Efficient Subgraph Frequency Estimation with G-Tries

Pedro Ribeiro and Fernando Silva

CRACS & INESC-Porto LA
Faculdade de Ciências, Universidade do Porto, Portugal
{pribeiro,fds}@dcc.fc.up.pt

Abstract. Many biological networks contain recurring overrepresented elements, called network motifs. Finding these substructures is a computationally hard task related to graph isomorphism. G-Tries are an efficient data structure, based on multiway trees, capable of efficiently identifying common substructures in a set of subgraphs. They are highly successful in constraining the search space when finding the occurrences of those subgraphs in a larger original graph. This leads to speedups up to 100 times faster than previous methods that aim for exact and complete results. In this paper we present a new efficient sampling algorithm for subgraph frequency estimation based on g-tries. It is able to uniformly traverse a fraction of the search space, providing an accurate unbiased estimation of subgraph frequencies. Our results show that in the same amount of time our algorithm achieves better precision than previous methods, as it is able to sustain higher sampling speeds.

Keywords: complex networks, network motifs, subgraph frequency, sampling, g-tries.

1 Introduction

A wide variety of real-life systems can be modeled and analyzed with complex networks [4]. It has been found that many of these networks contain recurring elements, called network motifs [15]. These are overrepresented subnetworks, i.e., subgraphs that appear in higher frequency than it would be expected in randomized networks with similar topological characteristics.

Network motif analysis has a broad multidisciplinary applicability. Just to name a few domains, it has been applied on biological systems (like in brain networks [20], protein-protein interactions [1] or gene regulation [5]), social networks [9]), engineering systems like electronic circuits [8] and even on software architecture [21]. Discovering these motifs is a computationally hard task closely related to the graph isomorphism problem. Currently, this is done by computing the frequency of subgraph classes of a determined size both in the original network and in a randomized ensemble of networks sharing similar topological features, namely the degree sequence.

Discovering subgraph frequencies is the main bottleneck of the whole computation, with an explosive combinatorial effect as the subgraph size increases.

V. Moulton and M. Singh (Eds.): WABI 2010, LNBI 6293, pp. 238–249, 2010.

This is typically tackled using one of two approaches: either we compute the frequency of each possible individual subgraph class, one at the time (*subgraph-centric* approach) [7], or we enumerate all subgraphs and then we compute which ones are isomorphic (*network-centric* approach) [15,23].

Recently we have proposed a new specialized data-structure, g-tries [17]. It takes advantage of common subgraph substructures in order to avoid redundant computations, matching an entire set of the subgraph classes at the same time in a given network. This leads to significant performance gains when compared to previous methods, up to one hundred times faster for some networks.

In the network-centric approach, approximation techniques have been developed in order to improve execution time at the cost of reducing the accuracy [11,23,16]. This is done by sampling a fraction of the subgraph occurrences, instead of exhaustively enumerating all of them.

Our main contribution is an efficient heuristic sampling algorithm for discovering network motifs using g-tries. We take the already existing g-trie exhaustive and complete algorithm and extend it in order to obtain an unbiased sample that can be used to estimate the desired subgraph frequencies. This leads to a new algorithm that, by taking advantage of g-tries, achieves higher sampling rates and thus is able to reach more accurate predictions than previous algorithms for the same computing time. To substantiate this claim, we empirically evaluate the sampling speed, accuracy and total execution time of the algorithm in a set of representative networks. Our results show that in the same amount of time our algorithm can potentially reach higher subgraph and graph sizes. It can also only sample subgraphs from a predefined set.

The remainder of this paper is organized as follows. Section 2 establishes a network terminology and gives an overview of related work. Section 3 overviews the used g-trie data structure and details our sampling algorithm. Section 4 discusses the obtained results on a set of representative networks. Section 5 concludes the paper, with comments on the results and possible future work.

2 Preliminaries

To ensure a coherent network terminology, we briefly review the main concepts and notation that will be used throughout the paper, and discuss related work.

