Parallel k-Clique Community Detection on Large-Scale Networks

Enrico Gregori, *Member*, *IEEE*, Luciano Lenzini, *Member*, *IEEE*, and Simone Mainardi

Abstract—The analysis of real-world complex networks has been the focus of recent research. Detecting communities helps in uncovering their structural and functional organization. Valuable insight can be obtained by analyzing the dense, overlapping, and highly interwoven k-clique communities. However, their detection is challenging due to extensive memory requirements and execution time. In this paper, we present a novel, parallel k-clique community detection method, based on an innovative technique which enables connected components of a network to be obtained from those of its subnetworks. The novel method has an unbounded, user-configurable, and input-independent maximum degree of parallelism, and hence is able to make full use of computational resources. Theoretical tight upper bounds on its worst case time and space complexities are given as well. Experiments on real-world networks such as the Internet and the World Wide Web confirmed the almost optimal use of parallelism (i.e., a linear speedup). Comparisons with other state-of-the-art k-clique community detection methods show dramatic reductions in execution time and memory footprint. An open-source implementation of the method is also made publicly available.

Index Terms—k-clique communities, parallel community detection method

1 Introduction

Complex networks—in the sense in which they are used in this paper—are essentially graphs modeling real-world complex systems. Detecting communities from these networks may be decisive in the understanding of their structural and functional properties [1], [2]. Examples include, but are not limited to, the Internet [3], [4] and the World Wide Web [5] as well as mobile phone [6], collaboration [7], citation [8], and biological [7] networks.

Specifically, the Internet topology at the Autonomous System (AS)[1] level has several zones which are extremely rich of connections. A study of such well interconnected zones can be helpful in the understanding of the complex dynamics at play in the business realm of the Internet. These dynamics can be to some extent inferred from the connections between ASes since they are the natural consequences of meticulous market strategies pursued by a myriad of interplaying forces. In one of our first analyses, we studied the cliques of ASes as we believe they represent the most tight concept of community—all possible pairs of ASes in a clique are *interacting* each other. In addition, it has been shown in [9] that the main properties of networks may be viewed as consequences of their underlying clique structure. We found that cliques of ASes were large in number, big-sized and highly overlapped. Leveraging on this result, we decided to adopt the k-clique community [2] definition to investigate the structure and the properties of the Internet at the AS level. The rationale behind this choice is that k-clique communities are unions of cliques well-interwoven and reachable each other through paths involving other cliques only. To the best of our knowledge, that of k-clique community is the only definition which is based on the concept of clique and at the same time:

- *Is formally defined*, i.e., is based on topological properties and uses neither heuristics nor function optimizations;
- *Is totally deterministic*, i.e., has no stochastic elements embedded and communities are not execution-dependent;
- *Allows overlap*, i.e., communities can be partially or also almost completely superimposed;
- *Is local*, i.e., each community exists independently of the other communities.

We used the k-clique community definition to analyze the Internet from a new perspective [4]. Nevertheless, this definition was successfully applied also in other areas. For example, in [10] Hui and Crowcroft identify k-clique communities among the participants of Infocom06 and the students in the MIT Media Laboratory and exploit this information to design efficient forwarding algorithms for mobile networks. Similarly, in [11] Hui et al. propose a distributed k-clique community detection method to be used for *social-based* message forwarding. k-clique communities also find application in social sciences. For example,

[1] At the AS level, the Internet topology consists of Internet service providers and Internet users (each one corresponding to an AS), interconnected through Border Gateway Protocol sessions, with the aim of exchanging traffic.

- *E. Gregori is with the Institute of Informatics and Telematics (IIT), Italian National Research Council (CNR), via Moruzzi 1, Pisa 56124, Italy. E-mail: enrico.gregori@iit.cnr.it.*
- *L. Lenzini is with the Department of Information Engineering (IET), University of Pisa, via Diotisalvi 2, Pisa 56122, Italy. E-mail: luciano.lenzini@iet.unipi.it.*
- *S. Mainardi is with the Institute of Informatics and Telematics (IIT), Italian National Research Council (CNR), via Moruzzi 1, Pisa 56124, and the Department of Information Engineering (IET), University of Pisa, via Diotisalvi 2, Pisa 56122, Italy. E-mail: simone.mainardi@iet.unipi.it.*

Manuscript received 30 Mar. 2012; revised 29 June 2012; accepted 12 July 2012; published online 25 July 2012.
Recommended for acceptance by J. Zhang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2012-03-0333.
Digital Object Identifier no. 10.1109/TPDS.2012.229.

1045-9219/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

in [12] they are used to capture the relationships characterizing the collaboration between scientists and the calls between mobile phone users.

The first method for extracting k-clique communities is the Clique Percolation Method (CPM) [13], which is prohibitively memory and time expensive. To the best of our knowledge, this had prevented k-clique communities from being extracted from large-scale networks such as those considered here. In this paper, we address CPM scalability issues and present the novel CPM On Steroids (COS). COS is the refinement of a working proof-of-concept, which is presented first to clarify the problem of parallel k-clique community detection. COS exploits parallel processing to reduce execution time and has a low memory footprint. Its maximum degree of parallelism, unbounded, user-configurable and input-independent, enables hardware resources to be used efficiently. In addition, we provide analytical tight upper bounds on its execution time and space requirements, which are given as function of: 1) the number of maximal cliques in the network; 2) the size of the maximal cliques; and 3) the number of processors available. These bounds prove that COS has a *linear* space dependence on the number of maximal cliques and a worst case execution time inversely proportional to the number of processors. By means of the aforesaid bounds we can answer questions such as "Is memory available on this hardware enough to extract k-clique communities from this network?" or "If the number of processors installed on a particular machine is doubled, would COS halve its execution time?". Therefore, we are providing a framework with which it is possible not only to extract k-clique communities efficiently, but also to estimate in advance the required amount of computing resources. These theoretical bounds are validated in a series of experiments. We experimentally measured a *linear speedup*: COS execution time halves when the number of processors it uses is doubled. Dramatic reductions in execution time and memory footprint are brought to light by comparisons with other state-of-the-art k-clique community detection methods. The implementation of COS is open-source and freely available [14].

Another major contribution of this paper is the innovative CONNECted componenTs MErging (CONNECT_ME) technique. By taking advantage of CONNECT_ME, it is possible to split a network into an arbitrary number of subnetworks with arbitrary topologies, and still be able to obtain its connected components. This novel low-complexity technique plays a key role in COS, by combining together partial results from all processors.

The remainder of this paper is structured as follows: The next section contains a brief overview of the efforts toward efficient community detection, with special emphasis on k-clique communities. In Section 3, we formulate the problem of k-clique community detection. We discuss CPM in Section 4, highlighting its scalability issues. In Section 5, we present the novel CONNECT_ME technique. A working proof-of-concept parallel k-clique community method and its worst case complexities are presented in Section 6. In the same section, we propose COS and two enhancements at the basis of its functioning. In Section 7, we examine COS performances via experiments. The paper concludes with a short summary in Section 8.

