Finding All Maximal Cliques in Very Large Social Networks

Alessio Conte[1], Roberto De Virgilio[2], Antonio Maccioni[2],
Maurizio Patrignani[2], Riccardo Torlone[2]
[1]Università di Pisa, Pisa, Italy
conte@di.unipi.it
[2]Università Roma Tre, Rome, Italy
{dvr, maccioni, patrigna, torlone}@dia.uniroma3.it

ABSTRACT

The detection of communities in social networks is a challenging task. A rigorous way to model communities considers maximal cliques, that is, maximal subgraphs in which each pair of nodes is connected by an edge. State-of-the-art strategies for finding maximal cliques in very large networks decompose the network in blocks and then perform a distributed computation. These approaches exhibit a trade-off between efficiency and completeness: decreasing the size of the blocks has been shown to improve efficiency but some cliques may remain undetected since high-degree nodes, also called hubs, may not fit with all their neighborhood into a small block. In this paper, we present a distributed approach that, by suitably handling hub nodes, is able to detect maximal cliques in large networks meeting both completeness and efficiency. The approach relies on a two-level decomposition process. The first level aims at recursively identifying and isolating tractable portions of the network. The second level further decomposes the tractable portions into small blocks. We demonstrate that this process is able to correctly detect all maximal cliques, provided that the sparsity of the network is bounded, as it is the case of real-world social networks. An extensive campaign of experiments confirms the effectiveness, efficiency, and scalability of our solution and shows that, if hub nodes were neglected, significant cliques would be undetected.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; G.2.3 [**Discrete Mathematics**]: Applications—*Maximal clique enumeration*

Keywords

Community detection, maximal clique enumeration, scale-free networks

1. INTRODUCTION

The detection of groups of densely connected nodes, usually called communities, is used to reveal fundamental properties of networks in a variety of domains such as sociology, bibliography, and biology [13, 18, 29]. A rigorous way to model communities considers maximal cliques, that is, maximal subgraphs in which any pair of nodes is connected by an edge. Maximal clique enumeration (MCE) is a paradigmatic problem in computer science and, due to its known complexity, several solutions have been proposed to deal with real-world scenarios [6, 14, 16, 33, 34].

When very large networks are involved, state-of-the-art strategies consist of decomposing the network into blocks that are independently processed in a distributed and parallel environment [8, 10, 14, 20, 31, 36, 38]. A crucial aspect of this approach is the choice of the size m of the blocks. Clearly, m is bounded by the dimension of the memory, but it has been shown that artificially reducing m to values as low as 1/100 or 1/1000 of the available memory results in a more efficient computation [8, 9, 10]. On the other hand, if the size of the blocks is too small, the effectiveness of the approach is compromised. In fact, consider a node n such that the graph induced by its neighborhood does not fit into a block. We call such a node *hub*. In any block of the decomposition a portion of the neighborhood of n will be necessarily omitted and, consequently, some maximal cliques involving n may remain undetected and some non-maximal cliques could be erroneously found.

Hence, while fixing the size of the blocks, state-of-the-art decomposition approaches also need to find a trade-off between efficiency and effectiveness. Even if efficiency is not an issue, effectiveness can be jeopardized since real-world social networks often contain nodes whose degree (i.e., the number of incident edges) is so high that their neighborhood does not fit into main memory altogether.

Actually, high degree nodes are connatural in *scale-free networks*, where the degree distribution of the nodes follows a power law. This property implies that the number of nodes with h connections to other nodes decreases exponentially as h increases and that the set of nodes with arbitrary high degree is not empty [2]. Several works in literature show that social networks, such as Facebook and Twitter, are scale-free [12, 35]. It has also been shown that scale-freeness is exhibited whenever the network has a growth mechanism based on preferential attachment [3, 11], that is, when new connections are distributed among nodes according to how many connections they already have. Hence, as social networks grow, this property is expected to be exacerbated.

In this paper, we address these limitations by proposing an approach to the problem of maximal clique enumeration in very large social networks that meets both the requirements of completeness and efficiency. The approach leverages on sparsity, another property of real-world networks, which basically means that the network can have very dense areas but, overall, nodes and edges are of the same order of magnitude.

Our solution is based on a two-level decomposition of the network. The first-level decomposition aims at recursively identifying and isolating tractable portions involving non-hub nodes only. Intuitively, this operation allows us to "brake" hub nodes by progressively decreasing their degree. The second-level decomposition suitably splits tractable portions of the network into small blocks that can be handled separately. Within a block, we then select the most promising state-of-the-art algorithm for enumerating its maximal cliques. A suitable procedure allows us to recognize and filter out those that are not maximal for the overall network. We formally show that this process is able to correctly detect *all* maximal cliques, provided that the sparsity of the network is bounded, as it is the case of real-world social networks.

We have performed a large number of experiments over data from real-world social networks showing that our approach is effective, efficient, and scalable. The experimentation confirms that in order to have an efficient computation it is convenient to choose a relatively small size of the blocks, which further increases the number of hub nodes. The experiments also confirm that, if hub nodes were neglected, significant cliques would remain undetected.

Summarizing, the contributions of this paper are the following.

- We propose a distributed approach to maximal clique enumeration in large social networks based on a novel decomposition strategy that, by suitably handling high-degree nodes, is able to progressively identify and isolate tractable portions of the network;
- We formally prove the correctness and the completeness of the approach;
- We provide experimental evidence of the efficiency and scalability of our solution and show that, if high-degree nodes were neglected, significant cliques would be undetected.

The rest of the paper is organized as follows. In Section 2 we provide a general overview of our technique. Section 3 describes in depth the two-level decomposition algorithm, Section 4 describes the computation of the maximal clique on a single block of the decomposition, and Section 5 provides the theoretical basis for the whole approach. In Section 6 we illustrate our campaign of experiments. Section 7 surveys the related work and Section 8 contains our conclusions.

2. OVERVIEW AND INTUITION

Our approach is based on a decomposition of the input network in smaller subgraphs called blocks that can partially overlap with each other. As we have discussed in the Introduction, this requires a careful choice of the size of the blocks, depending on hardware limitations and performance issues. Whichever the choice, let m be the maximum number of nodes that can fit in a block. The value of m identifies two types of nodes in the network: (i) the set N_h of *hub nodes* having degree greater or equal than m (i.e., those nodes that would not fit into a block with all their neighbors) and (ii) the set N_f of *feasible nodes* having degree less than m.

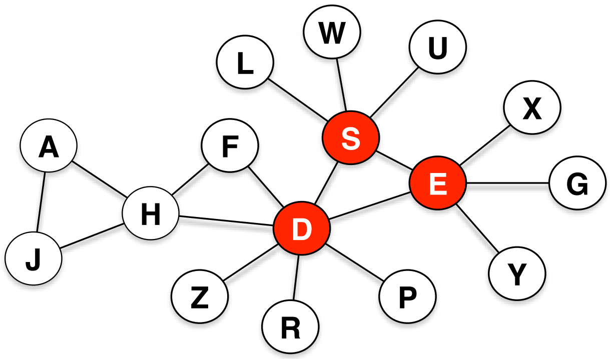


Figure 1: Feasible nodes (white) and hub nodes (red) when $m = 5$.

Consider, for example, the network in Figure 1 and suppose $m = 5$. The set N_h consists of the red-coloured nodes D, S, and E of degree 7, 5, and 5 respectively, whereas N_f consists of the remaining white nodes.

