OPEN ACCESS Freely available online

PLOS ONE

NetMODE: Network Motif Detection without Nauty

Xin Li[1]◔, Douglas S. Stones[2,3]*◔, Haidong Wang[1], Hualiang Deng[1], Xiaoguang Liu[4]*, Gang Wang[1]*

1 Nankai-Baidu Joint Laboratory, College of Information Technical Science, Nankai University, Tianjin, China, **2** Clayton School of Information Technology, Monash University, Victoria, Australia, **3** School of Mathematical Sciences, Monash University, Victoria, Australia, **4** College of Software, Nankai University, Tianjin, China

Abstract

A motif in a network is a connected graph that occurs significantly more frequently as an induced subgraph than would be expected in a similar randomized network. By virtue of being atypical, it is thought that motifs might play a more important role than arbitrary subgraphs. Recently, a flurry of advances in the study of network motifs has created demand for faster computational means for identifying motifs in increasingly larger networks. Motif detection is typically performed by enumerating subgraphs in an input network and in an ensemble of comparison networks; this poses a significant computational problem. Classifying the subgraphs encountered, for instance, is typically performed using a graph canonical labeling package, such as Nauty, and will typically be called billions of times. In this article, we describe an implementation of a network motif detection package, which we call *NetMODE*. NetMODE can only perform motif detection for k-node subgraphs when $k \leq 6$, but does so without the use of Nauty. To avoid using Nauty, NetMODE has an initial pretreatment phase, where k-node graph data is stored in memory ($k \leq 5$). For $k = 6$ we take a novel approach, which relates to the Reconstruction Conjecture for directed graphs. We find that NetMODE can perform up to around 30 times faster than its predecessors when $k \leq 5$ and up to around 20 times faster when $k = 6$ (the exact improvement varies considerably). NetMODE also (a) includes a method for generating comparison graphs uniformly at random, (b) can interface with external packages (e.g. R), and (c) can utilize multi-core architectures. NetMODE is available from netmode.sf.net.

Citation: Li X, Stones DS, Wang H, Deng H, Liu X, et al. (2012) NetMODE: Network Motif Detection without Nauty. PLoS ONE 7(12): e50093. doi:10.1371/journal.pone.0050093

Editor: Luís A. Nunes Amaral, Northwestern University, United States of America

Received July 3, 2012; **Accepted** October 18, 2012; **Published** December 18, 2012

Funding: This work was supported in part by NSFC of China (60903028, 61070014), Key Projects in the Tianjin Science & Technology Pillar Program (11ZCKFGX01100). Stones was also supported by an ARC grant. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

* E-mail: the_empty_element@yahoo.com (DSS); liuxg74@yahoo.com.cn (XL); wgzwp@163.com (GW)

◔ These authors contributed equally to this work.

Introduction

A *network motif* in a network G is a connected graph H that occurs significantly more frequently as an induced subgraph than would be expected in a "similar" random network. The term "network motif" was coined by [1,2], who discovered that they occur in several biological and artificial networks. Recently, network motifs have been found in a vast range of networks, and, in some cases, have been identified as functionally important. One prominent example is the 3-node feed-forward loop  in the E. coli transcription factor network [3]. For further reading, see [4,5]. See also Supporting Information S1 for an introduction to the graph theory concepts used in this paper.

Definition in practice

In practice, the detection of network motifs is typically implemented in the following way.

- *Step 1*: Compute the number of times that H occurs in G as an induced subgraph, call this number $f_G(H)$. We will assume that distinct copies of H may overlap (although, this is topical).
- *Step 2*: Generate an ensemble Ω of random graphs "similar" to the input network.
- *Step 3*: For each graph $R \in \Omega$, we compute the number of times H occurs in R, which we will denote $f_R(H)$.
- *Step 4*: Compute the probability p (called the *p-value*) that a similar graph $R \in \Omega$ contains the same or more copies of H than G. If the estimate for p is less than some user-defined threshold α, we declare H a network motif.

Performing these steps for all k-node connected subgraphs will be referred to as performing a k-node *subgraph census*.

In some motif detection programs, a *Z-score* is used instead of, or together with, the p-value above; it is given by $\frac{1}{\hat{\sigma}_H}(f_G(H) - \hat{\mu}_H)$ where $\hat{\mu}_H$ and $\hat{\sigma}_H$ are the sample mean and standard deviation of the dataset $\{f_R(H) : R \in \Omega\}$. However, we caution the reader that, in this context, the use of a "Z-score" does not imply a corresponding normal distribution.

Typically, a graph is described as *similar* to G if it has the same vertex set as G and if its vertices have the same in-degrees and out-degrees as in G. However, the treatment of loops and bidirectional edges can be addressed in different ways. We will describe how we address these issues in the "Modifications" section. In any case, we will refer to the similar graphs in Ω as the *comparison* graphs.

In practice, additional criteria are added, such as a minimum value for $f_G(H)$. Instead of frequency, concentration might also be considered [6], that is, the proportion of H amongst all k-node connected induced subgraphs in the network. The decision of which statistic to use can have a significant impact on whether or not H is declared a motif.

Software

Many packages have been developed that enable motif detection. Mainstream packages, those that can perform a k-node subgraph census, include Mfinder [7], MAVisto [8,9], FanMod [6,10,11], MODA [12], and Kavosh [13]. This is the family of network motif detection packages to which our program, NetMODE, belongs.

Of these, we will focus on FanMod and Kavosh, which are the most relevant: Mfinder and MAVisto are largely subsumed by FanMod; the authors of MODA have asked us to consider Kavosh instead (via private communication). We find Kavosh is typically faster than FanMod (see Table 1).

G-Tries [14–17] (see also [18]) is a data structure whose authors claim impressive speedups vs. FanMod. We do not include G-tries in the experiments in this paper since the G-tries data structure comprises only part of motif detection, although we give a theoretical analysis in the Theory section. Moreover, only recently has a "preliminary" program, called gtrieScanner, that implements the G-tries data structure become available; we make some remarks about gtrieScanner in the Conclusion.

There is also a range of other algorithms and packages that have some specialized functionality, or functionality related to motif detection, such as NeMo [19], which considers non-induced subgraphs.

There are also several papers that compare algorithms and software for network motif detection [20–26].