2.1 Graph Terminology

A *graph* G is composed by the set of vertices $V(G)$ and the set of edges $E(G)$. The *size* of a graph is $|V(G)|$, the number of vertices. A k-graph has size k. An edge is a pair $(a, b) : a, b \in V(G)$. If the graph is *directed* the order of the pair expresses direction, while in *undirected* graphs there is no direction in edges. The *neighborhood* of a vertex u is defined as $N(u) = \{v : (v, u) \vee (u, v) \in E(G)\}$. All vertices are assigned consecutive integer numbers starting from 0, and the comparison $v < u$ means that the index of v is lower than that of u. The adjacency matrix of a graph G is denoted as G_{Adj}, and $G_{Adj}[a][b]$ represents a possible edge between vertices with index a and b.

A *k-subgraph* G_k of a graph G is a k-graph such that $V(G_k) \subseteq V(G)$ and $E(G_k) \subseteq E(G)$. This subgraph is said to be *induced* if $u, v \in V(G_k)$ and $(u, v) \in E(G)$ implies $(u, v) \in E(G_k)$. Two graphs G and H are said to be *isomorphic* ($G \sim H$) if there is a one-to-one mapping between the vertices of both graphs where two vertices of G share an edge if and only if their corresponding vertices in H also share an edge.

2.2 Network Motifs and Frequency Count

In network motif discovery, frequency count is the central subproblem being addressed, and thus, we define it more precisely:

Definition 1 (Subgraph Counting Problem). *Given a set of subgraphs S_G and a graph G, count the number of all induced occurrences of subgraphs of S_G in G. Two occurrences are considered different if they have at least one node or edge that they do not share. Other nodes and edges can overlap.*

Note especially that we only count induced occurrences and how we distinguish occurrences. Although other frequency concepts exist [19], we resort to the standard definition for the network motif discovery problem [18]. It has direct implications on the number of occurrences and on the tractability of the problem, with no downward closure property [12] on the frequencies, i.e., a subgraph may appear more times than a subgraph contained in it.

2.3 Related Work

A general and informal survey on algorithms for network motifs discovery can be seen in [3], and [18] provides a more technical comparison of the algorithms. The overall most efficient exhaustive network-centric algorithms are ESU [23] and Kavosh [10]. MODA [16] and Grochow and Kellis [7] provide efficient subgraph-centric algorithms and we provided the g-trie data-structure for an efficient intermediate appproach [17].

Regarding heuristic approximate algorithms, there are three different approaches that we are aware of. Kashtan et al [11] propose to sample one subgraph at a time, following a random graph walk, which results in a biased estimator. RAND-ESU [23] algorithm provides unbiased sampling by associating probabilities with each recursive search tree branch of the ESU algorithm. MODA [16] chooses nodes with a probability proportional to their degree. Our sampling algorithm differs from all previous approaches since we use a different underlying data structure and its associated methodology.

3 Sampling Algorithm

3.1 G-Tries Data Structure

A g-trie is a data structure designed to store a set of graphs. It is conceptually inspired in prefix trees (trie) in the sense that it tries to identify common graph substructures in the same way a trie identifies common prefixes of sequences.

A g-trie is a tree where each tree node contains information about a single graph vertex and its correspondent connection to the vertices of ancestor tree nodes. Every node can have an arbitrary number of children and the path from the root to a node (possible a leaf) defines a single subgraph. Note that all descendants of a node share the same initial g-trie substructure and therefore have a common subtopology in graph terms. Figure 1 gives an example of a g-trie with 6 undirected subgraphs.

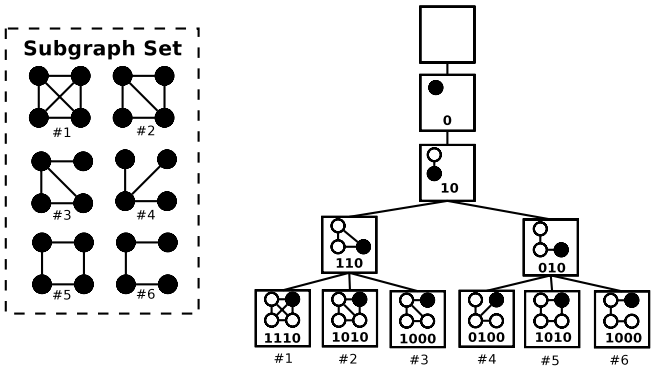

Fig. 1. A g-trie representing a set of 6 undirected subgraphs. Each g-trie node adds a new vertex (in black) to the already existing vertices in the ancestor nodes (in white). The connections to these nodes are represented by a sequence of boolean numbers indicating the corresponding adjacency matrix row.