Now, let C_f be the set of all maximal cliques of G involving at least one node in N_f and let C_h be the set of all maximal cliques in the network G_h induced[1] by the nodes in N_h. For example, in the network in Figure 1 we have that C_f includes the cliques {A, J, H} and {H, F, D}, as they both involve feasible nodes, while C_h includes the clique {D, S, E}, since G_h consists only of the nodes D, S, E and of the edges between them.

Our approach is based on the intuition that the set of all maximal cliques of the network G can be obtained from C_f and C_h alone. This is confirmed by Lemma 1 in Section 5, which establishes that the set of the maximal cliques of G is the union of C_f and the set C'_h obtained by filtering out from C_h any clique that is contained into a clique of C_f.

This result suggests that if we process separately the nodes in N_f and the nodes in N_h, no clique is left out. We then obtain an effective decomposition strategy which is also efficient since the neighbors of a feasible node fit into a block of size m by definition, while the degree of the nodes in the induced graph G_h is strongly reduced since G_h only involves a limited number of nodes in scale free networks. For instance, in the network of Figure 1, G_h is the cycle {D, S, E} and its maximum degree is two.

Regarding the computation of the cliques in C_f and C_h we proceed as follows.

C_f: As in [10], we compute a suitable partition of N_f and add to each set S of the partition the neighborhood in G of the nodes in S. The obtained sets of nodes, together with the edges between them, form the blocks of the decomposition. Observe that a node (including

[1] We recall that the subgraph of $G = (N, E)$ *induced* by a set of nodes $N' \subseteq N$ is the restriction of G to the nodes in N' and the edges between them.

the hub ones) may be included into several blocks as a neighbor node, together with a subset of its edges. Differently from [10], we allow for blocks of heterogeneous size and high connectivity that can be processed independently in an efficient way. Then, taking advantage of a decision tree, we apply on each block the most promising MCE algorithm based on the block characteristics. For instance, if the block is sparse, we find the maximal cliques with the algorithm in [17], while if the block is dense we adopt the algorithm described in [34].

C_h: We apply the whole approach recursively to G_h by partitioning its nodes N_h into two sets N'_f and N'_h of feasible and hub nodes, respectively. This is possible since the degree of the nodes in N_h is strongly reduced. The recursion produces a sequence of sets N'_f, N''_f, N'''_f, ... of decreasing size until there are no more hub nodes remaining.

In Section 5 we prove that, under the hypothesis that the input graph is sparse enough, this recursive process converges, in the sense that it ends with a bipartition involving only tractable nodes. In addition, in Section 6, we report that in all our experiments on real-world data sets the process needed at most a few recursive steps.

Summarizing, our approach consists of the following.

1. **First level decomposition:** we identify the set N_f of *feasible* nodes of G, whose degree is less than m, and the set N_h of *hub* nodes of G, whose degree is greater or equal than m.
2. **Recursive call:** if N_h is not empty, we build the subgraph G_h of G induced by the nodes in N_h and apply recursively the whole process to G_h.
3. **Second level decomposition:** given a set of feasible nodes N_f we compute a set of blocks by partitioning N_f and by adding to each node of a block its neighbors.
4. **Block analysis:** we apply a suitable MCE algorithm to each block generated by the second level decomposition to compute all its maximal cliques. The MCE algorithm is chosen from a collection of alternatives, taken from the literature, based on the properties of the block, as described in Section 4.
5. **Filtering:** the output is obtained by taking the union of the maximal cliques computed in step 4 and those computed in step 2, filtering out redundant cliques.

In the following sections we will describe in more detail the various steps of this strategy.

3. NETWORK DECOMPOSITION

Algorithm 1 (FIND-MAX-CLIQUES) describes our recursive procedure for computing maximal cliques. The CUT procedure (line 1) performs the first-level decomposition while the BLOCKS procedure (line 2) performs the second-level decomposition. In this section we describe in detail both of them.

Algorithm BLOCK-ANALYSIS (line 5) is discussed in Section 4. Procedure induced (line 6) accepts as input a graph G and a subset N_h of its nodes and computes the subgraph of G induced by N_h. Procedure filter (line 7) accepts as input two sets C_h and C_f of cliques and outputs all cliques in C_h that are not contained into some clique of C_f.

Algorithm 1: FIND-MAX-CLIQUES: Overall algorithm

```
Input : A graph $G = \langle N, E \rangle$ and a block size m.
Output: The set $C$ of the maximal cliques of $G$.
1 $\langle N_f, N_h \rangle \leftarrow$ CUT($G$,m);        /* $1^{st}$ level decomp.  */
2 $\mathcal{B} \leftarrow$ BLOCKS($G$, $N_f$, m);   /* $2^{nd}$ level decomp.  */
3 $C_f \leftarrow \emptyset$;
4 foreach $b \in \mathcal{B}$ do
5     $C_f \leftarrow C_f \cup$ BLOCK-ANALYSIS($b$);
6 $C_h \leftarrow$ FIND-MAX-CLIQUES(induced($G$,$N_h$),m);
7 $C'_h \leftarrow$ filter($C_h$, $C_f$);
8 return $C_f \cup C'_h$;
```

3.1 First level decomposition

Algorithm 2 describes the CUT procedure that is responsible of identifying the set N_f of feasible nodes and the set N_h of hub nodes. This is done by means of the procedure isfeasible (also called by procedure BLOCKS) that takes as input a set of nodes, the graph G and the maximum block size m and checks whether the union of the given nodes and all their neighborhoods in G has less than m elements. The set N_h of hub nodes is simply obtained, at line 5, as the difference between the nodes of G and N_f.

Algorithm 2: CUT: First-level decomposition

```
Input : A graph $G = \langle N, E \rangle$ and a block size m.
Output: The sets $N_f$ and $N_h$ of feasible and hub nodes
        of $G$, respectively.
1 $N_f \leftarrow \emptyset$;
2 foreach $n \in N$ do
3     if isfeasible({$n$},$G$,m) then
4         $N_f \leftarrow N_f \cup \{n\}$;
5 $N_h \leftarrow N - N_f$;
6 return $\langle N_f, N_h \rangle$;
```

3.2 Second level decomposition

Algorithm 3 describes the BLOCKS procedure, responsible of decomposing the input graph G into tractable blocks of maximum size m. The input graph G is assumed to have maximum degree $m - 1$. Here, we model blocks similarly to [10] but allow for blocks of heterogeneous sizes and leverage the adjacency of the nodes to put dense subgraphs into the same block. Hence, this step, in addition to distributing the computational load into tasks that could be accomplished separately in a distributed environment, also preprocesses the input producing internally homogeneous and compact chunks.

Blocks are defined sequentially in a greedy way. Each block will have *kernel* nodes, *border* nodes, and *visited* nodes. Each node of N_f is kernel node in exactly one block (i.e., kernel nodes form a partition of N_f). All the nodes of G that are adjacent to at least one kernel node of a block B and that are not kernel nodes of B are divided into border nodes and visited nodes of B, where visited nodes are those nodes that have been already used as kernel nodes for some previously defined block. The block is completed with all the edges among its nodes, irrespectively of the type.

For instance, consider again the network in Figure 1.

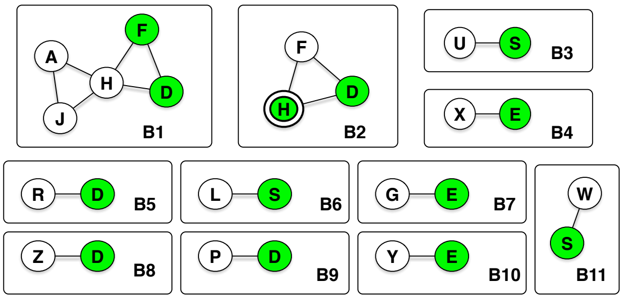


Figure 2: An example of graph decomposition obtained by focusing on the feasible nodes of the network of Figure 1.

