Scalable Partitioning of Directed Graphs Using Graphlets

Luce Le Gorrec, Philip A. Knight

Scalable Partitioning of Directed Graphs Using Graphlets

Luce le Gorrec and Philip Anthony Knight

Department of Mathematics and Statistics
University of Strathclyde, United Kingdom
{luce.le-gorrec,p.a.knight}@strath.ac.uk

Abstract. Community detection aims to partition a network into similar groups of nodes. Mainstream approaches focus on direct interactions (i.e. edges), but these are limiting for many community structures met in directed networks. Recently, analyses based on implicit interactions have been investigated. In particular, those based on graphlets have been identified as a promising tool. In this study we propose an algorithm to partition directed networks based on graphlets. Our method uses graphlets to produce an undirected representation of the network, and partitions this using the Louvain algorithm. It finally post-processes the resulting partitioning to address nodes disconnected from the undirected representation. We propose an implementation of the algorithm that is **versatile**, as it addresses a much larger set of graphlets than other existing methods. It is also **numerically efficient**, as it addresses networks with up to a few millions of nodes. Finally, it is **user-friendly**, since it provides the user with a large number of parameters and use cases that can be manually tuned, or used as default. On these three points, to the best of our knowledge, our proposed solution is the state-of-the-art.

Keywords: Community Detection · Directed Networks · Graphlets.

1 Introduction

Community detection is an essential tool in the analysis of complex networks. In essence, it groups together nodes that are similar, while separating those that are dissimilar. Community detection has proven useful in applications as diverse as predicting protein functions [8]; uncovering terrorism-related Twitter users [9]; analysing the history of mathematics [7], etc.. In the case of undirected networks, a simple and widely accepted definition of a community is that nodes within a community are densely connected, and loosely connected to the nodes outside—See Table 1.1 from [22]. Finding such communities has been an area of abundant research, and an ever increasing number of algorithms to uncover community structures have been proposed. Each makes its own interpretation of the meaning of "densely'" and/or "loosely" connected (intra-community density [16], cut [19], random walks [18], etc.), and the best way to reveal such a structure.

On the other hand, detecting communities when the network is directed is a much more sparsely populated field, with no such consensus about what a community should look like. This is because the very nature of a community is strongly application dependent. For instance, Figure 1 from [15] highlights community structures of different kinds: density-based, citation-based, and a flow-based community structures (from the left to the right). Algorithms designed to

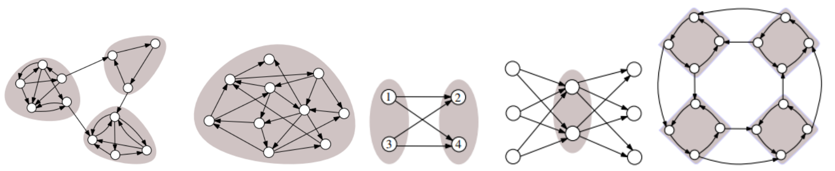


Fig. 1. Different kinds of community structures in directed networks (from [15]).

reveal communities within directed networks are thus dependent entirely on the structure of choice and a particular structure may require a bespoke method. For instance, InfoMap [18] aims to uncover flow-based community structures, while the algorithm from [6] focuses on citation-based communities. Algorithms for undirected networks have been retooled to cope with their directed counterparts by symmetrising the edges of the network adjacency matrix. These are generally better suited to uncover density-based community structures [15, Sections 4.1-2]. In summary, choosing the wrong algorithm can lead to inconsistent partitioning and different algorithms need to be used to uncover different kinds of community structures. This severely limits the utility of many existing methods.

Related Work. Recently, a framework has been proposed that builds a higher-order representations of networks based graphlets and partitions it using mainstream partitioning methods. This offers an attractive route to community detection, and has already given meaningful results in divers applications [13,20,1,21]. In large part, this is due to the fact that, with a judicious choice of graphlets, these higher-order representations should exhibit community structures which are groups of densely connected nodes. Yet, because of the novelty of this framework, there are only a few solutions that implement it, with room for improvement. Namely, the solution from [20] addresses uniquely undirected networks. Three related implementations for directed networks are proposed in [21,1,25]. They all use spectral algorithms to partition the networks, for which it is necessary to prescribe the number of clusters to find. The first cited of these can be used on all possible 3-node graphlets, and can deal with weighted networks, too. One can also use it to return the higher-order representation of the network. However, it is not suitable for large-scale networks. The two other methods can deal efficiently with all 3-node graphlets, even for very large networks. They can also address a single 4-node graphlet, known as the BiFan, but only for networks of moderate size. In both, the user has no access to the higher-order representation. None of these four methods is designed to deal with nodes that do not appear in the higher-order representation but it is not uncommon for such nodes to account for a large proportion of the node set.

Contributions. In this study, we present a solution—named MARGOT as in Motif-based pARtitioning of Graph with OrienTed edges—with three notable features. 1) **Versatility**. MARGOT addresses all 3-node and quadrangle graphlets, that is 142 graphlets overall. Moreover, it can produce interesting partial higher-order representations. 2) **Numerical efficiency**. MARGOT is parallelisable and able to address networks with millions of nodes, for both 3-node and quadrangles graphlets. 3) **User-friendliness**. MARGOT can be used end-to-end with default parameters, requiring only the path to the input graph and the destination of where to save the details of the output partitioning. But many parameters can be manually tuned to produce bespoke results.

The remainder of this paper is divided as follows: Section 2 presents definitions and concepts. These are used in Section 3 where we describe our method. In particular, Section 3.2 explores the numerical efficiency of the software. Finally, in Section 4 we use our software to analyse a real-world network.

2 Useful Objects and Concepts

2.1 Graphlets, Orbits, and Motif Adjacency Matrices

In this study, we investigate directed, unweighted graphs (also called networks), denoted by $G = (V, E)$, with $V \subset \mathbb{N}$ the set of nodes, and $E \subset V \times V$ the set of edges. Graphs are assumed to be without self-loops. Our method first builds a higher-order representation of a graph G based on a graphlet (synonymous to a motif in this study) that is called a Motif Adjacency Matrix (MAM) [1,21].