NetMODE

We will now introduce our network motif detection package NetMODE, **Net**work **MO**tif **DE**tection, designed to improve upon Kavosh for $k \leq 6$.

Table 1. Speedup of NetMODE vs. Kavosh.

G	v	e	k	run-time (sec.)	K-speedup	
				Kavosh	serial	4-core
1	67	182	3	0.5	0.5	1.8
Social			4	1.5	1.5	2.1
			5	8.5	3.2	2.4
			6	50.5	1.5	4.9
2	672	1276	3	3.9	1.0	7.1
E. coli			4	11.7	3.5	20.0
			5	79.6	6.2	15.7
			6	752.1	4.2	16.1
3	688	1079	3	10.5	4.4	13.2
Yeast			4	214.2	17.7	74.6
			5	5990.8	31.8	123.5
			6	119359.3	11.4	46.1
4	50	2540	3	20.7	7.6	27.0
Complete graph			4	460.7	25.5	107.6
			5	6971.7	31.7	122.1
			6	88025.3	20.4	81.0

doi:10.1371/journal.pone.0050093.t001

Modifications

NetMODE began development as a modified version of Kavosh, but over time was modified so substantially that we no longer consider it a Kavosh variant. We will now explain the differences.

Canonical labeling. In packages such as FanMod and Kavosh, the highly-optimized graph isomorphism package Nauty (available from cs.anu.edu.au/bdm/nauty/) is used to provide canonical labels for all of the k-node subgraphs encountered. Typically, they make millions, or even billions, of calls to Nauty. However, in small cases, the number of isomorphism classes of directed graphs is not overly large. Consequently, Nauty is called to perform the exact same task many times over. Moreover, often the majority of k-node subgraphs encountered fall into a handful of isomorphism classes.

NetMODE instead stores all canonical labels in memory. This scheme works easily for $3 \leq k \leq 5$, but for $k = 6$ we need to be a bit more clever, and for $k \geq 7$, this scheme seems totally impractical. Any significant run-time improvement achieved by NetMODE is the result of this preprocessing scheme.

Generating similar graphs. NetMODE has several methods for sampling similar graphs; which one to use should be determined based on the input network.

We employ several variants of a switching method similar to that described in [27]. The switching operation is quite simple: two edges (a,c) and (b,d) are randomly chosen, and are replaced by (a,d) and (b,c) provided no loops or parallel edges are introduced.

Kavosh uses a modified version of this switching process which has some unfortunate drawbacks. FanMod has three different options for the handling of bidirectional edges. In NetMODE, we modify Kavosh's switching method to allow FanMod-like functionality while still using Kavosh's edge selection method. Additional details regarding comparison graph generation methods used in Kavosh, FanMod, and NetMODE are given in Supporting Information S1.

In all of the above cases, the random graphs are sampled from a non-uniform distribution [28]. Furthermore, Ginoza and Mugler [29] discussed how the switching operations can change subgraph counts in a highly correlated manner. However, in practice, the switching method seems reasonable enough (when compared to uniform sampling). Note that different switching methods will incur different run-times.

An alternative method for sampling similar graphs has also been enabled in NetMODE. This option generates similar graphs uniformly at random, but is typically much slower than the switching methods. It implements the "local constant mode" using a variant of the Configuration Model. In some cases, it is extremely slow, but if the user finds it infeasible for a given input network, they may revert to a switching method. A related "stubs" method was implemented in Mfinder.

Subgraph enumeration. Kavosh utilizes a "revolving door" routine to iterate through combinations of vertices. (The "revolving door" routine minimizes the number of changes between each iteration, with the aim to improve computational efficiency.) NetMODE utilizes Kavosh's subgraph iteration procedure, except it does not use the revolving door algorithm; it does not seem to have any significant benefit for our purposes.

Structure

When NetMODE is run, it will first enter a *pretreatment phase*. If $3 \leq k \leq 5$, then a list of all $2^{k(k-1)}$ possible loop-free k-node directed graphs are stored in memory along with their canonical labels (obtained via a brute-force search). For $k \leq 4$, we experience negligible overhead, while for $k = 5$, we experience around 5

seconds overhead (this will differ on varying platforms). For $k=6$ this approach is infeasible, so we take an alternative approach that involves a special case of a variant of the Reconstruction Conjecture in graph theory. An introduction to the Reconstruction Conjecture is given in Supporting Information S1.

From a k-node graph G, we can generate k induced subgraphs on $k-1$ vertices, called *cards*, by deleting one vertex v (and the edges that have v as an endpoint). The multiset of cards from a given graph is called the *deck*, which we will denote $\mathrm{deck}(G)$. Note that if G and G' are isomorphic graphs, then $\mathrm{deck}(G)=\mathrm{deck}(G')$. By two independent computer searches we find that the converse is true for 6-node directed graphs barring a few exceptions, as detailed in Theorem 1.

Theorem 1 *If* G and G' *are two non-isomorphic loop-free* 6*-node directed graphs, then* $\mathrm{deck}(G)\neq\mathrm{deck}(G')$, *except when* $\{G,G'\}$ *belongs to the following list of exceptions:*
$\{56044064,56036128\},\{1133217864,1133217802\}$,
$\{1640468230,1640206094\},\{1657244500,1656986394\}$,
$\{1657252436,1656990300\},\{3311111272,3311111210\}$,
$\{3788211532,3788211470\},\{3838293818,3838289726\}$,
$\{3856371052,3856370990\}$.

Theorem 1 asserts that, with a few exceptions, loop-free 6-node directed graphs are determined (up to isomorphism) by their 5-node induced subgraphs. Thus, we instead store in memory the canonical labels for 5-node directed graphs, and use this data for canonical labeling 6-node subgraphs. In Theorem 1, graphs H are listed as their *graphID*, the number defined when the adjacency matrix is read as a binary number; it is also *canonical*, the minimum such number in the isomorphism class. The computation required to prove Theorem 1 has been performed independently by the authors Li and Stones (the code used has also been made available).