2 Related Work

Traditionally, the relevance of the discovered network communities has been traded off with the complexity required for their extraction. For example, k-core communities [15] can be obtained with low complexity [16] but they are loosely connected and nonoverlapping. Conversely, k-clique communities [2] are fine-grained, overlapping, and tightly connected but their extraction is extremely demanding in terms of computational resources [13]. Very little work has been done to avoid trading off the quality of the extracted communities for complexity. Parallelism has been proposed in [17] and [18] as a means to alleviate computational costs. In [17] Zhang et al. heuristically evaluate the *propinquity*, i.e., the probability that a pair of nodes is involved in a coherent community. They update the original network by adding (removing) edges if the propinquity is higher (lower) than a given threshold. A parallel method is used to update propinquity incrementally, to reflect network changes. Through this, they were able to extract meaningful communities from the huge Wikipedia linkage network. Rather than introducing a new definition of community, in [18] Sadi et al. propose a method to reduce the size of the networks. In parallel, they use a heuristic to locate quasicliques and assign them as nodes in a reduced graph to be used with standard community detection methods. These reduced graphs have a size which is approximately one half of the original size. Alternatively than exploiting parallelism, computational costs can be mitigated by designing efficient heuristics and greedy local function optimizations. To the best of our knowledge, methods proposed in [19] and [20] are two of the fastest (and best-performing according to [21]) optimization-based community detection methods. In Appendix B in the Supplemental Material, which can be found on the Computer Society Digital Library at http://doi.eeecomputersociety.org/10.1109/TPDS.2012.229, we analyse their performance on real-world networks.

The first k-clique community detection method is the CPM [13]. It first lists all the maximal cliques from the input network and then analyses the overlap between each possible pair of them. Although the number of maximal cliques in a network could be exponential with the number of nodes [22], none of the real-world networks we studied has a number of maximal cliques greater than few millions—this means that their actual number is from tens to tens of thousands orders of magnitude smaller that their maximum theoretical number. As a consequence, we were able to obtain the whole list of maximal cliques from the networks considered in this paper in at most a couple of minutes with a serial algorithm [23]. Therefore, we do not add anything new to this point and we refer the interested reader to [24, Section 5] for a review of such algorithms (or to [25], [26], and [27] for parallel algorithms). The real challenge is finding an efficient way to store and analyse the overlap between maximal cliques. In Section 4, we show that this has a complexity proportional to the *square* of the number of maximal cliques.

In [28] the first effort toward efficient k-clique community detection was made. The authors proposed the Sequential

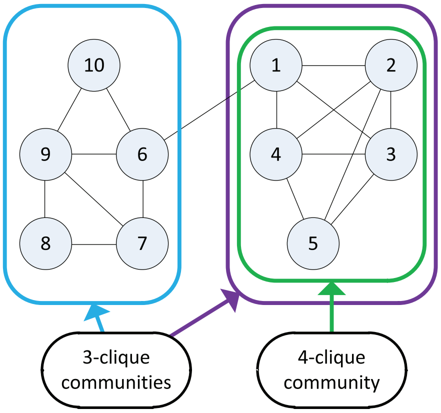


Fig. 1. A graph with its k-clique communities for $k = 4$ and $k = 3$.

Clique Percolation (SCP) method, which enables k-clique communities to be detected at multiple weight thresholds in a single run. Although SCP can detect communities on weighted networks, it cannot produce k-clique communities for each possible k in a single execution. Moreover, since it enumerates cliques rather than maximal cliques, it only works well on sparse networks. In fact, as also highlighted by the authors, given that a clique with size h contains $\binom{h}{k} \approx h^k/k!$ smaller cliques with size k, the huge number of cliques it generates on networks with fairly large maximal cliques—as those considered in this paper—prevents communities to be obtained in a reasonable amount of time.

In [29], we drew our first ideas on how to enhance CPM and proposed a simple parallel k-clique community detection method. Although this method is parallel and has a reduced memory footprint with reference to CPM, it does have several drawbacks. For example, its maximum degree of parallelism is: 1) upper bounded by the size of the largest maximal cliques; 2) strongly affected by maximal cliques distribution; and 3) a decreasing function of the execution time.

3 Problem Formulation

Let $G = (V, E)$ be an undirected, unweighted graph without isolated nodes (vertices) and self-edges. V is its vertex set and $E \subseteq V \times V$ its edge set. A k-clique in G is a subset of the vertex set $c \subseteq V$ such that there is an edge $(i, j) \in E$ between any two nodes $i, j \in c$ and $|c| = k$. A clique is a k-clique for some k. Two k-cliques are *adjacent* if they have $(k-1)$ nodes in common. Based on this notion of adjacency, we can define a k-clique community as follows:

Definition 1. *A k-clique community is the union of all the k-cliques that can be reached by each other through a series of adjacent k-cliques.*

Fig. 1 shows a graph with its k-clique communities at $k = 3$ and $k = 4$. At $k = 2$ there exists only one community, corresponding to the whole graph.

Now observe that each h-clique always contains a number $\binom{h}{k}$ of *adjacent* k-cliques for each $k \leq h$. For this reason an h-clique is (in a) k-clique community for each $k \leq h$. For example the clique $\{1, 2, 3, 4\}$ of Fig. 1 contains $\binom{4}{3} = 4$ adjacent 3-cliques: $\{1, 2, 3\}$, $\{1, 2, 4\}$, $\{1, 3, 4\}$, and $\{2, 3, 4\}$ and therefore is in a 3-clique community. Similarly, it has $\binom{4}{2} = 6$ adjacent 2-cliques and hence is also in a 2-clique community. Furthermore, if the h-clique is *not* contained in any other larger clique, i.e., it is a *maximal h-clique*, it can belong only to k-clique communities with $k \leq h$. This is because it cannot share a number of nodes greater than or equal to its size with any other clique. Otherwise it should be contained in a larger clique and it could therefore not be a maximal clique. For instance clique $\{6, 9, 10\}$ of Fig. 1, which is maximal since there not exists another larger clique containing it, is in a 3-clique community as well as in a 2-clique community, but it cannot belong to any community with size greater than or equal to 4. Instead, the clique $\{3, 4, 5\}$, which is not maximal, belongs also to a 4-clique community. These observations allow us to formulate an equivalent definition of k-clique community.

Definition 2. *A k-clique community is the union of all the maximal h-cliques, $k \leq h$, that can be reached by each other through a series of adjacent k-cliques.*

Accordingly, the problem of k-clique community detection on networks i) is to find all the possible unions of maximal cliques satisfying Definition 2; equivalently, ii) it is to find all the possible unions of cliques satisfying Definition 1. In the remainder of this paper we concentrate on formulation i).