Definition 1. *A **graphlet (motif)** is a small connected graph $\mathcal{M} = (\{1, ..., k\}, H)$. A graphlet $\mathcal{M}$ with k nodes is called a k-node graphlet.*

Definition 2. *Given a network $G = (V, E)$ and a graphlet $\mathcal{M} = (\{1, ..., k\}, H)$, we say that $S \subset V$ is an **occurrence of $\mathcal{M}$ in** G if $G_S = (S, E \cap S \times S)$ is isomorphic to $\mathcal{M}$. This is denoted $S \sim \mathcal{M}$, in which case $\mathcal{I}_{\mathcal{M},G}(S)$ is the set that contains all the isomorphisms from G_S to $\mathcal{M}$.*

Graphlet occurrences are exemplified in the left panel of Figure 2.

Definition 3. *The **Motif Adjacency Matrix of the graph** $G = (V, E)$ **built on graphlet** $\mathcal{M}$ is the weighted undirected graph $G_{\mathcal{M}} = (V, E_{\mathcal{M}})$ whose adjacency matrix $\mathbf{M} \in \mathbb{R}^{|V|\times|V|}$ is defined as:*

$$\mathbf{M}(i, j) = |\{S \subset V : i, j \in S \text{ and } S \sim \mathcal{M}\}|, \forall i \neq j.$$

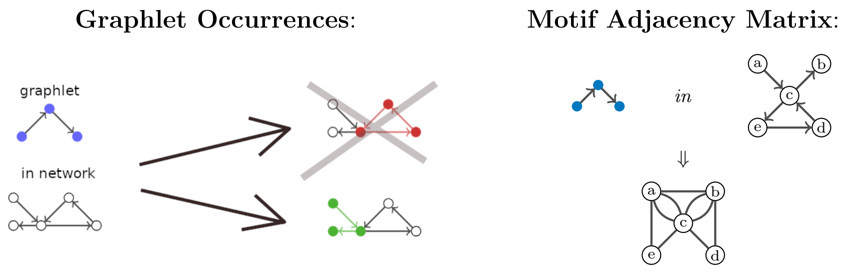


Fig. 2. Left: Red subgraph is not an occurrence of the blue graphlet in the network; Green subgraph is. **Right**: Occurrences of the blue graphlet in the network are $\{a, c, b\}$, $\{a, c, e\}$, and $\{d, c, b\}$, which result in the bottom MAM.

We note that in the literature it is common practice to use the term Motif Adjacency Matrix to denote both the adjacency matrix $\mathbf{M}$ and the associated graph $G_{\mathcal{M}}$.

In brief, the MAM of a graph $G = (V, E)$ given a graphlet $\mathcal{M}$ is an undirected weighted graph, built on V, and in which an edge between two nodes specifies the number of times these nodes appear together in an occurrence of $\mathcal{M}$. An example is provided in the right panel of Figure 2. Alternatively, a MAM can be defined along with a set of anchors [1], a subset of the graphlet set of nodes. When two nodes appear together in a graphlet occurrence, an edge is drawn in the MAM if these nodes can be mapped with nodes in the anchor set by an isomorphism. In our solution, anchors are related to the so-called orbits. Definitions are provided below, and an example of a MAM with anchors is provided in Figure 3.

Definition 4. *Given a k-node graphlet $\mathcal{M}$, we say that $u, v \in \{1, ..., k\}$ have the same* ***orbit*** *if there exists $\sigma \in \mathcal{I}_{\mathcal{M},\mathcal{M}}(\{1, ..., k\})$ s.t. $v = \sigma(u)$. This is an equivalence relation on $\{1, ..., k\}$, and we call* ***the orbits of*** *$\mathcal{M}$ a set of equivalence classes representatives, denoted $\mathcal{O}(\mathcal{M})$.*

Definition 5. *The* ***MAM of*** *G* ***built on*** *$\mathcal{M}$* ***with anchors*** *$\mathcal{A} \subset \mathcal{O}(\mathcal{M})$ is the graph $G^{\mathcal{A}}_{\mathcal{M}} = (V, E_{\mathcal{M}})$ whose adjacency matrix $\mathbf{M} \in \mathbb{R}^{|V|\times|V|}$ is:*

$$\mathbf{M}(i, j) = |\{S \subset V : i, j \in S \text{ and } \exists\sigma \in \mathcal{I}_{\mathcal{M},G}(S) : \sigma(i), \sigma(j) \in \mathcal{A}\}|, \forall i \neq j$$

We observe that a MAM with an anchor set that covers the orbit set is equivalent to a MAM without anchor set as defined in Definition 3.

2.2 Modularity and The Louvain Algorithm.

In our method, we use the Louvain algorithm [2] to partition MAMs. Recent studies show that it is still one of the most accurate and computationally efficient in discovering community structures within (weighted) undirected net-

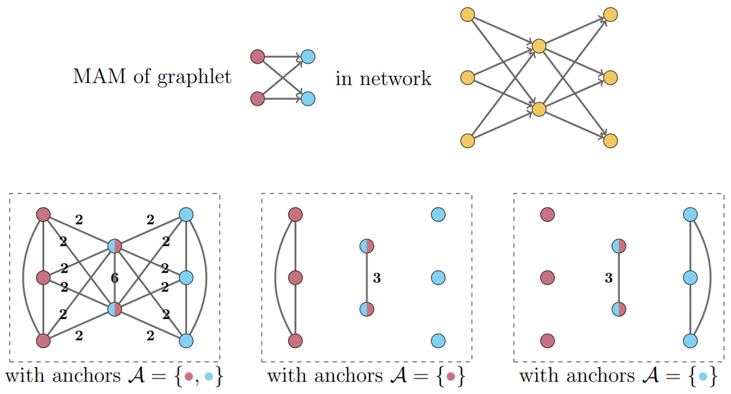

MAM of graphlet
in network
with anchors $\mathcal{A} = \{●, ●\}$
with anchors $\mathcal{A} = \{●\}$
with anchors $\mathcal{A} = \{●\}$

Fig. 3. MAMs of the yellow network built on a graphlet with 2 orbits (blue and red).

works [24,5]. The Louvain algorithm aims to maximise the so-called modularity [16], which measures the consistency of a community structure on a network, as defined below.