When running NetMODE for $3\leq k\leq 5$, on both the input network and the random comparison graphs, we iterate through all k-node subgraphs H, and simply increase the count corresponding to its canonical label. Running NetMODE for 6-node subgraphs is more complicated and involves two distinct stages. In Stage 1, we process the input network. We start with an empty list L, and as the algorithm proceeds:

- If H is an exception, we account for it separately (details are given in the Supporting Information S1), otherwise we continue.
- If $\mathrm{deck}(H)$ does not appear in L, we add a new entry to L (with count 1). Otherwise, $\mathrm{deck}(H)$ appears in L, so we simply increase its count. Searching through L is facilitated by a hash function; its details are described in the Supporting Information S1.

In Stage 2 we process the comparison graphs. When a 6-node subgraph H is found, we compute $\mathrm{deck}(H)$ and search for its hash value in L. If found, then we search for $\mathrm{deck}(H)$ amongst all entries with that hash value, and if $\mathrm{deck}(H)$ is found, we then increase its count. Otherwise, if its hash value or $\mathrm{deck}(H)$ is not found in L, we do nothing.

For $k=6$, note that we store, and hence return, the first graph isomorphic to H found in the input network, which is not necessarily in canonical form. Note also that, in all cases, NetMODE only returns the counts of subgraphs that are found in the input network.

Other features

Usability. The theory of network motifs is relatively new and has received some criticism, e.g. [30]. Consequently, we should be careful when making claims relating to network motifs. With this in mind, we have included a *verbose mode* in NetMODE, where we return the number of subgraphs isomorphic to H in both the input network G and the comparison graphs. This allows the user access to all of the data encountered during run-time, so that the user can perform their own independent analysis. Typically, this is a large amount of data, so verbose mode is intended for analysis via a separate program.

NetMODE uses stdin/stdout so that it can interface with well-established packages, such as R. We also provide some R code that can be run to enable interfacing with NetMODE in verbose mode (using the igraph extension). The user can therefore use R to e.g. draw motifs, check whether or not sufficiently many comparison graphs have been used, check whether the comparison graph counts are approximately normally distributed, and so on.

We also introduce a *burnin* feature, whereby the first few comparison graphs generated by the switching operation are discarded, resulting in a better ensemble of comparison graphs. The benefit of using burnin is that it essentially allows the switching process to start at a random starting point (rather than the input network); its use is standard practice in MCMC algorithms. In general, it is unclear what amount of burnin to use, but we found that 1000 discarded comparison networks seems reasonable in most cases.

Parallelism. We also design NetMODE to be capable of utilizing parallel processing in multi-core computers. We use a basic coarse-grained parallelism where an individual thread is responsible for performing subgraph census on one of the comparison graphs. Parallelism is only invoked for the comparison graphs, whereas the input graph is processed in serial. At this level, it is an embarrassingly parallel problem, in that no communication is required between individual threads once they are created. Message passing is handled by the pthread library. Each comparison graph computation is added to a queue, which is managed by a *thread manager*. The thread manager limits the number of simultaneous threads to some user-defined threshold.

Experimental Results

In this section we report the experimental results obtained from testing the various algorithms described in this paper.

Experimental setup

Unless otherwise specified, the platform used in these experiments is: AMD Phenom II X4 945 (4 cores) with 2×2 GB RAM running Red Hat Enterprise Linux 5.

The number of similar graphs generated for comparison is always 1000 (we do not endorse this number of comparison graphs be used in practice; users should decide for themselves based on the input network and value of k), and we use zero burnin. Each experiment is performed only once, since differences in run-times are negligible.

In its downloadable form, Kavosh wastes a lot of time printing to the screen and writing to files. To obtain a more reasonable comparison, all computations involving Kavosh are performed using a modified version in which printing is disabled; similarly we disable writing to the disk. The difference can be significant: e.g. 50 minutes vs. 80 seconds on the E. coli dataset, $k=5$.

Speedup

We define the *K-speedup* (resp. *F-speedup*) of NetMODE to be the number of times faster NetMODE is than Kavosh (resp. FanMod) when processing a given dataset. All variables (platform, number of comparison networks, etc.) are kept the same, except possibly

for the number of threads (which we will explicitly highlight, when relevant).

In Table 1, we list the K-speedup in a range of instances. For Table 1 we use the "fixed bidirectional edges" switching method, since it matches Kavosh's switching method. For the given input network, we use v to denote the number of nodes and e to denote the number of directed edges. We test NetMODE using four input networks: (a) a social network, (b) the metabolic pathway of E. coli, (c) the transcription network of S. cerevisiae (yeast), and (d) the complete directed graph on 50 vertices. Input networks (a), (b), and (c) were used in testing Kavosh and were packaged with the Kavosh source code. No description other than "a real social network" was given for input network (a) in [13].

The fourth input network was chosen since it has some easy-to-compute properties, e.g. in each of the 1001 networks (the input network and all 1000 comparison networks), there are exactly $\binom{50}{k}$ copies of $\vec{K}_k$. Therefore, the number of calls to Nauty by Kavosh is $1001 \cdot \binom{50}{k}$.

Table 2 gives some example run-times for FanMod and NetMODE under various switching methods. Notation: F = fixed bidirectional edges; NR = no regard; GC = global constant; LC = local constant; ULC = uniform local constant. In some instances, ULC mode is impractically slow.

Scalability

In this section we describe the results of experiments designed to test the scalability of NetMODE. For these experiments, we use the local constant switching method (LC), which is probably the most reliable of the possibilities. Other switching methods can take longer, cf. Table 2.

Size of input. Figure 1 plots the run-times of NetMODE and FanMod in a range of circumstances. In these experiments, we use "protein structure networks" (as defined in [1]), for a range of proteins, whose structures were obtained from the Protein Data Bank [31]. The protein ID numbers are: 1IEG, 1DNP, 1VR0, 1F3U, 1OLM, 1HI9, 2IDB, 2VV5, 7AHL, 1B65, 2UUB, 1V54, 1HBN, 1SUV, 1K1E, 1UW6, 1RVV, 1KP8, 1GR5, 2FUG, 1TZN (although, only 1IEG, 1DNP, 1VR0, and 1F3U were used for the $k=6$ experiments). These networks are undirected, where undirected edges are treated as bidirectional edges. In several instances, we were unable to run Kavosh for these networks, so we use FanMod for the run-time comparison. The networks are also available for download with NetMODE.

Number of cores

Figure 2 plots the run-time by the multi-core version of NetMODE as the number of utilized CPU cores varies. To perform this experiment, we switch to a platform with more cores: Red Hat Enterprise Linux 5; AMD Opteron Processor 6168 (4×12 cores); 16×4G RAM. The input network was the complete directed graph on 50 vertices.