Nodes D, E, and S are identified as hub nodes by procedure `FIND-MAX-CLIQUES` and will be processed in a subsequent recursive call of the same procedure. Figure 2 shows a possible decomposition of the network in eleven blocks obtained by focusing on the remaining non-hub nodes. In the figure kernel nodes are white, border nodes are green and visited nodes are double-marked. Note that all feasible nodes (white-filled in Figure 1) occur in exactly one block as kernel nodes (white-filled in Figure 2). Also, observe that each block of Figure 2 includes all the neighborhood of the kernel nodes. However, the hub nodes (D, E, and S) never occur as kernel nodes in any block. Instead, their neighborhood has been distributed among the various blocks. Finally, note that every maximal clique occurs in at least one block: this is an important property that allows us to independently process each block. If a maximal clique occurs in more than one block, only the occurrence without visited nodes will be considered. This is the case, for instance, for the maximal clique {H, F, D} that is detected both when processing $B1$ and when processing $B2$, but is discarded in the latter case since it contains a visited node.

We start to build a block B by picking a node n from N_f (line 4 of Algorithm 3) and adding it to the set K of kernel nodes of B (line 8). We then build: (i) the set V of visited nodes (line 9), composed of neighbors of nodes in K that are already used as kernel nodes in a previously defined block (we maintain these latter nodes in $\bar{K}$, which is updated at line 7), and (ii) the set H of border nodes (line 10), composed of neighbors of K that are not yet visited. Then we proceed by selecting one node of N_f that is a border node of B and transforming it into a kernel node of B (line 10). In order to produce blocks that correspond to dense graphs, we order the candidate border nodes based on the number of their adjacency with kernel nodes, and we stop either if we exceed the limit m by adding further nodes (line 5) or if all candidate border nodes have a number of adjacency with kernel nodes below a specified threshold.

4. MAXIMAL CLIQUES COMPUTATION

In order to find all maximal cliques in a block of the decomposition, we rely on a framework that leverages on a collection of algorithms taken from the literature with the goal of improving the overall performance of the computation.

The MCE problem has been subject of extensive study since the early 70's [6, 8, 10, 17, 21, 23, 34]. None of the available algorithms outperforms the others in every possible instance of the problem. However, some approaches tend to excel on graphs having specific properties. For example Eppstein *et al.* [17] propose an algorithm that runs in near-optimal time on graphs having small degeneracy[2]. On the contrary, this algorithm does not perform well on dense graphs where the degeneracy tends to be higher. On these graphs, the algorithm proposed by Tomita *et al.* [34] tends to be more efficient.

Algorithm 3: `BLOCKS`: Second-level decomposition

```
Input : A graph $G = \langle N, E \rangle$, a set $N_f$ of feasible
        nodes, and a block size m.
Output: A set of blocks $\mathcal{B}$.
1  $\bar{K} \leftarrow \emptyset$; $\mathcal{B} \leftarrow \emptyset$;
2  while $N_f \neq \emptyset$ do
3      $K, H, V \leftarrow \emptyset$;
4      $n \leftarrow$ select($N_f$);
5      while isfeasible($K \cup \{n\}$,$G$,m) do
6          $N_f \leftarrow N_f - \{n\}$;
7          $\bar{K} \leftarrow \bar{K} \cup \{n\}$;
8          $K \leftarrow K \cup \{n\}$;
9          $V \leftarrow N(n) \cap \bar{K}$;
10         $H \leftarrow N(n) - V$;
11         $n \leftarrow$ select($N_f \cap H$);
12     $\mathcal{B} \leftarrow \mathcal{B} \cup$ induced($G$, $K \cup H \cup V$);
13 return $\mathcal{B}$;
```

Our approach attempts at predicting, for each block, the *best-fit* among the available MCE algorithms, that is the one that achieves the best performance on it. The intuition behind this approach is that large heterogeneous networks yield blocks with very different characteristics, so that any algorithm would be suboptimal in a non-negligible portion of the blocks.

In order to efficiently predict the best-fit algorithm for a block, we first identified a set of easy-to-compute parameters to describe the block properties. Second, we selected a set of supporting data-structure and state-of-the-art MCE algorithms. Third, we measured the performance of each combination of data-structure/algorithm on a collection of heterogeneous graphs. Finally, we used the results as a training set to produce a decision tree aimed at selecting the best combination for a given block.

The parameters we used to classify blocks are the following: (a) number of nodes; (b) number of edges; (c) density; (d) degeneracy; and (e) the maximum value d^* for which the graph has at least d^* nodes with degree greater or equal than d^*. Parameter d^* can be computed in linear time and, intuitively, provides an estimate of the size of the densest portion of the graph, which we expect to dominate the performance of a search algorithm.

We considered three different data structures to represent the graph: adjacency matrices, bitsets, and adjacency lists (the latter including the inverted-table structure described in [17]).

As for the MCE algorithms, we implemented the following:

- **BKPivot**: one of the original algorithms proposed by

[2] See Section 5 for a formal definition of degeneracy.

Algorithm	Matrix	Lists	BitSets
`BKPivot [6]`	7	0	2
`Tomita [34]`	5	3	12
`Eppstein [17]`	0	2	0
`XPivot`	7	12	0

Table 1: Performance of the MCE algorithms.

Bron and Kerbosch [6]. It uses a pivot to avoid redundant recursive calls. The node of highest degree in the candidate set P is chosen as the pivot.

- **Tomita**: a variation of BKPivot by Tomita *et al.* [34]. It uses as pivot the node u that maximizes the size of $N(u) \cap P$, where $N(u)$ denotes the neighborhood of u.
- **Eppstein**: the algorithm by Eppstein and Strash [17]. It is based on a degeneracy ordering of the nodes to achieve a better complexity on sparse graphs.
- **XPivot**: a variation of BKPivot proposed by us. Like Tomita, it chooses the node that maximizes the size of $N(u) \cup P$, but the node u is chosen from the set of already visited nodes.

In Table 1 we show a performance comparison of the data-structure/algorithm combinations described above on a collection of 50 graphs, both synthetic (generated according to the models of Erdös-Renyi, Barabási-Albert and Watts-Strogatz models [2]) and real-world (taken from the SNAP project [22]). In particular, the table shows how many times a specific combination resulted the best performing among all the alternatives. It is apparent that no algorithm outperforms all the others in all cases.

Table 2 shows the maximum and minimum values of the adopted parameters in the collection and confirms that the graphs have heterogeneous properties.

Metric	Min value	Max value
`nodes`	50	685230
`edges`	199	6649470
`density`	0.00027	0.89
`degeneracy`	10	266
`d`*	15	713

Table 2: Ranges of adopted parameters for the chosen graphs.

We divided the graph collection in training and testing set with an 80/20 ratio. We then used the training set and the above parameters to generate the decision tree in Figure 3, launching the *recursive partitioning* algorithm in [32]. Each internal node of the tree contains a predicate on the parameters and has two children, associated with the predicate being `true` or `false` on the current block. Each leaf of the decision tree contains a data-structure/algorithm combination. Traversing the tree from the root to a leaf according to the values of the predicates yields the data-structure/algorithm combination that is the best-fit for the block.