Definition 6. *Given* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *the adjacency matrix of a weighted undirected graph* G *and a community structure* $\mathcal{C}$ *on* G *(that is a partitioning of* V*),* ***the modularity of*** $\mathcal{C}$ ***on*** G *is measured by*

$$\mathcal{Q}_\gamma(\mathbf{A}, \mathcal{C}) = \frac{1}{2m} \sum_{C \in \mathcal{C}} \sum_{u,v \in C} (a_{u,v} - \gamma \frac{k_u k_v}{2m}), \tag{1}$$

where $k_v = \sum_{w=1}^{n} a_{v,w}$ *is the (weighted) degree of node* v*,* $m = \frac{1}{2} \sum_{v=1}^{n} k_v$ *counts the edges, and* γ *is the so-called* ***resolution parameter*** *that helps to prevent the resolution limit [17].*

The Louvain algorithm aims to maximise the modularity via a multi-level heuristic. Starting with the trivial community structure in which each node has its own community, it follows an iterative 2-step scheme. 1) Each node is visited in turn and assigned to the community that produces the maximum gain of modularity, until no gain can be achieved by moving one node; 2) A meta-graph is built by merging nodes that belong to a same community into meta-nodes. These two steps are repeated until the obtained meta-graph cannot be simplified. It is possible to run Louvain from some initial partitioning: the meta-graph corresponding to this partitioning is first built, and the process is then run on this. We will use this idea in our post-processing step, to assign a community to nodes disconnected from the MAM.

3 Our Solution

In this section, we describe our proposed method and analyse its complexity.

3.1 The Algorithm

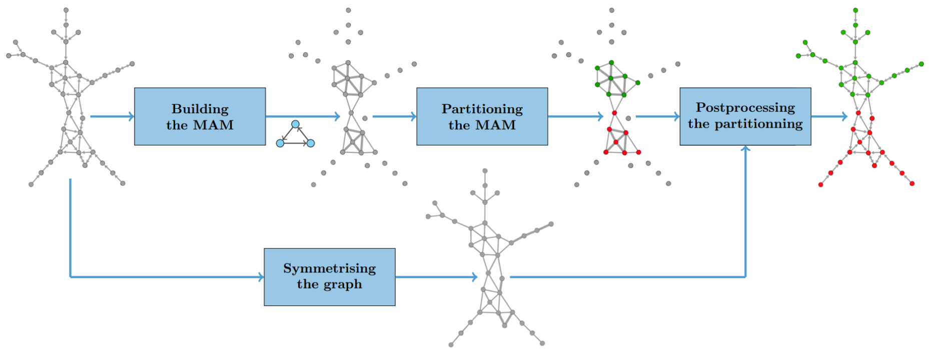
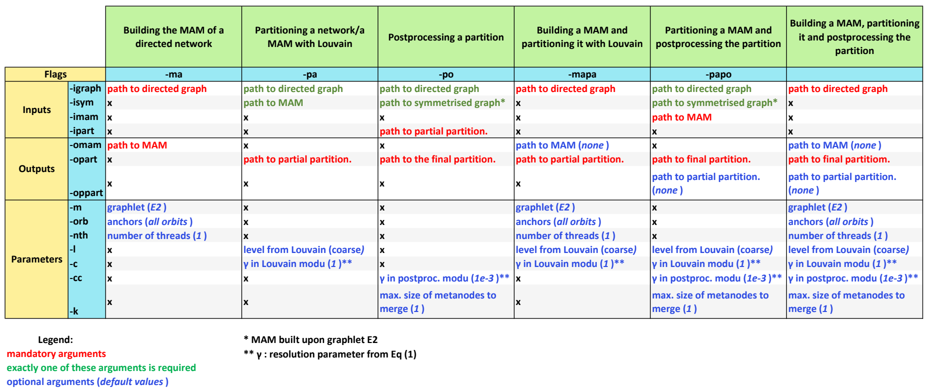


Fig. 4. Partitioning of the network using the 3-node loop. Node colours specify the community structure.

Given a directed graph and a graphlet, our algorithm performs the following steps: 1) The MAM of the graph is built based on the given graphlet; 2) The MAM is partitioned using the Louvain algorithm; 3) Nodes that are disconnected from the MAM are assigned to a community. This pipeline is illustrated in Figure 4. The three following paragraphs describe these stages. Our implementation enables each stage to be run independently or all together. Parameters can be tuned or used as default. The table in Figure 5 summarises this.



		Building the MAM of a directed network	Partitioning a network/a MAM with Louvain	Postprocessing a partition	Building a MAM and partitioning it with Louvain	Partitioning a MAM and postprocessing the partition	Building a MAM, partitioning it and postprocessing the partition
Flags		-ma	-pa	-po	-mapa	-papo	
Inputs	-igraph	path to directed graph	path to directed graph	path to directed graph	path to directed graph	path to directed graph	path to directed graph
	-isym	x	path to MAM	path to symmetrised graph*	x	path to symmetrised graph*	x
	-imam	x	x	x	x	path to MAM	x
	-ipart	x	x	path to partial partition.	x	x	x
Outputs	-omam	path to MAM	x	x	path to MAM (*none*)	x	path to MAM (*none*)
	-opart	x	path to partial partition.	path to the final partition.	path to partial partition.	path to final partition.	path to final partitiom.
	-oppart	x	x	x	x	path to partial partition. (*none*)	path to partial partition. (*none*)
Parameters	-m	graphlet (*E2*)	x	x	graphlet (*E2*)	x	graphlet (*E2*)
	-orb	anchors (*all orbits*)	x	x	anchors (*all orbits*)	x	anchors (*all orbits*)
	-nth	number of threads (*1*)	x	x	number of threads (*1*)	x	number of threads (*1*)
	-l	x	level from Louvain (coarse*)*	x	level from Louvain (coarse*)*	level from Louvain (coarse*)*	level from Louvain (coarse*)*
	-c	x	γ in Louvain modu (*1*)**	x	γ in Louvain modu (*1*)**	γ in Louvain modu (*1*)**	γ in Louvain modu (*1*)**
	-cc	x	x	γ in postproc. modu (*1e-3*)**	x	γ in postproc. modu (*1e-3*)**	γ in postproc. modu (*1e-3*)**
	-k	x	x	max. size of metanodes to merge (*1*)	x	max. size of metanodes to merge (*1*)	max. size of metanodes to merge (*1*)

Legend:
mandatory arguments
exactly one of these arguments is required
optional arguments (*default values*)

* MAM built upon graphlet E2
** γ : resolution parameter from Eq (1)

Fig. 5. Summary of the possible software use cases and parameters with default values.