From formulation i), it follows that the lower the k, the higher the number L_k of maximal h-cliques, $k \leq h$, among which to search for k-clique communities. If l_k denotes the number of maximal k-cliques in G, we can express this number as $L_k = \sum_{h=k}^{k_{max}} l_h$, where k_{max} is the maximal cliques maximum size. L_k is maximum for $k = 2$. In fact, L_2 is equal to the number $l = \sum_k l_k$ of maximal cliques in G.

4 CPM and Related Issues

In this section, we discuss the CPM, pointing out its scalability issues. We partition CPM into three subsequent phases for the sake of simplifying the presentation, namely: maximal cliques listing; clique-clique overlap matrix construction; k-clique community extraction. However, as already discussed in Section 2, maximal cliques listing does not represent an issue when dealing with real-world networks, at least with the ones we considered. For that reason, in the remainder of this section we concentrate only on the latter two phases.

4.1 Clique-Clique Overlap Matrix Construction

Given the whole list of maximal cliques, CPM builds a clique-clique overlap matrix as described in [30]. Each maximal clique is associated with a row (column) and the elements of the matrix represent the number of shared nodes between the corresponding maximal cliques. In the remainder of this paper, we assume maximal clique c_i to be always associated with row (column) i. Fig. 2a shows the list of the $l = 6$ maximal cliques extracted from the graph in Fig. 1, whereas Fig. 2b shows the resulting clique-clique overlap matrix. The clique-clique overlap matrix is symmetric and diagonal elements represent the size of the maximal cliques.

$c_0 = \{1,2,3,4\}$
$c_1 = \{2,3,4,5\}$
$c_2 = \{6,9,10\}$
$c_3 = \{6,7,9\}$
$c_4 = \{7,8,9\}$
$c_5 = \{1,6\}$

(a)

	0	1	2	3	4	5
0	4	3	0	0	0	1
1	3	4	0	0	0	0
2	0	0	3	2	1	1
3	0	0	2	3	2	1
4	0	0	1	2	3	0
5	1	0	1	1	0	2

(b)

Fig. 2. (a) The $l = 6$ maximal cliques $c_0, \cdots, c_5$ extracted from the graph in Fig. 1. (b) The resulting clique-clique overlap matrix. Maximal clique c_i is associated with row (column) i.

It is clear that with a standard storage format, the space complexity of the matrix scales quadratically with l; this is in spite of simple optimizations that take into account, for example, the symmetry of the matrix. More efficient storage formats have been proposed for sparse matrices [31], however experimental results have shown that clique-clique overlap matrices can be very dense, i.e., have almost all nonzero elements. This quadratic dependence on l represents the first scalability issue that makes CPM inapplicable on graphs that model real-world complex systems. The second issue concerns the worst case time complexity for computing the clique-clique overlap matrix which is in $\Omega(l^2)$ since overlap has to be computed for each of the $\binom{l}{2}$ possible pairs of maximal cliques.

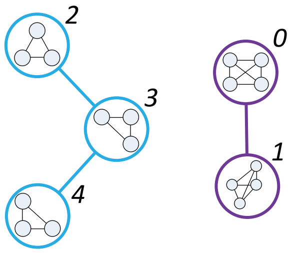
4.2 k-Clique Community Extraction

CPM extracts k-clique communities starting from the clique-clique overlap matrix as follows: It 1) puts at 1 every on-diagonal element greater than or equal to k and every off-diagonal element greater than or equal to $(k-1)$, and then, it 2) zeroes each other element, obtaining a binary matrix. Finally, it extract communities by carrying out a component analysis of this binary matrix.

Rather than accomplishing such analysis, we can relate k-clique communities to the connected components of a graph G_k, which we call henceforth the clique-clique graph. More precisely, if $G_k = (V_k, E_k)$ is a graph whose adjacency matrix is obtained according to 1 and 2, and if no node whose row (column) has all zero elements is in V_k, then k-clique communities are the unions of maximal cliques associated with nodes in the connected components of G_k. Indeed, it is easy to check that 1 and 2 assure that an edge exists between two nodes of G_k iff the corresponding maximal cliques have size greater than or equal to k and share at least $(k-1)$ nodes. Fig. 3b shows the binary matrix obtained from the clique-clique overlap matrix of Fig. 2b for $k = 3$. The row with index 5 contains only zeros since it relates to $c_5 = \{1, 6\}$, which cannot share $(k-1) = 2$ nodes with any other maximal clique. The resulting clique-clique graph $G_3 = (V_3, E_3)$ is shown in Fig. 3b. It has $|V_3| = L_3 = 5$ nodes. Edges represent the condition of having 2 nodes in common. The two connected components of G_3, highlighted with different colors, contain maximal cliques corresponding to the two 3-clique communities of G.

	0	1	2	3	4	5
0	1	1	0	0	0	0
1	1	1	0	0	0	0
2	0	0	1	1	0	0
3	0	0	1	1	1	0
4	0	0	0	1	1	0
5	0	0	0	0	0	0

(a)



(b)

Fig. 3. (a) The binary matrix obtained from the clique-clique overlap matrix of Fig. 2b for $k = 3$. (b) The resulting clique-clique graph G_3.

5 Connected Components Merging

In Section 4.2, we showed a relation between k-clique communities and the connected components of a graph. Here, we propose the innovative CONNECted componenTs MErging (CONNECT_ME) technique, which enables the connected components of the union of two graphs to be obtained without knowing their topologies. CONNECT_ME will be used in the next sections when combining parallel processors' partial results. Since CONNECT_ME has to manipulate disjoint sets to efficiently maintain the connected components, in this section, we also briefly discuss the *set union problem* and a well-known algorithm for its solution.

5.1 Connected Components as a Solution to the Set Union Problem

Connected Components, which are disjoint sets of nodes, can be obtained using any algorithm for solving the *set union problem* [32]. This problem consists in maintaining a collection F of disjoint sets under an intermixed sequence of $find_F$ and $union_F$ operations. $find_F(p)$ returns the canonical element of the set containing element p—the canonical element is an arbitrary but unique element identified within each set, which is used to represent the set. $union_F(P, Q)$ combines the sets whose canonical elements are P and Q into a single set, and make P the canonical element of the new set.

If we initialize F with $|V|$ singleton sets $\{v\}$ such that $v \in V$, we can obtain the connected components of a graph in F after $MERGE\_SETS(F, p, q)$ has been called on each edge $(p, q) \in E$ [33]. $MERGE\_SETS$, which is presented in Method 1, retrieves the canonical elements P and Q of the sets containing p and q via two $find_F$ operations. If $P \neq Q$, then p and q are in two different sets which are merged with a $union_F$.

