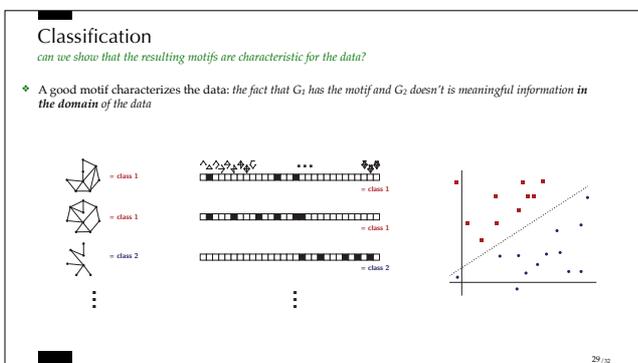
We then apply our method and compare the frequency of subgraphs found to their log-factor.

Note that:

- Many very high frequency subgraphs exist, which are not motifs.
- The motifs are extremely low-frequency.
- With just ten instances, the motif is discovered.
- Some subgraphs that overlap with the motif (containing triangles or quadrilaterals) are also returned as motifs.

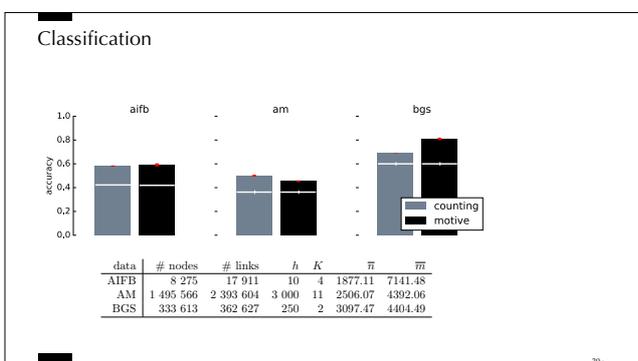


Here is the result of one of the tests on a real-world graph. We plot the log factors for three different null models. Note that the Erdos Renyi model disagrees with the others, but the Degree-sequence and Edgelist models follow a very similar trend. We see this in all four datasets tested, which suggests that for many datasets the edgelist model will be an acceptable proxy for the degree sequence model.



To test whether the motifs returned by our method, using the EL model, are useful, we perform a graph classification experiment. We take a graph classification dataset and check whether the 29 graphs of 3, 4, or 5 nodes are motifs.

This gives us a binary vector of size 29, which we use as a feature vector for the graph. If the motif judgements are good at characterizing the graphs, the resulting feature vector should allow us to solve the classification task. We apply a linear SVM and report the test accuracy.



We can mainly look to improve over the majority-class baseline, indicated by the white line. On all three datasets, our method significantly outperforms this.

Compared to the traditional method (in gray) we outperform it significantly on one dataset, underperform significantly on another, and we see no significant difference on a third. Our method is much more scalable than the traditional method, so there will be many cases where a potential drop in performance is acceptable.

Scaling up

what can we do on one compute node?

data	disk	n	m	$ \mathcal{M} $	mem.	t	preload	search	motifs
wiki-nl ^a		1 M	13 M	3-6	16 Gb	16		7m	8
				3-6	5 Gb	16		13m	8
				3-6	2 Gb	1		25m	8
				10	11 Gb	1		2h 41m	0
wiki-en ^b	✓	12 M	378 M	3-6	1 Gb	1	8m	1h 30m	8
				3-6	2 Gb	1	4h 58m	6h 6m	10
				8	8 Gb	1		6h 5m	23
twitter ^c	✓	53 M	1 963 M	3-6	6 Gb	1	17h 12m	33h 19m	0
				7	8 Gb	1		54h 26m	0
friendster ^d	✓	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

31/32

Finally we see what can be done on a single compute node.

We use a single compute node with 8 cores and 64 Gb for these experiments.

Looking at the second-to-last line, we see that if we use the full resources of the node, we can perform a full motif analysis, returning many candidates in under 9 hours. If we further limit the resources to those of an average consumer laptop, the time of analysis increases to as much as two days, but the experiment is still feasible.

Conclusions

- ◆ We moved the goal-posts a bit, but the result is a very scalable method with some additional advantages
 - Conservative hypothesis test
 - Accurate computation of low p-values (if we get a high compression)
 - No accurate search for or count of instances required. *Any* set of instances provides a valid test.
 - Comparison between multiple motif sizes
- ◆ Many subtleties to use and interpretation
 - Should we care about multiple testing?
 - Is the choice of null model important?
 - These are discussed at length in the paper. Feel free to ask in the Q&A or at vu@peterbloem.nl.

32/32

In conclusion, we present a very scaleable approach to motif analysis, and more generally, to pattern mining using null-hypothesis testing. Our approach has many other benefits, including the use of a strictly conservative hypothesis test and very accurate computation of low p-values.

There are many subtleties to the interpretation of these results. These are discussed at length in the paper, and we would, of course, be happy to answer any questions in the Q&A session.