2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining

Towards a Faster Network-Centric Subgraph Census

Pedro Paredes Pedro Ribeiro
CRACS & INESC-TEC
DCC-FCUP, Universidade do Porto, Portugal
pparedes@dcc.fc.up.pt, pribeiro@dcc.fc.up.pt

***Abstract*—Determining the frequency of small subgraphs is an important computational task lying at the core of several graph mining methodologies, such as network motifs discovery or graphlet based measurements. In this paper we try to improve a class of algorithms available for this purpose, namely network-centric algorithms, which are based upon the enumeration of all sets of k connected nodes. Past approaches would essentially delay isomorphism tests until they had a finalized set of k nodes. In this paper we show how isomorphism testing can be done during the actual enumeration. We use a customized g-trie, a tree data structure, in order to encapsulate the topological information of the embedded subgraphs, identifying already known node permutations of the same subgraph type. With this we avoid redundancy and the need of an isomorphism test for each subgraph occurrence. We tested our algorithm, which we called FaSE, on a set of different real complex networks, both directed and undirected, showcasing that we indeed achieve significant speedups of at least one order of magnitude against past algorithms, paving the way for a faster network-centric approach.**

***Keywords*—Complex Networks, Graph Mining, Subgraphs, G-Tries, Network Motifs, Graphlets**

I. Introduction

Many real-world systems can be modeled as complex networks, which are emerging as an ubiquitous and very flexible model for a multitude of applications [1]. There are a large number of methodologies and measurements available when we want to characterize these networks [2]. A recent trend has been to look for interesting groups of nodes within the network. These groups may have a relatively large size, as is the case in community detection [3], but they can also be of smaller sizes, like it is the case on network motifs discovery [4] or graphlet based metrics [5].

These two last methodologies have been successfully applied in the social networks domain. For instance, motifs have been used to characterize and classify co-authorship networks [6] and wikipedia edition networks [7]. Additionally, graphlets have been used to provide a complete characterization of social networks, allowing the selection of an adequate model [8].

At the core of these methods lies the computation of subgraph frequencies, also known as doing a *subgraph census*. For instance, network motifs are defined as statistically overrepresented subgraphs, which is translated into having a higher frequency than expected [4]. This means that we need to compute the amount of times each subgraph appears not only on the original network, but also on a large ensemble of similar randomized networks [9]. The problem is that calculating the frequency of subgraphs is *computationally hard*, since we are dealing with a task closely related to the *subgraph isomorphism problem*, which is NP-complete [10]. The execution time grows exponentially as we increase the size of the original network or the size of the subgraphs being analyzed, which limits the applicability. Given the ubiquity of complex networks, being able to do a faster subgraph census would therefore have an impact on a multitude of fields, enlarging the feasible limits for this computation.

Past algorithms for computing subgraph frequencies can be divided in three major conceptual approaches. *Network-centric* algorithms, such as ESU [11] or Kavosh [12], take a look at the entire network and given a certain integer k they output the frequency of all existing k-sized subgraphs. By contrast, *subgraph-centric* algorithms, such as Grochow and Kellis [13], search for one specific single subgraph type at a time, not reusing any kind of information on subsequent searches. At the middle of these two approaches, we have *set-centric* algorithms such as g-tries [14], [15], which are given as input a certain set of subgraphs and then proceed by computing the frequency of all the subgraphs in that set.

In this paper we will be dealing with network-centric approaches, for solving the problem of finding the frequency of all subgraphs of a certain size. Note that for this specific task, subgraph-centric methods would need to search individually for all possible k-sized subgraphs, while set-centric methods would need to receive as input the same set of all possible k-sized subgraphs, regardless of having no guarantees that all possible subgraph types will appear on the network being analyzed. In their essence, network-centric methods reside on two major steps: enumeration of connected sets of k nodes and isomorphism tests to determine to which subgraph type each node set belongs. Past classical approaches do this "separately": the enumeration part gives origin to sets of k nodes; afterward, each set, which corresponds to one occurrence of a subgraph, is the input to an isomorphism computation (typically by calculating a canonical labeling) so that the correspondent subgraph type frequency can be incremented. This means that we will have as many isomorphic computations as the number of occurrences of subgraphs, while the number of actual different subgraph types is much smaller.

ASONAM'13, August 25-29, 2013, Niagara, Ontario, CAN

We present the FaSE algorithm, a solution that is geared precisely towards avoiding redundancy on these tests. Instead of delaying the isomorphism testing to when a node set is "completed", we do it at the same time the enumeration is being done. We use a customized version of the g-trie, a tree-like data structure designed for storing sets of graphs, in order to represent the topology of the subgraphs being enumerated. Each time a node is added to the set, we either create a new ramification of the tree or we follow an already known path, corresponding to a topologically equivalent subgraph. A path from the root node to any leaf in the created tree corresponds to a different node permutation of a certain graph type. We compute a canonical labeling for each leaf in order to identify its subgraph type but we avoid the need for that computation on all other occurrences of the same type whose node permutation corresponds to an automorphism, that is, corresponds to the same path in the tree. In the end we get an internal subgraph representation which is aware of the topology, allowing for better understanding of what is really being identified as common substructure and opening the way for further improvements based on that knowledge.