Method 1: $MERGE\_SETS(F, p, q)$

```
Input: A collection $F$ of disjoint sets and two
       elements $p$ and $q$
Ensure: sets containing $p$ and $q$ are merged in $F$
begin
    $P \leftarrow find_F(p)$
    $Q \leftarrow find_F(q)$
    if $P \neq Q$ then
        $union_F(P, Q)$
```

To the best of our knowledge, the fastest algorithm for the solution of the set union problem is presented and analyzed in [34]. This algorithm represents each set in F as a *rooted tree*[2] whose nodes are the elements in the set and whose root is the canonical element. Each node has an outgoing link to its *father*-node—itself if the root—in the tree. $find_F(p)$ returns the root of the tree containing p and $union_F(P,Q)$ combines the trees whose roots are P and Q, by making P the new root of Q. If two simple optimization rules are applied, this algorithm reaches an $O(f\alpha(f,g))$ worst case time complexity for f operations on g initially singleton sets, assuming $f=\Omega(g)$. α is a functional inverse of Ackermann's function. This function grows very slowly and for all practical purposes is a constant no larger than four [32]. In Appendix D.1 in the Supplemental Material, which is available online, we give an example of the dynamic evolution of a collection F of disjoint sets.

5.2 The CONNECT_ME Technique

We now present CONNECT_ME which, starting from the connected components of two graphs $H_1=(V_1,E_1)$ and $H_2=(V_2,E_2)$, $V_2\subseteq V_1$, enables the connected components of their union $H_1\cup H_2$ to be obtained. If F_1 and F_2 contain disjoint sets equivalent to the connected components of H_1 and H_2, respectively, $CONNECT\_ME(F_1,F_2)$, produces the connected components of the union of the two graphs without any information neither on the edges E_1 nor on the edges E_2. $CONNECT\_ME$ is described in Method 2. For each element $u\in V_2$, the canonical element U of the set containing u is found in F_2 and both U and u are merged in F_1. The basic idea behind this method is: "given that u and U are in the same connected component of H_2, they must also be in the same connected component of $H_1\cup H_2$". The idea is formalized in the following theorem.

```
Method 2: $CONNECT\_ME(F_1,F_2)$
  Input: Two collections of sets, $F_1$ and $F_2$,
         corresponding to the connected
         components of the graphs $H_1=(V_1,E_1)$
         and $H_2=(V_2,E_2)$, $V_2\subseteq V_1$, respectively.
  Ensure: Disjoint sets in $F_1$ correspond to the
          connected components of $H_1\cup H_2$
begin
    foreach $u\in V_2$ do
        $U\leftarrow find_{F_2}(u)$
        if $U\neq u$ then
            $MERGE\_SETS(F_1,U,u)$
```

Theorem 1. *If F_1 and F_2 are collections of sets corresponding to the connected components of graphs $H_1=(V_1,E_1)$ and $H_2=(V_2,E_2)$, $V_2\subseteq V_1$, then $CONNECT\_ME(F_1,F_2)$ ensures F_1 contains sets corresponding to the connected components of $H_1\cup H_2$.*

Proof. See Appendix C.1 in the Supplemental Material, which is available online. □

A general approach, which uses CONNECT_ME to merge the connected components of an arbitrary number of subgraphs, is developed in the next section.

6 Detecting k-Clique Communities in Parallel

In this section, we first present a k-clique community detection method which serves as a working proof-of-concept for addressing the following issues:

1. the ability to reduce the execution time by exploiting parallel architectures;
2. the need to efficiently distribute the load between the processors;
3. the ability to drastically reduce memory requirements by avoiding the use of clique-clique overlap matrices;
4. the need to analytically determine the set of resources to be provisioned.

Then, we describe the COS which includes some additional mechanisms that further reduce execution time.

6.1 A Working Proof-of-Concept

The proof-of-concept CPM On Steroids (COSpoc) is described in Method 3. COSpoc is designed for a p-processor shared-memory architecture. The method takes as input $c_0,\ldots,c_{l-1}$, where c_i contains the list of nodes in the ith maximal clique in G and $c_i\prec c_j$ iff c_i has a size greater than or equal to the size of c_j.

```
Method 3: $COSpoc(c_0,\cdots,c_{l-1})$
  Input: $c_0,\cdots,c_{l-1}$ // $c_i$=list of nodes in the $i$-th
         maximal clique in $G$
  Output: $(k_{max}-1)$ collections of disjoint sets $F_k$,
          $k\in[2,k_{max}]$, corresponding to the
          $k$-clique communities of $G$
1  begin
2      all processors $q$ s.t. $q\in[0,p-1]$ do in parallel
           // Initialize collections of disjoint sets
3          foreach $k\in[2,k_{max}]$ do
4              $F_{q,k}\leftarrow<L_k$ singletons
               $\{0\},\cdots,\{L_k-1\}>$
           // Extract $k$-clique communities
5          foreach $i\in[0,l-1]$ s.t. $i \bmod p=q$ do
6              for $j\leftarrow i+1$ to $l-1$ do
7                  $ov_{i,j}\leftarrow OVERLAP(c_i,c_j)$
8                  foreach $k\in[2,ov_{i,j}+1]$ do
9                      $MERGE\_SETS(F_{q,k},i,j)$
       // Join partial results
10     foreach $k\in[2,k_{max}]$ do
11         foreach $q\in[1,p-1]$ do
12             $CONNECT\_ME(F_{0,k},F_{q,k})$
13     return $F_{0,k}$, $k\in[2,k_{max}]$
```

Immediately after the beginning, in line 2, p processors start their execution in parallel. At first, each processor q, $q\in[0,p-1]$, initializes $(k_{max}-1)$ collections $F_{q,k_{max}},\ldots,$

[2] In the remainder of this section, we use rooted trees to graphically represent disjoint sets.

$F_{q,3}, F_{q,2}$ on which it will be the only one to operate on. Collection $F_{q,k}$ has size L_k. Processor q uses $F_{q,k}$ for extracting the connected components of a subgraph $G_{q,k} = (V_k, E_{q,k})$ of the clique-clique graph G_k. This subgraph has the same vertex set V_k of G_k and an edge set $E_{q,k} \subseteq E_k$ which is determined by the condition in line 5. Formally, $E_{q,k} = \{(i,j) \in E_k : q = i \bmod p \wedge j > i\}$. Processor q obtains the connected components of each subgraph $G_{q,k}$ as follows: First, it executes $OVERLAP(c_i, c_j)^3$ to obtain the number $ov_{i,j}$ of nodes in common between maximal cliques c_i, c_j. Then, since c_i and c_j belong to the same k-clique community for each $k \in [2, ov_{i,j} + 1]$, it merges disjoint sets containing i and j in $F_{q,2}, \dots, F_{q,ov_{i,j}+1}$.