Accuracy

Since NetMODE allows uniform local constant (ULC) random comparison graph sampling, we can inspect the accuracy of NetMODE's results. In Table 3 we list some concentration-based Z-scores returned by Mfinder in local constant mode, estimated using 10^7 samples (denoted LCs), FanMod in local constant (LC) mode, and NetMODE in both LC and ULC modes. We also list some frequency-based Z-scores returned by Mfinder in LC mode, and NetMODE in both LC and ULC modes.

Discussion

Experiment

In the experiments in this paper, the primary focus is on run-time, since this is the main advantage for the end user. Memory requirements will typically be easily met, even on modest computers.

From our experiments, we find that Kavosh is usually faster than FanMod (i.e. the speedup vs. FanMod will typically be slightly greater than a speedup vs. Kavosh). To put this into a broader context, FanMod admits a host of functionality, such as allowing colored graphs as input networks, and the use of a sampling method (rather than complete enumeration). Neither Kavosh nor NetMODE have these features. NetMODE specializes on one specific task, and is faster than FanMod at this one task. However, NetMODE also offers multi-core parallelism, a uniform method for generating comparison graphs, and the ability to access the comparison graphs' subgraph counts, which are all important features that are absent from FanMod.

Tables 1 and 2 give run-times and speedups of NetMODE in a range of circumstances. We see that NetMODE can attain substantial speedups vs. Kavosh and FanMod. We also see a noticeable drop in speedup from $k=5$ to $k=6$ as NetMODE switches to a different mode. Table 2 also compares the run-times of NetMODE and FanMod under different methods for generating comparison graphs. We see that, while some switching methods are slower in NetMODE, they are also correspondingly slower in FanMod.

Figure 1 shows that the speedup achieved by NetMODE is roughly constant with the size of the input network. Figure 2 shows a near-linear increase of speedup with the number of cores utilized. In some instances, we see speedups of over 1000, and expect that this would be increased further if we were to use more cores.

We give a comparison of Z-scores returned by FanMod, Mfinder, and NetMODE in Table 2. We see that NetMODE's results resemble that of the uniform null model. FanMod's results, which typically resemble the uniform null model, are sometimes wildly different. Specifically, the graphs with graphID 78  and 238  have surprising Z-scores. We can exclude the possibility of this discrepency being the result of a bug with NetMODE since NetMODE and Mfinder's results are comparable. This indicates a bias in FanMod's results in some cases. Further experimentation relating to FanMod's bias is given in Supporting Information S1.

Theory

Maximum theoretical speedup. If S, I, and N are the run-times of the components "generate similar graphs", "iterate through subgraphs", and "canonically label subgraphs", then NetMODE could achieve a speedup vs. Kavosh (or FanMod) of at most

$$\text{speedup} \leq \frac{S+I+N}{S+I}, \tag{1}$$

since these programs share the S and I components.

Methods for decreasing I (along with N) are already available, such as estimating the concentration of subgraphs via sampling, or searching for only a single k-node subgraph. These approaches offer a trade-off of accuracy or scope for run-time. These methods will still have a theoretical best speedup of $(S+I+N)/S$.

An alternative approach is to compute the p-values analytically, which is not easy to do with the established definition of a

Table 2. Run-times (sec.) for Kavosh, FanMod, and NetMODE under various switching methods.

	Yeast; 4-node subgraph census				
	F	**NR**	**GC**	**LC**	**ULC**
Kavosh	214.2	–	–	–	–
FanMod	–	318.0	319.0	318.0	–
NetMODE	12.1	12.6	12.0	12.3	–
NetMODE 4-core	2.9	2.9	3.0	3.0	–
	Social; 6-node subgraph census				
	F	**NR**	**GC**	**LC**	**ULC**
Kavosh	50.5	–	–	–	–
FanMod	–	464.0	139.0	147.0	–
NetMODE	33.5	87.2	31.4	34.9	34.5
NetMODE 4-core	10.3	24.1	10.1	10.8	11.6

doi:10.1371/journal.pone.0050093.t002

"similar" graph. Such an approach was use in NeMo [19], but for a different distribution of comparison graphs.

Some other programs avoid the use of Nauty by using symmetry breaking. These include the program by Grochow and Kellis, which searches for one k-node subgraph at a time, and both MODA and G-tries, which utilize a tree-like data structure to guide their subgraph searching (although in dissimilar ways). Another approach was offered by Baskerville and Paczuski [32], which uses a heuristic based on vertex invariants; it is able to distinguish all isomorphism classes of undirected graphs for $k \leq 6$ and most isomorphism classes for $k \leq 8$.

Another bound dominates when $N \to \infty$ and $S = o(N)$. In these instances, if s and t are respectively the average times taken to canonically label a given subgraph by NetMODE and Kavosh, then the speedup will be bounded above by t/s. To estimate t/s, we ran NetMODE and Kavosh on the input network $\vec{K}_{200}$ (with 0 comparison graphs), which gave speedups of

$$21.2, \qquad 31.7, \qquad 35.3, \qquad 23.2$$

for $k = 3,4,5,6$, respectively.

Remarks on G-tries. Ribeiro and Silva [15] claimed speedups of G-tries vs. their own version of FanMod of over 100 in some cases. However, there are several properties of their experimental design and results that indicate these results would not be applicable to an end user.

Since G-tries describes a data structure rather than a full program, we will discuss Program G, a hypothetical program which incorporates G-tries, as described by [15].

G-tries vs. NetMODE. In order to perform a k-node subgraph census, Program G generates random comparison graphs and iterates through all k-node connected subgraphs, thus its run-time contains the components measured by S and I. Any significant speedup of Program G vs. FanMod is therefore achieved wholly by reducing N (i.e. by avoiding the use of Nauty), but this is achieved in a much more straightforward fashion in NetMODE.

Excluding S. The experiments for benchmarking G-tries were designed to test only the components improved by G-tries. As such, the authors have excluded the time it takes to generate similar graphs (i.e. S).

