Detection of Communities in Directed Networks based on Strongly p-Connected Components

Vincent Levorato, Coralie Petermann

Detection of Communities in Directed Networks based on Strongly p-Connected Components

Vincent Levorato
University of Orléans
LIFO - Orléans, France
vincent.levorato@univ-orleans.fr

Coralie Petermann
Ecole Pratiques des Hautes Etudes
LaISC - Paris, France
coralie.petermann@laisc.net

Preprint Version

Abstract

A lot of algorithms in communities detection have been proposed particularly for undirected networks. As methods to find communities in directed networks are few, our contribution is to propose a method based on strongly and unilaterally connected components, and more specifically on strongly p-connected components in directed graphs. The result is a clustering of nodes giving good results in generated graphs according to several clustering evaluation measures, and which practical time complexity remains acceptable.

1. Introduction

The field of complex systems have highlighted the central role played by networks in many phenomena. Analyzing the structure and the dynamic of a complex network is a difficult task which researchers are trying to answer. A partial response to this problem is the identification of equivalent roles of the elements composing such a network, which can be of several natures (social, biological, etc.) [1]. To solve this problem, one can use some techniques of different fields like *clustering* in data mining or *graph partitions* in graph theory. This kind of idea is generalized by a research field known as *communities detection*. With the expansion of the web, most of the works dedicated to communities detection focus on social networks of the web. Habitually, we represent networks as graphs, and a lot of papers are dealing with this subject since some researchers found some special properties characterizing these real networks [2]–[4]. Most of the papers on communities detection use graphs as a mathematical representation of a network, and more precisely undirected graphs [5]. However, a lot of networks have complex relations and are expressed as *directed* links. There are few algorithms that identify communities in oriented networks, and it is easier to ignore link orientation. Moreover, transforming a directed graph into an undirected one causes a loss of information, and the detected communities don't really make sense [6]. As few works propose communities detection in directed networks, finding these communities remains a difficult task [5].
In this paper, we focus on the communities detection into directed networks, and on the definition of a community in such a network based on *connected components*. First, we give some basic notions of graph theory, and definitions of connected components, which is the main subject of our work. Then, we explain how we define a community in a directed network, and how our method works. Finally, we present results based on benchmarks dedicated to the communities detection and on measures validating our clustering algorithm.

2. Graph Theory Notions

2.1. Graph definitions

In this article, we consider only *directed graphs* (also noted *digraph*). Formally, a graph $G = (V, A)$ is the pair composed of [7]:

1) a set $V = \{x_1, x_2, ..., x_n\}$ named *vertices* or *nodes*.
2) a family $A = (a_1, a_2, ..., a_n)$ of elements of the Cartesian product $V \times V = \{(x, y)/x \in V, y \in V\}$ named *arcs*.

We usually denote n as the amount of vertices (also noted $|V(G)|$) and m the amount of arcs (also noted $|A(G)|$). A *path* P is composed of k arcs such as $P = (a_1, a_2, ..., a_i, ..., a_k)$ where for every arc a_i the terminal end coincides with the initial end of a_{i+1}. Several equivalent notations can be used: $P = ((x_1, x_2), (x_2, x_3), ...) = [x_1, x_2, ..., x_k, x_{k+1}\ ] = P[x_1, x_{k+1}]$. A *chain* is, like a path, an alternating sequence of vertices and edges, where an edge is an arc without orientation.

2.2. Connected Components

As our work focus on connected components in a directed graph, we give a reminder of some definitions [8]:

- a *weakly connected component* WCC of a digraph is a subgraph where: $\forall x, y \in WCC$, there is a chain between x and y.
- an *unilaterally connected component* UCC of a digraph is a subgraph where: $\forall x, y \in UCC$, there is a path between x and y, or there is a path between y and x.
- a *strongly connected component* SCC of a digraph is a subgraph where: $\forall x, y \in SCC$, there is a path between x and y, and there is a path between y and x.

A *circuit* is a path such that the first node of the path corresponds to the last. It can be viewed as an oriented cycle.

In this paper, we use a special case of strongly connected component named *strongly p-connected component* by [9] which is related to l-edge-connectivity [10] (we use *p-connected* notation instead of *n-connected* to avoid confusion):

- a *strongly p-connected component* p-SCC of a digraph is a subgraph where: $\forall x, y \in$ p-SCC, there is a path of length p or less between x and y, and there is a path of length p or less between y and x, with $p \geq 2$.