We tested our approach on a set of representative real world simple directed and undirected complex networks with varied topological properties. We compared our results with the best known network-centric approaches. We show that we are able to obtain considerable speedups, being roughly an order of magnitude faster than past methods which need to compute isomorphism for all subgraph occurrences.

The remainder of this paper is organized as follows. Section II defines the problem being solved and talks about related work. Section III describes in detail our proposed methodology. Section IV shows our experimental results. Section V concludes the paper and also gives some directions for future work.

II. Preliminaries

A. Graph Terminology

To ensure coherence in the used terminology we briefly review the notation used. A *graph* G is composed by the set of vertices $V(G)$ and the set of edges $E(G)$, represented by pairs $(a, b) : a, b \in V(G)$. The *size* of a graph, denoted by $|V(G)|$, is the number of vertices in the graph. A k-graph has size k. All vertices are assigned consecutive integers starting from 0. The *neighborhood* of a vertex v is defined as $N(v) = \{u : (v, u) \in E(G) \vee (u, v) \in E(G)\}$.

A *subgraph* G_k of a graph G is a k-graph where $V(G_k) \subseteq V(G)$ and $E(G_k) \subseteq E(G)$. This subgraph is said to be *induced* when $\forall u, v \in V(G_k) : (v, u) \in E(G) \leftrightarrow (v, u) \in E(G_k)$ and is said to be *connected* when all pairs of vertices have a sequence of edges connecting them. The neighborhood of a subgraph, $N(G_k)$, is the set of all the neighbors of its vertices not included in G_k. The *exclusive neighborhood* of a vertex v of graph G relative to a subgraph G_k is defined as $N_{exc}(v, G_k) = \{u : u \in N(v) \wedge u \notin N(G_k) \wedge u \notin G_k\}$.

Two graphs G and H are *isomorphic*, denoted as $G \sim H$, if there is a bijection between $V(G)$ and $V(H)$ such that two vertices are adjacent in G if and only if their corresponding vertices in H are adjacent.

B. Problem Definition

We will now define more precisely the problem we are trying to solve:

Definition 1 (Subgraph Census Problem): Given an integer k and a graph G, determine the frequency of all connected induced k-subgraphs of G. Two occurrences of a subgraph are considered different if they have at least one node that they do not share.

We would like to emphasize that we are only concerned with subgraphs that are both connected and induced. Note also how we distinguish occurrences. Other possible frequency concepts do exist [16], but here we resort to the standard definition. This has direct implications on the number of existing subgraphs, with no downward closure on the frequencies, since a subgraph may appear more times than a subgraph contained in it.

C. Related Work

ESU [11] and Kavosh [12] are two of the best exact network-centric algorithms and represent our base enumeration method. Although they use different enumeration algorithms, they both work by traversing all occurrences of k connected nodes, with an isomorphic test being applied to each occurrence. Our proposed algorithm differs from these approaches because it integrates the isomorphism tests into the enumeration process, as we will see in the following sections.

For a subgraph-centric approach, we refer the reader to the work by Grochow and Kellis [13], that show how to compute the exact frequency of a single subgraph by breaking symmetries. Note that this is conceptually different from our work in this paper, given that a census would require separate computations for each different subgraph type.

Our previous work with g-tries [14], [15] is an example of a set-centric method. It takes advantage of common substructure and symmetry breaking conditions to create an efficient algorithm for computing the exact count of a given set of subgraphs. Although the data-structure used here has similarities, our approach in this paper is conceptually different because it is network-centric and does not require providing an initial list of subgraphs to search for.

In this same line of thought, if the set of subgraphs to consider if small enough, there is the possibility of doing some precalculations that reduce the computational work needed when computing the frequencies on an actual network data set. For instance, one can cache the result of isomorphic tests [17] or explore specific combinatoric features [18]. Our work differs, because we avoid the need to rely on a pre-defined set of subgraphs to consider. We target generalness and applicability on any kind of networks and subgraphs

In this paper we are mainly concerned with exact frequencies, but we would like to point out that there exist sampling alternatives for providing approximate results, such as Kashtan et al. [19], Rand-ESU [11], Rand-gtries [20] or GUISE [21].

III. A Faster Subgraph Census

Our proposal to improve the efficiency of the subgraph census is to exploit the topology of subgraphs to partition them into intermediate classes that are calculated as we are enumerating occurrences. All subgraphs belonging to a particular class are isomorphic and therefore we only need to compute a single isomorphism test per class type. This contrasts sharply with the past approach of doing one isomorphic test per occurrence, greatly reducing the number of isomorphism tests needed.