To illustrate the discrepancy, in the power network tested in [15], we find that S amounts to more than 47% and 12% of the total run-time in FanMod, when performing a 3-node and 4-node subgraph census, respectively (undirected graphs, complete enumeration). Consequently, it is theoretically impossible for Program G to achieve a speedup vs. FanMod of more than around 3 or 9, respectively, in this example (which is much less than the 100+ speedups reported). This issue reduces as k increases, since S becomes relatively smaller.

Measurements. The run-times by the version of FanMod used in [15] appear to be much larger than can be achieved by an authentic version of FanMod. For example, in Table 4 we list the average run-time (sec.) per comparison graph to perform a k-node subgraph census given by [15] on the "power" network, along with that achieved by an authentic version of FanMod run on a slower computer (Windows XP, 1.66 GHz, 1 GB RAM).

Parallelism

NetMODE can utilize multi-core architecture, such as in many modern computers. Considering that multi-core hardware has been widespread for some time (particularly in bioinformatics), the inability of other programs to utilize parallel hardware is a serious limitation.

Figure 2 illustrates that the multi-core version achieves a near-linear speedup (which should be expected in such embarrassingly parallel problems). The computer used in this experiment has 48 CPU cores, so increasing the number of threads used beyond 48 did not achieve anything significant.

We also made an attempt to accelerate the subgraph searching method on a graphics processing unit (GPU). We use NVIDIA's CUDA parallel computing architecture, and testing was performed on the NVIDIA GeForce GTX 480. We tried two approaches:

1. We collect a number of graphs and distribute them to the GPU's blocks; each kernel performs a separate subgraph searching procedure. After the kernel exits we combine the results from several graphs.
2. We search a single graph in one call, requiring each block to search a subset of the vertices. When one random graph is generated, we launch a kernel. After the kernel exits we combine the results from that single graph, and move onto the

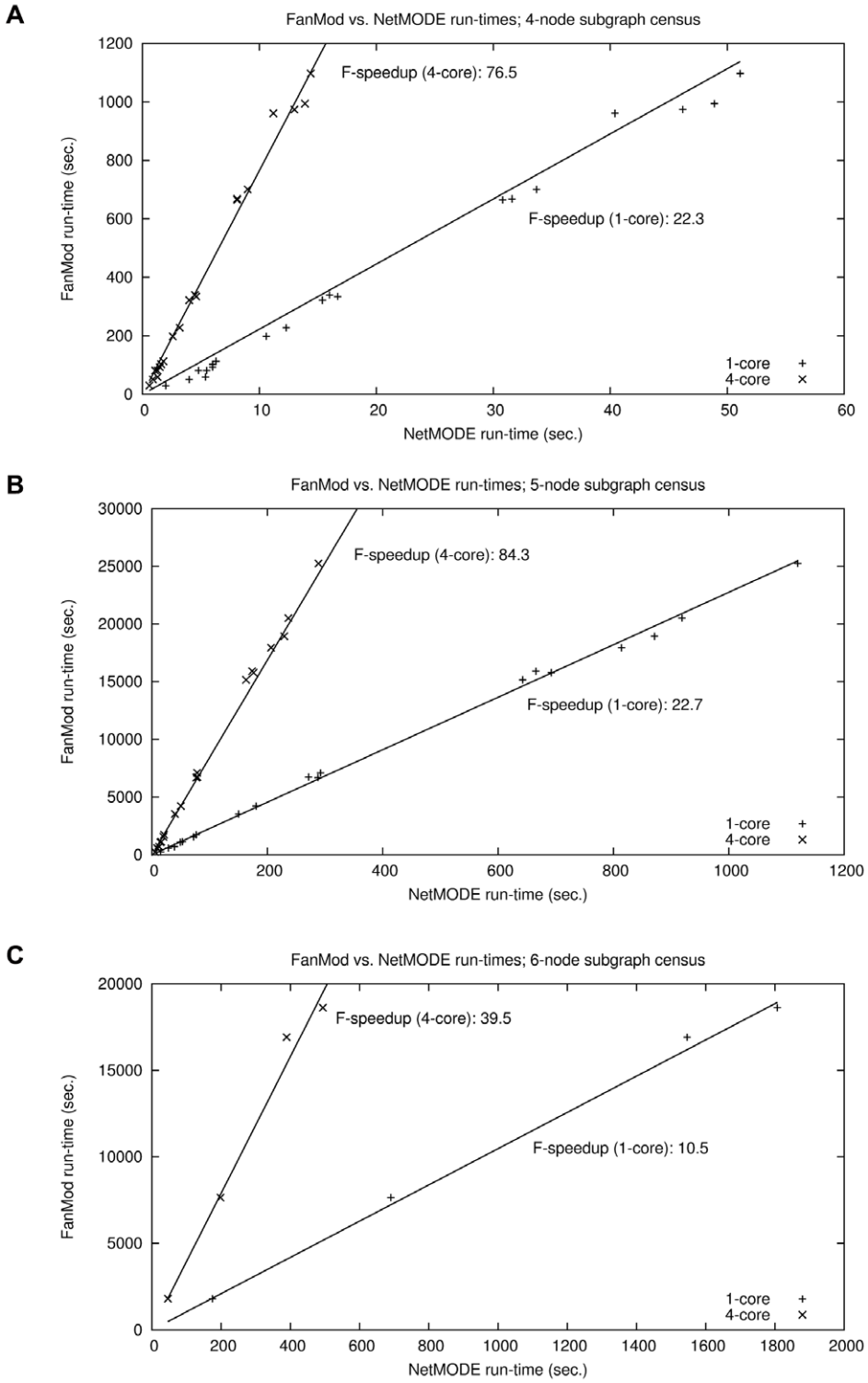

Figure 1. Run-times of NetMODE and FanMod on various protein structure networks.
doi:10.1371/journal.pone.0050093.g001

next graph. This method was also carefully optimized with the kernel stream.