As said, each g-trie node needs to specify the connections of is vertex to all ancestor ones (and to itself). This can be done in several ways, but in our current implementation we just store the correspondent part of the adjacency matrix. If the graphs are undirected, we store in each node the adjacency matrix row up to that vertex. If the graphs are directed we also store the adjacency matrix column up to that vertex, because we must specify ingoing and outgoing connections. In any case, given a path from the root to a node, we have a fully specified graph. The g-trie root node is empty since there are two possible direct child nodes: a vertex with or without a connection to itself.

Considering that we want an unique and univocal representation of a set of graphs, we use a canonical adjacency matrix. This guarantees that any subgraph will always lead to the same path traversing the tree. There are many possible choices here, and we opted for the lexicographically bigger adjacency matrix. This favors the occurrence of more common substructures with higher degree nodes appearing in lower tree depth levels.

This capability of identifying common subtopologies is the main strength of a g-trie. We are compressing information and avoiding redundant storage. But more than that, at a later stage, when using the g-trie to search for sugraphs and when matching a specific vertex in the g-trie, we are matching at the same time all possible descendant subgraphs stored in the g-trie.

In order to avoid subgraph symmetries, g-tries also store symmetry breaking conditions of the form $a < b$ indicating that the vertex in position a should have

a graph index smaller than vertex in position b. Similar to what was done in [7], these conditions establish an order for the vertices of the same symmetry group and guarantee that each subgraph can be found only once. More details on this can be seen in our previous work [17].

For the sake of clarity, from now on we will use the term node to refer to a g-trie tree node, and vertex to refer to a vertex of the stored graphs. Given a g-trie node T, we will use $T.vertex$ to refer the new vertex of that node (represented in black in Figure 1), and $T.in[i]$ and $T.out[i]$ to refer to the boolean value of the new vertex having respectively an ingoing or outgoing connection to the vertex with index i, i.e., the new node represented in the ancestor of depth i. Note that if the g-trie stores undirected graphs, then $T.in[i] = T.out[i]$ (and in fact T.out is not even stored in memory). We will also use $T.cond$ to denote the set of conditions that break symmetries for the descendant nodes that correspond to a full graph. $T.root$ denotes the g-trie root node and $T.isGraph$ indicates if the node is the end vertex of a graph (in fig. 1 this corresponds to all leaf nodes).

3.2 Exact Subgraph Frequency

Given a g-trie T and a graph G, the g-trie matching algorithm will find the occurrences of all graphs of T as subgraphs of G, as shown in [17]. The basic idea is to find a set of vertices of G that match completely with a path in T, and we heavily constraint our search by using the information stored about connections and symmetry breaking conditions. For the sake of clarity, we show the matching algorithm, in Algorithm 1, with a subtle modification. It encapsulates some of the work in the `matchingVertices()` function, thus allowing for a logical separation of the recursion calls and the isomorphic matching.

At any stage, V_{used} represents the currently partial match of graph vertices to a g-trie path. We start with the g-trie root children nodes and call the recursive procedure `match()` with an initial empty matched set (line 2). The later procedure starts by creating a set of vertices that completely match the current g-trie node (line 4). We then traverse that set (line 5) and recursively try to expand it through all possible tree paths (lines 7 and 8). If the node corresponds to a full subgraph, then we have found an occurrence of that subgraph (line 6). Note that at this time no isomorphic test is needed, since this was implicitly done as we were matching the vertices.