Our algorithm, which we call FaSE (following the etymology FAst Subgraph Enumeration) has two main parts, tightly integrated with each other: enumeration and encapsulation. The enumeration part is where each individual subgraph occurrence from the main network is generated. This is done by an incremental growth of a connected set of vertices. The encapsulation part is where topological information about the subgraph classes is stored, along with an increment on the frequency of the respective class. Each time the enumeration adds a new vertex to the current set, we generate a label describing the connections of this new vertex. This is done using a generic labeling process, which we call LS-Labeling, that generates different classes of subgraphs. For the actual storage we use a tree data structure that behaves as a customized g-trie, in which we use the LS-Labeling as the separator, that is, as the tree edges. The entire process is explained with more detail in the following sections.

A. Subgraph Enumeration

The process of enumerating every subgraph occurrence is a generic operation, meaning it allows different enumeration approaches. Any algorithm that enumerates all connected sets of k vertices in a larger network is acceptable, provided that it does so in a way that incrementally grows the set, node by node. The main idea is that an algorithm that transitions from a state to another by adding a new node at the time allows each enumerated occurrence to be labeled according to the transitions it took to reach the final state.

ESU [11] and Kavosh [12] are our base network-centric subgraph census algorithms and both fulfill the enumeration requisites. They have similar complexities and execution times, although they work differently. Given the space constraints, and for the sake of better understanding, we will only describe with more detail an ESU implementation, showing how we can integrate it in our methodology. We note however that the same process could indeed be followed by swapping the usage of ESU with Kavosh, keeping all other parts intact.

1) ESU algorithm: The ESU algorithm enumerates all k-subgraphs of network once and only once. It works by maintaining two vertex lists: V_S and V_E (in the original paper [11] $V_{Subgraph}$ and $V_{Extension}$). The first one represents the currently selected subgraph (set of connected vertices) and the latter a list of all neighbor vertices that can be added. Initially, it chooses each vertex v in the original graph G and sets $V_S = \{v\}$ and $V_E = N(v)$. Then it recursively removes each element u of V_E and sets $V_S = V_S \cup \{u\}$ and $V_E = V_E \cup \{v \in N_{exc}(u, V_S) : v > V_S[0]\}$ ($V_S[0]$ is the 0-th element of V_S). The usage of the exclusive neighborhood, along with the condition $v > V_S[0]$, breaks symmetries and guarantees that each occurrence is only found once. Note that $v > V_S[0]$ means that the added vertices must have a label larger than $V_S[0]$. After this operation, a recursive call is made. When the size of V_S reaches k it means a new occurrence of a k-subgraph was found.

Because of the way it works, ESU creates an implicit recursive search tree. Each node of that tree is a distinct state that has two parameters: V_S and V_E. Hence, it fits our initial description of the suitable enumeration algorithms, since each time a recursive call is made the labeling process can be done and the current vertex set can therefore be classified. Figure 1 illustrates a search tree induced by the ESU algorithm.

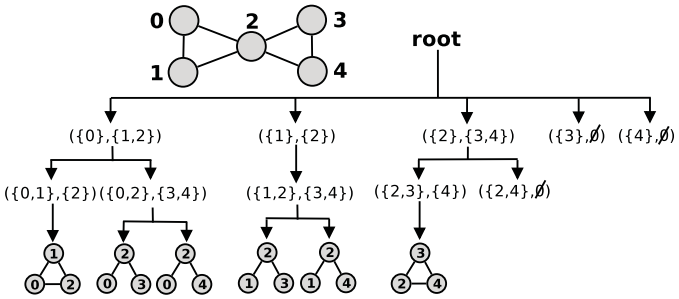

Fig. 1. An example induced ESU search tree for a 3-subgraph census.

B. Encapsulating Isomorphism Information in a Tree

While doing the enumeration we need to gradually store the information gathered. Since we are adding one vertex at a time, we want a data structure that mimics this behavior. At the same time, we want to make use of the relationships between labelings to save memory. A tree complies with the demands and naturally fits to the idea of having an hierarchical identification of common topology. The data structure we use is based on our previous work with g-tries [14], which can be thought of as "prefix trees for graphs", but the set-up and usage of this data-structure is slightly altered. In the remainder of this paper, and to avoid confusion, the term node will be used to refer to tree nodes, while the term vertex will be used for network and subgraph vertices.

1) G-Tries: The customized g-trie used in this work is a tree structure whose nodes represent abstract models of graphs, as exemplified in Figure 2. The base model, the root, represents the empty graph. Each node stores topological information and a counter representing the respective frequency. For each vertex that is added to this initially empty graph there is a labeling as described above such that it traces the newly added connections in a deterministic way, meaning that for a different graph of the same kind the label is the same. This labeling represents an edge on the g-trie, that is, the label sets the destiny node (of the edge). If there is no edge with that label yet and consequently no node to represent the current graph, they are both created. As a result, if the insertion of two graphs in the g-trie ends in the same tree node, we can be assured that the two graphs are isomorphic.