When each processor p has terminated its parallel execution, connected components of the subgraphs $G_{q,k}$ are merged together in the loop starting at line 10. For each k, $F_{0,k}$ is updated with the connected components of $G_{q,k}$, $q \in [1, p-1]$, through $CONNECT\_ME(F_{0,k}, F_{q,k})$. Therefore, after the ith iteration of the loop, according to Theorem 1, $F_{0,k}$ contains the connected components of a graph $\bigcup_{q \in [0,i]} G_{q,k}$. Now, by observing that the remainder of a division by p is always a number between 0 and $p-1$, each possible pair of maximal cliques is processed since we have p processors with indices $q \in [0, p-1]$. Hence, $\bigcup_{q \in [0,p-1]} G_{q,k} = G_k$ and $F_{0,k}$ contains the connected components of G_k after the loop starting at line 10 has completed. These connected components are equivalent to the k-clique communities of G. COSpoc worst case time complexity is given in the following theorem—the assumptions, reasonable, and well-supported by the experiments, are discussed in Appendix C.2 in the Supplemental Material, which is available online.

Theorem 2. *If operations on collections of disjoint sets are in $O(1)$, perfect load balancing is achieved and overlap is calculated through binary searches, then $COSpoc(c_0, \dots, c_{l-1})$ worst case time complexity is in:*

$$O\left(\frac{l^2}{p} k_{max} \log_2 k_{max}\right). \quad (1)$$

Proof. See Appendix C.2 in the Supplemental Material, which is available online. □

Although COSpoc complexity is inversely proportional to the number of processors, the bound derived does not allow to analytically determine the speedup, i.e., the ratio between the execution time of the sequential method and the execution time of the parallel method. However, as we discuss in Section 7, we experimentally measured a *linear* speedup of the method, which is as good as we can possibly hope for. Worst case space complexity is given in the following theorem.

Theorem 3. *$COSpoc(c_0, \dots, c_{l-1})$ worst case space complexity is in:*

$$O(p \cdot l \cdot k_{max}).$$

Proof. See Appendix C.2 in the Supplemental Material, which is available online. □

This complexity depends linearly on l, while in CPM this dependence is quadratic. The substantial reduction in the space required enabled COS to extract k-clique communities from real-world networks, such as those shown in Section 7. The advantages arising from the linear dependence of the space on l far outweigh the disadvantages arising from the linear dependence on p. In fact, $l^2 \gg p$ in any realistic case. The lack of dependence of CPM on p is due to the fact that it is not a parallel algorithm.

6.2 CPM on Steroids

In this section, we introduce COS. Compared to the proof-of-concept COSpoc, in COS we drastically reduce the number of operations on collections of disjoint sets, by ensuring that MERGE_SETS is called at most one time for each possible pair of maximal cliques. To achieve this improvement we: 1) use a *sliding window* over the clique-clique overlap matrix, and 2) exploit the fact that k-clique communities are nested [35]—nested in the sense that each k-clique community is contained in one and only one h-clique community for each $h < k$.

The enhanced method COS, designed for a p-processors shared-memory architecture, uses a sliding window to efficiently process the clique-clique overlap matrix in chunks of configurable size. Indeed, when multiple flows of execution are available, the clique-clique overlap matrix can be used to facilitate cooperation between threads. However, we reduced the standard $O(l^2)$ worst case space complexity to $O(W)$, with W constant and user-configurable. We achieved this reduction with a negligible worst case time complexity since the most expensive operation consists in solving a second-order equation. For a detailed description of the sliding window see Appendix A.1 in the Supplemental Material, which is available online.

For each chunk, parallel operations are divided into two blocks. Two synchronization points are introduced at the end of each block. In the first parallel block the $OVERLAP(c_i, c_j)$ is computed for *each* pair of maximal cliques associated with rows of the matrix in the current chunk. When the overlap is computed, it is written to the buffer. Write operations are performed simultaneously since no two processors ever write to the same location. In the second parallel block the overlap is analyzed to extract the connected components of the clique-clique graphs G_k, according to a strictly decreasing order of k, i.e., from k_{max} down to 2. More precisely, for each chunk, processors keep updated the connected components of subgraphs of $G_{k_{max}}$. Then, they exploit the information already encoded in these connected components to keep updated the connected components of a subgraph $G_{k_{max}-1}$ and so on until G_2. This information can be exploited in accordance to the theorem in [35]. The theorem guarantees that each k-clique community is contained in one and only one h-clique community, $h \in [2, k]$, implying $G_k \subseteq G_{k-1}$ for each k.

A comprehensive description of COS, including its complexity analysis, is omitted here due to space limits but is given in Appendix A in the Supplemental Material, which is available online. We demonstrate that COS has a

[3] In practice, if c_i and c_i are represented as ordered vectors, this function can be efficiently implemented by performing a binary search on the larger vector for each element of the smaller one.

TABLE 1
Graphs Used in the Experiments

Graph	Type	Ref.	$\vert V\vert$	$\vert E\vert$
LINX	Auton. Systems	[37]	345	14,188
NDwww	Web	[38]	325,729	1,090,108
SFwww	Web	[39]	281,903	1,992,636
CAquery	Web	[40]	9,664	15,969
Yeast	Protein Interact.	[41]	2,361	6,646
NetSci	Collaboration	[42]	1,589	2,742
Erdos	Collaboration	[38]	6,927	11,850
Geom	Collaboration	[43]	7,343	7,343
Amazon	Prod. co-purch.	[44]	403,394	2,443,408
AstroPh	Collaboration	[45]	18,772	198,050
CondMat	Collaboration	[45]	23,133	93,439
HepTh	Citation	[45]	27,770	352,285
EmlEnron	Communication	[46]	36,692	183,831

TABLE 2
Features of the Graphs Used in the Experiments

Graph	l	k_{max}	μ	σ^2	l_2	$\tilde{s}$ [GB]
LINX	384,494	34	23.01	11.29	1	113.11
NDwww	495,947	155	3.15	1.49	294,706	37.72
SFwww	1,055,936	61	7.00	5.65	108,831	835.4
CAquery	17,548	10	1.92	0.18	15,660	> 0.01
Yeast	5,012	9	2.45	0.39	3,644	> 0.01
NetSci	741	20	2.88	0.94	349	> 0.01
Erdos	9,210	8	2.47	0.32	6,503	0.01
Geom	5,817	22	2.68	0.85	3,167	0.01
Amazon	1,023,572	11	3.82	0.75	264,874	536.09
AstroPh	36,428	57	6.87	3.88	2,236	1.09
CondMat	18,502	26	3.95	1.01	3,888	0.2
HepTh	464,873	23	7.71	1.99	15,466	188.1
EmlEnron	226,859	20	8.08	1.36	14,070	42.17

worst case space complexity in $O(\, l \cdot (p + k_{max}) + W\,)$. In addition, we prove that COS can achieve the same worst case time complexity of COSpoc. Nevertheless, by comparing COSpoc and COS in Appendix B.2 in the Supplemental Material, which is available online, we highlighted great reductions in execution time. This is mainly due to the strongly reduced number of operations on collections of disjoint sets.