Unfortunately, neither method displayed significant performance. This comes as a surprise, considering how useful the GPU is for detecting "sequence motifs" (see e.g. [33–35]). We believe this is mainly a result of the divergence within the subgraph searching procedure. The GPU is optimized for performing similar operations in different threads, so the differences between the searching procedure (caused by the randomness of the comparison graphs) together with the enormous memory require-

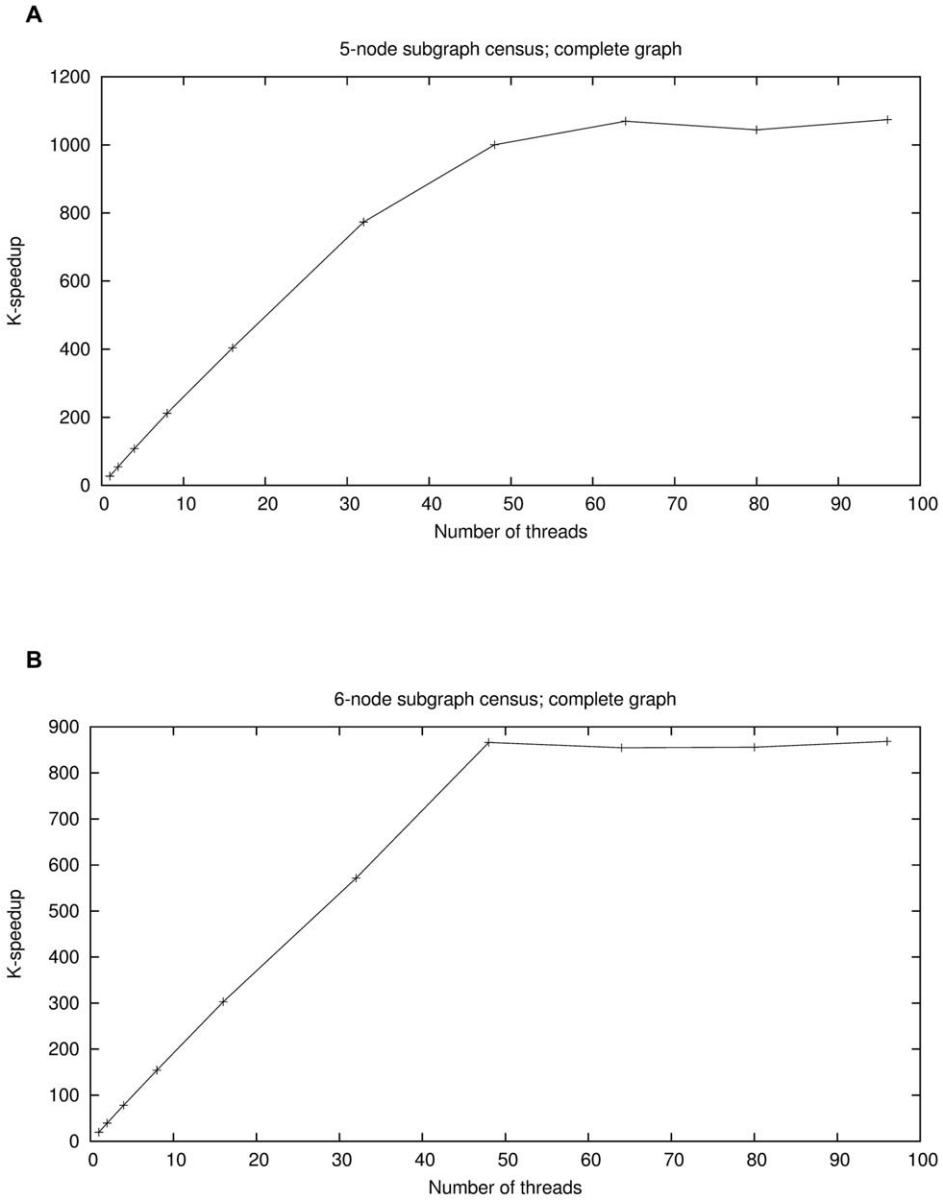

Figure 2. Speedup of NetMODE vs. Kavosh as the number of threads varies. Kavosh took 713 seconds and 134670 seconds, respectively.
doi:10.1371/journal.pone.0050093.g002

ments slow down the kernel. To help future researchers, these programs are made available for download with NetMODE.

Combinatorial explosion

There are extremely many connected digraphs on k nodes for large k. The number of non-isomorphic connected k-node digraphs is given by Sloane's A003085 and the number of non-isomorphic connected k-node undirected graphs is given by Sloane's A001349; see oeis.org. The first few values are listed in Table 5.

This property is sometimes referred to as "combinatorial explosion". If we were to run e.g. a 7-node subgraph census on a directed graph, the sheer number of non-isomorphic 7-node digraphs will give rise to many estimated p-values of 0 (the reader can readily verify this with FanMod). Unless we use extremely many comparison graphs, mixed in the list of subgraphs with an estimated p-value of 0 will be many insignificant subgraphs. In fact, this problem is still a significant concern at $k = 6$.

An alternative is to use Z-scores instead of p-values, but this was shown to be an unreliable statistic by Picard et al. [36] (and more about this will appear in [37]); we include Z-scores in NetMODE mainly for reasons of comparison. Thus, to distinguish between the subgraphs with an estimated p-value of 0, the user should repeat the experiment using a larger number of comparison graphs. Practically, this is a double blow, not only does the computational time increase rapidly with k, but also increases linearly with N (which should be chosen to increase rapidly with k).

In NetMODE, we enable the user to use more comparison graphs through parallelism (and by simply being faster than previous programs). FanMod (and Mfinder) instead offers the option of estimating subgraph concentrations via subgraph sampling (rather than complete enumeration).

Table 3. Concentration and frequency Z-scores returned by FanMod, Mfinder, and NetMODE; E. coli network, 3-node subgraph census.

	Concentration				Frequency		
	Mfin.	FanM.	NetM.	NetM.	Mfin.	NetM.	NetM.
gID	LCs	LC	LC	ULC	LC	LC	ULC
6	−3.65	−3.35	−3.49	−3.66	−11.94	−11.60	−11.86
12	−4.10	−4.25	−4.54	−4.27	−13.13	−12.60	−12.02
14	2.66	2.98	2.50	2.55	−10.18	−9.49	−9.65
36	−5.96	−6.09	−6.14	−6.34	−12.80	−12.70	−13.17
38	16.34	16.40	16.75	16.75	15.47	15.59	16.08
46	1.12	1.22	1.17	1.25	1.20	1.09	1.18
74	3.55	3.76	3.26	3.42	−8.92	−8.33	−8.61
78	−16.56	**−46.11**	−15.97	−16.24	−21.94	−21.60	−20.62
98	5.16	5.01	5.08	4.73	5.42	4.95	4.53
102	3.42	3.06	3.32	3.25	3.21	2.80	3.09
110	11.91	11.84	11.65	11.62	11.71	11.56	11.14
238	19.05	**98.99**	18.37	18.98	18.83	18.77	18.23

doi:10.1371/journal.pone.0050093.t003

Conclusion

We have implemented a network motif detection package, NetMODE, designed to improve on Kavosh and FanMod by minimizing the time taken for canonical labeling. While other packages, such as FanMod, offer a host of functionality, NetMODE's function is more specialized: it consideres only uncolored graphs, performs only complete enumeration and searches for k-node motifs for $3 \le k \le 6$. The goal for NetMODE is to perform this task faster than its predecessors, which has been achieved through a preprocessing scheme.