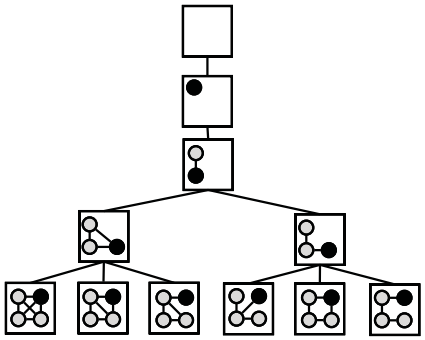

Fig. 2. An example g-trie storing six different undirected 4-graphs. The black vertices are the ones being added at each depth level.

In terms of actual functioning, our g-trie works by maintaining a current node, which represents the current graph (since it is inserted node by node), initially the root node. It implements two procedures: `Deepen` and `Jump`. The first one is responsible for inserting a new node in the current graph, thus dropping one level in the tree ("*deepening*"). It requires a labeling as an argument and simply sets the current node as the corresponding node, incrementing its count and creating it along with the due edge in case they both do not exist. We use sequence prefix trees ("tries") to represent the list of labelings that correspond to the edges, guaranteeing linear time searching and insertion. The `Jump` procedure does the opposite of `Deepen`, traversing the tree to the parent node.

To insert graphs into the g-trie we take advantage of the fact that the enumeration process builds a search tree, with a new recursive call being made each time a new node is added. The idea is to perform `Deepen` before each recursive call and `Jump` right after. This is because all occurrences generated from a certain state will share a common subtopology correspondent to the partial set generated and therefore they also share the same labeling. In practice, this means all enumerated occurrences will be in a descendant tree node.

In conclusion, the g-trie can be abstracted as a simple tree governed by the labelings. In fact, the only reason it represents graphs in its nodes is because of the way the labeling is used. So this structure is very generalized and can be applied to any labeling.

2) LS-Labeling: As said above, it is a generic labeling and only works with the condition that it creates classes of graphs that are isomorphic. It acts as the core of the g-trie and has direct influence in the branching factor of the tree, meaning that different labelings will result in different run times. In an abstract perfect environment, the LS-Labeling would be a function that actually computed isomorphisms, but since that calculation itself is computationally hard, the time saved by using the g-trie would be spent on calculating the labelings. So we want the LS-Labeling function to be polynomial, setting up a trade off in the amount of calculation time spent on the labeling and on the g-trie (and also on the isomorphism calculations). Furthermore, we want a labeling that is able to be computed incrementally as new vertices are added, so that we can use only the part pertinent to any new vertex being added (the other existing parts will already be stored on ascendant nodes).

For our work we tried essentially two simple and intuitive labelings: "adjacency list" and "adjacency matrix", corresponding to the equivalent graph data-structures, as their names suggest. However, when a new vertex is added we only store its connections to vertices already in the constructed set. Consider now the undirected case, when adding the k-th vertex. For the adjacency list we have a list of at most $k-1$ integers where an integer is present if there is an edge to the node with that label. For the adjacency matrix we have a sequence of $k-1$ boolean values indicating if there is an edge to the node whose label corresponds to the position in the sequence. Figure 3 gives an example of these two undirected labelings. In the directed case, we store in this way both the ingoing and outgoing connections.

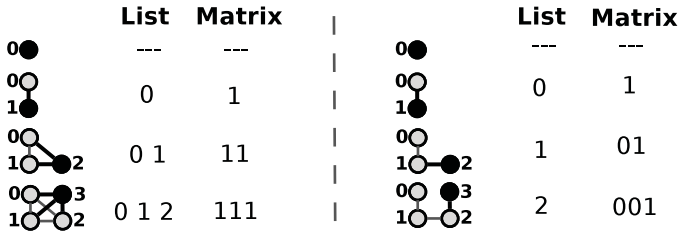

Fig. 3. Two different valid LS-Labeling schemes on two example graphs. Black vertices are the ones being added.

We can easily prove the correctness of both approaches, but since they are methodically equivalent we will refer to both by considering their structure. Notice that the correction property is that the labeling creates classes of isomorphic subgraphs, as stated in the beginning of the section. Thus we can conclude that the labeling is correct since we can find a bijection between any two subgraphs belonging to a particular labeling class by simply following the order of the enumeration (that is implicit in the actual labeling) and consider the permutation relative to this order. Hence they are all isomorphic and we have our correctness proof.

In conclusion, LS-Labeling is a powerful and flexible operation since its generic nature allows for multiple types of labelings. This means we can regulate the trade off of calculation time in the g-trie and in the labeling itself. Besides it makes the algorithm work for different formulations of the problem. For instance, the algorithm is exactly the same whether the graph is undirected or directed, with the only difference relying on the labeling itself. Also, it allows more complex examples to be considered, such as colored graphs, weighted graphs or multigraphs, just by using an adequate type of LS-Labeling.

C. The FaSE algorithm

Algorithm 1 presents an overview of the complete FaSE algorithm, which incorporates ESU enumerations with LS-Labeling and g-tries. The expression `+=` is used to express incrementation by a certain value.