The testing set was used to evaluate the effectiveness of this approach. Figure 4 shows the total time taken by our approach to process the testing set and the five best performing combinations. Note that the use of the decision tree achieves better performance than any other algorithm taken singularly.

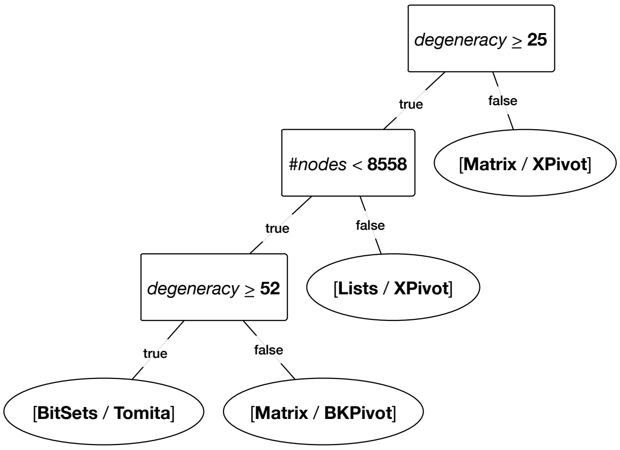


Figure 3: The decision tree for selecting the most suitable MCE algorithm.

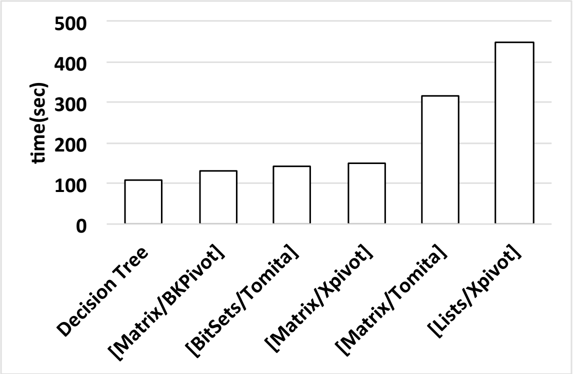


Figure 4: Times to compute cliques with or without a decision tree.

Algorithm 4 describes in detail the `BLOCK-ANALYSIS` procedure that computes all maximal cliques of the block given as input.

First, a suitable `MCE` procedure is identified by using the decision tree described above (line 1).

As described in Section 3.2, the purpose of Algorithm 4 is to find all maximal cliques that have at least one node in K, but no node in the set V of visited nodes of the input block. In line 3 we initialize $\bar{V}$ with V. For each node k in the set K of kernel nodes of the input block, Algorithm `MCE`$(k, P, \bar{V})$ enumerates all maximal cliques that contain k and no node in $\bar{V}$ as long as all the neighbors of k are in $P \cup \bar{V}$. One can observe that all neighbors of k are either in the set H of border nodes of the input block, or in K or in $\bar{V}$. Therefore, procedure `BLOCK-ANALYSIS` detects all maximal cliques containing a node of K and no node in V. Finally, after k is visited, it is added to $\bar{V}$ since all cliques containing k have been found.

5. THEORETICAL BASIS

In this section we prove under what conditions our approach is correct and complete. Namely, Lemma 1 proves

Algorithm 4: `BLOCK-ANALYSIS`: Clique detection

```
Input  : A block $\langle N = K \cup H \cup V, E\rangle$.
Output: The maximal cliques $\mathcal{C}$ of $B$ that have at least
        one node in $K$, but no node in $V$.
MCE $\leftarrow$ bestfit($B$);
$P \leftarrow K \cup H$;
$\bar{V} \leftarrow V$;
foreach $k \in K$ do
    $N_k \leftarrow N(k) \cap P$;
    $\mathcal{C} \leftarrow \mathcal{C} \cup$ MCE($k, P \cap N_k, \bar{V} \cap N_k$);
    $P \leftarrow P - \{k\}$;
    $\bar{V} \leftarrow \bar{V} \cup \{k\}$;
return $\mathcal{C}$;
```

that `FIND-MAX-CLIQUES` (Algorithm 1 in Section 3) actually computes all maximal cliques of the input network. Theorem 1, instead, shows that `FIND-MAX-CLIQUES` terminates its recursive calls whenever the input network is sparse.

Sparsity is a well known property of social networks and can be formally measured in terms of their low *degeneracy* [35]. The degeneracy of a network, also called *coreness*, is the highest value d for which the network contains a d-core[3]. Hence, a network with a low degeneracy is inherently sparse. The degeneracy of a network can be easily computed, even in a distributed environment (see, e.g., [4]), and it is usually much lower than the maximum degree of the network. Indeed, real-world social networks have low degeneracy [35].

Lemma 1. *Let N_1 and N_2 be any bipartition of the nodes of a graph G. Let C_1 be the set of the maximal cliques of G containing at least one node of N_1 and let C_2 be the set of maximal cliques of the subgraph of G induced by the nodes in N_2. The set of the maximal cliques of G is the union of C_1 and the set C_2' obtained by filtering out from C_2 any clique that is contained into a clique of C_1.*

Proof. Let K be a maximal clique of the network. We show that K is in $C_1 \cup C_2'$. We have two cases: (i) at least one node of K is in N_1 or (ii) all nodes of K are in N_2. In the first case K belongs to C_1 and, hence, it is also in the union of C_1 and C_2'. In the second case K is in C_2 and, since by hypothesis K is maximal, it is also in C_2', and hence in the union of C_1 and C_2'.

Conversely, let K be a clique in the union of C_1 and C_2'. We show that K is a maximal clique. Suppose, for a contradiction, that K' is a clique containing K and having a vertex v in addition to the vertices of K. One of the following three cases applies: (a) at least one node of K belongs to N_1; (b) all nodes of K belong to N_2 and v also belongs to N_2; or (c) all nodes of K belong to N_2 and v belongs to N_1. In Case (a), both K and K' belong to C_1, contradicting the hypothesis that C_1 is composed of maximal cliques. In Case (b), both K and K' belong to C_2, contradicting the hypothesis that C_2 is composed of maximal cliques. Finally, in Case (c), K belongs to C_2 while K' belongs to C_1. However, since K is contained into K', K does not belong to C_2', contradicting the hypothesis that K belongs to the union of C_1 and C_2'. □

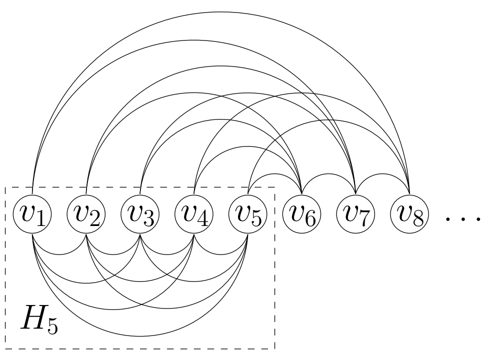


Figure 5: The construction for $m = 4$ of graph H_n used to prove Statement 2 of Theorem 1.

The following theorem shows that, if the network is sparse, the recursive algorithm `FIND-MAX-CLIQUES` converges, in the sense that it ends with a bipartition involving only tractable nodes.