Building Motif Adjacency Matrices. MAMs are extremely useful since they transform directed networks into weighted undirected networks that highlight desired structural properties, as illustrated in Figure 6. Here, a directed network is built

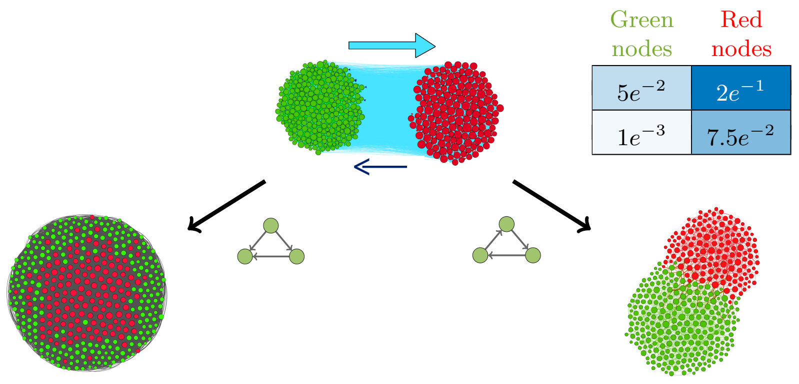


Fig. 6. A network from a SBM and two of its MAMs (graphlets in green). MAM visualisations are obtained using the Force Atlas algorithm from Gephi.

using the stochastic block model (SBM) shown at the top right panel. Two kinds of structures could be of interest: the core-periphery structure with red nodes as core and green nodes as periphery, and the two-block structure that separates red from green nodes. For both targets, it is possible to pick a graphlet whose associated MAM exhibits the desired structure, as illustrated in the bottom part of the figure. But building the MAM is the bottleneck for our method, and more generally for the framework evoked in Section 1, in terms of scalability [1]. The complexity of methods for enumerating graphlet occurrences generally grows with the size of graphlets. Thus, we limit ourselves to 3-node graphlets, and quadrangles which are 4-node graphlets containing a 4-edge undirected cycle. Indeed, two efficient algorithms to enumerate respectively closed 3-node graphlets (called triangles) and quadrangles in undirected graphs were proposed in [3]. When the graph is connected, they are both theoretically proven to be upper-bounded by the same time complexity, namely $O(m^{1.5})$ with m the number of edges. As this is the best complexity we are aware of for the enumeration of occurrences of such graphlets, we have adapted these algorithms to directed networks. For open 3-node graphlets (called wedges), we use the algorithm proposed in [1]. The graph datastructure used to implement those algorithms is from SNAP [14]. Furthermore, all of these algorithms can parallelised straightforwardly in a manner that resembles the Node-Parallel version of [4]. We have done this using OpenMP. Finally, for all the different graphlets addressed in the software, we adapted the implemented methods so that orbit-based anchored MAMs can be built.

In the software, the graphlet used to build a MAM is specified by the parameter `-m`, and orbits to use as anchors by `-orb`. The list of all graphlet identifiers with their orbits are listed in the file *GraphletIdentifiersWithOrbits.pdf*. Using the default value `-m E2`, the representation of the input network G produced by the software is the undirected, weighted graph with adjacency matrix $\mathbf{A} + \mathbf{A}^T$, where $\mathbf{A}$ is the adjacency matrix of G.

Partitioning. After the first stage of the algorithm we have built a MAM that represents an undirected weighted graph having the same set of nodes as the input network. The second stage partitions this MAM using the Louvain algorithm. The modularity function that Louvain aims to maximise suffers from the so-called resolution limit that can cause it to overlook well-defined communities if they are too small [17]. To circumvent this, the resolution parameter γ from Definition 6 can be manually tuned in our implementation by means of the argument `-c`. Another way to avoid enormous communities is to force Louvain to stop before its final pass, which can be done in our implementation using the argument `-l`. Our implementation is adapted from the Louvain software from `https://sourceforge.net/projects/louvain/files/GenericLouvain/`.

Finally, note that the MAMs returned by the software are textual edge lists: a text file in which each a row indicates an edge by three numbers, namely its source node, target node, and its weight. This is a classical format for graphs, and it is thus possible to use other algorithms to partition the MAMs, independently from our software. Our post-processing can then been applied using the partitioning at hand.

Post-Processing. It is clear from Definition 3 that any nodes from the input graph which are not involved in any graphlet occurrence will be disconnected from the MAM. Such nodes can account for a large part of the node set, and it may be important to assign a community to them afterwards. To this end, we apply a home-brewed constrained version of Louvain on the input network symmetrised as for `-m E2`, with the partial partitioning obtained after stage 2 as a starting point. A meta-graph is built based on the input symmetrised graph and the input partial partitioning: nodes that belong to the same community are merged together to form meta-nodes. There is also one meta-node for each node disconnected from the MAM, so that all the nodes from the initial graph are taken into account in this meta-graph. Louvain is then applied on this meta-graph, but with additional constraints to prevent mergers of some of the initial partitions. Namely, the initial partitions are split onto two groups: those that can be merged (free partitions), and those that cannot (constrained partitions). The Louvain algorithm is then modified as follows: 1) In the modularity maximisation procedure only the free meta-nodes (partitions) are visited, and they can only be merged within constrained partitions; 2) When new meta-graphs are built, the new meta-nodes are considered free only if all the nodes (partitions) that they contain are free. From a software perspective, whether a partition is free or constrained in the initial partitioning is dependent on the number of nodes that it contains. This can be tuned using the parameter `-k`. Using the default parameter, free partitions are disconnected nodes (partitions that contain at most one node).