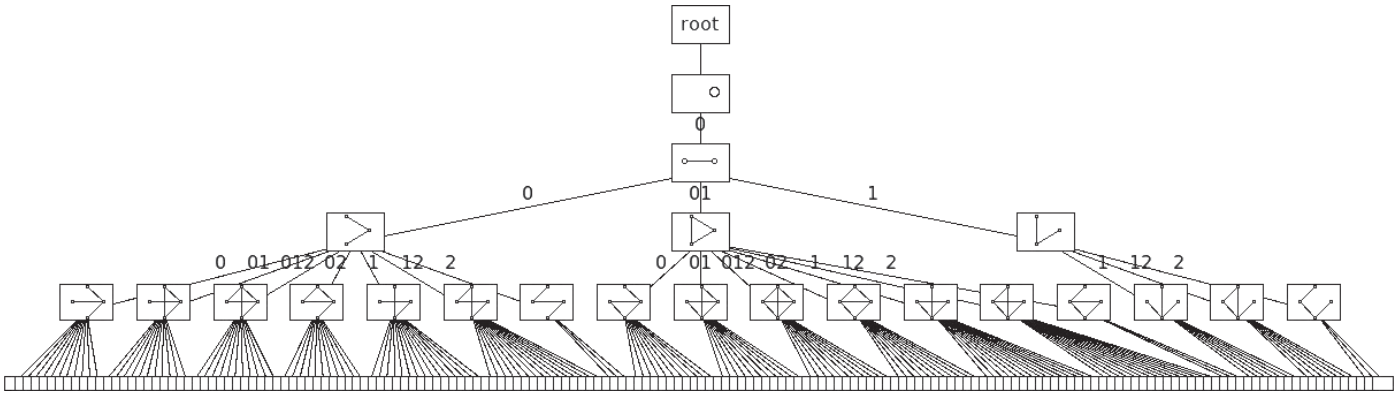

Fig. 4. An example g-trie with list LS-Labeling after searching for 5-subgraphs. Details of the lower level are hidden for visual clarity.

Algorithm 1 The $FaSE$ Algorithm

Input: A graph G and a subgraph size k
Result: Frequencies of all k-subgraphs of G

1: **procedure** FASE(G, k)
2: $EnumerateAll(G, k, \emptyset, 0)$
3: **for all** n in $GTrie.leaves()$ **do**
4: $frequency[CanonicalLabel(n.Graph)]$ += $n.count$

5: **procedure** ENUMERATEALL(G, K, S, d) ▷ S:subgraph; d:depth
6: **if** $d = K$ **then**
7: $GTrie.current.count$ += 1
8: **else**
9: **while** $nS \leftarrow EnumerateNext(S)$ **do**
10: $w \leftarrow nS.NextNode()$
11: $nL \leftarrow LSLabel(S, w)$
12: $GTrie.Deepen(nL)$
13: $nS.Subgraph \leftarrow nS.Subgraph \cup w$
14: $EnumerateAll(G, K, nS, d + 1)$
15: $GTrie.Jump()$

The procedure `enumerateAll()` passes through all occurrences of the subgraphs and increments the counter when at the desired size. In the end, the actual frequencies are implicitly stored at g-trie leaves. However, two different leaves may correspond to the same subgraph isomorphic type, and we still need to compute a canonical labeling to disambiguate and guarantee that the final results will reflect the real frequencies of each type. In our current implementation, this is done through `nauty` [22], a third-party efficient isomorphism toolkit, but any other algorithm that would produce a canonical labeling could be used. Figure 4 shows a visual representation of what a g-trie annotated with the list LS-Labeling would look like after running FaSE. Note how at the same level we end up having different nodes representing the same subgraph type, due to the order in which the vertices were added to the graph. Nevertheless, the number of leaves will be just a very small fraction of the total number of occurrences, and that is why we gain a significant amount of computation time: we only do one isomorphic test per leaf, and not per occurrence.

Also note that in the specific case of the ESU algorithm the actual state S used in the `EnumerateNext()` procedure is also changed on each iteration of the **while** loop. To increase the efficiency of the algorithm, the enumeration algorithms can be hard coded to explore low level features of the specific algorithms, as was done in our current implementation.

IV. Experimental Results

To evaluate the performance of the proposed algorithm, FaSE was implemented in C++, using ESU as the enumeration algorithm[1]. All tests were run on a Linux machine with an Intel Core 2 6600 (2.4GHz) and 2GB of memory. We used a wide range of directed and undirected networks, showcasing general applicability. All weights or self-loops were ignored. The networks features are summarized in Table I. The `StarWars` network was assembled by us using a script from the Star Wars IV film, where vertices are characters in the movie and two vertices are connected if there is a scene where both characters speak.

We implemented both the list and adjacency LS-Labeling functions, but in all the networks the list method ended up having slightly better execution time, and so we opted to only show the results obtained with it. We compared our results with the two base network-centric methods: ESU and Kavosh. The former through its publicly available tool FanMod [28] and the latter through its original source code [12], both of which also use `nauty` [22] for isomorphisms. We compared the time needed for doing a complete k-subgraph census on the original networks as we varied k from 3 to 9. For practical reasons, we only show execution times up to 5 hours. The results are detailed in Table II, with speedup indicating the relative time gain versus ESU and Kavosh.

