Large-Scale Clique Cover of Real-World Networks☆

Alessio Conte

Dipartimento di Informatica, Università di Pisa, Italy

Roberto Grossi

Dipartimento di Informatica, Università di Pisa, Italy

Andrea Marino

Dipartimento di Statistica, Informatica, Applicazioni, Università di Firenze, Italy

Abstract

The edge clique cover (ECC) problem deals with discovering a set of (possibly overlapping) cliques in a given graph, such that each edge is part of at least one of these cliques. This problem finds applications ranging from social networks to compiler optimization and stringology. We consider classical and new variants of the ECC problem, addressing the quality of the cliques found and proposing more structured criteria other than traditional measures (such as minimum number of cliques). We describe fast practical algorithms for implementing some heuristics, the fastest one taking $O(md_G)$ time for a connected graph with m edges and degeneracy $d_G \leq \min\{\Delta, 2\sqrt{m}\}$ (also known as k-core number), where Δ is the maximum node degree. For real-world networks, with millions of nodes, such as social networks, the possibility of getting a result is constrained to the running time, which should be (almost) linear in the size of the network. Our algorithm for finding ECCs of large networks has linear-time performance in practice because d_G is small, as our experiments show on real-world networks whose number of nodes ranges from thousands to several millions.

Keywords: Edge Clique Cover, Network analysis, Graph algorithms

1. Introduction

Consider an undirected connected graph $G = (V, E)$ with $m = |E|$ edges and $n = |V|$ nodes. Its cliques are subsets of nodes $C \subseteq V$ that are pairwise adjacent (i.e. there is an edge $\{u, v\} \in E$ for every pair of nodes $u, v \in C$, $u \neq v$): we equivalently see a clique as its set C of nodes, or its corresponding induced subgraph $G[C] = (C, \{\{u, v\} \in E \mid u, v \in C\})$. The edge clique cover (ECC) problem deals with selecting a set of (possibly overlapping) cliques in G, such that each edge in E belongs to at least one of these cliques. Figure 1 shows two example of ECCs. This problem and its variants have many applications, such as applied statistics [2, 3], behavioral and cognitive networks [4], compiler optimization [5], complex networks [6, 7], computational biology [8], computational geometry [9], dynamic networks [10], email networks [11], financial networks [12], SAT solvers [13], security [14], social networks [15], and stringology [16].

In the literature, problems related to ECC have been independently addressed in the mid 70's in social network analysis [18, 19]. From a computational point of view, the related problem of vertex clique cover, where the cliques are disjoint and the vertices are to be covered, is one of Karp's original problems shown to be NP-complete in his

☆A shorter and preliminary version of this paper is available in [1]. This paper adds an improved analysis and experimental study of several variants. Work partially supported by JST CREST, Grant Number JPMJCR1401, Japan.

Email addresses: conte@di.unipi.it (Alessio Conte), grossi@di.unipi.it (Roberto Grossi), andrea.marino@unifi.it (Andrea Marino)

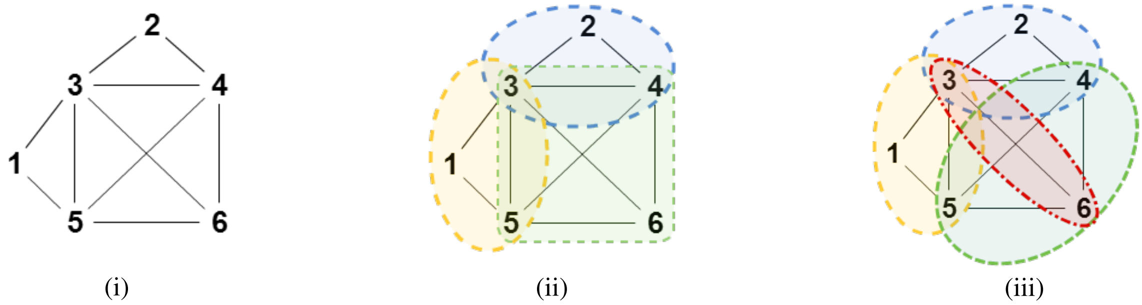


Figure 1: Graph (i) taken from [17], and two examples of ECCs in (ii) and (iii).

famous 1972 paper. The ECC appeared with different terminology such as keyword conflict [20], covering by cliques, intersection graph basis, or clique partition [21], as discussed in [22]. The latter reports that the problem of finding an ECC having the *minimum*[1] number of cliques is NP-hard [23], even for planar graphs [24] or graphs with degree at most 6 [25]. It can be solved polynomially for some restricted classes of graphs [25, 26, 27, 23]. As it can be seen, the previous work mainly aims at minimizing the number of covering cliques for this NP-hard problem. The parameterized exact algorithms at the state of the art [28] are probably optimal [22]. However, from the experiments reported in the paper, it seems that they do not scale well to large graphs. The computational difficulty of the problem is also due to the fact that it is not approximable within a factor of n^ϵ, where n is the number of vertices and $\epsilon > 0$, unless P=NP [29].

When considering large real-world networks some heuristic algorithms for cliques work surprisingly well (e.g. [30, 31, 32, 33, 34]), whereas approximation algorithms with some guarantee are still too expensive. Due to the massive size of available networks, it is convenient to consider algorithms or heuristics taking (almost) linear time in the size of the network. This comes as an invitation to study the ECC problem on these networks. Looking at the literature from this point of view, Gramm et al. [35] give an efficient implementation of the original Kellerman heuristics [20], with the post-processing step of Kou et al. [17]. Interestingly, even if this heuristic dates back to 1973, its effectiveness in terms of quality of the solution (i.e. the minimal number of cliques found) is still the state of the art. However, these algorithms do not have linear bounds. For a graph with n vertices and m edges, the original results by Kellerman finds a clique cover in $O(nm^2)$ time. The post-processing of Kou et al. requires at least $\Omega(nm)$ time in the worst case. Gramm et al. show how to obtain the same heuristics solution just in total $O(nm)$ time, using $O(n^2)$ space.

In this paper, we present new algorithmic tools for heuristics on ECCs that are tailored for large real-word networks. Our view is well represented by the *clique cover graph* shown in Figure 2 for the example in Figure 1. This bipartite graph has nodes representing the cliques of the ECC on the left, and nodes representing the vertices of the input network on the right. There is an edge to indicate that a given vertex is part of the given clique. On the right, we may represent the edges of the network rather than the nodes. This graph is reminiscent of the clique representation in intersection graph theory [36], and its formal definition is given in Section 2.

The classical well-studied measure in the ECC problem is the minimum number of cliques, which corresponds to having the minimum number of nodes on the left side of the clique cover graph of the candidate ECCs. Recently also the minimum assignment objective has been proposed, which minimizes the sum of the sizes of the cliques [37]: this corresponds to minimizing the number of edges in the clique cover graph. Further information on the quality of the solution can be obtained from the cover graph, which suggests using other quality measures when evaluating heuristics for the ECC problem. For example, we suggest in Section 2 to use also the weight of an ECC, the clique size distribution, or the cover index distribution.

Our first contribution is a simple, yet very effective, framework to design fast algorithmic tools that scale well for real-world large graphs and perform well according to a variety of measures based on the clique cover graph, when compared to the state of the art. They require $O(m+n)$ space and their time cost can be upper bounded as $O(md_G)$ time for a connected graph G with m edges and degeneracy $d_G \leq \min\{\Delta, 2\sqrt{m}\}$, where Δ is the maximum node degree.[2]

[1]Finding any ECC is easy, for example the trivial one made up of just the single edges.

[2]The degeneracy d_G (also known as k-core number) of a graph G is the smallest g such that every subgraph of G has a vertex of degree at most

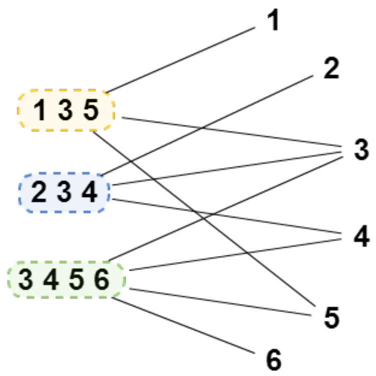


Figure 2: Clique cover graph for the ECC in Figure 1(ii).

The actual running time is linear in m in our experiments as d_G is small in real-world networks. Our algorithms require $O(m+n)$ space and their time cost can be upper bounded as $O(m \log \Delta + k\,\Delta\, d_G)$, where $k \leq m$ is the number of cliques in the ECC. This cost becomes $O(m\, d_G)$ time in some cases, thus improving performance, i.e. time and space needed to get the solution.