Furthermore, since the purpose of this stage is to assign disconnected nodes to existing communities, the modularity to maximise in this constrained approach is tuned with a very low default resolution parameter (10^{-3}). This can be changed using the argument `-cc`.

3.2 Resource Consumption

Here, n is the number of nodes in the graph, m is the number of edges, and k_v is the (undirected) degree of a node v. Graphlets are denoted by their identifiers as defined in [10].

Theoretical run-time complexity of building a MAM. The algorithms proposed in [3] to list triangles and quadrangles in undirected networks have a proven theoretical upper-bound of run-time complexity equal to $O(m^{1.5})$ with ideal data structures. When extended to directed networks and using the SNAP data structure, the complexity becomes $O(\sum_v k_v^2)$. To build MAMs of wedges, we implemented the method proposed in [1], which also has complexity $O(\sum_v k_v^2)$.

Theoretical run-time complexity of Louvain. A fine bound for the complexity of the Louvain algorithm is unrealistic since the sizes of objects (number of communities, number of (meta-)nodes in the graph) vary greatly from iteration to iteration. An upper-bound proposed in [5] is $O(n\log(n))$, where the authors also find that in practice Louvain is the quickest algorithm from their benchmark, even though others have lower upper-bounds.

Practical resource consumption. To assess the scalability of the implemented method, we first run it on a benchmark of randomly generated networks with average degree 10 where the number of nodes varies from 100K to 1.6M. The experiments were run on a desktop with an I9-7060X processor with 32 logical cores and 128GB RAM. We terminate any process that needs more than 3 hours of run-time. In Figure 7, we record the elapsed time and maximum resident set size taken to build a MAM given a 3-node graphlet (left) or a quadrangle (right) averaged over 2 random networks and a subset of graphlets. We observe that it takes a bit longer to compute MAMs for wedges than for triangles but the memory consumption is the same. For 3-node graphlets, MAMs of largest networks can be built in less than 2 minutes with only one thread. For both 3-node and quadrangle graphlets we note the substantial reduction in elapsed time provided by a parallelisation using up to 8 threads: with only one thread (blue curve) it is not possible to compute quadrangle-based MAM of the largest graphs within the time limit. For more than 8 threads, the elapsed time is not reduced, though the memory used to build quadrangle based MAMs is increased.

It is well known that the performance of algorithms on random networks may be unrepresentative of real-world performance. Thus we have assessed our method on five large real-world networks from `https://networks.skewed.de/` whose statistics are provided in Table 1. Elapsed time and memory consumption for building MAMs are summarised in Figure 8. Processes were killed if unfinished within 2 hours. We observe two substantial differences from random

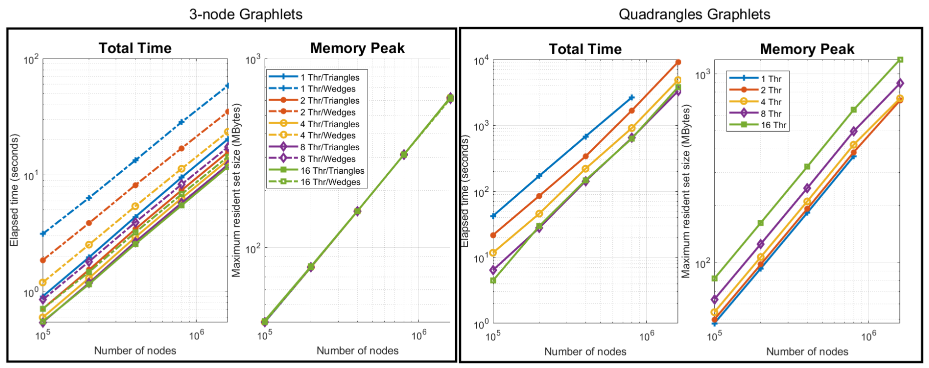


Fig. 7. Total elapsed time and maximum resident set size to build MAMs.

Network name	Number of nodes	Number of edges
Dallas_Fortworth_Arlingto	187K	466K
Stanford_Web	282K	2.3M
MySpace_Aminer	853K	5.6M
WikiTalk_fr	1.4M	2.4M
Trec_Web	1.6M	8.1M

Table 1. Statistics of real-world networks.

networks. First, it takes much longer to build MAMs for wedges. This is related to the high frequency of these graphlets. For Trec_web and Stanford_Web, it takes more than 1GB of RAM to build the T74 MAM. Second, we observe that the memory consumption distribution is uneven and can reach high values. Three networks require more than 20GB of RAM to build MAMs on Q204 (up to 50GB for the largest network). Apart from these specific differences, the method behaves similarly for real-world and random networks. Elapsed times for graphlets other than wedges are comparable, and using several threads cuts the run-time deeply. Figure 9 shows how the elapsed time of the whole process is distributed between the three stages of the method on real-world networks using 8 threads to build MAMs. The partitioning and post-processing steps were run 5 times for each case, and the time shown here is averaged. We observed that when our method is applied on quadrangles and on wedges generally more than half of the total time is spent building the MAM.