This definition can be easily generalized to other types of components.

3. Communities detection

3.1. Community definition

Concerning undirected networks, a lot of works has been done mainly in: graph partitioning [11], [12], clustering [13]–[17] and random walks [18]–[22]. Some of these algorithms have been adapted to oriented networks with varying degrees of success [23]. If the definition of a community is blurry [5], the communities appear to be relatively similar sets of nodes strongly interrelated and more weakly associated with the rest of the network. We can also imagine a community in term of network with dense communication. Based on this idea, we can define a community as a group of elements where each element can strongly communicate with other elements: this reflects the fact that there is always a bidirectional flow that connects two elements of a community. This vision can be related to the definition of the strongly connected components (SCC) we have in graph theory. As the SCC of a graph are maximal, these components are often not precise enough to make appear real communities. We answer to this problem by using strongly p-connected components (p-SCC) as subdivisions of the initial SCC. We can establish a hierarchical classification of components corresponding itself to a hierarchical classification of communities (Fig. 1). Thus, we consider *the detection of communities as the finding of p-SCC*. Note: in this article, we only work on clustering without overlapping.

Figure 1. Hierarchical Classification of Components.

3.2. Our Method

One of the most known algorithm to find SCC is the Tarjan's algorithm [24] which uses depth-first search (DFS). As we want to find some p-SCC, the exploration of the graph is bounded by p (the maximum path length in the component) in term of arcs. This operation is realized by a breadth-first search (BFS). Here is a description of each step of our method which computes p-SCC:

1) Choose a random start vertex x_s.
2) Make a BFS beginning at x_s.
3) If, during the BFS, a circuit with x_s is found, all nodes of the circuit are added to the current component.
4) Stop the BFS if current path length reaches p, and add current component to the list of p-SCC.
5) Remove nodes of the last added component of the graph.
6) Go to step 1 if the graph is not empty.

The method is using a FIFO (i.e., First In, First Out) queue like a standard BFS. An example of the execution of our algorithm is given in Figure 2.

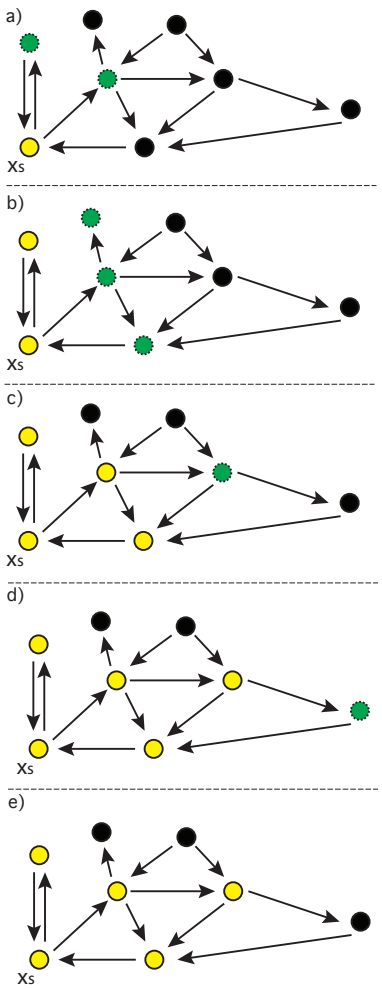

Figure 2. Example of execution of our algorithm with $p = 4$.

In the illustration of Fig. 2, we seek 4-SCC. At step *a)*, the neighborhood of x_s is explored. These nodes are contained in paths which are potential circuits (dashed nodes). At next step *b)*, we find a circuit of length 2. Step *c)* finds a circuit of length 3 and deselects a node which is a leaf (upper node). Step *d)* finds a circuit of length 4. Step *e)* stops the algorithm because we reach the maximum path length p.

As our method starts exploration from a random vertex, results would be different between several executions, but it can be considered as a small issue as we will note in the experiment part. A bigger problem is the disconnection of nodes: the algorithm can find singletons as communities. In the previous example of the Figure 2 at step *e)*, three nodes remains disconnected. To avoid this problem, a final phase groups communities having a too small size (which is a parameter to adjust). Having our p-SCC considered as core communities, we merge all "small" communities with large ones, if there is an arc in common between them, which extends some of the p-SCC to UCC (Fig. 3). This phase reduces errors that can be obtained by selecting nodes at random for each BFS.