Generating the set of matching vertices is done in the `matchingVertices()` procedure. The efficiency of the algorithm heavily depends on the above mentioned constraints as they help in reducing the search space. To generate the matching set, we start by creating a set of candidates (V_{cand}). If we are at a root child, then all graph vertices are viable candidates (line 10). If not, we select from the already matched vertices that are connected to the new vertex (line 12), the one with the smallest neighborhood (line 13), reducing the possible candidates (line 14). Then, we traverse the set of candidates (line 16) and if one respects all connections to ancestors (lines 17 to 19), and respects at least one set of symmetry breaking conditions for a possible descendant subgraph (line 19), we add it to the set of matching vertices (line 20).

Algorithm 1. Finding subgraphs of g-trie T in graph G.

```
1: procedure matchAll(T, G)
2:     for all children c of T.root do match(c, ∅)
3: procedure match(T, V_used)
4:     V = matchingVertices(T, V_used)
5:     for all vertex v of V do
6:         if T.isGraph then foundMatch()
7:         for all children c of T do
8:             match(c, V_used ∪ {v})
9: function matchingVertices(T, V_used)
10:    if V_used = ∅ then V_cand := V(G)
11:    else
12:        V_conn = {v : v = V_used[i], T.in[i] ∨ T.out[i], i ∈ [1..|V_used|]}
13:        m := m ∈ V_conn : ∀v∈ V_conn, |N(m)| ≤ |N(v)|
14:        V_cand := {v ∈ N(m) : v ∉ V_used}
15:    Vertices = ∅
16:    for all v ∈ V_cand do
17:        if ∀i∈[1..|V_used|]:
18:            T.in[i] = G_Adj[V_used[i]][v]  ∧ T.out[i] = G_Adj[v][V_used[i]] then
19:            if ∃C ∈ T.cond : V_used + v respects C then
20:                Vertices = Vertices ∪ {v}
21:    return Vertices
```

3.3 Uniform Sampling

Algorithm 1 creates an exhaustive and complete enumeration of all subgraph occurrences. Our contribution to the existing g-tries methods is to sample only a fraction of all the occurrences. Similarly to what was done in [23], we will be trading accuracy for execution speed. The main idea is that each search branch is only chosen with a certain probability as depicted in Algorithm 2. Note that it is exactly the same as the previous algorithm with the exception of the indicated lines 3 and 9.

Algorithm 2. Sample subgraphs of g-trie T in graph G. Probability of each occurence is P, with $P = \prod P_d$, where P_d is probability of depth d.

```
1: procedure sampleAll(T, G)
2:     for all children c of T.root do
3:         With probability P_0 do sample(c, ∅)                    ▷ changed line
4: procedure sample(T, V_used)
5:     V = matchingVertices(T, V_used)
6:     for all node v of V do
7:         if T.isGraph then foundMatch()
8:         for all children c of T do
9:             With probability P_{T.depth} do sample(c, V_used ∪ {v})   ▷ changed line
```

In order to follow a probabilistic approach, the algorithm uses a set of probabilities associated to each g-trie depth:, $\{P_0, P_1, \ldots, P_{gtrie\_max\_depth}\}$ where $0 \leq P_i \leq 1$. Any given node of depth d will therefore only be reached with probability $P_0 \times \ldots \times P_{d-1}$. With this, we can produce an unbiased estimator of the frequency count of a single subgraph. Let P_i be the probability associated with depth i and $F_{sample}(G_k)$ be the number of occurrences of the k-subgraph G_k found in G by the `sampleAll()` procedure of Algorithm 2. Then, an unbiased estimator $\hat{F}(G_k, G)$ of the total number of occurrences of G_k in G is given by the following equation:

$$\hat{F}(G_k, G) = \frac{F_{sample}(G_k, G)}{P_0 \times P_1 \times \ldots \times P_{k-1}} \tag{1}$$

We say that the estimator is unbiased because any occurrence of G_k can be found with equal probability, and as we increase the probabilities, the estimator gets closer to the real value. In fact, if we choose $P_i = 1$ for all i, then the result is the same as the original complete algorithm.