Theorem 1. *Let G be a graph and let G_i, with $i = 1, 2, 3\ldots$ be a sequence of subgraphs of G such that $G_1 = G$ and G_i, for $i > 1$ is the graph induced by the nodes of G_{i-1} of degree greater or equal than m. Let the degeneracy d of G be strictly less than $m + 1$.*

1. *There is a value q such that all G_j, with $j \geq q$, are empty graphs.*
2. *There exists a graph with n nodes for which q is $\Omega(n)$.*

Proof. Statement 1 is proved by observing that graphs G_i, with $i > 1$, are obtained from G by iteratively removing nodes of degree less or equal than m. For i large enough, such iterative removal coincides with a recursive removal and, hence, leads by definition to the $(m + 1)$-core of G, which is the empty graph since $d < m + 1$.

Statement 2 is proved by producing a graph H_n with n nodes and whose degeneracy is $d < m+1$ such that $q \in \Omega(n)$, as follows. Start from H_1 composed of the isolated node v_1 and, for $j = 2, 3, \ldots, n$, obtain H_j by adding a node v_j to H_{j-1}. For $j \leq m + 1$ connect v_j to all previously inserted nodes, so that, H_j, with $j \leq m + 1$, is a complete graph on the first j nodes (see Figure 5 where $m = 4$). For $j > m + 1$ connect v_j to the previous m nodes that have lower degree. It is easy to check that:

(a) v_j has degree m in H_j, for any $j > m+1$. For example, in Figure 5 node v_6 has degree 4 in H_6.

(b) v_{j-1} has degree $m + 1$ in H_j, for any $j > m + 2$. For example, in Figure 5 node v_6 has degree 5 in H_7.

(c) $v_1, v_2, \ldots, v_{j-2}$ have degree greater than m in H_j, for any $j > m + 3$. For example, in Figure 5 nodes $v_1, v_2, \ldots, v_6$ have degree greater than 4 in H_8.

Therefore, for $j > m+3$, the three conditions (a),(b), and (c) hold and the removal of all nodes of degree less or equal than m from H_j only removes v_j, yielding H_{j-1}. This implies that: (i) recursively removing all nodes of degree less or equal than m from H_n yields the empty graph, i.e., the degeneracy of H_n is less than $m + 1$ and (ii) $\Omega(n)$ removals are needed to obtain the empty graph from H_n. □

[3] The *d-core* of a graph is obtained by recursively removing nodes with degree less than d.

Network	# of nodes	# of edges	Maximum degree
twitter1	2,919,613	12,887,063	39,753
twitter2	6,072,441	117,185,083	338,313
twitter3	17,069,982	476,553,560	2,081,112
facebook	4,601,952	87,610,993	2,621,960
google+	6,308,731	81,700,035	1,098,000

Table 3: The data sets used in the experimentation.

Theorem 1 proves that, in order to guarantee that all maximal cliques are detected, FIND-MAX-CLIQUES only requires that m is chosen to be greater than $d-1$, where d is the degeneracy of the network (Section 6 shows how to pick a good value for m). We remark that, although the very special graph described in the proof of Theorem 1 requires $\Omega(n)$ recursive steps, in all our experiments with real-world data sets the process needed at most a few of them (see Section 6).

6. EXPERIMENTAL RESULTS

We implemented our approach for maximal clique enumeration into a C++ system using OpenMPI v1.8 library. This section reports the results of the experimentation of the system.

6.1 Benchmark Environment

We deployed our system on a 10-nodes time-shared cluster, where each machine is equipped with 8 GB DDR3 RAM, 4 CPUs 2.67 GHz Intel Xeon with 4 cores and 8 threads, running Scientific Linux 5.7, with the TORQUE Resource Manager process scheduler. The system is provided with the Lustre file system v2.1. The performance of our system has been measured with respect to data loading (i.e., decomposition), and all maximal cliques computation time (i.e., block analysis).

For our experiments we used some of the largest available social networks (see Table 3) taken from SNAP [22] and from the Koblenz Konect repositories[4]. In particular we considered three portions of the "follower network" of Twitter (labeled twitter1, twitter2 and twitter3 in Table 3), the friendship network of Facebook enriched with posts to user's wall (labeled facebook), and "circles" data from Google+ (labeled google+). All these data sets are scale-free networks and provide a significant number of hub nodes. Figure 6 shows a truncated degree distribution of all considered data sets: as discussed in the Introduction, all networks follow a power law for which most of the nodes (i.e. 91% of the total, on average) provide a degree included in the range $[1, 20]$. Nevertheless, on average, in each data set the amount of possible hub nodes (i.e. they provide the maximum degree) represents the 3% of the total set of nodes.

6.2 Network Decomposition

We distributed the input data set among the ten machines of our cluster: each data set is locally split into files whose records contain triples in the format $\langle n_1, e, n_2 \rangle$, where n_1 and n_2 are the labels of the nodes and e is the label of the edge between them. To speed-up the process we encoded node and edge labels with hashes.

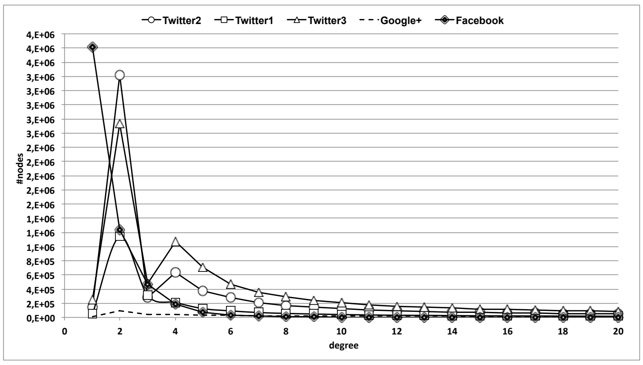


Figure 6: Truncated degree distribution of data sets.

On each data set we ran Algorithm FIND-MAX-CLIQUES three times on each machine and measured the average time used to produce the blocks (including the I/O time). Figure 7 shows for each data set the average time to perform the two-level decomposition with respect to the ratio m/d, where m is the maximum number of nodes in a block and d is the maximum node degree. In the experiment we considered five ratios (i.e. 0.9, 0.7, 0.5, 0.3, and 0.1) obtained by decreasing m. As the block size limit decreases, the number of blocks increases and consequently it increases also the time to perform the decomposition. It also causes the increase of the number of hub nodes as well as the increase of the number of maximal cliques involving hubs (see Section 6.3).

We remark that for $m/d \in \{0.5, 0.9\}$ all data sets required two iterations of the first-level decomposition, while for $m/d \in \{0.1, 0.3\}$ all data sets were decomposed after three iterations. This confirms what formally enunciated in Theorem 1. The results in Figure 7 confirm the feasibility of the approach.

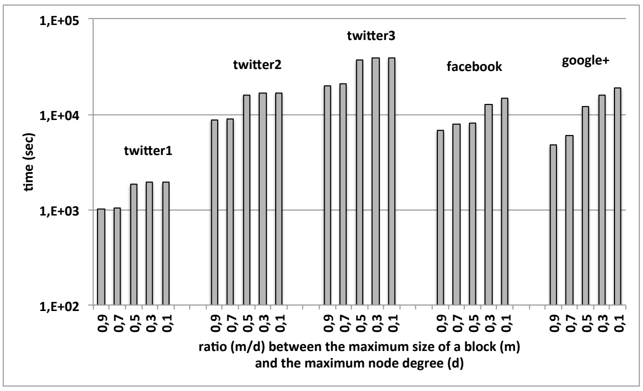


Figure 7: Times to compute the decomposition.