The first algorithm (Alg. 1) finds a p-SCC starting from a given node x_s. This method is called while the graph is not empty (Alg. 2) and the merging operation is then applied on found p-SCC (Alg. 3).

Algorithm 1:

Method pSCC(graph G, vertex x_s, integer p)
{C:set, paths:list, P:path, A(x):neighborhoods of x}
$C \leftarrow \emptyset$
$paths \leftarrow \emptyset$
$paths.add([x_s])$
while $paths \neq \emptyset$ **do**
  $P \leftarrow paths.removeFirstPath()$

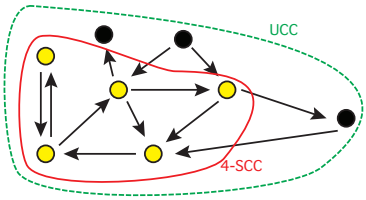

Figure 3. Merging communities in UCC

```
    if x_s = P.x_{k+1} then
        C ← C ∪ P
    else
        for each y ∈ A(P.x_{k+1}) do
            PP ← P
            PP.add(y)
            if |PP| ≤ p + 1 then
                paths.add(PP)
            end if
        end for
    end if
end while
```

Algorithm 2:

```
Method AllpSCC(graph G, integer p)
{C:set, SCC:family of sets}
SCC ← ∅
while V(G) ≠ ∅ do
    x ← random node of V(G)
    C ← pSCC(G, x, p)
    V(G) ← V(G) − C
    SCC ← SCC ∪ C
end while
return SCC
```

Algorithm 3:

```
Method MergeSCC(graph G ,family SCC, integer minCsize)
{C,smallC:sets, smallSCC:family of sets, A(x):neighborhoods of x}
smallSCC ← ∅
for each C ∈ SCC do
    if |C| ≤ minCsize then
        smallCC ← smallCC ∪ C
    end if
end for
SCC ← SCC − smallSCC
while smallSCC ≠ ∅ do
    smallC ← smallSCC.remove()
    for each x ∈ smallC do
        for each C ∈ SCC do
            if A(x) ∩ A(C) then
                C ← C ∪ {x}
            end if
        end for
    end for
end while
return SCC
```

Note: if the graph is not weakly-connected, we apply our method on each weakly connected component of the graph.

For now, the complexity of our algorithm is exponential in the maximum length of the paths. The worst case for our method can be obtained by applying it on a complete graph K_n (each pair (x, y) of nodes has an arc from x to y and from y to x). As the method *pSCC* keeps in memory all current paths, starting with a random node, we have $n-1$ initial paths. Then we exhaustively explore the neighborhoods of each path ends. The number of operations is equal to: $(n-1)^p$. The worst case time complexity can be written as $O(n^p)$ with p the maximum path length. Although this complexity is exponential, it is parametrized by the maximum length of paths. It can be related to the Tarjan's algorithm [25] which enumerates all circuits in a digraph with a complexity in time of $O((n \cdot m)(c+1))$ (c is the amount of elementary circuits and is exponential). In real networks, the worst case should not be met because they are used to be sparse [16], and our experiments show that the best value for p is no more than 4 with a low experimental complexity. The space complexity follows the same rules.

4. Experiments

To test the efficiency of our method, we use specific benchmarks dedicated to the detection of communities. Graph generators have been introduced allowing complex tests of realistic undirected [26] and directed [27] graphs with a reference dataset of communities. However, these graph generators are not intended to model the phenomena which result from real networks, they are simply trying to imitate some of their properties while imposing a structure in the community. We are aware that the community structures of real networks are certainly more complex than that generated graphs. The problem is that there is no real large graphs for which a reference partitioning is known. So we test our method on directed versions of these graphs, and we then apply it to some real network datasets to estimate practical time efficiency.
The most common measure of similarity in the literature to compare two partitions is the Normalized Mutual Information (NMI) [28] which had been generalized to covering partitions (GNMI) by [29]. This measure is quite close to the standard NMI even when communities do not overlap. However, the use of a single test measurement is not sufficient. The evaluation of clustering is an important field of machine learning domain, and a lot of different measures which compare two clustering exist. In addition to the NMI, we measured the performance of our algorithm to find communities using: Adjusted Rand Index, Jaccard index, and F-measure. We compare our scores to InfoMap method [22] which gives good results in undirected [30] and directed [23] graphs. Obviously, we compare ourselves only to the directed version of their algorithm.