Our second contribution is an experimental study of heuristics for the ECC problem on real-word networks that is based on the clique cover graph: we do not merely look at the minimization problem as we consider multiple measures on the cliques emerging from an ECC. We perform an experimental analysis on a set of real-world and synthetic networks: the comparison among 20 variants of our algorithms and the state of the art involved a data set of 44 networks to select the best variants. We adopted some measures based on the clique cover graph, and compared with the state of the art on an extensive data set. For example, ECC-`rc`, one of our most effective algorithms, is faster than existing methods and it improves in practice the quality of the solution, also according to the traditional measures such as the minimum number of cliques in ECC.

We hope that our findings can inspire further work on network analysis that uses clique cover graphs, as the latter ones could be customized to deal with new quality measures that depend on the application domains. The paper is organized as follows. Section 2 defines the edge clique cover, the clique cover graph and its related quality measures. Section 3 describes our framework and discuss its correctness and complexity. Sections 4 presents our experimental study. Conclusions are drawn in Section 5.

2. Clique Cover

For our study, we consider a network as an undirected connected graph $G = (V, E)$, where V is the set of n vertices and E is the set of m edges. A clique $C \subseteq V$ is a set of vertices that are pairwise connected by edges in E, e.g., $\{u, v\} \in E$ for each pair of distinct vertices $u, v \in C$. (Equivalently, the induced subgraph $G[C]$ is complete.) In the following we will write that an edge $x, y \in C$ when both its endpoints x and y are in the clique C. A clique C is *maximal* if there is no other clique C' such that $C \subset C'$. A clique is *trivial* if it is maximal and contains only two nodes (i.e. $|C| = 2$).

An *edge clique cover* for G (in short, ECC) is a set of cliques $C_1, C_2, \ldots, C_k$ such that *(1)* no clique C_i is contained in another clique C_j, where $i \neq j$, and *(2)* for each edge $\{u, v\} \in E$ there is at least one clique C_i such that $u, v \in C_i$ [17, 35].[3] Note that an ECC always exists (e.g. choose $k = m$ and the individual edges as cliques). Maximality on each C_i is not required in the definition of ECC, but it is not difficult to see how to transform each C_i into a maximal

g. Equivalently, it is the maximum among the smallest degrees in the induced subgraphs of G. The degeneracy order is very useful for finding cliques [33, 34].

[3]Other definitions exist in the literature. In a vertex clique cover, condition *(2)* is replaced by asking that for each vertex v there is a clique C_i such that $v \in C_i$. In a clique partition, the cliques $C_1, C_2, \ldots, C_k$ form a partition of V. Note that storing $C_1, C_2, \ldots, C_k$ in place of G according to these definitions make loosing information regarding the edges of G as the single edges connecting C_i to C_j with $i \neq j$ are not stored. We focus on ECC as we can reconstruct G from it.

Algorithm 1: General framework

Input: A graph $G = (V, E)$
Output: A cover C of G
All edges in E are marked as *uncovered*
$C \leftarrow \emptyset$
while *there are uncovered edges* **do**
    $u, v \leftarrow$ `SELECT_UNCOVERED_EDGE()`
    $R \leftarrow$ `FIND_CLIQUE_OF`(u, v)
    $C \leftarrow C \cup \{R\}$
    Mark all edges of R as covered

clique, if this is needed. An ECC is *minimal* if there is no clique contained in the union of the others, namely, it is $C_i \nsubseteq \cup_{j \neq i} C_j$ for $1 \leq i \leq k$. The clique number of G is the smallest k such that an ECC of k cliques exists for G.

Beside the possible definitions of cliques, the quality of a solution, i.e. a clique set, has been often measured as its cardinality, that is, the quality of the ECC $C_1, \ldots, C_k$ is simply k. Hence, the so called MINIMUM ECC aims at minimizing the parameter k. Although finding a MINIMUM ECC has been already proved to be hard and not arbitrarily approximable, heuristics are often satisfying since in practice the quality of a solution could also depend on the application.

2.1. Clique Cover Graph

It is convenient to define a *clique cover graph* for the given ECC $C_1, \ldots, C_k$ to better highlight its properties and assess its qualities (see Figure 2). It is a bipartite undirected graph $G' = (V'_1 \cup V'_2, E')$, such that V'_1 contains a node for each clique, hence $|V'_1| = k$, and V'_2 contains a node for each vertex in V, hence $|V'_2| = n$. There is an edge $u_1, u_2 \in E'$, where $u_1 \in V'_1$ represents a clique C_i in the ECC and $u_2 \in V'_2$ represents a vertex $v \in V$, if and only if $v \in C_i$.

Depending on the application, we may define a clique cover graph with V'_2 representing the edges in E, where $|V'_2| = m$. In general, the choice of V'_2 representing either the vertices in V or the edges in E will be clear from the context.

While traditional work aims at minimizing the cardinality of V_1 (i.e., finding an ECC of minimum cardinality k or giving an approximation) and E' (i.e., minimum assignment), other possibilities can be offered. Not only we can obtain the measures previously known in the literature by looking at the clique cover graphs for the possible ECCs of G, but we can also define some additional ones in a smooth way.

1. Finding an ECC of *minimum weight* $\sum_{i=1}^{k} |C_i|$ is equivalent to finding an ECC whose clique cover graph has the minimum number of edges (see also [37]).

2. Finding the *clique size distribution* in the given ECC is equivalent to the degree distribution of the nodes in V'_1 for the corresponding clique cover graph.

3. Finding the *covering index distribution* in the given ECC, namely, computing the number y of elements from V'_2 that are contained in x cliques of the ECC, for all feasible values of x and y, is equivalent to the degree distribution of the nodes in V'_2 for the corresponding clique cover graph.

We devote a significant part of the paper (Section 4) to an experimental study of the properties of the clique cover graph in real-world networks.

3. Framework and Variants

Our general approach for finding ECCs relies on a simple algorithmic framework, summarized in Algorithm 1, taking a graph $G = (V, E)$ as input. It begins with an empty ECC C, and during the steps, cliques are added to C. At any step the edges in E are partitioned into *covered* and *uncovered*: edge $\{u, v\}$ is covered if there exists a clique $C_i \in C$ such that $u, v \in C_i$; it is uncovered otherwise. Initially, all edges in E are uncovered. As long as there are uncovered edges, one of them is chosen by `SELECT_UNCOVERED_EDGE`, described in Section 3.1. Let this edge be $\{u, v\}$. We then find a new clique R that contains u, v using `FIND_CLIQUE_OF`, described in Section 3.2. Note that the clique R must be new by definition as $\{u, v\}$ is uncovered. We then add R to C, and mark all edges $x, y \in R$ as covered.

3.1. Operation `SELECT_UNCOVERED_EDGE`

Given a node $u \in V$, we denote its set of neighbors by $N(u) = \{v \mid \{u, v\} \in E\}$ and its degree by $d(u) = |N(u)|$. We also denote the set of uncovered edges by $U \subseteq E$, and define the uncovered neighbors of $u \in V$ as $N_U(u) = \{v \mid \{u, v\} \in U\}$. Consequently, the *uncovered degree* of u is $d_U(u) = |N_U(u)|$. When $d_U(u) > 0$, we call u *eligible*.

We consider three variants of `SELECT_UNCOVERED_EDGE`, denoted by `r`, `m`, `M`, `U`,[4] assuming that the edge set U is nonempty, as shown in Table 1.

`r`	*random*: return an uncovered edge $\{u, v\}$ from U uniformly at random.
`m`	*min-degree*: choose an eligible node u of minimum degree $d(u)$, and return an arbitrary edge $\{u, v\} \in U$ (i.e. $v \in N_U(u)$).
`M`	*max-degree*: as above, except that u has maximum degree $d(u)$.
`U`	*max-uncovered-degree*: choose an eligible node u of maximum uncovered degree $d_U(u)$, and return an arbitrary edge $\{u, v\} \in U$.

Table 1: Variants of `SELECT_UNCOVERED_EDGE`

3.2. Operation `FIND_CLIQUE_OF`

Given an uncovered edge $\{u, v\}$ in the input graph G, we can find a clique R containing $\{u, v\}$ by following the idea of the Bron-Kerbosh (abbreviated BK) technique [38], as illustrated in Algorithm 2.