As seen, the parameters P_i control the search. Regarding the accuracy, we should avoid small values of probability for lower depths, closer to the root. Its effect is to increase the variance of the result because any disregarded branch in lower depths may correspond to entire parts of the graph, and therefore may correspond to a higher number of subgraph occurrences not found. As to the execution time, the opposite happens. Very high probabilities in the lower depths will increase the execution time, since more parts of the search tree will have to be computed. For example, in the extreme case of having all probabilities equal to one except the last one, in the higher possible depth d, means that in practice we will explore all possible subgraphs of depth $d - 1$.

Picking the parameters is therefore a delicate choice that will influence both the accuracy and speed of our method. Section 4 gives more details on actual useful real parameters. Note that if only k-subgraphs are being sought, than all complete subgraphs of the tree will correspond to leaf nodes and therefore the probability at depth k should always be 1 since when we are at that point, all computation needed to identify the occurrence is already made (no isomorphism test is needed after that), and choosing any value smaller than 1 would only decrease the number of samples without any gain in execution time.

The main benefit of our sampling algorithm regarding previous ones, is that it is able to sample only the desired set of subgraphs (mfinder and ESU can only sample the entire set of possible k-subgraphs and MODA can only sample the occurrences of a particular single subgraph). To our best knowledge, this is the first algorithm doing that.

The quality of the estimation depends on many factors. A fully fledged analytical determination of tight bounds on error margins is very complicated since we do not know beforehand the distribution of the subgraphs that we are looking for. For example, if the subgraph is very well spread in the entire subgraph, we will have less variance than if all occurrences are clustered in a small number of nodes, where a search branch not followed can imply a significant number of occurrences not found.

3.4 Network Motif Discovery

With the algorithms previously defined we can discover all network k-motifs in the following way: first we find all k-subgraphs that occur in the original graph using another algorithm (for example `ESU`). Then we build a g-trie with those k-subgraphs and only search that particular set in the similar ensemble of randomized networks. Eventually, if we have other conditions, like a minimum frequency in the original graph, we can already discard some subgraphs and take advantage of the fact that we can search only for the ones that interest us.

4 Results

In order to evaluate the performance of our proposed algorithm (which from now on we will call `RAND-GTRIE`) we implemented it using C++. Isomorphisms and canonical labellings were computed using the `nauty` tool [14]. All tests were made on a computer with an Intel Core 2 6600 (2.4GHz) with 2GB of memory. We used four different biological networks from different domains, with varied topological features that are summarized in Table 1.

Table 1. Networks used for experimental testing of RAND-GTRIE

Network	Nodes	Edges	Directed	Description	Source
`Social`	62	159	no	Social network of a community of dolphins	[13]
`Neural`	297	2345	yes	Neural network of *C. elegans*	[22]
`Metabolic`	453	2025	yes	Metabolic network of *C. elegans*	[6]
`Protein`	2361	6646	no	Protein-protein inter. of *S. cerevisiae*	[2]

In all tests the construction of the g-trie in itself was a very small fraction of the execution time, and we could even store and reuse canonical labellings on other program runs. On the network discovery problem, the g-trie can be computed once, at the beginning, and then reused for the ensemble random networks. Given this, we chose to leave the g-trie creation out of the picture when stating execution time. For the purposes of this section, we also limited the choice of probability parameters to three levels of quality. In order to sample a fraction f of all k-subgraph occurrences, we can opt for one of the following levels:

- `high`: $\{P_0 = 1, \ldots, P_{k-3} = 1, P_{k-2} = f, P_{k-1} = 1\}$
- `medium`: $\{P_0 = 1, \ldots, P_{k-4} = 1, P_{k-3} = \sqrt{f}, P_{k-2} = \sqrt{f}, P_{k-1} = 1\}$
- `low`: $\{P_0 = \sqrt[k-1]{f}, \ldots, P_{k-2} = \sqrt[k-1]{f}, P_{k-1} = 1\}$

Our first test was to analyze the speed at which `RAND-GTRIE` is able to generate samples. For that we counted how many k-subgraphs per second it was able to generate, both with a complete enumeration (all $P_i = 1$) and with only 10% of the k-subgraphs obtained by sampling with `high` quality level. We also compared the performance with RAND-ESU, the present most efficient network centric method that also allows sampling in a way similar to ours. For that, the publicly

available FanMod tool was used, with the same probabilities at the same depths. FanMod is also implemented in C++ and uses nauty for isomorphism. All sizes between 3 (the minimum acceptable for a subgraph to be taken in account) and 6 (the maximum that guarantees computation in a matter of a few hours) were used. We first used ESU to discover all the k-subgraphs in the original graph, constructed a g-trie with those and then used it to estimate the frequency (as we would do with the randomized networks if we were discovering motifs). Figure 2 details the results obtained.