Finally, we compare the complexity of building MAMs in MARGOT using 8 threads, and in the solutions proposed in [21] (denoted ORFE) and in [1,25] (denoted SNAP). We use the Python version of ORFE, and SNAP is implemented in C++, as is our solution. Ours is the only parallel method. The times taken into account include reading the graph, building the MAM, and writing it into a text file. Elapsed times and maximum resident set sizes are indicated in Table 2 for some of the random and real-world networks presented earlier. To push the methods to their limits, wedges are chosen among those that usually produce very dense MAMs in real-world networks. The quadrangle tested is the only one for which SNAP is capable of building a network's MAM. Apart for graphlet

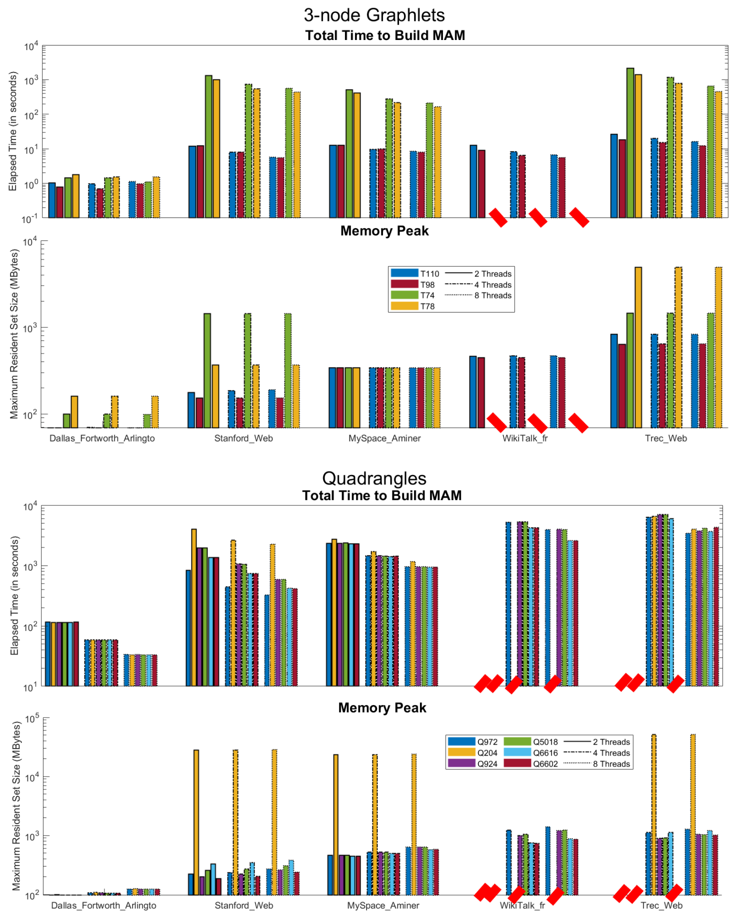


Fig. 8. Elapsed time and memory consumption for real-world networks, for 3-node graphlets (top) and quadrangles (bottom). Killed processes are highlighted with a red slash on the x-axis.

in Trec_Web, MARGOT always outperforms the two other solutions in terms of elapsed time, significantly for the most complex cases. It is also the only one which can return the MAM for every pair network/graphlet tested here. On the other hand, it may need a huge amount of RAM (up to 107GB for on Trec_Web). Generally though, it tends to use less RAM than ORFE, and more than SNAP. Lastly we remark that while our solution uses 8 threads, it is a long way from being 8 times faster than SNAP for 3-node graphlets. We hypothesise that this is due to our node-oriented parallelisation which may quickly lead to several idling threads as observed in [4].

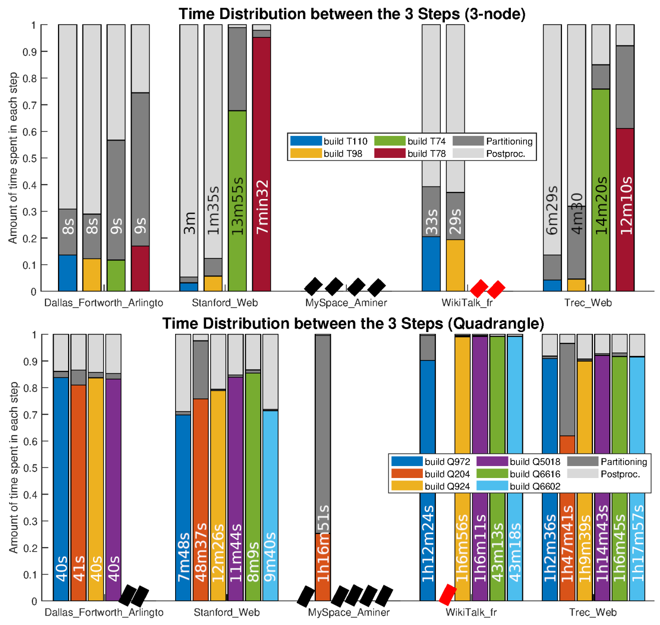


Fig. 9. Proportion of time spent in each step. A black slash on the x-axis indicates that a graphlet does not occur in a network. Total times are indicated by the bars.

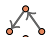
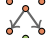
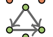
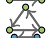
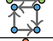
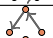
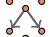
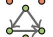
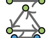
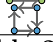
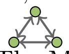
		Random Networks						Real-World Networks						
		Elapsed time			Max. Res. Set Size			Elapsed time			Max. Res. Set Size			
		Margot	Orfe	Snap	Margot	Orfe	Snap	Margot	Orfe	Snap	Margot	Orfe	Snap	
100K		3s	15s	3s	0.39	0.36	0.11	3m29s	◇	9m18s	0.86	◇	0.27	Stanford_Web
		2s	10s	2s	0.24	0.29	0.08	3m14s	◇	7m8s	0.58	◇	0.19	
		<1s	4s	<1s	0.04	0.15	0.03	3s	◇	7s	0.32	◇	0.12	
		<1s	4s	<1s	0.04	0.15	0.03	2s	15s	5s	0.15	0.34	0.12	
		3s	♣	1h23m	0.06	♣	0.03	23m33s	♣	†	25	♣	†	
1.6M		49s	3m53s	55s	5.8	4.2	1.7	3m48s	2m29s	16m19s	2.1	58	0.77	Trec_Web
		34s	2m40s	40s	3.6	2.6	1.2	17m13s	1h	18m50s	107	82	27	
		7s	59s	10s	0.59	1.6	0.48	9s	2m1s	12s	1.1	58	0.48	
		7s	51s	10s	0.59	1.6	0.48	5s	49s	7s	0.62	0.7	0.47	
		52m50s	♣	†	0.89	♣	†	47m51s	♣	†	45	♣	†	

Table 2. Elapsed time and Memory peak (in GB) to build and save MAMs. Legend: ♣: Not implemented. ◇: Runtime error: "nnz too large". †: Killed after 2hours.