Algorithm 2: `FIND_CLIQUE_OF`

```
Input: A graph G = (V, E), an uncovered edge {u, v} ∈ E
Output: A clique R containing {u, v}
R ← {u, v}
P ← N(u) ∩ N(v)
z ← EXTRACT_NODE(P)
while z ≠ null do
    R ← R ∪ {z}
    P ← P ∩ N(z)
    z ← EXTRACT_NODE(P)
```

After initializing R with $\{u, v\}$, the candidate set P is built by intersecting the neighbors of u and v. Using `EXTRACT_NODE`, a node z is suitably extracted from P, if it exists (i.e. $z \neq$ *null*), and added to R as z is surely adjacent to all the vertices in R. Since z is now part of R, the vertices in the candidate set P must be also neighbors of z, so P is updated with the intersection with $N(z)$. In general, each of the vertices in P is a neighbor of all the vertices in R. The expansion of R terminates when either P is empty (and thus R is maximal) or z cannot be found in P: in both cases it is z is *null*.

The descriptions of the variants of `EXTRACT_NODE` (P) are shown in Table 2. As it can seen, sometimes z can be *null* even if P is nonempty. If P is empty, all variants return $z =$ *null*, so the descriptions assume without loss of generality that P is nonempty.

We give some details on the variants. The pivoting variant `p` is inspired by the result of Tomita et al. [39], originally aimed at minimizing the number of calls to the BK algorithm. In our scenario we use it as a greedy heuristic to find a

[4] As it should be clear, it does not seem useful for variant `U` to choose the minimum degree node.

`r`	*random*: return a vertex z uniformly at random.
`p`	*pivoting*: return any vertex z with maximum $\|N(z) \cap P\|$ (as in [39]).
`c`	*clean*: return any vertex z (if any) with maximum $\|N_U(z) \cap R\| > 0$; otherwise, return $z = null$.
`s`	*semi-clean*: return any vertex z (if any) with maximum $\|N_U(z) \cap R\| > 0$; else, return any vertex z (if any) with maximum $\|N_U(z) \cap P\| > 0$; otherwise, return $z = null$.
`d`	*dirty*: return any vertex z (if any) with maximum $\|N_U(z) \cap R\| > 0$; else, return any vertex z (if any) with maximum $\|N_U(z) \cap P\| > 0$; otherwise, return any vertex z.

Table 2: Variants of `EXTRACT_NODE` to select a vertex $z \in P$

large clique: adding z to the current result set will maximize the size of P at the next step, as P will only retain the neighbors of z in P. The clean variant `c` aims at maximizing the number of uncovered edges that will become covered after adding z to the current clique R. The semi-clean variant `s`, in case that the clean variant fails, attempts to make the clean variant successful at the next call, as the candidate set P will meet the condition. Indeed, adding z to R will cause a node in the resulting P to have uncovered edges to the resulting R (as there is at least one that has has an uncovered edge to z). Finally, the `d` variant has the same aim as `s`, but never returns $z = null$ (unless P is empty).

Remark 1. Recall that we are proposing a heuristics for a hard problem. For example, the traditional task of minimizing the number of cliques could benefit by a variant of `EXTRACT_NODE` (P) that finds the largest clique containing edge $\{u, v\}$, but unfortunately the latter problem is NP-hard. We therefore propose variants that are fast to compute.

In the following, we use the notation ECC-*xy* to denote the variant of our framework given in Algorithm 1, where $x \in \{\mathtt{r}, \mathtt{m}, \mathtt{M}, \mathtt{U}\}$ encodes one of the edge selection variants described in Table 1, and $y \in \{\mathtt{r}, \mathtt{p}, \mathtt{c}, \mathtt{s}, \mathtt{d}\}$ encodes one of the node extraction variants described in Table 2. For example, ECC-`rc` selects an uncovered edge $\{u, v\}$ in a uniformly random fashion, and then it finds the clique containing $\{u, v\}$ by extracting each time the node z (if it exists) with the maximum number of uncovered edges to R. Hence, we have a total of 20 variants of our framework, and we expect just a subset of them to be effective for ECC (e.g. see Table 6 in Section 4).

3.3. Implementation and analysis

In this section we consider the complexity of the variants of our framework, and give some details on the ECCs they find. In the following, we make the fairly standard assumption of being able to iterate adjacency lists in time proportional to their size, and test adjacency between nodes in constant time.[5] First, we show that they correctly compute an ECC as defined in Section 2.

Lemma 2. *All the variants of Algorithm 1 correctly compute an* ECC *of the input graph G.*

Proof Let ECC-*xy* be any variant of Algorithm 1 where $x \in \{\mathtt{r}, \mathtt{m}, \mathtt{M}, \mathtt{U}\}$ and $y \in \{\mathtt{r}, \mathtt{p}, \mathtt{c}, \mathtt{s}, \mathtt{d}\}$. Let $C_1, C_2, \ldots, C_k$ be the cliques of the ECC in their order of discovery by ECC-*xy*, namely, C_i is discovered before C_j iff $i < j$. We will prove that *(1)* no clique C_i is contained in another clique C_j, where $i \neq j$, and *(2)* for each edge $\{u, v\} \in E$ there is at least one clique C_i such that $u, v \in C_i$. The latter point is easily met by ECC-*xy* as the while loop in Algorithm 1 terminates when all edges are covered.

We thus focus on point *(1)*. Suppose by contradiction that $C_i \subseteq C_j$ for $i \neq j$. First, we observe that it cannot be $C_i = C_j$ as they both must be found from one uncovered edge, and the first discovered of the two would cover all the edges of the other. Thus, it must be $C_i \subset C_j$. Second, it must be $i < j$ for the same reason: if it were $j < i$, then C_i would contain only covered edges.

[5] This second operation is easily achieved by means of, e.g., perfect hashing tables, which may be initialized in the beginning of the algorithm taking $O(m)$ time and space. Otherwise, performing binary searches on the (ordered) adjacency lists would save the extra memory and only add a factor $O(\log \Delta)$ to the final complexity of the approach.

Now we get a contradiction for different reasons.

When $y \in \{\mathtt{r}, \mathtt{p}, \mathtt{d}\}$, the cliques in their ECCs are all maximal. Indeed, a node $z \in P$ is returned as long as P is nonempty, and the clique found is maximal when P is empty since if it could be incremented with a node, that node would be in P. As by definition a maximal clique cannot be part of a larger clique, we have $C_i \not\subset C_j$ which contradicts what previously found.

As for $y = \mathtt{s}$, recall that C_j must contain at least one uncovered edge in $C_j \setminus C_i$. Since $C_j \setminus C_i \subseteq P$, one endpoint of that edge would have been chosen as z in the while loop for C_i, thus further extending C_i by at least one vertex. This is a contradiction as C_i was returned as it could not be further extended.

Finally, when $y = \mathtt{c}$, returning C_i means that all the edges having one endpoint in C_i and the other endpoint in $C_j \setminus C_i$ are covered, otherwise C_i could be expanded. Furthermore, all edges in C_i are then covered. Thus C_j must be discovered from an edge with both ends in $C_j \setminus C_i$. A moment of reflection shows that it is impossible to reach C_i's vertices from the endpoints u and v (of that uncovered edge) using other uncovered edges during the execution of `FIND_CLIQUE_OF`(u, v) in ECC-xc, as there is no clean edge between $C_j \setminus C_i$ and C_i. Thus it is a contradiction that C_i is contained in C_j. □

Remark 3. Looking at the above proof, we can see that it is always $k \leq m$ (k being the number of cliques in the covering). Moreover, ECC-xy discovers maximal cliques when $y \in \{\mathtt{r}, \mathtt{p}, \mathtt{d}\}$, while this is not necessarily true when $y \in \{\mathtt{c}, \mathtt{s}\}$. Note that the resulting ECC is not necessarily minimal, but we can apply the method in [17] to make it so.

We now consider how to implement `SELECT_UNCOVERED_EDGE` efficiently in ECC-xy (see Section 3.1). We store the set U of uncovered edges as an unordered array, which supports random access and deletion. Both operations can be performed in constant time with some bookkeeping: each edge in U knows its position in the array and, whenever it is selected to be marked as covered and deleted from U, its position is filled with the last entry in the array, and the array size conceptually decreases by 1.

The above is enough to implement variant $x = \mathtt{r}$ in constant time. As for the other variants $x \in \{\mathtt{m}, \mathtt{M}, \mathtt{U}\}$, we store the set of eligible nodes in at most Δ unordered lists $L_1, L_2, \ldots, L_\Delta$, where $\Delta = \max_{u \in V} d(u)$ is the graph's maximum degree. Specifically, eligible node u with $d(u) = i$ is stored in L_i. A similar organization is maintained also for the uncovered degree $d_U(u)$ of each eligible node u. With some bookkeeping, it takes constant time to be removed from a list or move u between lists.