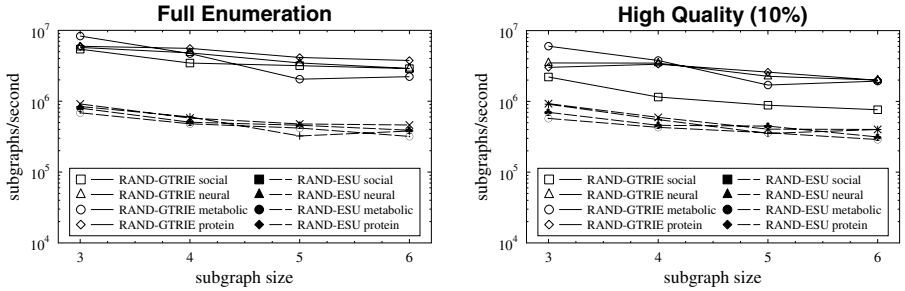

Fig. 2. Sampling speed of RAND-GTRIE and RAND-ESU

The main aspect to note is that RAND-GTRIE is always faster, being an order of magnitude faster. This was also the case for all other networks tested, with the more extreme speedup bigger than 100×, for a power grid network [22]. RAND-GTRIE also appears to scale well with an increasing subgraph size, as is the case with RAND-ESU, since the sampling rate is sustained. Mfinder, the other major alternative for sampling, was shown to be much slower than RAND-ESU and it does not scale well [23].

In order to test the accuracy of our algorithm, we applied all levels of sampling quality, while increasing the fraction of subgraphs being sampled, taking note of the percentage of subgraphs correctly identified. We considered an estimate to be accurate when it was within a 20% error margin of the correct perfect value. We took 100 samples for each fraction and level and only considered the estimate correct when at least 80 of those samples were accurate. The results for two of the networks are shown on fig. 3. As expected, higher probabilities in lower depths correspond to better sampling quality (less variance).

If we measure the execution time for the exact same tests, we can see that the opposite happens, with better quality sampling taking more execution time as detailed in fig. 4. All quality levels have an execution growth proportional to the percentage of samples, but higher quality levels have a minimum time bigger than lower quality minimum time. For example, on the protein network, sampling just 0.1% of the subgraphs in high level of quality takes more then 6% of the time it takes to do a full enumeration. This is because we are traversing the entire tree up to depth $k-2$. Judging by our empirical tests, 10% on high level exhibits a good balance between execution time and sampling quality, but depending on the situation, any other values can be used.

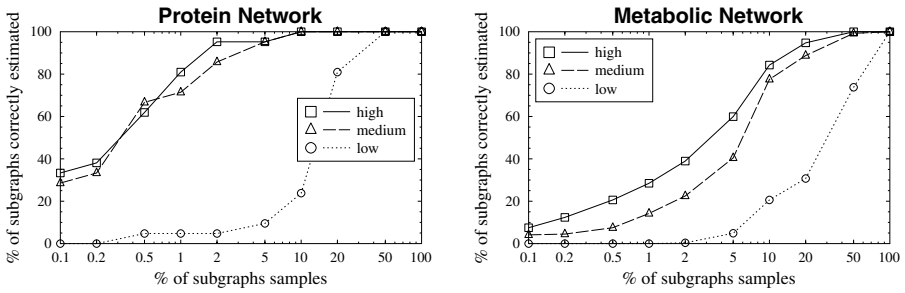

Fig. 3. Accuracy of `RAND-GTRIE` for 5-subgraphs, measured in percentage of correctly estimated subgraphs as the percentage of samples grows