7 Experimental Results

We implemented COSpoc and COS in Appendix C, which is available online, in a freely and publicly available software [14]. Maximal Cliques were listed using the open-source implementation of the (serial) Bron-Kerbosh (BK) algorithm available in the `igraph` library [36]. For parallel programming we used the standard POSIX Threads.[4] We used a CPM implementation available in CFinder[5] [2] and a python SCP implementation retrieved from http://www.lce.hut.fi/~mtkivela/kclique.html. The machine on which we ran the experiments has four Intel Xeon processors E7-4850[6] and 128 GB RAM. It runs a GNU/Linux Operating System (OS) with a kernel Linux 3.0.6.

Graphs used in the experiments, together with the type of complex system they model, their references and their number of nodes $|V|$ and edges $|E|$ are reported in Table 1. LINX graph was obtained according to [37] whereas the others were retrieved from [38] and [47]. All the graphs were considered undirected, unweighted, without isolated nodes and without self- and multiple-edges. Table 2 reports the total number of maximal cliques l in each graph, their maximum size k_{max}, their average size $\mu = l^{-1}\sum_k k \cdot l_k$, their variance $\sigma^2 = l^{-1}\sum_k l_k(k-\mu)^2$ and the number l_2 of maximal cliques with size 2. In addition, a fine estimation $\tilde{s}$ of the size of the clique-clique overlap matrix CPM has to build is reported. This estimation was computed as the square of the number of maximal cliques with size strictly greater than 2, assuming that a byte is used for each element. With this estimation it is possible to know, a priori, which graphs can be processed by CPM on our 128 GB memory machine. The accuracy of this estimation was experimentally validated by monitoring CPM runtime memory footprint in Appendix B in the Supplemental Material, which is available online. In the experiment we compare CPM runtime memory footprint with that of COS. We show that COS maximum memory footprint is, at most, approximately equal to the size of the sliding window buffer. Therefore, the upper bound on its worst case space complexity can actually be considered W in practice. In the aforementioned Appendix B, we also compare COS and COSpoc execution time, with the aim of demonstrating that the techniques introduced in COS dramatically improve the overall performance. In fact, COS is from one to two orders of magnitude faster than COSpoc. In addition, still in Appendix B, which is available in the online supplemental material, we performed an experiment to determine how changes in the sliding window buffer size W impact on the execution time. We observed that for buffer sizes greater than 8 GB the execution time is minimized and its almost constant values suggest a low sensitivity of COS to changes in the sliding window buffer size in this range. We have chosen the size $W = 32$ GB as the default size for subsequent experiments.

In Fig. 4, we show the execution time of COS versus an exponentially increasing number of threads. Plotted values do not include the time to list maximal cliques with the BK algorithm. LINX, NDWWW, and AstroPh were chosen as inputs since their execution times always differ for at least one order of magnitude, regardless of the number of threads. An ideal case reference (dashed line), where doubling the number of threads halves the execution time, is drawn for each input. It is worth noting how COS reaches—or is very close to—the ideal case, at least up to 32 threads. The distance from the ideal case that is experienced for 64 and 80 threads is due to the fact that our hardware's physical degree of parallelism is 40. Hence, we can conclude that COS speedup on our machine is *linear* with the number of physical cores available.

[4] POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995).

[5] Experiments were carried out with version 2.0.5, 64 bit.

[6] 24M Cache, 2.00 GHz, 10 cores, 20 threads, HT capable.

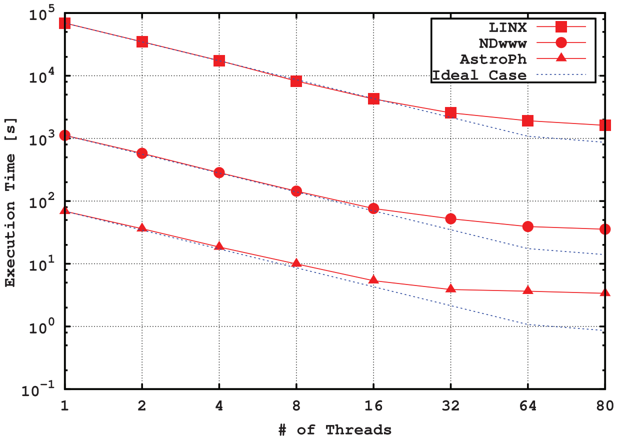


Fig. 4. Execution time of COS versus number of threads.

In Fig. 5, we show the time COS took to execute on graphs for which it was not possible to run CPM on. In particular, it was not possible to execute CPM on SFwww, Amazon, and HepTh due to their clique-clique overlap matrix size, exceeding the amount of memory available on our hardware. Conversely, despite matrix sizes of LINX, AstroPh, and EmlEnron would allow CPM to run, we (on LINX and AstroPh after two days) or the OS (on EmlEnron after 12 hours) stopped the execution. Values plotted include the time taken by serially extracting maximal cliques. Even with the inclusion of this time, COS continues to achieve a very good speedup. Hence—as also stated in Section 2—maximal cliques listing time is negligible if compared with the total time.

In Fig. 6, we compare COS, CPM, and SCP execution times. On our hardware, we were able to execute successfully CPM on NDwww, CAquery, Yeasy, NetSci, Erdos, Geom, and CondMat. We were able to run also SCP on all these graphs except NDwww. Since SCP is designed to extract k-clique communities for a given k, we obtained its execution time by summing the times it takes to extract k-clique communities for each possible value of k. Execution times of both CPM and SCP are given in the bottom right corner of Fig. 6. In the other plots of the same figure, we show the values of two execution time ratios, namely: CPM/COS (in red), and SCP/COS (in blue). For each number of threads in the x-axis, we computed the ratio CPM/COS (SCP/COS) by dividing the execution time of CPM (SCP) with that of COS executed with the corresponding number of threads. These ratios, which are always greater than 1, reveal that COS is always faster than both CPM and SCP. In particular, it is always more than 10 times faster than SCP on any input, even when run with 1 thread. By increasing the number of threads, we see that it becomes 100 to more than 1,000 times faster. Best performance is obtained for CondMat where COS terminate its 80-threaded execution in one 10,000th the time it takes SCP. Very good execution time reductions are experienced also with reference to CPM. In this case, shortening in the execution time starts from a few units for single threaded executions, and reach 10-20 when the number of threads is increased. Although these reductions are important in both cases, absolute execution times are too small (only NDwww takes more than 10 seconds) to enable the identification of a clear link between number of threads and COS execution time variations. Finally, with this experiment we can say that SCP, although designed to overcome its drawbacks, it is actually slower than CPM and differences in their execution time always exceed the order of magnitude. In Appendix B in the Supplemental Material, which is available online, we also compare COS with other state-of-the art methods, detecting communities different from the k-clique communities. For the comparison we carefully selected 6 of the best-performing methods available in the literature. We observed that COS performance is as good as the fastest methods on some graphs, even when it is not executed in parallel. Performance degradations are observed on graphs with an extremely high number of maximal cliques with large sizes.