We have also designed NetMODE to be functional on multi-core parallel architectures. Running NetMODE on multi-core hardware should be straightforward to use even for less computer savvy users: just add e.g. "-t 2" to the command line to use two threads. For portability, we have developed NetMODE to use stdin/stdout, and allow interfacing with the popular R statistical package. Since NetMODE is substantially faster than FanMod (and can be run in parallel), it offers the ability to search for k-node motifs for $3 \le k \le 6$ (using complete enumeration) within larger input networks than FanMod would practically cope with.

One apparent technique for practically extending NetMODE's functionality to $k \ge 7$ is, in the preprocessing phase, store in memory only the canonical labels of the subgraphs that are likely to be encountered (reverting to calling Nauty whenever other subgraphs are encountered). In this case, which subgraphs to preprocess could be determined from testing, or from an auxiliary file. However, we have not explored this avenue in NetMODE since we consider the problem of combinatorial explosion to be overwhelming at this stage, and, in any case, the majority of research interest so far has been for subgraphs with 6 or fewer nodes.

Brendan McKay suggested another way to improve preprocessing times: generating and storing in memory only the k-node digraphs whose degree sequences are in "order" with respect to the vertex labels (for directed graphs, the "order" could be the lexicographic order on in-degree/out-degree pairs). In this scheme, when we encounter a k-node subgraph, we first permute its vertices so that its degree sequence is in order, then recall its canonical label from memory.

Table 4. Examples of run-time discrepency between G-tries's FanMod and the authentic version of FanMod.

k	G-tries's FanMod	authentic FanMod	articial speedup
3	0.91	0.052	≥17.6
4	3.01	0.202	≥14.9
5	12.38	1.043	≥11.9

doi:10.1371/journal.pone.0050093.t004

G-tries run-time

After the submission of this paper, gtrieScanner, a network motif detection package based on G-tries was released on Ribeiro's webpage (http://www.dcc.fc.up.pt/gtries/). This gave the present authors an opportunity to compare the two programs. Recall that both programs improve upon previous methods by minimizing the canonical labeling component, i.e. N in (1).

Ribeiro commented that the current release of gtrieScanner (version 0.1) is a "preliminary" version of a program that implements the G-tries data structure. Thus, we feel it would be misleading to give a full-blown comparison of this version of gtrieScanner vs. NetMODE. However, we make the following observations:

- For the experiments we performed, the relevant subgraph counts, Z-scores, and p-values between NetMODE and gtrieScanner were consistent.
- For performing a k-node subgraph census with $3 \le k \le 5$, the run-times of gtrieScanner and NetMODE were comparable. This suggests that both methods have improved upon previous methods by minimizing the time taken to canonically label subgraphs. Differences in e.g. preprocessing times and memory usage causes some fluctuation between the run-times of gtrieScanner and NetMODE.
- For a 6-node subgraph census, gtrieScanner performed much faster than NetMODE for undirected networks (but returned a segmentation fault for directed networks). Moreover, unlike NetMODE, gtrieScanner can also perform 7-node to 9-node subgraph census for undirected networks.
- For large input networks, gtrieScanner used a considerable amount more memory than NetMODE. We gleam from the source code, that gtrieScanner stores e.g. the input network's

Table 5. The number of k-node connected digraphs and undirected graphs, respectively.

k	A003085	A001349
1 1 1	1	1
2	2	1
3	13	2
4	199	6
5	9364	21
6	1530843	112
7	880471142	853
8	1792473955306	11117

doi:10.1371/journal.pone.0050093.t005

adjacency matrix in memory, thereby requiring $O(v^2)$ memory, where v is the number of nodes in the input network. This approach is not scalable, and gtrieScanner is virtually unusable for graphs with 10^5 or more nodes.

Note also that, unlike NetMODE, gtrieScanner is not (yet) a parallelized program.

Supporting Information

Supporting Information S1 A document giving an introduction to the concepts required to understand this paper along with an outline of how to use NetMODE.
(PDF)

Acknowledgments

The authors would like to thank Rosemary Bailey and Peter Cameron for helpful discussions regarding the uniform sampling method, and Sebastian Wernicke for a helpful discussion regarding the switching method.

Author Contributions

Conceived and designed the experiments: XL DSS HW HD XL GW. Performed the experiments: XL DSS HW HD XL GW. Analyzed the data: XL DSS HW HD XL GW. Contributed reagents/materials/analysis tools: XL DSS HW HD XL GW. Wrote the paper: XL DSS HW HD XL GW.