4.1. Evaluation methods

In a clustering evaluation, there is a correspondence between reference and predicted clusters, expressed by the *confusion matrix*. The confusion matrix can be considered to list the frequencies of occurrence common to reference-predicted pairs clusters. The mean information content of the confusion matrix is, by definition, identical to the mean entropy of reference-predicted pairs that resulted from the experiment. We use the following notations: $I(X;Y)$ is the mutual information of X and Y with $I(X;Y) = H(X) - H(X|Y)$, $H(X)$ and $H(Y)$ are the marginal entropies, $H(X|Y)$ and $H(Y|X)$ are the conditional entropies, and $H(X,Y)$ is the joint entropy of X and Y which are two discrete random variables. All measures are between 0 (worst matching) and 1 (best matching).

4.1.1. Normalized Mutual Information [28]. It is defined as the mutual information between the cluster assignments and a pre-existing labeling of the dataset normalized by:

$$NMI(X,Y) = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}}$$

4.1.2. Jaccard Index [31]. The Jaccard index is used in statistics to compare the similarity and diversity between samples. Given two sets A and B, the Jaccard index is expressed as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

A set-theoretic interpretation of information shows that $I(X;Y)/H(X,Y)$ is equivalent to the Jaccard Index.

4.1.3. Adjusted Rand Index [32]. The adjusted form of the Rand Index, the Adjusted Rand Index, which is used for increasing the sensitivity of the index value, is defined as:

$$AdjutedIndex = \frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex}$$

It can be related to the adjusted measure for the mutual information:

$$AI(X;Y) = \frac{I(X;Y) - E\{I(X;Y)\}}{max\{H(X),H(Y)\} - E\{I(X;Y)\}}$$

4.1.4. F-measure [33]. The formula for the clustering F-measure is defined in [34]. The recall, precision, and F-measure for natural class K_i (reference) and cluster C_j (predicted) are defined as $Recall(K_i,C_j) = n_{ij}/|K_i|$, $Precision(K_i,C_j) = n_{ij}/|C_j|$ and :

$$F(Ki;Cj) = \frac{2 \times Recall(K_i,C_j) \times Precision(K_i,C_j)}{Recall(K_i,C_j) + Precision(K_i,C_j)}$$

where n_{ij} corresponds to a cell of the confusion matrix.

4.2. Results on generated graphs

Our tests have been made on several digraphs generated by [23] and having different parameters:

- n the number of nodes of the graph
- k_{avg} the average node degree, k_{max} the maximum node degree
- min_c, max_c min. and max. community size
- μ the mixing parameter i.e. for each vertex the ratio between the number of outgoing arcs and the number inside his community.

As done in [30], we move one criteria for each test (degree, mixing parameter, and community size), but the difference is that we did our tests on graphs of different sizes, from $n = 1000$ to $n = 10000$. Each score corresponds to a mean over 50 graph generations. The parameters used for our method is $p = 4$ (maximum path length) and $minCsize = 3$ (minimum community size) which give the best results on all types of graph. All tests are using JUNG graph (Java) library [35] and have been runned on a lambda laptop. Our method is noted *LP Method* in the following tests.

4.2.1. Standard Graph. We set the graph generator parameters with reference values as follows: $k_{avg} = 15, k_{max} = 50, min_c = 20, max_c = 50, \mu = 0.1$. The results of the measures appear in Table 1. It appears that our algorithm gives always good results which are increasing with the number of nodes.

		Amount of nodes					
Algorithm	**Measures**	1000	2000	3000	4000	5000	10000
LP Method	ARI	0.95	0.97	0.98	0.98	0.98	0.99
	Jaccard	0.91	0.95	0.96	0.96	0.97	0.98
	F-Measure	0.97	0.98	0.99	0.99	0.99	0.99
	NMI	0.93	0.95	0.97	0.97	0.98	0.99
InfoMap	ARI	0.62	0.64	0.65	0.67	0.67	0.69
	Jaccard	0.47	0.48	0.49	0.50	0.50	0.53
	F-Measure	0.69	0.71	0.72	0.74	0.75	0.77
	NMI	0.41	0.46	0.48	0.50	0.51	0.52

Table 1. Efficiency test while increasing n.