The most relevant fact to notice is that in all cases FaSE outperforms both ESU and Kavosh, as expected, with the speedup being approximately an order of magnitude faster. More than that, as the size k grows the general tendency is for the speedup to grow, with bigger gains in situations

[1]Our implementation, along with test data, can be consulted on the following url: http://www.dcc.fc.up.pt/gtries/fase/

TABLE I. COMPLEX NETWORKS USED IN THE EXPERIMENTS

Network	Directed	Nodes	Edges	Avg. Degree	Type	Source
StarWars	No	51	157	3.08	Social	Our Own
Jazz	No	198	2,742	13.85	Social	Arenas [23]
Power	No	4,941	6,594	1.33	Geo-spacial	Newman [24]
AstroPh	No	18,772	198,050	10.55	Social	SNAP [25]
Gloss	Yes	72	118	1.64	Semantic	Pajek [26]
Neural	Yes	297	2,359	7.94	Biological	Newman [24]
Email	Yes	1,133	10,902	9.62	Social	Arenas [27]
Foldoc	Yes	13,356	120,700	9.04	Semantic	Pajek [26]

TABLE II. DETAILED EXPERIMENTAL RESULTS FOR THE 8 COMPLEX NETWORKS USED.

Network	K	Subgraphs found		FaSE		ESU		Kavosh	
		Types	Occurrences	Time (s)	Leaves	Time (s)	Speedup	Time (s)	Speedup
StarWars	3	2	1,449	<0.01	3	<0.01	—	<0.01	—
	4	6	12,958	<0.01	17	0.04	**23.5**	0.03	**17.6**
	5	21	98,426	0.01	171	0.39	**30.7**	0.21	**16.5**
	6	106	630,369	0.08	2,406	3.12	**38.0**	1.90	**23.1**
	7	699	3,445,808	0.58	26,692	21.95	**38.0**	13.26	**23.0**
	8	5,601	16,320,648	3.55	203,687	133.34	**37.6**	78.18	**22.0**
	9	41,790	67,883,236	19.08	1,133,749	(*)	—	395.90	**20.7**
Jazz	3	2	67,414	<0.01	3	0.14	**31.8**	0.06	**13.6**
	4	6	1,833,618	0.15	17	4.24	**28.9**	2.55	**17.4**
	5	21	49,500,654	4.65	171	143.64	**30.9**	89.3	**19.2**
	6	112	1,266,953,062	140.84	3,113	(**)3,630.00	**25.8**	2,912.43	**20.7**
	7	853	30,166,157,456	3,946.81	106,417	>5h	—	>5h	—
Power	3	2	17,631	<0.01	3	0.04	**14.3**	0.02	**7.1**
	4	6	63,401	0.01	17	0.14	**15.6**	0.09	**10.0**
	5	21	268,694	0.04	170	0.67	**17.5**	0.48	**12.5**
	6	101	1,260,958	0.19	1,771	4.11	**21.6**	2.91	**15.3**
	7	626	6,340,413	1.07	14,441	23.60	**22.0**	17.57	**16.4**
	8	4,516	33,494,650	6.08	96,219	155.44	**25.6**	113.53	**18.7**
	9	31,543	183,453,978	35.84	565,387	(*)	—	722.91	**20.2**
AstroPh	3	2	10,046,498	1.02	3	15.69	**15.4**	8.88	**8.7**
	4	6	923,805,607	95.92	17	(**)2,272.00	**23.7**	1,244.03	**13.0**
	5	21	108,845,670,981	12,411.83	171	>5h	—	>5h	—
Gloss	3	4	558	<0.01	14	<0.01	—	<0.01	—
	4	24	3,229	<0.01	122	<0.01	—	<0.01	—
	5	162	18,636	0.01	783	0.06	**8.8**	0.05	**7.4**
	6	907	101,435	0.04	4,482	0.34	**9.6**	0.41	**11.6**
	7	4,716	513,982	0.22	23,508	1.94	**8.7**	3.51	**15.7**
	8	22,789	2,428,915	1.45	115,094	11.04	**7.6**	27.22	**18.7**
	9	103,229	10,756,147	9.55	526,053	(*)	—	205.72	**21.5**
Neural	3	13	47,322	0.01	45	0.09	**16.7**	0.04	**7.4**
	4	197	1,394,259	0.13	1,846	2.21	**17.5**	1.71	**13.5**
	5	7,072	43,256,069	4.73	76,214	102.14	**21.6**	91.03	**19.3**
	6	286,376	1,309,307,357	170.96	2,499,645	(**)4,420.00	**25.9**	4,636.43	**27.1**
Email	3	2	85,729	0.01	3	0.13	**12.9**	0.07	**6.9**
	4	6	1,905,641	0.17	17	3.87	**22.6**	2.52	**14.7**
	5	21	49,324,622	5.56	171	117.49	**21.1**	79.99	**14.4**
	6	112	1,375,965,126	182.45	3,113	(**)4,186.00	**22.9**	2,929.36	**16.1**
Foldoc	3	13	2,553,830	0.35	45	3.97	**11.2**	2.17	**6.1**
	4	198	228,272,189	27.80	2,304	903.39	**32.5**	308.78	**11.1**
	5	8,345	29,621,881,964	3,735.20	141,115	>5h	—	>5h	—

(*) FanMod accepts only 8 as the maximum subgraph size.