References

1. Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, et al. (2002) Network motifs: Simple building blocks of complex networks. Science 298: 824–827.
2. Shen-Orr S, Milo R, Mangan S, Alon U (2002) Network motifs in the transcriptional regulation network of Escherichia coli. Nature Genetics 31: 64–68.
3. Alon U (2007) Network motifs: theory and experimental approaches. Nature Reviews Genetics 8: 450–461.
4. Maslov S, Sneppen K, Alon U (2003) Handbook of Graphs and Networks: From the Genome to the Internet, Wiley-VCH, chapter Correlation profiles and motifs in complex networks. pp. 168–198.
5. Schwöbbermeyer H (2008) Network motifs. In: Analysis of Biological Networks. Hoboken, NJ: Wiley. pp. 85–111.
6. Wernicke S, Rasche F (2006) FANMOD: a tool for fast network motif detection. Bioinformatics 22: 1152–1153.
7. Kashtan N, Itzkovitz S, Milo R, Alon U (2004) Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. Bioinformatics 20: 1746–1758.
8. Schreiber F, Schwöbbermeyer H (2004) Towards motif detection in networks: frequency concepts and exible search. In: Merelli E, et al., editors. Proc International Workshop on Network Tools and Applications in Biology (NETTAB'04). pp. 91–102. Available: http://www2.ipk-gatersleben.de/~schreibe/publications/SchreiberS04.pdf. Accessed 2012 Nov 21.
9. Schreiber F, Schwöbbermeyer H (2005) MAVisto: a tool for the exploration of network motifs. Bioinformatics 21: 3572–3574.
10. Wernicke S (2005) A faster algorithm for detecting network motifs. In: Casadio R, Myers G, editors. Proc Of the 5th International Workshop on Algorithms in Bioinformatics. Lecture Notes in Computer Science, volume 3692. Berlin: Springer. pp. 165–177.
11. Wernicke S (2006) Efficient detection of network motifs. IEEE/ACM Transactions on Computational Biology and Bioinformatics 3(4): 347–359.
12. Omidi S, Schreiber F, Masoudi-Nejad A (2009) MODA: An efficient algorithm for network motif discovery in biological networks. Genes Genet Syst 84: 385–395.
13. Kashani ZRM, Ahrabian H, Elahi E, Nowzari-Dalini A, Ansari ES, et al. (2009) Kavosh: a new algorithm for finding network motifs. BMC Bioinformatics 10.
14. Ribeiro P, Silva F (2010) Efficient subgraph frequency estimation with g-tries. In: Proceedings of the 10th International Conference on Algorithms in Bioinformatics (WABI '10). Berlin: Springer-Verlag. pp. 238–249.
15. Ribeiro P, Silva F (2010) G-tries: an efficient data structure for discovering network motifs. In: Proc. 25th ACM Symposium On Applied Computing – Bioinformatics Track. New York: ACM. pp. 1559–1566.
16. Ribeiro P, Silva F, Lopes L (2010) Efficient parallel subgraph counting using g-tries. In: IEEE Cluster. pp. 217–226. Available: http://doi.ieeecomputersociety.org/10.1109/CLUSTER.2010.27. Accessed 2012 Nov 21.
17. Ribeiro P, Silva F, Lopes L (2010) Parallel calculation of subgraph census in biological networks. In: Proc 1st International Conference on Bioinformatics. pp. 56–65. Available: http://dblp.dagstuhl.de/db/conf/biostec/bioinformatics2010.html. Accessed 2012 Nov 21.
18. Ribeiro P, Silva F, Lopes L (2011) A parallel algorithm for counting subgraphs in complex networks. In: Fred A, Filipe J, Gamboa H, editors, Biomedical Engineering Systems and Technologies, volume 127. Berlin: Springer. pp. 380–393.
19. Koskas M, Grasseau G, Birmelé E, Schbath S, Robin S (2011) NeMo: Fast count of network motifs. In: Book of Abstracts for Journées Ouvertes Biologie Informatique Mathématiques (JOBIM) 2011. pp. 53–60.
20. Bruno F, Palopoli L, Rombo SE (2010) New trends in graph mining: Structural and node-colored network motifs. International Journal of Knowledge Discovery in Bioinformatics 1: 81–99.
21. Ciriello G, Guerra C (2008) A review on models and algorithms for motif discovery in proteinprotein interaction networks. Brief Funct Genomic Proteomic 7: 147–156.
22. Hu JL, Gao L, Qin GM (2009) Evaluation of subgraph searching algorithms for detecting network motifs in biological networks. Front Comput Sci China 3: 412–416.
23. Qin GM, Gao L, Hu JL (2009) A review on algorithms for network motif discovery in biological networks. Acta Electronica Sinica 37: 2258–2265.
24. Ribeiro P, Silva F, Kaiser M (2009) Strategies for network motifs discovery. In: Proc E-SCIENCE '09. pp. 80–87.
25. Wan R, Mamitsuka H (2009) Biological Data Mining in Protein Interaction Networks, IGI Global, chapter Discovering Network Motifs in Protein Interaction Networks. pp. 117–143. doi:10.4018/978-1-60566-398-2.ch008.
26. Wong E, Baur B, Quader S, Huang CH (2011) Biological network motif detection: principles and practice. Brief Bioinform. doi:10.1093/bib/bbr033.
27. Milo R, Kashtan N, Itzkovitz S, Newman MEJ, Alon U (2003) On the uniform generation of random graphs with prescribed degree sequences. Available: http://arxiv.org/pdf/condmat/0312028.pdf. Accessed 2012 Nov 21.
28. Artzy-Randrup Y, Stone L (2005) Generating uniformly distributed random networks. Phys Rev E 72: 056708.
29. Ginoza R, Mugler A (2010) Network motifs come in sets: Correlations in the randomization process. Phys Rev E : 011921.
30. Konagurthu AS, Lesk AM (2008) On the origin of distribution patterns of motifs in biological networks. BMC Systems Biology 2.
31. Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, et al. (2000) The protein data bank. Nucleic Acids Res 28: 235–242.
32. Baskerville K, Paczuski M (2006) Subgraph ensembles and motif discovery using an alternative heuristic for graph isomorphism. Phys Rev E 74: 051903.
33. Chen C, Schmidt B, Weiguo L, Müller-Wittig W (2008) GPU-MEME: Using graphics hardware to accelerate motif finding in DNA sequences. In: Chetty M, Ngom A, Ahmad S, editors. Pattern Recognition in Bioinformatics, Third IAPR International Conference Proceedings (PRIB 2008). Lecture Notes in Computer Science, volume 5265. Berlin: Springer. pp. 448–459.
34. Liu Y, Schmidt B, Liu W, Maskell DL (2010) CUDAMEME: Accelerating motif discovery in biological sequences using CUDA-enabled graphics processing units. Pattern Recognition Letters 31: 2170–2177.
35. Yu L, Xu Y A parallel Gibbs sampling algorithm for motif finding on GPU. In: 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications. New York: IEEE. pp. 555–558.
36. Picard F, Daudin JJ, Koskas M, Schbath S, Robin S (2008) Assessing the exceptionality of network motifs. J Comput Biol 3: 347–359.
37. Brand M, Stones DS (2012) The trouble with network motifs: an analytical perspective. In preparation.