4 Analysis of a citation network

Here we use our software to analyse a citation network that contains 27,770 manuscripts published on arXiv between January 1992 and April 2003 whose main category is High Energy Physics Theory. An edge from manuscript a to manuscript b indicates that a cites b. We downloaded this network (`cit-HepTh`) from `https://snap.stanford.edu/data/`. Nodes are labelled via their arXiv identifiers, enabling calls to arXiv API.

4.1 An Unexpected Graphlet

In a citation network, edges are expected to point backwards in time. But if a was submitted before b an edge could point the other way if: 1) a was revised, and its last revision occurs after b was published; 2) Authors of a are aware of b before its submission, e.g. they are also authors of b, or collaborators of b's authors. Given this, occurrences of graphlet T238 =  are highly unexpected. Yet, 36 papers are involved in T238 occurrences. The MAM they form is provided in Figure 10. It is composed of 11 cliques. A deeper investigation highlights that

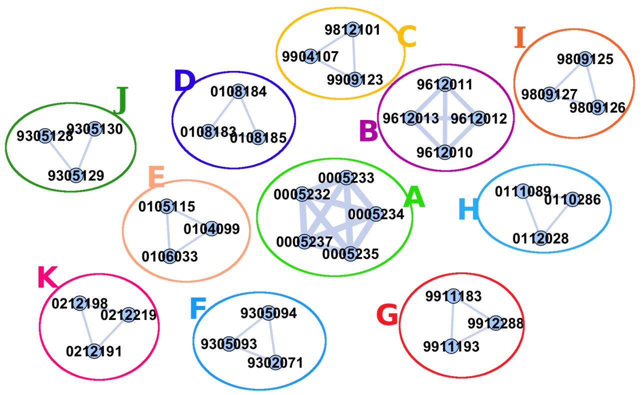
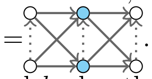


Fig. 10. MAM of `cit-HepTh` given graphlet T238.

cliques A, B, C, D, I, J and K are due to co-authorship among the clique papers. Cliques E and F are due to collaborations (visits and exchanges mentioned in acknowledgements), and also co-authorship for F. Cliques G and H are due partly to co-authorship and partly to revisions (discussions about other papers from the cliques added in the article bodies after the first submission).

4.2 A Consistent and Explicable Partitioning of the Network

We use our software to find a consistent partitioning of `cit-HepTh`. In a citation network, two papers sharing lots of in- and out-neighbours are expected to be similar. Hence, the software is used to build a MAM-like graph based on graphlet $\mathcal{M}$ = . The weight of an edge between a and b is the number of times a and b play the role of the blue nodes in a subgraph isomorphic to a 6-node graphlet that contains $\mathcal{M}^1$. Only edges with weight greater than 30 are kept. The largest connected component of this graph is then partitioned within MARGOT, resulting in the partitioning $\mathcal{P}_M$. We compare this to the partitioning obtained by applying Louvain directly on the subgraph of `cit-HepTh` induced by this largest component, which we denote $\mathcal{P}_L$. We first observe that $\mathcal{P}_M$ is much more consistent than $\mathcal{P}_L$: the modularity values being respectively 0.86 and 0.62. We also use the Φ matrix from [11] to observe the clusterings: $\Phi(C, K)$ is the average level to which nodes from community C belong to community K. From Figure 11-left, we see that intra-values of Φ for $\mathcal{P}_M$ are always greater than those from $\mathcal{P}_L$, while inter-values are generally lower. This confirms that $\mathcal{P}_M$ is more consistent than $\mathcal{P}_L$ in terms of graph structure. We see from Figure 11-right that $\mathcal{P}_M$ is essentially a refinement of $\mathcal{P}_L$.

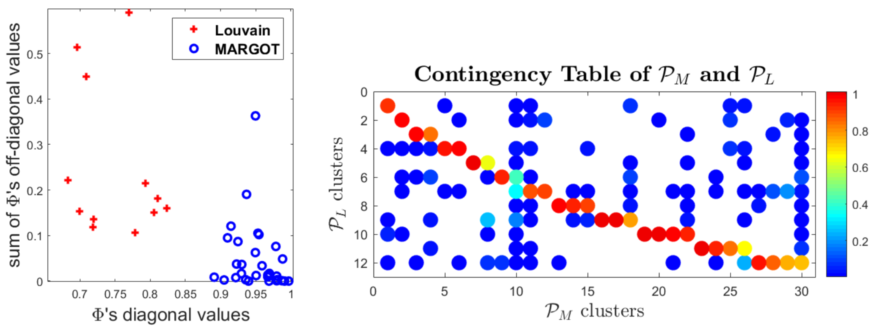


Fig. 11. Left: Φ's diagonal versus off-diagonal values (summed over rows), for each cluster from $\mathcal{P}_M$ and $\mathcal{P}_L$. Right: Contingency table whose columns are normalised.

We also find that both $\mathcal{P}_M$ and $\mathcal{P}_L$ can be accurately recovered using papers' lists of authors. Namely, given $\mathcal{P} = \mathcal{P}_M$ or $\mathcal{P}_L$, we create a new partitioning $\mathcal{P}^*$ by assigning a paper x to the cluster C^* such that

$$C^* = \underset{C \in \mathcal{P}}{argmax}(\underset{a \in \mathcal{A}(x)}{max} P(x \in C | a \in \mathcal{A}(x))),$$

with $\mathcal{A}(x)$ the list of x's authors, and $P(x \in C | a \in \mathcal{A}(x))) = \dfrac{|\{y \in C : a \in \mathcal{A}(y)\}|}{|\{y : a \in \mathcal{A}(y)\}|}$ from Bayes' theorem. The adjusted mutual information (AMI) [23] is equal to 0.49 when $\mathcal{P} = \mathcal{P}_M$ and 0.47 when $\mathcal{P} = \mathcal{P}_L$ (or respectively 0.43 and 0.41 if we remove authors who wrote a unique paper). In any case these values are far enough from 0, which means that $\mathcal{P}^*$ is globally close to $\mathcal{P}$. If we use key-words from papers' title and abstract instead of authors, the partitionings are even closer: AMI are equal to 0.76 and 0.75 for $\mathcal{P}_M$ and $\mathcal{P}_L$ respectively (0.6 for both by removing words that appear in a unique paper). So authors and

[1] Indicated edges must exist in the given direction only, or not at all for dotted ones.

key-words explain the observed partitionings. Thus, analysing groups of authors and key-words responsible for those clusters should provide insights about the dataset.