For variants $x \in \{\mathtt{M}, \mathtt{U}\}$, we can keep the largest i such that L_i is nonempty. Since eligible nodes can be only deleted or moved to lists with lower indices, the amortized cost is constant for each change. For variant $x = \mathtt{m}$, we maintain a priority queue on the values i such that L_i is nonempty, and each operation takes $O(\log \Delta)$ time in the worst case.[6] We thus have the following cost.

Lemma 4. *Operation* `SELECT_UNCOVERED_EDGE` *in* ECC-*xy can be implemented in linear space, taking (amortized) constant time for variants* $x \in \{\mathtt{r}, \mathtt{M}, \mathtt{U}\}$ *and* $O(\log \Delta)$ *time for variant* $x = \mathtt{m}$*, where* Δ *is the graph's maximum degree.*

We now discuss how to implement `FIND_CLIQUE_OF` efficiently in ECC-xy (see Section 3.2). Operation `EXTRACT_NODE`(P) takes constant time by storing P as unordered array with random access and deletion with variant $y = \mathtt{r}$ (see above for $x = \mathtt{r}$). For the other variants, we observe that the sizes of P and U decrease while that of R increases, thus the values of $|N(z) \cap P|$ and $|N_U(z) \cap P|$ decrease and that of $|N_U(z) \cap R|$ can change. For these values, we thus use unordered lists $L_1, L_2, \ldots, L_\delta$ as discussed above for the variants $x \in \{\mathtt{m}, \mathtt{M}, \mathtt{U}\}$, where $\delta = \min\{d(u), d(v)\} \leq \Delta$ and $\{u, v\}$ is the uncovered edge on which `FIND_CLIQUE_OF` is launched. We maintain priority queues at the cost of $O(\log \delta)$ time per query or deletion. Consequently, `EXTRACT_NODE`(P) takes constant time for variant $y = \mathtt{p}$, and $O(\log \delta)$ time for the remaining variants $y \in \{\mathtt{c}, \mathtt{s}, \mathtt{d}\}$. As a result, we have the following cost, since both $|P|, |R| \leq \delta$.

Lemma 5. *Operation* `FIND_CLIQUE_OF`(u, v) *in* ECC-*xy can be implemented to find the clique R in linear space, taking* $O(|R| \min\{d(u), d(v)\})$ *time.*

[6] Time can be lowered for instance considering double-linked lists containing increasing indexes of non-empty L_i, but in practice this cost is small.

Proof Looking at Algorithm 2, let $\delta = \min\{d(u), d(v)\}$. The initialization of R takes constant time and that of P takes $O(\delta)$ time. As we saw, the extraction of z and its addition to R take $O(\log \delta)$ time, repeated for $|R| - 2$ times, thus giving a cost of $O(|R| \log \delta)$ time. The cost of updating P with $N(z)$ takes $O(\delta)$ time, and is repeated for $|R| - 2$ times, thus giving a cost of $O(|R|\, \delta)$ time, which is the dominant cost of `FIND_CLIQUE_OF`. □

Finally we can give an upper bound to the cost of ECC-xy in our algorithmic framework.

Theorem 6. *For an undirected graph G with n vertices and m edges, Algorithm 1 finds an* ECC *$C_1, C_2, \ldots, C_k$ in $O(m + n)$ space and $O(m \log \Delta + \sum_{i=1}^{k} |C_i| \min\{d(u_i), d(v_i)\})$ time ($k \leq m$), where u_i, v_i is the uncovered edge leading to the discovery of clique C_i, and Δ is G's maximum degree.*

Proof Since each C_i corresponds to at least one distinct uncovered edge, we have $k \leq m$. We pay $O(m \log \Delta)$ time to maintain the set of uncovered edges and choose one of them, by Lemma 4. Furthermore, finding clique C_i, for $1 \leq i \leq k$, takes $O(|C_i| \min\{d(u_i), d(v_i)\})$ time by Lemma 5. The cost follows. □

3.4. Improved bound

We finally show that some variants of the algorithm allow for a tighter bound. In particular, we restrict our attention to ECC-xy with $x \in \{\mathtt{r}, \mathtt{M}, \mathtt{U}\}$, as they allow for a faster (constant time) `SELECT_UNCOVERED_EDGE` operation, and $y = \mathtt{c}$ which guarantees a smaller result size.

To refine the bound we will also use the *degeneracy* d_G of the graph G: this is as the smallest number k such that every subgraph of G has a vertex of degree *at most* k. Equivalently, this is the highest k which the graph allows a k-core, i.e., a subgraph in which all nodes have degree at least k. For this reason the degeneracy is also known as *core number*. Note that $d_G \leq \Delta$ and that, if C is a clique, $d_G \geq |C| - 1$ (since C is a $(|C| - 1)$-core). The degeneracy is of particular interest in the analysis of graph algorithms since it is often small on real-world networks: this is exemplified by the values in Tables 3 and 5. It is also well known that graphs allow a *degeneracy ordering* (see [33, 40]), that is, an ordering in which every node v has at most d_G neighbors occurring after v in the ordering, where d_G is the degeneracy of G. This ordering may be found in linear time, but this is not relevant for our purpose.

Firstly, let us prove the following statement which will be key for the final complexity.

Lemma 7. *Given a graph $G = (V, E)$ with n nodes, m edges and degeneracy d_G, we have that $\sum_{\{u,v\} \in E} \min\{d(u), d(v)\} \leq m d_G$.*

Proof Assume knowledge of the degeneracy ordering of G. Furthermore, for any edge $e = \{u, v\}$, assume without loss of generality u to be the node occurring earlier in the degeneracy ordering (i.e., before v). Consider now the sum $\sum_{\{u,v\} \in E} d(u)$: if we call $t(u_i)$ the number of times that the degree of node u_i is added up in the summation, we can equivalently rewrite this as $\sum_{i=1}^{n} d(u_i) \cdot t(u_i)$. We have that the degree of u_i is added up at most as many times as the neighbors of u_i occurring *after* u_i in the degeneracy ordering, since we assumed u_i to be the one occurring earlier. Thus, by definition of degeneracy ordering, $t(u_i)$ is at most d_G. Finally, we have $\sum_{\{u,v\} \in E} d(u) = \sum_{i=1}^{n} d(u_i) \cdot t(u_i) \leq d_G \sum_{i=1}^{n} d(u_i) = d_G \cdot m$. As, for any edge $\{u, v\}$, $\min\{d(u), d(v)\}$ is clearly not larger than the degree of the extreme occurring earlier in a degeneracy ordering, the statement follows. □

Finally, we are ready to give our bound:

Theorem 8. *The cost of* ECC-*xy (Algorithm 1), for $x \in \{\mathtt{r}, \mathtt{M}, \mathtt{U}\}$ and $y = \mathtt{c}$, is $O(m d_G)$, where m is the number of edges and d_G the degeneracy of the input graph.*

Proof We will prove the statement by refining the bound given in Lemma 5 for the case $y = \mathtt{c}$.

By Lemma 4, the amortized cost of operation `SELECT_UNCOVERED_EDGE` for the considered variants is just $O(1)$ per clique, thus is negligible.

Consider now the expansion of the set R in Algorithm 2, starting from edge $\{u, v\}$, which is uncovered. Recall that R is a clique, thus it is fully contained in the neighborhood $(N(x) \cup \{x\})$ of any node x which belongs to it. In the beginning, this we have to consider $\min\{d(u), d(v)\}$ possible nodes for addition to R. At any step, assume we selected node z, so we have to restrict the set P to just nodes that are neighbors of z. Since $y = \mathtt{s}$, there is at least one uncovered edge $\{z, x\}$ between z and a node already in R. Furthermore, as x is in R, we have $|P| \leq d(x) + 1$. To intersect P and $N(z)$ we thus need just $O(\min\{d(x), d(z)\})$ time (note that we do not need to actually *find* the edge). We also have to update the priority queues, but this is dominated by the previous cost as it takes $O(\log |P|) = O(\min\{d(x), d(z)\})$ time.

Let E' be a set containing the edge $\{u, v\}$ and all edges $\{z, x\}$ considered during the expansion. We have that the cost of Algorithm 2 is bounded by $\sum_{\{u,v\}\in E'} \min\{d(u), d(v)\}$. Furthermore, every edge in E' was uncovered when it was considered: this means that, during the whole execution of Algorithm 1 every edge may be considered *at most* once in a single execution of Algorithm 2. The total cost of the algorithm is thus $O(\sum_{\{u,v\}\in E} \min\{d(u), d(v)\})$, which by Lemma 7 is $O(md_G)$. □

4. Experimental Evaluation of the Framework

This section presents an analysis of the results provided by our algorithmic framework, describing its quality and performance. In Section 4.1, we show the performance of one of our variants in particular, which is ECC-`rc`, giving a detailed comparison with the state of the art in Section 4.1, and showing that ECC-`rc` consistently outperforms the baseline in terms of both running time and quality of the result. In Section 4.2, we further show the behaviour of our approach, varying the different choices for the operations `SELECT_UNCOVERED_EDGE` and `FIND_CLIQUE_OF`. We show that different variants of ECC-*xy* may be used to produce covers with different properties (e.g., average or maximum clique size), which may be desired in different applications.