6.3 Clique computation

For evaluating the computation times of our approach we ran Algorithm BLOCK-ANALYSIS three times on all blocks and measured the average overall time (including the I/O time). Figure 8 shows the average response time in seconds to compute all maximal cliques with respect to the values 0.9, 0.7, 0.5, 0.3, and 0.1 for m/d. All times refer to a serial pro-

[4] Available at http://konect.uni-koblenz.de/downloads/\#rdf

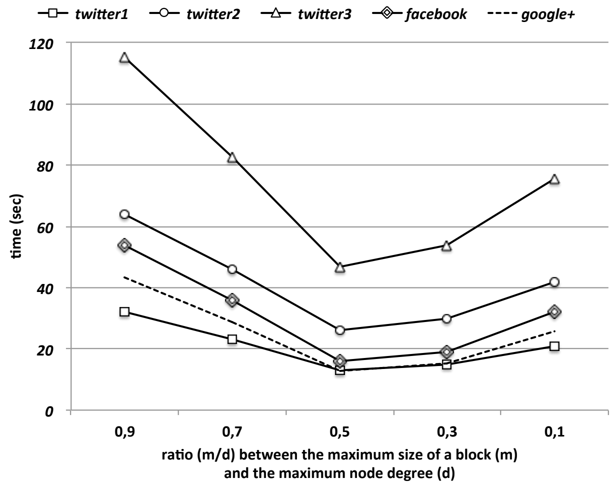


Figure 8: Times to compute all maximal cliques.

cessing (i.e., they do not account for the speed-up due to simultaneous computations on distributed platforms).

Efficiency. Our experiments confirm that the running times benefit from relatively small values of m. The fact that the overall performance is improved when smaller blocks are involved is likely due to the efficiency of the clique detection algorithms on small instances. Hence, it can be argued that the decomposition phase is playing the role of a preprocessing step for the MCE problem, producing blocks that can be regarded as approximate solutions to be refined by an exact MCE algorithm.

For small values of m/d (i.e., 0.3 and 0.1) we have many blocks and the performance of the entire process are affected by an increasing overlap among the neighborhood of each block and an increasing communication overhead among the machines of the cluster. As shown in Figure 8, the value $m/d = 0.5$ is a common "saddle point" for all data sets.

Effectiveness. Figure 9.(a) and Figure 10.(a) show the number of cliques computed with respect to the same five ratios used above and Figure 9.(b) and Figure 10.(b) show the average size of the cliques. In all the figures, white bars denote maximal cliques computed from the blocks built from the feasible nodes, while gray bars refer to maximal cliques computed from the blocks built from the hub nodes. Figure 9.(a) and Figure 10.(a) clearly show the contribution of our approach: in all the experiments we had a non-negligible number of maximal cliques involving hub nodes only, that could be omitted or could induce the erroneous detection of non-maximal cliques if the techniques described in this paper were not adopted. In particular, as the ratio between the block size and the maximum node degree decreases, the portion of maximal cliques involving only hub nodes is significantly increased (i.e. reducing m artificially increases the number of hub nodes).

Figure 9.(b) and Figure 10.(b) focus on the size of the produced cliques. It turns out that the sizes of the cliques involving only hub nodes are comparable with (and, in average, greater than) the sizes of the cliques involving feasible nodes. This is more apparent when the ratio m/d is smaller (i.e., 0.3 and 0.1). Furthermore, observe that the cliques involving only hub nodes are comparable in size with the biggest cliques contained in the network. Hence, even when the cliques computed on the hub nodes are a small percentage, they are among the most significant when their size is considered.

In order to better estimate how much significant are the maximal cliques composed exclusively of hub nodes, we focused on the 200 largest maximal cliques. Figure 11 shows the percentage of maximal cliques computed on the feasible nodes and the percentage of maximal cliques computed on the hub nodes (with respect to the same five values of m/d used for Figures 9 and 10). The percentage of maximal cliques computed on the hub nodes grows significantly around the value $0.5m/d$. In particular, for values of $m/d \in [0.1, 0.5]$, the percentage of maximal cliques computed on hub nodes is between 20% and 80% for all data sets. This confirms that decreasing the block size for boosting efficiency has a dramatic impact on the number of significant maximal cliques that would be lost if the techniques described in this paper were not adopted.

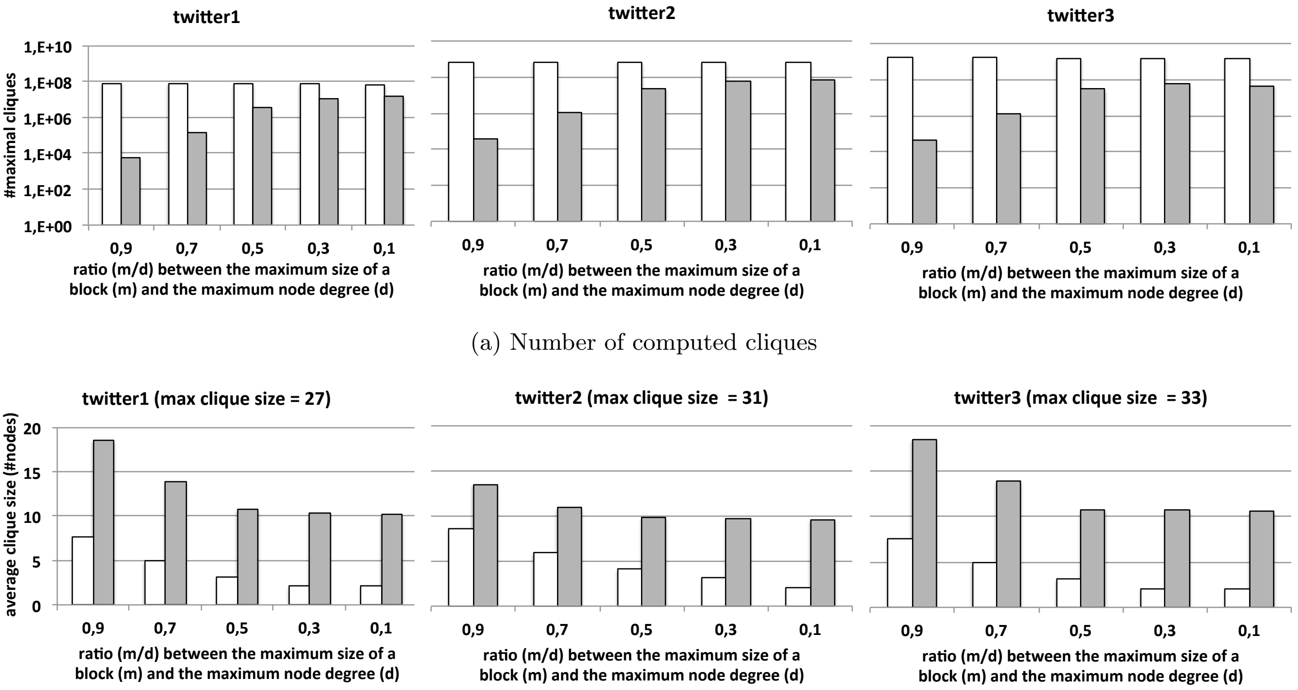
7. RELATED WORK

Despite a long research history, the MCE problem has recently re-emerged as one of the key research topics of graph mining. Due to the NP-completeness of the problem, traditional algorithms for enumerating maximal cliques rely on pruning techniques in order to reduce the search space and speed up the execution [27, 33]. With the increasing dimension of nowadays social networks such algorithms are not satisfactory anymore because the size of the input network often exceeds the available memory.

To address this issue, new approaches have been introduced [8, 10, 30, 36, 38, 7]. They usually rely on a decomposition phase that splits the graph into (partially overlapping) blocks and on distributed computation on the independent blocks to detect all the maximal cliques therein.