4.2.2. Graph with Bad Defined Communities. We set the graph generator parameters as standard graph values and $n = 5000$, changing only the mixing parameter form $\mu = 0.1$ to $\mu = 0.5$. The results of the measures appear in Table 2. The efficiency to find communities decreases with increase of μ which makes sense because each node has many links within and outside its own community, and it becomes more and more difficult to find communities.

		Mixing parameter				
Algorithm	**Measures**	0.1	0.2	0.3	0.4	0.5
LP Method	ARI	0.98	0.92	0.79	0.59	0.33
	Jaccard	0.97	0.86	0.66	0.43	0.20
	F-Measure	0.99	0.96	0.89	0.78	0.62
	NMI	0.98	0.92	0.81	0.66	0.45
InfoMap	ARI	0.67	0.48	0.33	0.19	0.13
	Jaccard	0.50	0.32	0.20	0.11	0.07
	F-Measure	0.75	0.62	0.51	0.38	0.31
	NMI	0.51	0.30	0.15	0.03	0.00

Table 2. Efficiency test while increasing μ.

4.2.3. Undense Graph. We set the graph generator parameters as standard graph values changing only the average nodes degree $k_{avg} = 7$ and $n = 5000$. The results of the measures appear in Table 3. The parameters of the generated graph gives an average number of 155 communities. On average, we find the same number of communities with $minCsize = 10$ and have the best NMI score. The other measures decrease when $minCsize$ and NMI score increase. One can see the interest of using several evaluation measures. In the others tests, all measures were more or less correlated. In the case of undense graph, the choice of the minimum size of a community is important.

4.3. About practical complexity

4.3.1. Generated Graphs. Generated graphs used for our test are composed from 1000 to 250 000 nodes. The results are displayed in Table 4, and show that our algorithm performs well even at 250 000 nodes (3 min 20 s).

	ARI	Jaccard	F-Measure	NMI	$\|C\|$
LP ($minCsize = 3$)	0.68	0.51	0.76	0.56	$\simeq 260$
LP ($minCsize = 5$)	0.60	0.43	0.73	0.57	$\simeq 195$
LP ($minCsize = 10$)	0.48	0.39	0.69	0.62	$\simeq 155$
LP ($minCsize = 15$)	0.23	0.13	0.62	0.61	$\simeq 130$
InfoMap	0.59	0.42	0.64	0.37	$\simeq 265$

Table 3. Efficiency test on undense graphs.

Nodes	1000	5000	10 000	50 000	100 000	250 000
Time	2s	4s	7s	36s	1min 10s	3min 20s

Table 4. Practical time complexity in generated graphs.

4.3.2. Real Networks. The datasets[1] used are:

- *Epinions social network*: This is a who-trust-whom online social network of a a general consumer review site Epinions.com with 75 879 nodes and 508 837 arcs.
- *EU email communication network*: The network was generated using email data from a large European research institution. It contains 265 214 nodes and 420 045 arcs.

The first network is processed by our algorithm in less than 5 mins. The second one is treated in 58 minutes. The main difference in time processing can be explained by the merging operation. For Epinions, we obtain 58 026 clusters before merging, and 1572 clusters after merging. For the email network, we obtain 256 962 clusters before merging, and 174 clusters after merging. This observation is related to the results we obtain in undense graphs. The density of the email network is smaller than the Epinions network, and the choice of the size of the smallest community is important.

5. Conclusion

In this paper, we focused on communities detection using the p-connected components approach. It really gives very good results on generated graphs, and has an acceptable ratio $quality/time\ execution$. Our main contribution is a simple and efficient algorithm for detecting communities using graph strongly and unilaterally connected components properties. We validate our algorithm by several performance tests of different natures. We are aware that using artificial networks is not sufficient to fully validate our results, and we have to extend our work to real networks which have communities reference datasets. As the algorithm makes a "naive" search in the graph, we should optimize it to obtain a better complexity. Another possible issue is the choice of the parameter $minCsize$ in our method which needs to be studied. Moreover, the case of overlapping communities is not treated in this paper and should be taken in account in a future work.

References

[1] M. Newman, A.-L. Barabási, and D. J. Watts, *The Structure and Dynamics of Networks*. Princeton University Press, 2006.

[2] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, pp. 167–256, 2003.