Datasets. All the networks in our experiments have been collected from LASAGNE [41].[7]

Real-world networks include autonomous system, biological, citation, collaboration, communication, word-adjacency, peer-to-peer, social networks, and web networks. Synthetic networks include networks generated using Erdős-Rényi [43], forest fire [44], and Kronecker [45]. For more details about the graphs used, we refer to Table 3, where we report for each network the number of nodes, the number of edges, the number of trivial cliques, and the degeneracy. It is worth observing that, although the degeneracy d_G can be up to $2\sqrt{m}$, its value is much less for these data sets. A more extensive sample of networks is considered in Section 4.1 in order to study the scalability of one of our variants (ECC-`rc`).

Competitors. Looking at the state of the art, we took as a *baseline* what we call `g-ALG`: this is the heuristic algorithm by Gramm et al. [35], which corresponds to the original algorithm by Kellerman [20], with the postprocessing step of Kou et al. [17] to remove redundant cliques, and performance improvements described in [35]. Interestingly, `g-ALG`[8] is still the state of the art, as far as we know, in terms of the traditional cost that minimizes the number of cliques found. For the sake of completeness, it is worth remarking that `g-ALG` was designed for minimizing the number of cliques and not to optimize the other criteria, for which no algorithm has been specifically provided. Moreover, [37] pointed out the speed and the quality of `g-ALG` also considering the minimum assignment objective, compared to an exact (exponential time) algorithm. Gramm et al. [35, 28] consider also an exact parameterized algorithm to find an ECC with the minimal number of cliques but it does not seem to scale for large graphs due to its larger running time. Hence we believe that `g-ALG` is the choice to compare with our solutions.

We implemented our variants ECC-*xy* (Sections 3.1–3.2) and `g-ALG` in Java 1.8.0_25.[9] The software is available at `https://github.com/Pronte/ECC`. Our computing platform was a 24 core machine with Intel(R) Xeon(R) CPU E5-2620 v3 at 2.40GHz, with 128GB of shared memory. The operating system was a Ubuntu 14.04.2 LTS, with a Linux kernel version 3.16.0-30.

We performed a large-scale preliminary selection of these 20 variants and `g-ALG`.

[7] We remark that LASAGNE stores graphs taken from various sources. Many of the graphs shown in Table 3 are originally from the better known SNAP repository [42].

[8] Which is equivalent to Kellerman's algorithm [20] in terms of quality

[9] We also considered the OCaml implementation of `g-ALG` kindly provided by the authors: while for small networks we obtained results consistent with our implementation of `g-ALG`, the performance of the original implementation did not allow us to process large networks in our dataset.

Category	Graph	Nodes	Edges	Trivial Cliques	Degeneracy
autonomous system	itdk0304_rlinks_undirected	192 244	609 066	248 035	32
biological	Drosophila_melanogaster	10 625	40 781	33 300	14
biological	Homo	1 027	1 166	980	3
biological	hprd_pp	9 465	37 039	17 657	14
biological	ppi_gcc	37 333	135 618	71 734	25
citation	cit-HepPh	34 546	420 876	24 986	30
citation	cit-HepTh	27 770	352 284	15 467	37
citation	citeseer	259 217	532 040	343 466	9
citation	hep-th-citations_MAX	27 400	352 021	15 289	37
collaboration	Newman-Cond_mat_95-99	22 015	58 578	58 350	8
collaboration	advogato	7 418	42 892	5 100	28
collaboration	ca-AstroPh	18 771	198 050	2 236	56
collaboration	ca-CondMat	23 133	93 439	3 888	25
collaboration	ca-GrQc	5 241	14 484	1 606	43
collaboration	ca-HepPh	12 006	118 489	2 588	238
collaboration	ca-HepTh	9 875	25 973	3 558	31
communication	email-Enron	36 691	183 830	14 069	43
communication	email-EuAll	265 214	364 480	253 357	38
word-adj	darwinBookInter_st	7 381	44 207	4 714	37
word-adj	frenchBookInter_st	8 325	23 841	9 683	17
word-adj	japaneseBookInter_st	2 704	7 998	2 916	15
word-adj	spanishBookInter_st	11 586	43 065	8 177	30
p2p	p2p-Gnutella31	62 586	147 891	142 503	6
social	soc-Slashdot0811	77 360	469 180	231 064	55
social	soc-Slashdot0902	82 168	504 230	247 909	56
social	trust	49 288	381 036	64 870	72
social	wiki-Vote	7 115	100 761	8 655	53
synthetic	Gnp_1e4	10 000	59 849	59 061	8
synthetic	Gnp_2e3	2 000	8 994	8 698	6
synthetic	Gnp_5e3	5 000	24 809	24 307	7
synthetic	forest1e4	10 000	49 354	2 698	36
synthetic	forest1e4_2	10 000	153 925	2 967	101
synthetic	forest5e4	50 000	243 441	13 740	40
synthetic	kron14	8 156	24 482	21 016	6
synthetic	kron16	30 429	65 494	63 790	6
synthetic	ud_1e3	1 000	16 727	9	77
synthetic	ud_1e4	10 000	313 726	16	285
synthetic	ud_2e3	1 999	35 697	30	108
synthetic	ud_5e3	4 998	97 027	41	165
synthetic	us_1e3	1 000	14 334	0	23
synthetic	us_1e4	10 000	242 533	0	33
synthetic	us_2e3	2 000	37 928	0	30
synthetic	us_5e3	5 000	135 833	0	35
web	GoogleNw	15 763	148 585	6 756	102

Table 3: A summary of our datasets

measure	**goal**	**vs** g-ALG
number of cliques	MIN	5%
maximum clique size	MAX	2%
average clique size	MAX	-5%

(a) clique size distribution

measure	**goal**	**vs** g-ALG
maximum node covering	MIN	23%
average node covering	MIN	11%

(b) node covering index distribution

measure	**goal**	**vs** g-ALG
maximum edge covering	MIN	29%
average edge covering	MIN	12%

(c) edge covering index distribution

measure	**goal**	**vs** g-ALG
average time	MIN	62%
coefficient of variation — 0.57		

(d) speedup

Table 4: Summary of the direct comparison between ECC-rc and g-ALG for each measure and for the speed

Quality guidelines and results. Our quality guidelines for the experiments are based on the clique cover graphs $G' = (V'_1 \cup V'_2, E')$ for the graphs $G = (V, E)$ in the datasets, as described in Section 2.1. In particular we considered the following ones.

(a) Clique size distribution: for each feasible value t, we compute the number of cliques in the ECC that have size t. In the clique cover graph, this is equivalent to the degree distribution of V'_1. With this information, we also get the number of cliques, the maximum clique size, and the average clique size.

(b) Node covering index distribution: given a node u in G, let $c(u)$ be the number of cliques containing u in the ECC. For each feasible value t, we compute the number of nodes u having $c(u) = t$. In the clique cover graph where $V'_2 = V$, this is equivalent to the degree distribution of V'_2. With this information, we also get the maximum $c(u)$ and the average value.

(c) Edge covering index distribution: as in (b), except that the cover index $c(u, v)$ is the number of cliques containing edge $\{u, v\}$ in the ECC. In the clique cover graph where $V'_2 = E$, this is equivalent to the degree distribution of V'_2.

For the sake of completeness, for each distribution we have also considered the *coefficient of variation*, in short *CV*. The CV is a measure of dispersion of a distribution and it corresponds to the standard deviation divided by the mean. This measure is used as a normalized way to evaluate the variability of a covering (the closer to 0, the better is).

4.1. Evaluation of ECC-rc

In this section, we focus on evaluating our variant ECC-rc, comparing it to the state of the art (g-ALG). Recall that the ECC-rc uses the clean extraction of nodes where the uncovered edge selection uses the random variant. We chose evaluate specifically this variant as it seems to be the one offering the best trade-off between running time and quality of the result, also in terms of the "standard" quality measure, i.e., the number of cliques found, as it will be shown in Section 4.2. The take-home lesson of this section is that ECC-rc is usually the best choice for computing a small ECC, especially in large graphs.[10]

For the comparison, we used the graphs in Table 3. Moreover, in order to show the ability of ECC-rc to efficiently process large graphs, we also ran it on a larger set of 160 graphs, with up to 783 million edges.