`ExtMCE` [8, 38] is the first algorithm that handles graphs that do not fit into main memory. It starts the search for maximal cliques from a sub-portion of the whole graph, called H*-graph. However, `ExtMCE` works under the assumption that the H*-graph fits into main memory, which may be again too restrictive with real-world networks.

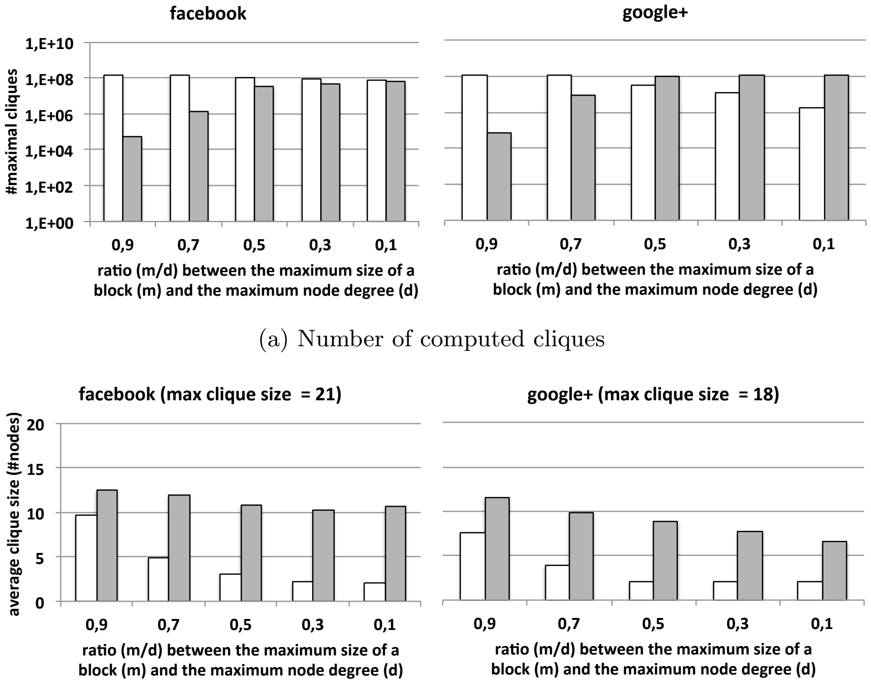
The same authors improved their approach introducing the `EmMCE` algorithm [10] that takes advantage of parallelization to reduce I/O overhead and to distribute computation loads. As confirmed also by our experimentation, in [10] it is shown that producing blocks of much smaller size than the available memory yield better time performance. At the same time, though, algorithm `EmMCE` assumes that the neighborhood of each node fits within a block. This clearly poses a trade-off between efficiency and correctness. In fact when the neighborhood of a node does not fit into a single block some of its maximal cliques may be discarded and some non-maximal cliques could be erroneously detected. Furthermore, even if efficiency was not an issue, correctness and completeness are lost whenever the graph has nodes of degree so high that their neighborhood does not fit into main memory. Trying to address this problem in [10] it is suggested to decompose the graph considering nodes in increasing degree order. This results into artificially augmenting the size of a graph fitting into a block, since, when a hub node is chosen as a kernel node, its neighborhood would be



(a) Number of computed cliques

(b) Average number of nodes per clique

Figure 9: The results of the experimentation on `twitter1`, `twitter2`, and `twitter3` data sets. White bars refer to cliques computed from the feasible nodes while gray bars refer to cliques containing only hub nodes.



(a) Number of computed cliques

(b) Average number of nodes per clique

Figure 10: The results of the experimentation on `facebook` and `google+` data sets. White bars refer to cliques computed from the feasible nodes while gray bars refer to cliques containing only hub nodes.

largely composed by visited nodes, and edges among visited nodes could be omitted from the block. Nevertheless, all neighbors of a hub node need to be stored into the block as before, and the block size limit still hinder correctness.

Chang *et al.* [7] find all maximal cliques in polynomial delay: cliques are found one after the other and the time-complexity of finding the next clique in the sequence is polynomial. They improve over the previously polynomial-delay fastest algorithm for MCE [24] by using a strategy that partitions the graph into low and high degree nodes.

The authors in [38] focus on the skews of the parallel computation of cliques, since the analysis of few blocks takes far more time than the rest. They also propose algorithms that can incrementally update the maximal cliques when the graph is updated.

Xing *et al.* [36] use a recursive algorithm, called BMC, for partitioning the network into blocks. Then, they use an algorithm based on MapReduce to compute the cliques present in each block. Since BMC generates blocks having similar size, inter-block cliques are skipped and the approach is not complete. Gregori *et al.* [20] and Rossi *et al.* [30] find the maximal k-cliques and the largest cliques in a parallel way, respectively. These approaches can not be adapted to find all maximal cliques.

Computations over massive networks often take advantage of distributed graph processing systems such as GraphLab/PowerGraph [19] and Pregel/Giraph [25]. They provide: (a) a fault-tolerant infrastructure for processing distributed data; (b) a graph partitioning technique; and (c) an abstract computational model for implementing algorithms. While we could benefit from the infrastructures and abstract models, the partitioning techniques of such systems (point (b) above) are not suitable in the MCE context. They usually use random partitioning (i.e., hash partitioning) which is proven to be the worst possible partitioning for scale-free networks [15]. Instead, as shown in Section 6, we benefit from a decomposition that produces dense chunks of different size.

Maximal Clique Enumeration is especially used for detecting communities in social networks. Several approaches for community detection, rely on a relaxed concept with respect to the enumeration of maximal cliques and consider each subgraph that approximates a clique as a community [29, 39, 28, 1]. In the remaining part of this section we briefly review some of them.

WalkTrap [28] computes random traversals to individuate communities. The heuristic idea of WalkTrap is that a random path would likely stay "trapped" inside a subgraph of highly connected nodes. The random path cliques do not give any warranty on the quality of the solutions as, choosing randomly, they might not retrieve a tight community.

There are several approaches that find communities as the subgraphs resulting from the clustering of the edges in the network (see, for example, [1]). They uniquely assign each individual to a cluster. Clearly, this assumption is not suitable for social networks where an individual may belong to multiple communities. To face this aspect, a series of works have been proposed in order to allow overlapping communities (see the survey in [37]).

Differently from all approaches mentioned above, SCD [29] employs a parallel strategy to detect the subgraphs that maximize the number of contained triangles, since this measure is indicative of how tight is a community. In [39] it is introduced the concept of *k-mutual-friend* to find communities and, additionally, it is described a system to browse the communities in a visual manner.

Finally, there are approaches that retrieve communities in terms of *k-plexes*, which are relaxations of cliques in which a node can miss at most k neighbours [5, 26].

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a novel technique for computing all the maximal cliques of an arbitrarily large network in a distributed environment. The approach relies on a two-level decomposition strategy that allows us to achieve efficiency by suitably lowering the size of the blocks without jeopardizing completeness. This is confirmed by a number of theoretical results showing the correctness and completeness of the technique over sparse graphs, a natural property of real-world social networks.

An extensive campaign of experiments conducted over real-world scenarios has shown the efficiency and scalability of our proposal. We have also demonstrated experimentally that, if our technique was not adopted, a significant portion of the most relevant cliques would have been lost.

In the future, we plan to explore the possibility of extending our approach to relaxed definitions of communities, such as k-cliques, k-clubs, k-clans, and k-plexes. We are also interested in studying an incremental version of our approach that takes into account the evolution of the social network.