[3] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *SCIENCE*, vol. 286, pp. 509–512, 1999.

[4] S. H. Strogatz, "Exploring complex networks," *NATURE*, vol. 410, pp. 268–276, 2001.

[5] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 74–174, February 2010.

[6] E. A. Leicht and M. E. J. Newman, "Community structure in directed networks," *Phys. Rev. Lett.*, vol. 100, no. 11, p. 118703, 2008.

[7] C. Berge, *Graphes et Hypergraphes*. Paris: Dunod, 1970.

[8] F. Harary, *Graph Theory*. Addison-Wesley, Reading, 1969.

1. source: Stanford Large Network Dataset Collection (http://snap.stanford.edu/data/)

[9] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[10] R. Diestel, *Graph Theory*, 4th ed. Springer-Verlag Berlin, July 2010.

[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *ACM conference SIGCOMM*, 1999, pp. 251–262.

[12] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 1, pp. 291–307, 1970.

[13] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[14] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *Proceedings of the National Academy of Sciences*, vol. 101, 2004.

[15] S. Fortunato, V. Latora, and M. Marchiori, "Method to find community structures based on information centrality," *Phys. Rev. E*, vol. 70, no. 5, p. 056104, 2004.

[16] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, no. 6, p. 066111, 2004.

[17] L. Donetti and M. A. Muñoz, "Improved spectral algorithm for the detection of network communities," in *Proceedings of the 8th Granada Seminar - Computational and Statistical Physics*, 2005, pp. 1–2.

[18] S. V. Dongen, "Graph clustering by flow simulation," Ph.D. dissertation, University of Utrecht, Netherlands, 2000.

[19] H. Zhou and R. Lipowsky, "Network brownian motion : A new method to measure vertex-vertex proximity and to identify communities and subcommunities." in *Proceedings of International Conference on Computational Science (ICCS)*, vol. 3038. Lecture Notes in Computer Science, 2004, pp. 1062–1069.

[20] L. Yen, D. Vanvyve, F. Wousters, F. Fouss, M. Verleysen, and M. Saerens, "Clustering using a random walk based distance measure." in *Proceedings of European Symposium on Artificial Neural Networks (ESANN)*, 2005.

[21] P. Pons and M. Latapy, "Computing communities in large networks using random walks," *Journal of Graph Algorithms and Applications*, vol. 10, no. 2, pp. 191–218, 2006.

[22] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences USA*, pp. 1118–1123, 2008.

[23] A. Lancichinetti and S. Fortunato, "Community detection algorithms: A comparative analysis," *Phys. Rev. E*, vol. 80, p. 056117, 2009.

[24] R. E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972.

[25] ——, "Enumeration of the elementary circuits of a directed graph," *SIAM J. Comput.*, vol. 2, no. 3, pp. 211–216, 1973.

[26] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Phys. Rev. E*, vol. 78, no. 4, p. 046110, 2008.

[27] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Physical Review E*, vol. 80, no. 1, pp. 1–8, 2009.

[28] L. Danon, A. Díaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 9, pp. P09 008–09 008, 2005.

[29] A. Lancichinetti, S. Fortunato, and J. Kertész, "Detecting the overlapping and hierarchical community structure in complex networks," *New Journal of Physics*, vol. 11, no. 3, p. 033015, 2009.

[30] E. Navarro and R. Cazabet, "Détection de communautés, étude comparative sur graphes réels," in *1ères Journées Modèles et l'Analyse des Réseaux : Approches Mathématiques et Informatique*, 2010.

[31] G. W. Milligan, S. C. Soon, and L. M. Sokol, "The effect of cluster size, dimensionality, and the number of clusters on recovery of true cluster structure," *Pattern Analysis and Machine Intelligence IEEE Transactions*, vol. PAMI-5, no. 1, pp. 40–47, 1983.

[32] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[33] C. J. van Rijsbergen, *Information Retrieval*, 2nd ed. London: Butterworth, 1979.

[34] B. C. M. Fung, K. Wang, and M. Ester, "Hierarchical document clustering using frequent itemsets," in *SIAM International Conference on Data Mining*, 2003.

[35] J. Madadhain, D. Fishera, P. Smyth, S. White, and Y. B. Boey, "Analysis and visualization of network data using jung," *Journal of Statistical Software*, pp. 1–25, 2005.