[10]If, however, the goal is maximizing one specific measure among the ones considered, we refer the reader to Section 4.2.

4.1.1. Quality of the cover

Our results are shown in Table 4, which can be read as follows. The first column indicates the measures described above[11] while the second column indicate whether it is good to minimize or maximize the measure in the corresponding row. The third column reports how our ECC-rc compared to g-ALG: a value of $p\%$ for the minimization indicates that the measure found by our algorithm is $p\%$ smaller than that provided by g-ALG; for the maximization, it indicates that the measure is $p\%$ larger than that by g-ALG.[12]

It is worth observing that ECC-rc finds a cover with slightly fewer cliques than g-ALG, i.e., smaller by 5%. The improvement of ECC-rc for node and edge covering indexes is more significant: Minimizing the average covering corresponds to minimizing the assignments, and the solutions of g-ALG have been observed to be often close to the optimum [37] for this metric. However, ECC-rc improves on the average assignments by 11%, and on the maximum node covering (i.e., assignments of the most assigned node) by 23%. As for the edge covering, we improve the average by 12% and the maximum by 29%.

We can also note how ECC-rc ran faster than g-ALG by 62% on average. However, we will show in the following how the difference becomes much more dramatic on large graphs.

4.1.2. Performance

Figure 3 shows the performance comparison between ECC-rc and g-ALG for the graphs in Table 3. For each graph in the dataset (ordered by number of edges) we draw a red cross which is the ratio between the time needed by g-ALG and the one needed by ECC-rc, as a function of the number of the edges. In other words, a point at height 1 on the y-axis (green line) means that g-ALG took the same time as ECC-rc to process the corresponding graph, and a point at height 10 means that g-ALG was 10 times slower.

The average time used by ECC-rc to obtain the covering is 62% smaller than the one used by g-ALG, as also shown in Table 4. This means that our algorithm employs on average almost one third of the time employed by g-ALG to complete the task. For the sake of completeness, in Table 4 we report also the CV of the speedup which is equal to 0.57, as it can varies significantly. However, looking at Figure 3 we can see that, in the case of larger graphs, the situation is much worse for g-ALG: the plot shows that very often the performance of g-ALG is several orders of magnitude worse than that by ECC-rc. The variability seems to dramatically affect the performance of g-ALG as the number of edges increases.

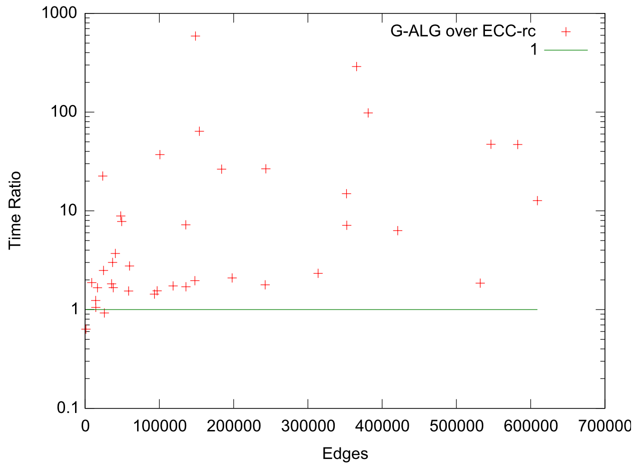


Figure 3: Ratio between the time used by g-ALG and ECC-rc for each graph, as a function of the number of the edges (y-axis is in log-scale).

Finally, we considered the performance of ECC-rc on larger graphs: we took a collection of 160 real and synthetic

[11] Except for the coefficients of variation, for which an upper bound is provided in Table 6.

[12] In other words, let a be the measure found by our best performing algorithm and b be the measure found by g-ALG; in the case of minimization we report $p = (1 - a/b) \cdot 100$, while in the case of maximization we report $p = (a/b - 1) \cdot 100$.

networks from SNAP [42] and LASAGNE [41], including those in Table 3 and several larger ones. For brevity, rather than reporting all 160 graphs, we show just the largest ones in Table 5.

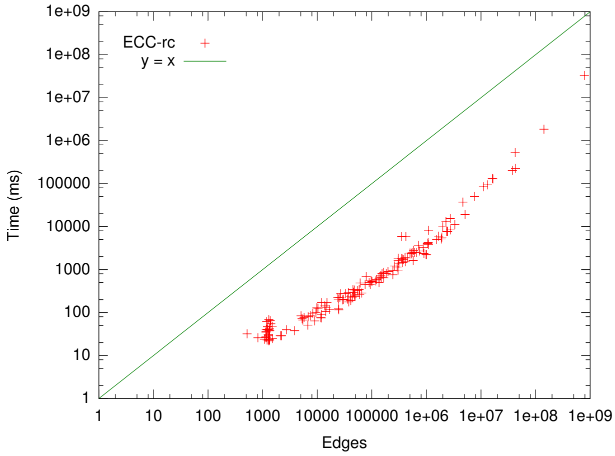


Figure 4: Time used by ECC-rc to compute clique covers for 160 real-world graphs as a function of the number of edges.

Figure 4 reports the time used by ECC-rc to compute clique covers as a function of the number of edges in the graph. In other words, for each graph in our dataset, we have run ECC-rc placing a red cross in position (x, y) whether the graph has x edges and ECC-rc spent y milliseconds to finish the computation. Looking at the plot, the red crosses seem to be disposed over a line which is parallel to the line $y = x$ suggesting a linear running time of our algorithm. This hypothesis is confirmed by the results of a linear regression over the original data (not in log scale): the time y can be related to the number of edges x by using $y = a \cdot x + b$, where $a = 0.041$ and $b = -71005.486$; these estimates have respectively p-value $7.563 \cdot 10^{-136}$ and 0.0174 (the correlation is 98.961%). Due to the statistical significance of our tests, we argue that ECC-rc is linear when dealing with large real-world networks. It is reasonable to assume that this is due, at least partly, to the degeneracy, which may be linear in the worst case, but is in practice small in real-world graphs.

This feature clearly emerges with the largest networks we considered (see Table 5): for the largest one, a snapshot of the web, our algorithm terminates after approximately 9 hours, which is reasonable if we consider the size of the network and the fact it has been processed on a single core, and in line with the performance obtained on the rest of the dataset.

Finally, it is worth remarking that, while this algorithmic framework is designed for —typically sparse— real-world graphs, ECC-rc was tested in [46] and shown to perform remarkably well on dense graphs too.[13]

Category	Graph	Nodes	Edges	Degeneracy	Time (s)
col.	imdb	913 201	37 588 613	1 297	202
web	enwiki	13 834 640	42 336 692	208	526
social	LiveJournal1	4 847 571	42 851 236	372	225
P2P	p2p	5 792 297	142 038 401	853	1 845
web	web	39 454 463	783 027 125	588	32 734

Table 5: Sample of the largest networks we considered with the running time of ECC-rc.

4.2. *Comparison of our Approaches*

In this section, we report our comparison concerning the quality of the results provided by our variants. Our results are reported in Table 6, which has the same format of Table 4, reporting also which algorithms ECC-xy resulted the best performers according to the measure in the row.

As shown in Table 6, the following variants emerged from the rest when considering various quality measures.

[13]For context, the description of ECC-rc was taken by the authors of [46] from the preliminary version of this paper available at [1].

measure	goal	best algs	vs g-ALG
number of cliques	MIN	ECC-rs, ECC-ms	12%,12%
maximum clique size	MAX	ECC-Mp, ECC-Up	6%,6%
average clique size	MAX	ECC-Mp, ECC-mp	26%,27%
coefficiente of variation — range: 0.013–0.016			

(a) clique size distribution

measure	goal	best alg.	vs g-ALG
maximum node covering	MIN	ECC-Ms	37%
average node covering	MIN	ECC-rc	11%
coefficient of variation — range: 0.026–0.035			

(b) node covering index distribution

measure	goal	best alg.	vs g-ALG
maximum edge covering	MIN	ECC-Ms	87%
average edge covering	MIN	ECC-rc	23%
coefficient of variation — range: 0.021–0.026			

(c) edge covering index distribution

Table 6: Summary of the best-performing algorithms for each measure

- ECC-xp: the pivoting extraction of nodes where the uncovered edge selection uses the variant $x \in \{\mathtt{M}, \mathtt{m}, \mathtt{U}\}$ of max degree, min degree, or max uncovered degree.
- ECC-xs: the semi-clean extraction of nodes where the uncovered edge selection uses the variant $x \in \{\mathtt{M}, \mathtt{m}, \mathtt{r}\}$ of max degree, min degree, or random.
- ECC-rc: the clean extraction of nodes where the uncovered edge selection uses the random variant.