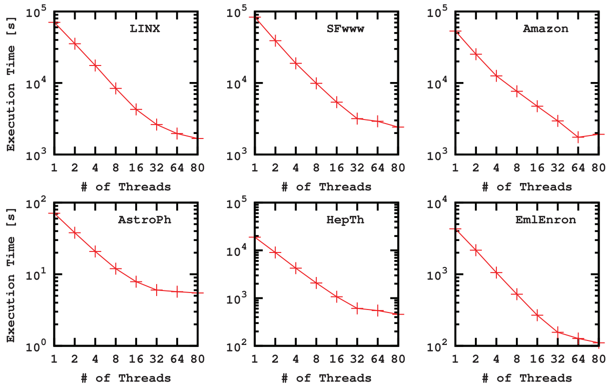


Fig. 5. Execution time of COS versus number of threads.

8 Discussion and Conclusions

In this paper, we addressed the problem of extracting k-clique communities in parallel from real-world networks such as the Internet and the World Wide Web. We

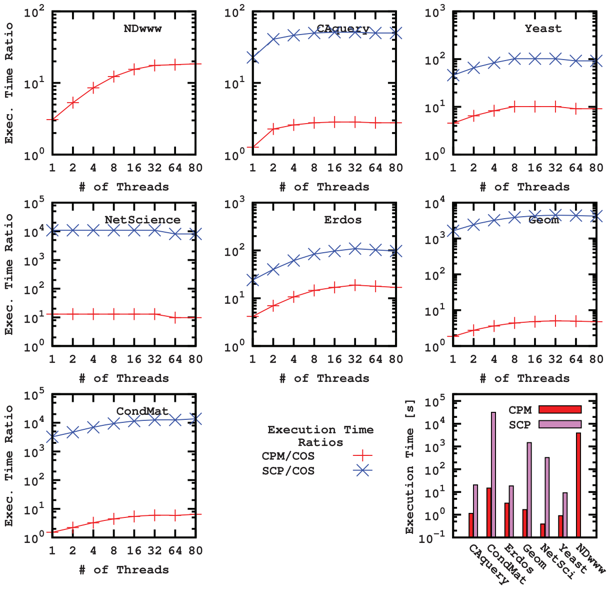


Fig. 6. CPM/COS (SCP/COS) execution time ratio and execution times of CPM and SCP.

theoretically analyzed the existing CPM, highlighting its scalability issues. The identification of these scalability issues enabled us to design and develop COS. COS efficiently extracts k-clique communities, with low memory requirements and has an unbounded, user-configurable degree of parallelism. Analytical tight upper bounds on COS execution time and space requirements, providing strong evidence about its efficiency, are presented as well. A key role in COS is played by the innovative CONNECted ComponenTs MErging (CONNECT_ME) technique. With this technique we can obtain the connected components of a network, even if it has previously been split into and arbitrary number of subnetworks that could be processed in parallel. Through extensive experiments run on real-world networks, we showed that COS has a linear speedup and constantly outperform all the other state-of-the-art k-clique community detection methods in terms of both space requirements and execution time. In our opinion, it should be the method of choice for k-clique communities extraction aiming at very high performance and low resource requirements. As a future work we plan to extend the design of COS for a message-passing architecture and to investigate its performance on mega-scale networks such as Wikipedia, Facebook, and Twitter.

References

[1] S. Fortunato, "Community Detection in Graphs," *Physics Reports,* vol. 486, nos. 3-5, pp. 75-174, 2010.

[2] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, "Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society," *Nature,* vol. 435, no. 7043, pp. 814-818, 2005.

[3] E. Gregori, L. Lenzini, and C. Orsini, "k-Dense Communities in the Internet AS-Level Topology," *Proc. Third Int'l Conf. Comm. Systems and Networks (COMSNETS),* 2010.

[4] E. Gregori, L. Lenzini, and C. Orsini, "K-Clique Communities in the Internet AS-Level Topology Graph," *Proc. 31st Int'l Conf. Distributed Computing Systems Workshops (ICDCSW '11),* 2011.

[5] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the Web for Emerging Cyber-Communities," *Computer Networks,* vol. 31, nos. 11-16, pp. 1481-1493, 1999.

[6] J.P. Onnela, J. Saramäki, J. Hyvönen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and A.L. Barabási, "Structure and Tie Strengths in Mobile Communication Networks," *Proc. Nat'l Academy of Sciences of USA,* vol. 104, no. 18, pp. 7332-7336, 2007.

[7] M. Girvan and M. Newman, "Community Structure in Social and Biological Networks," *Proc. Nat'l Academy of Sciences of USA,* vol. 99, no. 12, pp. 7821-7826, 2002.

[8] P. Chen and S. Redner, "Community Structure of the Physical Review Citation Network," *J. Informetrics,* vol. 4, no. 3, pp. 278-290, 2009.

[9] J.-L. Guillaume and M. Latapy, "Bipartite Graphs as Models of Complex Networks," *Physica A: Statistical Mechanics and Its Applications,* vol. 371, no. 2, pp. 795-813, 2006.

[10] P. Hui and J. Crowcroft, "Human Mobility Models and Opportunistic Communications System Design," *Philosophical Trans. Royal Soc. A: Math., Physical and Eng. Sciences,* vol. 366, no. 1872, pp. 2005-2016, 2008.

[11] P. Hui, E. Yoneki, S.Y. Chan, and J. Crowcroft, "Distributed Community Detection in Delay Tolerant Networks," *Proc. ACM/ IEEE Second Int'l Workshop Mobility in the Evolving Internet Architecture,* pp. 7:1-7:8, 2007.

[12] G. Palla, A. Barabasi, and T. Vicsek, "Quantifying Social Group Evolution," *Nature,* vol. 446, pp. 664-667, Apr. 2007.

[13] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, "Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society - Supplementary Material," *Nature,* vol. 435, no. 7043, pp. 814-818, 2005.

[14] E. Gregori, L. Lenzini, and S. Mainardi, "Parallel K-Clique Community Detection on Large-Scale Networks," http://cosparallel.sourceforge.net/, 2013.

[15] K. Saito, T. Yamada, and K. Kazama, "Extracting Communities from Complex Networks by the k-Dense Method," *IEICE Trans. Fundamentals of Electronics Comm. Computer Sciences,* vol. E91-A, no. 11, pp. 3304-3311, 2008.

[16] V. Batagelj and M. Zaversnik, "An o(m) Algorithm for Cores Decomposition of Networks," *J. Computing Research Repository,* vol. 0310049, 2003.

[17] Y. Zhang, J. Wang, Y. Wang, and L. Zhou, "Parallel Community Detection on Large Networks with Propinquity Dynamics," *Proc. ACM SIGKDD 15th Int'l Conf. Knowledge Discovery and Data Mining,* pp. 997-1006, 2009.

[18] S. Sadi, S.G. Ögüdücü, and A.S. Etaner-Uyar, "An Efficient Community Detection Method Using Parallel Clique-Finding Ants," *Proc. IEEE Congress on Evolutionary Computation,* pp. 1-7, 2010.