(**) Overflow problem in its own reported enumeration time and so we used elapsed time.

where the total execution time is higher, the ones which benefit more from a faster algorithm. The specific speedups are network-dependent and are related to the number of subgraph occurrences versus the number of leaves in our g-trie. Note that by itself the ratio between these two quantities cannot be used to accurately predict the speedup because it only represents aggregated values and the specific times needed for the isomorphic tests are subgraph dependent.

Our current implementation uses ESU for enumeration. In order to be sure that we are really improving the underlying algorithm and not just creating a better implementation of ESU, we tested the execution times when not using g-tries and doing isomorphic tests trough `nauty` for each occurrence, mimicking the original behavior of ESU. We observed that our implementation was slightly faster than FanMod, but within the same order of magnitude. Thus, if we compute the speedup of FaSE against this implementation, we still are roughly an order of magnitude faster. Noting that Kavosh is slightly faster than ESU, we will potentially benefit from this if we implement the enumeration part resorting to Kavosh instead of ESU.

The core of our gains resides on improving the isomorphism tests, which is the major bottleneck of a subgraph census. We experimented with our implementation stripping it from everything but the enumeration process. This means no g-trie was created, nor isomorphic tests were performed. Of course that in this case we would not be able to actually compute the census, but this can give a better sense of how much time the enumeration process takes when compared to the isomorphic tests. In fact, for the ESU case, the enumeration is just a small fraction of the entire computation, and the weight of the enumeration decreases as the size of the subgraph increases.

The `Gloss` network is the one where we achieve the worst speedups, but it is also an outlier on all of our results. In fact this is also the only case where FanMod is quicker than Kavosh. Furthermore, our own implementation is also slower than FanMod in this network. One possible explanation is that there is something specific in FanMod implementation that gives it an advantage for this particular data set.

Another aspect is that we currently use simple adjacency matrix as the graph representations, which is unfeasible for larger networks. We experimented with an adjacency list representation and a naive binary search for checking edge existence and we were able to run FaSE on networks with more than 1 million nodes. We could not compare with Kavosh because it does not support such larger networks, but in comparisons with FanMod we had better execution times, albeit with relatively smaller speedups. We expect that the usage of more efficient and specialized graph primitives will greatly improve our times.

Finally, we point the reader to the number of leaves our algorithm is generating. This is directly connected to the memory needed by the g-trie and it means we cannot go to much larger subgraph sizes than the ones depicted here, for the general case. Nevertheless, as k grows, the number of possible subgraph types also grows super exponentially and simply storing the frequency results may become prohibitive.

V. Conclusion

In this paper we presented FaSE, a novel algorithm that tackles the subgraph enumeration problem through a network-centric approach. It identifies common substructures using a tree data-structure allowing for savings in the number of isomorphic tests needed. The results obtained show that FaSE consistently outperforms the classic network-centric algorithms that need one isomorphic test per occurrence, being one order of magnitude faster.

FaSE is very generic, allowing for the usage of different enumeration methods, as well as different LS-Labeling functions. We intend to use this in the future, by implementing potentially better enumeration algorithms (we will start with Kavosh) and by experimenting with different LS-Labelings, paving the way for more complex setups, such as multigraphs and colored or weighted graphs. Note that the generalness of our approach implies that it will more naturally adapt to these scenarios. For instance, a method that relies on having a pre-generated set would be unfeasible for a colored graph, because the total number of possible subgraph types increases super exponentially. In our case, the algorithm would naturally only include the occurring types, which can be a very small fraction of the entire set of possibilities.

Given that the work presented here only gives exact results, we also intend to work on a sampling version, able to trade accuracy for even better execution times.

The main limitation of FaSE is the number of leaves it generates when comparing with the number of existing subgraph types. We intend to reduce the number of leaves by taking advantage of the fact that our intermediate data-structure is aware of the topology. We wish to experiment both with new LS-Labelings and with compressing the g-trie by acknowledging the topological equivalence of different tree nodes. The end goal is to have a g-trie with only has many leaves has different subgraph types, produced dynamically during the enumeration without any previous knowledge about the network or subgraph types.

Acknowledgments

This work was partially funded by FEDER/ON2 and FCT project NORTE-07-0124-FEDER-000059. Pedro Ribeiro is funded by an FCT Research Grant (SFRH/BPD/81695/2011).