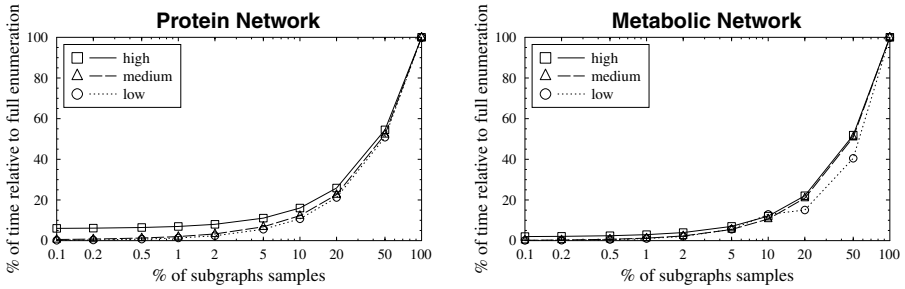

Fig. 4. Execution time of `RAND-GTRIE` for 5-subgraphs, relative to the time a full enumeration with g-tries takes (i.e., with $P_i = 1$ for every i)

If we take a closer look to what `RAND-GTRIE` is computing, we can see that the more rare subgraphs are the ones with less estimating quality. This is because a smaller number of occurrences will obviously imply more variance in the estimated values (a "miss" has more weight). For example, with `high` level setting and 10% of samples on the `metabolic` network we have 84.27% subgraph classes estimated correctly. Almost all of the ones not identified appear less than 100 times in the sample, and therefore are estimated to appear less than 1000 times in the original network. On the other hand, with the same `high` level setting and only 0.1% of samples, the 7.49% that were estimated correctly correspond to subgraph classes that were sampled at least 1000 times, which means that they are estimated to occur more than one million times in the original graph.

Finally, regarding motifs, we experimented to discover all motifs of sizes 3 to 6 in the four networks, using 10% sampling with `high` quality level, and we were able to find more than 90% of the motifs that a full enumeration would find. More than that, we spend on average less than 20% of the time it would take using g-tries full enumeration. If we take into account that g-tries are themselves a more efficient data structure than previous methods, we can magnify even more the speedup and potentially reach previously unfeasible network and subgraph sizes. Note that since we can choose the subgraphs that we are looking for, we can even experiment with different probability parameters for different subgraphs, thus paving the way for a more adaptive algorithm.

5 Conclusion

In this paper we presented a novel sampling algorithm for discovering network motifs. It employs as a basis the g-trie data structure, an efficient specialized tree that uses common topologies in subgraphs in order to heavily constraint the search. By associating a probability with each tree depth, it is able to uniformly traverse a fraction of the whole search space. With this it provides an unbiased estimator for the real frequency of the associated subgraphs, and a way of efficiently discovering motifs.

Our algorithm offers many parametrization choices and it is also capable of sampling subgraphs solely from a predefined set, in opposition to having to sample among all of the subgraphs of a determined size, or only sampling one individual subgraph. This has a direct beneficial impact on the execution time and we are able to produce accurate results spending less execution time than previously existent methods.

In the future we intend to exploit even more this property and create an adaptive version of our sampling algorithm that is able to make an initial estimation and then keep refining it for the subgraphs that do not have enough estimation quality. For example, one could remove all frequent subgraphs from the g-trie and only repeat the search for the more rare ones, with an higher fraction of samples. We also intend to study the impact of the original graph labeling on the sampling quality, since our symmetry breaking conditions rely on this order.

Finally, we will also apply our methodology in real-life problems, analyzing networks at scales that were not possible before, attempting to unveil new larger network motifs.

Acknowledgments

Pedro Ribeiro is funded by an FCT Research Grant (SFRH/BD/19753/2004).