[19] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast Unfolding of Communities in Large Networks," *J. Statistical Mechanics: Theory and Experiment,* vol. 2008, no. 10, p. P10008, 2008.

[20] M. Rosvall and C.T. Bergstrom, "Maps of Random Walks on Complex Networks Reveal Community Structure," *Proc. Nat'l Academy of Sciences of USA,* vol. 105, no. 4, pp. 1118-1123, 2008.

[21] A. Lancichinetti and S. Fortunato, "Community Detection Algorithms: A Comparative Analysis," *Physical Rev. E,* vol. 80, p. 056117, 2009.

[22] J. Moon and L. Moser, "On Cliques in Graphs," *Israel J. Math.,* vol. 3, pp. 23-28, 1965.

[23] C. Bron and J. Kerbosch, "Algorithm 457: Finding All Cliques of an Undirected Graph," *ACM Comm.,* vol. 16, no. 9, pp. 575-577, 1973.

[24] I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo, "The Maximum Clique Problem," *Handbook of Combinatorial Optimization,* pp. 1-74, Kluwer Academic Publishers, 1999.

[25] N. Du, B. Wu, L. Xu, B. Wang, and X. Pei, "A Parallel Algorithm for Enumerating all Maximal Cliques in Complex Network," *Proc. IEEE Sixth Int'l Conf. Data Mining Workshops,* pp. 320-324, 2006.

[26] M.C. Schmidt, N.F. Samatova, K. Thomas, and B.-H. Park, "A Scalable, Parallel Algorithm for Maximal Clique Enumeration," *J. Parallel Distributed Computing,* vol. 69, pp. 417-428, 2009.

[27] Y. Zhang, F.N. Abu-Khzam, N.E. Baldwin, E.J. Chesler, M.A. Langston, and N.F. Samatova, "Genome-Scale Computational Approaches to Memory-Intensive Applications in Systems Biology," *Proc. ACM/IEEE Conf. Supercomputing,* 2005.

[28] J.M. Kumpula, M. Kivelä, K. Kaski, and J. Saramäki, "Sequential Algorithm for Fast Clique Percolation," *Physical Rev. E,* vol. 78, no. 2, p. 026109, 2008.

[29] E. Gregori, L. Lenzini, S. Mainardi, and C. Orsini, "Flip-cpm: A Parallel Community Detection Method," *Proc. 26th Int'l Symp. Computer and Information Sciences (ISCIS),* pp. 249-255, 2011.

[30] M.G. Everett and S.P. Borgatti, "Analyzing Clique Overlap," *Connections,* vol. 21, no. 1, pp. 49-61, 1998.

[31] J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide,* Z. Bai ed. Soc. for Industrial and Applied Math., 2000.

[32] R.E. Tarjan and J. van Leeuwen, "Worst-Case Analysis of Set Union Algorithms," *J. ACM,* vol. 31, pp. 245-281, 1984.

[33] D. Eppstein, Z. Galil, and G.F. Italiano, "Dynamic Graph Algorithms," *Algorithms and Theory of Computation Handbook,* first ed., M.J. Atallah and S. Fox, eds., CRC Press, Inc., 1998.

[34] R.E. Tarjan, "Efficiency of a Good but Not Linear Set Union Algorithm," *J. ACM,* vol. 22, pp. 215-225, 1975.

[35] E. Gregori, L. Lenzini, and C. Orsini, "k-Clique Communities in the Internet AS-Level Topology Graph," technical report, http://puma.isti.cnr.it/, 2010.

[36] G. Csardi and T. Nepusz, "The Igraph Software Package for Complex Network Research," http://igraph.sf.net, 2013.

[37] E. Gregori, A. Improta, L. Lenzini, and C. Orsini, "The Impact of Ixps on the As-Level Topology Structure of the Internet," *Computer Comm.,* vol. 34, no. 1, pp. 68-82, 2011.

[38] V. Batagelj and A. Mrvar, "Pajek Datasets," http://vlado.fmf.uni-lj.si/pub/networks/data/, 2013.

[39] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub, "Exploiting the Block Structure of the Web for Computing PageRank," technical Report, Stanford Univ., 2003.

[40] J.M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," *J. ACM,* vol. 46, pp. 604-632, 1999.

[41] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen, "Topological Structure Analysis of the Protein-Protein Interaction Network in Budding Yeast," *Nucleic Acids Research,* vol. 31, no. 9, pp. 2443-2450, 2003.

[42] M.E.J. Newman, "Finding Community Structure in Networks Using the Eigenvectors of Matrices," *Phys. Rev. E,* vol. 74, 2006.

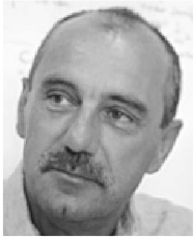
[43] B. Jones, "Computational Geometry Database," ftp://ftp.cs.usask.ca/pub/geometry/, 2012.

[44] J. Leskovec, L.A. Adamic, and B.A. Huberman, "The Dynamics of Viral Marketing," *ACM Trans. Web,* vol. 1, article 5, 2007.

[45] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph Evolution: Densification and Shrinking Diameters," *ACM Trans. Knowledge Discovery from Data,* vol. 1, article 2, 2007.

[46] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations," *Proc. 11th ACM SIGKDD Int'l Conf. Knowledge Discovery in Data Mining,* pp. 177-187, 2005.

[47] J. Leskovec, "Stanford Large Network Dataset Collection," http://snap.stanford.edu/data/, 2013.



Enrico Gregori received the laurea degree in electronic engineering from the University of Pisa in 1980. He has contributed to several national and international projects on computer networking. He has authored more than 100 papers in the area of computer networks, published in international journals and conference proceedings, and is coauthor of the book Metropolitan Area Networks (Springer, 1997). His current research interests include Internet measurements and data analysis, ad hoc networks, sensor networks, wireless LANs, quality of service in packet-switching networks, and evolution of TCP/IP protocols. He is the member of the IEEE.

Luciano Lenzini received the degree in physics from the University of Pisa, Italy. In 1994, he joined the Department of Information Engineering, University of Pisa as a full professor. His current research interests include the design and performance evaluation of MAC protocols for wireless networks, architectures and protocols for mesh networks, and the Quality of Service provision in integrated and differentiated services networks. He is currently on the editorial boards of computer networks and the *Journal of Communications and Networks* as an area editor for wireless networks. He is a member of the IEEE.

Simone Mainardi received the BSc and MSc degrees in computer engineering from the University of Pisa, in 2009 and 2010, respectively. Currently, he is working toward the PhD degree at the Department of Information Engineering, University of Pisa and a research associate at the Institute of Informatics and Telematics of the Italian National Research Council. His research interests include parallel and distributed algorithms, Internet measurements and data analysis, complex network analysis, and network evolutionary models.