References

[1] L. Costa, O. Oliveira Jr, G. Travieso, F. Rodrigues, P. Boas, L. Antiqueira, M. Viana, and L. Da Rocha, "Analyzing and modeling real-world phenomena with complex networks: a survey of applications," *Adv Phys*, vol. 60, pp. 329–412, 2011.

[2] L. F. Costa, F. A. Rodrigues, G. Travieso, and P. R. V. Boas, "Characterization of complex networks: A survey of measurements," *Adv Phys*, vol. 56, p. 167, 2007.

[3] S. Fortunato, "Community detection in graphs," *Phys Rep*, vol. 486, no. 3-5, pp. 75–174, 2010.

[4] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network Motifs: Simple Building Blocks of Complex Networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[5] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, vol. 26, no. 6, pp. 853–854, 2010.

[6] S. Choobdar, P. Ribeiro, S. Bugla, and F. Silva, "Co-authorship network comparison across research fields using motifs," in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE, August 2012, pp. 147–152.

[7] G. Wu, M. Harrigan, and P. Cunningham, "Characterizing wikipedia pages using edit network motif profiles," in *3rd Int. workshop on search and mining user-generated contents (SMUC)*. New York, NY, USA: ACM, 2011, pp. 45–52.

[8] E. Janssen, M. Hurshman, and N. Kalyaniwalla, "Model selection for social networks using graphlets," *Internet Mathematics*, 2012.

[9] P. Ribeiro, F. Silva, and M. Kaiser, "Strategies for network motifs discovery," in *IEEE International Conference on e-Science*, ser. e-Science, 2009, pp. 80–87.

[10] S. A. Cook, "The complexity of theorem-proving procedures," in *ACM Symposium on Theory of computing*, ser. STOC. New York, NY, USA: ACM, 1971, pp. 151–158.

[11] S. Wernicke, "Efficient detection of network motifs," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 347–359, 2006.

[12] Z. Kashani, H. Ahrabian, E. Elahi, A. Nowzari-Dalini, E. Ansari, S. Asadi, S. Mohammadi, F. Schreiber, and A. Masoudi-Nejad, "Kavosh: a new algorithm for finding network motifs," *BMC bioinformatics*, vol. 10, no. 1, p. 318, 2009.

[13] J. Grochow and M. Kellis, "Network motif discovery using subgraph enumeration and symmetry-breaking," *Research in Computational Molecular Biology*, pp. 92–106, 2007.

[14] P. Ribeiro and F. Silva, "G-tries: an efficient data structure for discovering network motifs," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. ACM-SAC, 2010, pp. 1559–1566.

[15] P. Ribeiro and F. Silva, "Querying subgraph sets with g-tries," in *2nd ACM SIGMOD Workshop on Databases and Social Networks*, ser. DBSocial. ACM, 2012, pp. 25–30.

[16] F. Schreiber and H. Schwobbermeyer, "Towards motif detection in networks: Frequency concepts and flexible search," in *International Workshop on Network Tools and Applications in Biology*, ser. NetTAB, 2004, pp. 91–102.

[17] X. Li, D. S. Stones, H. Wang, H. Deng, X. Liu, and G. Wang, "Netmode: Network motif detection without nauty." *PLoS One*, vol. 7, no. 12, p. e50093, 2012.

[18] D. Marcus and Y. Shavitt, "Efficient counting of network motifs." in *ICDCS Workshops*. IEEE Computer Society, 2010, pp. 92–98.

[19] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon, "Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs," *Bioinformatics*, vol. 20, no. 11, pp. 1746–1758, 2004.

[20] P. Ribeiro and F. Silva, "Efficient subgraph frequency estimation with g-tries," in *International Workshop on Algorithms in Bioinformatics*, ser. WABI, vol. 6293. Springer, 2010, pp. 238–249.

[21] M. Bhuiyan, M. Rahman, M. Rahman, and M. A. Hasan, "Guise: Uniform sampling of graphlets for large graph analysis," in *IEEE International Conference on Data Mining*, ser. ICDM, 2012, pp. 91–100.

[22] B. McKay, "nauty," `http://cs.anu.edu.au/~bdm/nauty/`, 2012.

[23] P. M. Gleiser and L. Danon, "Community structure in jazz," *Advances in Complex Systems*, vol. 06, no. 04, pp. 565–573, 2003.

[24] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, pp. 440–442, 1998.

[25] J. Leskovec, J. M. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Transactions on Knowledge Discovery From Data*, vol. 1, no. 1, 2007.

[26] V. Batagelj and A. Mrvar, "Pajek datasets," `http://vlado.fmf.uni-lj.si/pub/networks/data/`, 2006.

[27] R. Guimera, L. Danon, A. Diaz-Guilera, F. Giralt, and A. Arenas, "Self-similar community structure in a network of human interactions," *Phys. Rev. E*, vol. 68, p. 065103, 2003.

[28] S. Wernicke and F. Rasche, "Fanmod: a tool for fast network motif detection," *Bioinformatics*, vol. 22, no. 9, pp. 1152–1153, 2006.