5 Conclusions

In this study we presented our solution for detecting communities within large directed networks. It relies upon a recently proposed framework that consists in partitioning the MAM of an input network given a graphlet [1,21,20], and which can address several kinds of pattern-based communities, with judicious choice of graphlets. One novelty of our work lies in the range of graphlets that it addresses. It is also more efficient than existing solutions, in terms of time complexity, for building MAMs—which is the bottleneck of this framework. It is also the first implementation to ensure that all nodes from the initial network are assigned to a community. Finally, our software provides a wide range of tuneable parameters and set-ups, giving the user the freedom to use all or parts of our solution. Using our software to analyse a citation network, we were able to discover interesting patterns, and to refine partitionings returned by another algorithm, thus illustrating how useful this can be in analysing real-world datasets.

We envisage three major improvements to our current solution, one being current work-in-progress: 1) We are working to decorrelate the anchor set from the orbit set. Indeed, there are some cases in which this correlation can be limiting, for instance in finding the bipartite sets in a (nearly) bipartite network [12]; 2) Our parallelisation is not optimal and we would like to improve it by using an edge-oriented approach, as proposed in [4]; 3) Our current solution partitions the MAM using Louvain algorithm, but is designed for other partitioning methods to be easily used instead. We would like to include some other partitioning algorithms directly within the software, enabling the user to choose among a range of techniques that may be more suitable. Finally, the range of addressed graphlets enabled by our solution leads to the difficulty of choosing the most suitable one(s). We are currently working on providing guidance to help practitioners in this choice, notably by automatically discovering graphlets able to produce suitable partitionings.

MARGOT is available at `https://github.com/acaen/MARGOT`.

References

1. Benson, A.R.: Tools for higher-order network analysis. Ph.D. thesis, Stanford University (2017)
2. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment **2008**(10), P10008 (2008)
3. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM Journal on computing **14**(1), 210–223 (1985)
4. Danisch, M., Balalau, O., Sozio, M.: Listing k-cliques in sparse real-world graphs. In: Proceedings of the 2018 World Wide Web Conference. pp. 589–598 (2018)

5. Dao, V.L., Bothorel, C., Lenca, P.: Community structure: A comparative evaluation of community detection methods. Network Science **8**(1), 1–41 (2020)
6. Gamgne Domgue, F., Tsopze, N., Ndoundam, R.: Community structure extraction in directed network using triads. International Journal of General Systems **49**(8), 819–842 (2020)
7. Gargiulo, F., Caen, A., Lambiotte, R., Carletti, T.: The classical origin of modern mathematics. EPJ Data Science **5**(1), 26 (2016)
8. Gaugain, C., Barriot, R., Fichant, G., Quentin, Y.: Classification of abc transporters using community detection. In: Classification as a Tool for Research, pp. 501–508. Springer (2010)
9. Gialampoukidis, I., Kalpakis, G., Tsikrika, T., Papadopoulos, S., Vrochidis, S., Kompatsiaris, I.: Detection of terrorism-related twitter communities using centrality scores. In: Proceedings of the 2nd international workshop on multimedia forensics and security. pp. 21–25 (2017)
10. le Gorrec, L., Knight, P.A., Caen, A.: Learning network embeddings using small graphlets. Social Network Analysis and Mining **12**(1), 1–21 (2022)
11. le Gorrec, L., Mouysset, S., Ruiz, D.: Doubly-Stochastic Scaling Unifies Community Detection (Apr 2022), `https://hal.archives-ouvertes.fr/hal-03633062`, working paper or preprint
12. Holme, P., Liljeros, F., Edling, C.R., Kim, B.J.: Network bipartivity. Physical Review E **68**(5), 056107 (2003)
13. Klymko, C., Gleich, D., Kolda, T.G.: Using triangles to improve community detection in directed networks. arXiv preprint arXiv:1404.5874 (2014)
14. Leskovec, J., Sosič, R.: Snap: A general-purpose network analysis and graph-mining library. ACM Transactions on Intelligent Systems and Technology (TIST) **8**(1), 1 (2016)
15. Malliaros, F.D., Vazirgiannis, M.: Clustering and community detection in directed networks: A survey. Physics reports **533**(4), 95–142 (2013)
16. Newman, M.E.: Analysis of weighted networks. Physical review E **70**(5), 056131 (2004)
17. Reichardt, J., Bornholdt, S.: Statistical mechanics of community detection. Physical review E **74**(1), 016110 (2006)
18. Rosvall, M., Axelsson, D., Bergstrom, C.T.: The map equation. The European Physical Journal Special Topics **178**(1), 13–23 (2009)
19. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on pattern analysis and machine intelligence **22**(8), 888–905 (2000)
20. Tsourakakis, C.E., Pachocki, J., Mitzenmacher, M.: Scalable motif-aware graph clustering. In: Proceedings of the 26th International Conference on World Wide Web. pp. 1451–1460 (2017)
21. Underwood, W.G., Elliott, A., Cucuringu, M.: Motif-based spectral clustering of weighted directed networks. Applied Network Science **5**(1), 1–41 (2020)
22. Veldt, L.N.: Optimization Frameworks for Graph Clustering. Ph.D. thesis, Purdue University Graduate School (2019)
23. Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. The Journal of Machine Learning Research **11**, 2837–2854 (2010)
24. Yang, Z., Algesheimer, R., Tessone, C.J.: A comparative analysis of community detection algorithms on artificial networks. Scientific reports **6**(1), 1–18 (2016)
25. Yin, H., Benson, A.R., Leskovec, J., Gleich, D.F.: Local higher-order graph clustering. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 555–564 (2017)