For each distribution, we also report the range of values for the CV for all the variants of ECC-*xy* we considered. As it can be seen in Table 6, for each measure there is at least one algorithm in our framework which outperforms g-ALG.

Figure 5 reports the plots for distributions (a)–(c) for the graphs as-itdk0304_rlinks (autonomous system network), cit-HepPh (citation network), and email-Enron (communication network). It displays the distribution achieved by g-ALG and the best algorithms reported in Table 6. For the sake of clarity, in the case of ties we have decided to show in the plots just one of the options, so that the reported methods are: ECC-ms, ECC-Mp, ECC-Ms, and ECC-rc.

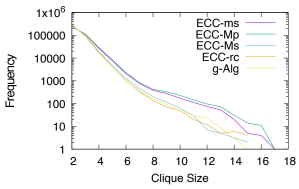
ECC-Mp achieves more often a larger maximum clique, that is the maximum value on the x-axis such that the line is draw in the plots in the first column of Figure 5 (namely, (a-1),(a-2), and (a-3)). On the other hand, the area under curve of clique size distributions in these latter plots corresponds to the number of cliques returned by each algorithm: the plots confirm that ECC-ms is the more effective, since the y-axis is in log scale and ECC-ms seems to achieve much less small cliques with respect to the other methods.

For the sake of completeness we show the plots for the node covering distribution in the second column of Figure 5, where the situation appears more complicated. However, numerically, we have seen that ECC-Ms and ECC-rc correspond to the curve more likely to be close to the y-axis, meaning that the maximum and average covering index induced by these distribution is more likely to be smaller. The same can be said looking at the third column of Figure 5, where the edge covering distribution is shown: in this case, the higher improvement permits to appreciate better the lower maximum and average covering index of the edges in the solutions of ECC-Ms and ECC-rc.

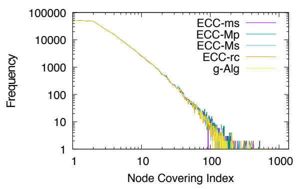
In the rest of this section, we give more details on the experimental study of these measures for distributions (a)–(c).

4.2.1. Analyzing Clique Size Distribution

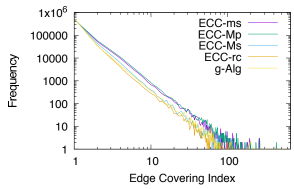
Number of Cliques. In the great majority of cases we found that ECC-rs and ECC-ms are the most effective. On the average, they provide about 9% fewer cliques than g-ALG. In general, the latter finds more cliques than all ECC-*xy* `as-itdk0304_rlinks`



(a-1) clique size distribution

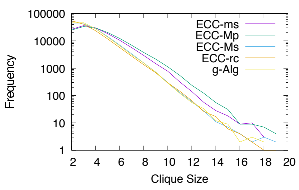


(b-1) node covering distribution

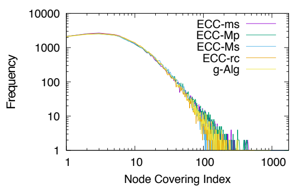


(c-1) edge covering distribution

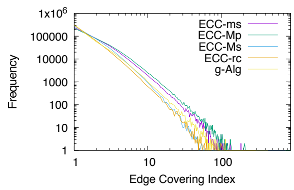
`cit-HepPh`



(a-2) clique size distribution

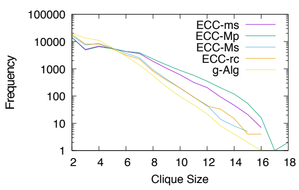


(b-2) node covering distribution

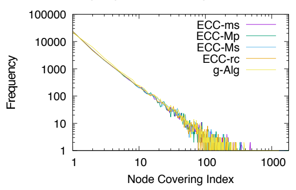


(c-2) edge covering distribution

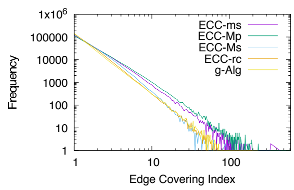
`email-Enron`



(a-3) clique size distribution



(b-3) node covering distribution



(c-3) edge covering distribution

Figure 5: Distributions (a)–(c) of the best performing algorithms (mentioned in Table 6) together with G-ALG for a sample of three graphs, namely, `as-itdk0304_rlinks` (autonomous system network), `cit-HepPh` (citation network), and `email-Enron` (communication network). The axes are in logarithmic scale, except for the x-axis of (a-1), (a-2), and (a-3). The plots can be zoomed in the electronic version of this paper.

in more than 60% of the cases. Moreover, many edges of real-world networks do not belong to any triangle and hence they must be covered by *trivial* cliques (i.e. of size 2). The ratio between the number of trivial cliques and the number of edges seems to be positively correlated with the inverse of the average degree in the graph, i.e. n/m: the correlation is quite high indeed, 0.7. We have thus considered the difference between the number of cliques found by each algorithm and the number of trivial cliques. Also in this case, the best methods to minimize the number of cliques are ECC-rs and ECC-ms and they find 12% fewer cliques than g-ALG.

Maximum clique size dimension. In all the experiments the most effective algorithm in finding a maximum size clique are ECC-Mp and ECC-Up. In particular they are the most effective, respectively, for the 95% and 90% of the cases. On the average both of them find a clique that is 6% larger than that of g-ALG. In all the cases, g-ALG never finds a maximum clique larger than the one found by the two algorithms above. When considering all ECC-*xy*, the algorithm finding the smallest among all is g-ALG for 60% of the cases.

Average clique size. In all the cases ECC-mp and ECC-Mp achieve the maximum average size among all the algorithms. In the 85% of the cases, the average size of the cliques in the solution provided by g-ALG is smaller than the average size of the cliques found by ECC-mp or ECC-Mp. The average clique sizes returned by ECC-mp and ECC-Mp are, respectively, 27% and 26% larger than the one by g-ALG.

CV of clique size distribution. We observe that the CV ranges between 0.013 (ECC-rs) and 0.016 (ECC-Mp). The CV of g-ALG is 0.015. This means that the overall variability of the distributions is quite similar when comparing the algorithms by averaging each of them on all the graphs.

4.2.2. Node Covering Index

Mean node covering. In order to avoid redundancy it could be preferable to minimize this measure, that is minimizing the space occupied by the solution. Evaluating $\sum_{u \in V} c(u)$ corresponds to evaluate the sum of the sizes of the cliques in terms of nodes, i.e. $\sum_{i=1}^{k} |C_i|$, so that the average node covering index is $\frac{\sum_{i=1}^{k} |C_i|}{n}$. The method ECC-rc achieves more often (in the 45% of the cases) a solution whose mean is less than all the others. On the average this value is 11% smaller than that of g-ALG. In the 22% of the cases the minimum mean is achieved by ECC-mc. On the average its value is 6% smaller than that of g-ALG. Note that the latter never finds the minimum mean among all the algorithms.

Max node covering. The smallest maximum node covering index is achieved by ECC-Ms in almost 62% of the networks, never by g-ALG. On the average, the former achieves a max node node covering index that is 37% smaller than the one provided by the latter.

CV. The coefficient of variation ranges in between 0.026 (ECC-rc) and 0.035 (ECC-Ud). An higher variability is achieved by ECC-*x*r and ECC-*x*d, while a lower variability is achieved for the variants of ECC-*x*c.

4.2.3. Edge Covering Index

Mean edge covering. The algorithms ECC-rc and ECC-mc achieve the lowest mean edge covering for the majority of the networks: they are the best algorithms, respectively, in the 38% and in the 36% of the cases. In all the remaining instances, g-ALG is never the most effective. On the average the mean edge covering index of the solution returned by ECC-rc and ECC-mc is, respectively, 23% and 13% smaller than that by g-ALG. Note that the average edge covering index is equal to $\frac{\sum_{i=1}^{k} |C_i| \cdot |C_i - 1|}{2 \cdot m}$, where $C_1, C_2, \ldots, C_k$ are the the cliques in ECC.

Max edge covering. For half of the networks, the algorithm achieving the minimum among the maximum edge covering indexes is ECC-Ms. Other effective algorithms for are ECC-rc and ECC-mc (respectively, in the 18% and 12% of the cases). Also for this measure, g-ALG is never the most effective. On the average the maximum edge covering indexing of a solution found by ECC-Ms is 87% smaller than that by g-ALG.