Acknowledgements

The authors are grateful to Lorenzo Dolfi and Gabriele De Capoa for their contribution in the development of the tools described in this paper. Research supported in part by the MIUR project AMANDA "Algorithmics for MAssive and Networked DAta", prot. 2012C4E3KT_001.

9. REFERENCES

[1] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, 2010.

[2] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, 2002.

[3] A.-L. Barabási and E. Bonabeau. Scale-free networks. *Scientific American*, 288(5):50–59, 2003.

[4] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

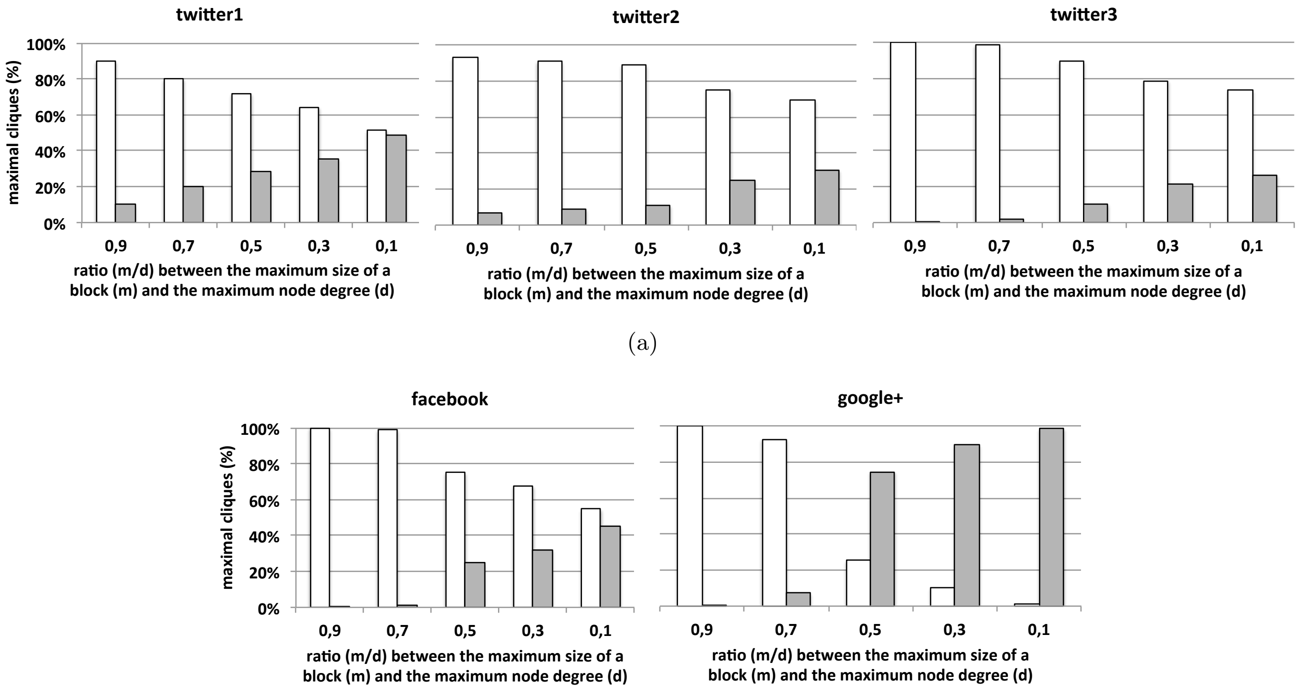
[5] D. Berlowitz, S. Cohen, and B. Kimelfeld. Efficient enumeration of maximal k-plexes. In *SIGMOD*, pages 431–444, 2015.

[6] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[7] L. Chang, J. X. Yu, and L. Qin. Fast maximal cliques enumeration in sparse graphs. *Algorithmica*, 66(1):173–186, 2013.

[8] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks by h*-graph. In *SIGMOD*, pages 447–458, 2010.

[9] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks. *ACM Trans. Database Syst.*, 36(4):21, 2011.



(a)

(b)

Figure 11: An analysis of the 200 largest maximal cliques of each data set. White bars represent the percentage of these cliques computed from the feasible nodes while gray bars represents the percentage of these cliques containing only hub nodes.

[10] J. Cheng, L. Zhu, Y. Ke, and S. Chu. Fast algorithms for maximal clique enumeration with limited memory. In *KDD*, pages 1240–1248, 2012.

[11] K. Choromański, M. Matuszak, and J. Miekisz. Scale-free graph with preferential attachment and evolving internal vertex structure. *Journal of Statistical Physics*, 151(6):1175–1183, 2013.

[12] A. Cui, Z. Zhang, M. Tang, and Y. Fu. Emergence of scale-free close-knit friendship structure in online social networks. *CoRR*, abs/1205.2583, 2012.

[13] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[14] N. Du, B. Wu, L. Xu, B. Wang, and X. Pei. A parallel algorithm for enumerating all maximal cliques in complex network. In *ICDM Workshops*, pages 320–324, 2006.

[15] Q. Duong, S. Goel, J. M. Hofman, and S. Vassilvitskii. Sharding social networks. In *WSDM*, pages 223–232, 2013.

[16] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *ISAAC*, pages 403–414, 2010.

[17] D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. In *SEA*, pages 364–375, 2011.

[18] S. Fortunato. Community detection in graphs. *CoRR*, abs/0906.0612, 2009.

[19] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, pages 17–30, 2012.

[20] E. Gregori, L. Lenzini, and S. Mainardi. Parallel k-clique community detection on large-scale networks. *Trans. Parallel Distrib. Syst.*, 24(8):1651–1660, 2013.

[21] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theor. Comput. Sci.*, 250(1-2):1–30, 2001.

[22] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2015.

[23] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In T. Hagerup and J. Katajainen, editors, *Algorithm Theory - SWAT 2004*, volume 3111 of *Lecture Notes in Computer Science*, pages 260–272. Springer Berlin Heidelberg, 2004.

[24] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *SWAT*, pages 260–272, 2004.

[25] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.

[26] B. McClosky and I. V. Hicks. Combinatorial algorithms for the maximum k-plex problem. *J. Comb. Optim.*, 23(1):29–49, 2012.

[27] P. R. J. Östergård. A fast algorithm for the maximum

clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[28] P. Pons and M. Latapy. Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, 10(2):191–218, 2006.

[29] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *WWW*, pages 225–236, 2014.

[30] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, and M. M. A. Patwary. Fast maximum clique algorithms for large graphs. In *WWW (Companion Volume)*, pages 365–366, 2014.

[31] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs. In *SIGKDD*, pages 1222–1230, 2012.

[32] T. M. Therneau and E. J. Atkinson. An introduction to recursive partitioning using the RPART routines. Technical report, Division of Biostatistics 61, Mayo Clinic, 1997.

[33] E. Tomita and T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. Global Optimization*, 44(2):311, 2009.

[34] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, 2006.

[35] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[36] J. Xiang, C. Guo, and A. Aboulnaga. Scalable maximum clique computation using mapreduce. In *ICDE*, pages 74–85, 2013.

[37] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Comput. Surv.*, 45(4):43, 2013.

[38] Y. Xu, J. Cheng, A. W. Fu, and Y. Bu. Distributed maximal clique computation. In *International Congress on Big Data*, pages 160–167, 2014.

[39] F. Zhao and A. K. H. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB*, 6(2):85–96, 2012.