References

1. Albert, I., Albert, R.: Conserved network motifs allow protein-protein interaction prediction. Bioinformatics 20(18), 3346–3352 (2004)
2. Bu, D., Zhao, Y., Cai, L., Xue, H., Zhu, X., Lu, H., Zhang, J., Sun, S., Ling, L., Zhang, N., Li, G., Chen, R.: Topological structure analysis of the protein-protein interaction network in budding yeast. Nucl. Acids Res. 31(9), 2443–2450 (2003)
3. Ciriello, G., Guerra, C.: A review on models and algorithms for motif discovery in protein-protein interaction networks. Brief Funct. Genomic Proteomic 7(2), 147–156 (2008)
4. da Costa Luciano, F., Oliveira Jr., O.N., Travieso, G., Rodrigues, F.A., Villas Boas, P.R., Antiqueira, L., Viana, M.P., da Rocha, L.E.C.: Analyzing and modeling real-world phenomena with complex networks: A survey of applications. ArXiv e-prints 0711(3199) (2007)
5. Dobrin, R., Beg, Q.K., Barabasi, A., Oltvai, Z.: Aggregation of topological motifs in the escherichia coli transcriptional regulatory network. BMC Bioinformatics 5, 10 (2004)

6. Duch, J., Arenas, A.: Community identification using extremal optimization. Phys. Rev. E. (Stat. Nonlin. Soft Matter Phys.) 72, 027104 (2005)
7. Grochow, J., Kellis, M.: Network motif discovery using subgraph enumeration and symmetry-breaking. Research in Computational Molecular Biology, 92–106 (2007)
8. Itzkovitz, S., Levitt, R., Kashtan, N., Milo, R., Itzkovitz, M., Alon, U.: Coarse-graining and self-dissimilarity of complex networks. Phys. Rev. E (Stat. Nonlin. Soft Matter Phys.) 71(1 Pt. 2) (January 2005)
9. Juszczyszyn, K., Kazienko, P., Musial, K.: Local topology of social network based on motif analysis. In: Lovrek, I., Howlett, R.J., Jain, L.C. (eds.) KES 2008, Part II. LNCS (LNAI), vol. 5178, pp. 97–105. Springer, Heidelberg (2008)
10. Kashani, Z., Ahrabian, H., Elahi, E., Nowzari-Dalini, A., Ansari, E., Asadi, S., Mohammadi, S., Schreiber, F., Masoudi-Nejad, A.: Kavosh: a new algorithm for finding network motifs. BMC Bioinformatics 10(1), 318 (2009)
11. Kashtan, N., Itzkovitz, S., Milo, R., Alon, U.: Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. Bioinformatics 20(11), 1746–1758 (2004)
12. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: IEEE International Conference on Data Mining, p. 313 (2001)
13. Lusseau, D., Schneider, K., Boisseau, O.J., Haase, P., Slooten, E., Dawson, S.M.: The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. can geographic isolation explain this unique trait? Behavioral Ecology and Sociobiology 54(4), 396–405 (2003)
14. McKay, B.: Practical graph isomorphism. Cong. Numerantium 30, 45–87 (1981)
15. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: simple building blocks of complex networks. Science 298(5594), 824–827 (2002)
16. Omidi, S., Schreiber, F., Masoudi-Nejad, A.: Moda: An efficient algorithm for network motif discovery in biological networks. Genes & genetic systems 84(5), 385–395 (2009)
17. Ribeiro, P., Silva, F.: G-tries: an efficient data structure for discovering network motifs. In: ACM Symposium on Applied Computing (2010)
18. Ribeiro, P., Silva, F., Kaiser, M.: Strategies for network motifs discovery. In: 5th IEEE International Conference on e-Science. IEEE CS Press, Oxford (2009)
19. Schreiber, F., Schwobbermeyer, H.: Towards motif detection in networks: Frequency concepts and flexible search. In: Proc. of the Int. Workshop on Network Tools and Applications in Biology (NETTAB 2004), pp. 91–102 (2004)
20. Sporns, O., Kotter, R.: Motifs in brain networks. PLoS Biology 2 (2004)
21. Valverde, S., Solé, R.V.: Network motifs in computational graphs: A case study in software architecture. Phys. Rev. E (Stat. Nonlin. Soft Matter Phys.) 72(2) (2005)
22. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-world networks. Nature 393(6684), 440–442 (1998)
23. Wernicke, S.: Efficient detection of network motifs. IEEE/ACM Trans. Comput. Biol. Bioinformatics 3(4), 347–359 (2006)