CV. The coefficient of variation of the edge covering distributions for all the algorithms range between 0.021 (ECC-Ms) and 0.026 (ECC-Mc, ECC-Uc, and Kellerman algorithm).

5. Conclusions

Inspired by the *clique cover graph*, in this paper we have introduced several measures to assess the quality of a solution rather just the classical ones, e.g. minimizing the number of cliques.

To deal with these measures, we proposed new and simple algorithms which are part of a simple framework and scale well to deal with large real-world networks. Moreover, we showed that our algorithms improve the state of the art according to all the several measures we considered based on the clique cover graph, experimenting our 20 variants of the framework on a data set of 44 networks. Our algorithms improve in practice the quality of the solution, also according to the traditional measures. In particular, one of our variants emerged as particularly effective both for quality and running times, which can be considered linear in practice, as shown by our extensive experiments.

References

[1] A. Conte, R. Grossi, A. Marino, Clique covering of large real-world networks, in: Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016, 2016, pp. 1134–1139. `doi:10.1145/2851613.2851816`.

[2] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, H. P. Piepho, R. Schmid, Algorithms for compact letter displays: Comparison and evaluation, Computational Statistics and Data Analysis 52 (2) (2007) 725–736.

[3] H.-P. Piepho, An algorithm for a letter-based representation of all-pairwise comparisons, Journal of Computational and Graphical Statistics 13 (2) (2004) 456–466. `doi:10.1198/1061860043515`.

[4] H. Bernard, P. D. Killworth, L. Sailer, Informant accuracy in social network data iv: a comparison of clique-level structure in behavioral and cognitive network data, Social Networks 2 (3) (1979) 191 – 218. `doi:https://doi.org/10.1016/0378-8733(79)90014-5`.

[5] S. Rajagopalan, M. Vachharajani, S. Malik, Handling irregular ILP within conventional VLIW schedulers using artificial resource constraints, in: CASES, 2000, pp. 157–164.

[6] M. Behrisch, A. Taraz, Efficiently covering complex networks with cliques of similar vertices, Theoretical Computer Science 355 (1) (2006) 37–47.

[7] J.-L. Guillaume, M. Latapy, Bipartite structure of all complex networks, Inf. Process. Lett. 90 (5) (2004) 215–221.

[8] F. Abu-Khzam, N. N. Baldwin, M. Langston, N. Samatova, On the relative efficiency of maximal clique enumeration algorithms, with application to high-throughput computational biology, in: International Conference on Research Trends in Science and Technology, 2005, pp. 1–10.

[9] P. Agarwal, N. Alon, B. Aronov, S. Suri, Can visibility graphs be represented compactly?, GEOMETRY: Discrete & Computational Geometry 12.

[10] V. Stix, Finding All Maximal Cliques in Dynamic Graphs, Computational Optimization and Applications 27 (2) (2004) 173–186.

[11] G. Creamer, R. Rowe, S. Hershkop, S. J. Stolfo, Segmentation and automated social hierarchy detection through email network analysis, in: Lecture Notes in Computer Science, Vol. 5439 LNAI, 2009, pp. 40–58.

[12] V. Boginski, S. Butenko, P. M. Pardalos, Statistical analysis of financial networks, Computational Statistics & Data Analysis 48 (2) (2005) 431–443.

[13] A. Ignatiev, A. Morgado, J. Marques-Silva, Cardinality encodings for graph optimization problems, in: C. Sierra (Ed.), Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, ijcai.org, 2017, pp. 652–658.

[14] B. Nina, T. Ko, T. Moy, J. Smrcka, J. Turnley, B. Wu, Emergent clique formation in terrorist recruitment., The AAAI-04 Workshop on Agent Organizations: Theory and Practice.

[15] S. Wasserman, K. Faust, Social Network Analysis: Methods and Applications, Vol. 8, Cambridge university press, 1994.

[16] J. Helling, P. Ryan, W. Smyth, M. Soltys, Constructing an indeterminate string from its associated graph, Theoretical Computer Science 710 (2018) 88 – 96. `doi:https://doi.org/10.1016/j.tcs.2017.02.016`.

[17] L. T. Kou, L. J. Stockmeyer, C.-K. Wong, Covering edges by cliques with regard to keyword conflicts and intersection graphs, Communications of the ACM 21 (2) (1978) 135–139.

[18] J. Berger, Status characteristics and social interaction: an expectation-states approach, Elsevier Scientific Pub. Co., 1977.

[19] F. Roberts, Discrete Mathematical Models, with Applications to Social, Biological, and Environmental Problems, Prentice-Hall, 1976.

[20] E. Kellerman, Determination of keyword conflict, IBM Technical Disclosure Bulletin 16 (2) (1973) 544–546.

[21] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Computers and Intractability (1979) 340.

[22] M. Cygan, M. Pilipczuk, M. Pilipczuk, Known algorithms for edge clique cover are probably optimal, in: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2013, pp. 1044–1053.

[23] J. Orlin, Contentment in graph theory: Covering graphs with cliques, Indagationes Mathematicae (Proceedings) 80 (5) (1977) 406–424.

[24] M.-S. Chang, H. Müller, On the tree-degree of graphs, in: A. Brandstädt, V. B. Le (Eds.), Graph-Theoretic Concepts in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 44–54.

[25] D. N. Hoover, Contentment in graph theory: Covering graphs with cliques, J. Math. Combinatorial Comput. 11 (1992) 187–208.

[26] W.L.Hsu, K. Tsai, Linear time algorithms on circular-arc graphs, IPL: Information Processing Letters 40.

[27] W. D. Wallis, J. Wu, Generalized Clique Coverings of Chordal Graphs, Australasian Journal of Combinatorics 1 (1990) 233–236.

[28] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, Data reduction and exact algorithms for clique cover, Journal of Experimental Algorithmics (JEA) 13 (2009) 2.

[29] C. Lund, M. Yannakakis, On the hardness of approximating minimization problems, Journal of the ACM 41 (5) (1994) 960–981.

[30] J. Abello, P. M. Pardalos, M. G. C. Resende, On Maximum Clique Problems In Very Large Graphs, American Mathematical Society, 1999.
[31] E. A. Akkoyunlu, The enumeration of maximal cliques of large graphs, SIAM Journal on Computing 2 (1) (1973) 1–6.
[32] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, L. Zhu, Finding maximal cliques in massive networks, ACM Transactions on Database Systems 36 (4) (2011) 1–34.
[33] D. Eppstein, D. Strash, Listing all maximal cliques in large sparse real-world graphs, in: Proceedings of the 10th International Conference on Experimental Algorithms, SEA'11, Springer-Verlag, 2011, pp. 364–375.
[34] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, M. M. A. Patwary, Fast maximum clique algorithms for large graphs, in: WWW, 2014, pp. 365–366.
[35] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, Data reduction, exact, and heuristic algorithms for clique cover., in: ALENEX, SIAM, 2006, pp. 86–94.
[36] P. Erdos, A. W. Goodman, L. Pósa, The representation of a graph by set intersections, Canad. J. Math 18 (106-112) (1966) 86.
[37] J. M. Ennis, C. M. Fayle, D. M. Ennis, Assignment-minimum clique coverings, J. Exp. Algorithmics 17 (2012) 1.5:1.1–1.5:1.17.
[38] C. Bron, J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, Communications of the ACM 16 (9) (1973) 575–577.
[39] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques and computational experiments, Theoretical Computer Science 363 (1) (2006) 28–42.
[40] A. Conte, R. Grossi, A. Marino, L. Versari, Sublinear-Space Bounded-Delay Enumeration for Massive Network Analytics: Maximal Cliques, in: 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), Vol. 55 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2016, pp. 148:1–148:15. doi:10.4230/LIPIcs.ICALP.2016.148.
[41] U. of Florence, LASAGNE, www.pilucrescenzi.it/wp/networks/, [Online; accessed March 2019] (2018).
[42] SNAP, Stanford Network Analysis Project, snap.stanford.edu/data/, [Online; accessed March 2019] (2018).
[43] P. Erdős, A. Rényi, On the evolution of random graphs, Bull. Inst. Internat. Statist 38 (4) (1961) 343–347.
[44] J. Leskovec, J. Kleinberg, C. Faloutsos, Graphs over time: Densification laws, shrinking diameters and possible explanations, in: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, KDD '05, ACM, 2005, pp. 177–187.
[45] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, Z. Ghahramani, Kronecker graphs: An approach to modeling networks, The Journal of Machine Learning Research 11 (2010) 985–1042.
[46] J. Helling, P. Ryan, W. Smyth, M. Soltys, Constructing an indeterminate string from its associated graph, Theoretical Computer Science 710 (2018) 88–96.