A Sparse Matrix Approach for Covering Large Complex Networks by Cliques

Wali Mohammad Abdullah and Shahadat Hossain

University of Lethbridge, Lethbridge, Alberta, Canada
{w.abdullah,shahadat.hossain}@uleth.ca

Abstract. A classical NP-hard problem is the *Minimum Edge Clique Cover (minECC)* problem, which is concerned with covering the edges of a network (graph) with the minimum number of cliques. There are many real-life applications of this problem, such as in food science, computational biology, efficient representation of pairwise information, and so on. Borrowing ideas from [8], we propose using a compact representation, the intersection representation, of network data and design an efficient and scalable algorithm for minECC. Edges are considered for inclusion in cliques in degree-based orders during the clique construction step. The intersection representation of the input graph enabled efficient computer implementation of the algorithm by utilizing an existing sparse matrix package [11]. We present results from numerical experiments on a representative set of real-world and synthetically constructed benchmark graph instances. Our algorithm significantly outperforms the current state-of-the-art heuristic algorithm of [4] in terms of the quality of the edge clique covers returned and running time performance on the benchmark test instances. On some of the largest graph instances whilst existing heuristics failed to terminate, our algorithm could finish the computation within a reasonable amount of time.

Keywords: Adjacency Matrix · Clique Cover · Intersection Matrix · Ordering · Sparse Graph.

1 Introduction

The graph kernel operations, such as identification of and computation with dense subgraphs, frequently arise in areas as diverse as sparse matrix determination and complex network analysis [14, 13]. In social networks, identification of special interest groups or characterization of information propagation are examples of frequently performed network analytics tasks [23]. The *Edge Clique Cover problem (ECC)* considered in this paper is concerned with finding a collection of complete subgraphs or cliques such that every edge and every vertex of the input graph is included in some clique. The computational challenge is to find an ECC with the smallest number of cliques (*minECC*). The minECC problem is computationally intractable or NP-hard [16].

Effective representation of network data is critical to meeting algorithmic challenges for exactly or approximately solving intractable problems, especially

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

when the instance sizes are large and sparse. In this paper, we use sparse matrix data structures to enable compact representation of sparse network data based on an existing sparse matrix framework [11] to design efficient algorithms for the minECC problem.

Let $G = (V, E)$ be an undirected connected graph, where V is the set of vertices, and E is the set of edges. A clique is a subset of vertices such that every pair of distinct vertices are connected by an edge in the induced subgraph. In graph G, an edge clique cover of size k is a decomposition of set V into k subsets $C_1, C_2, \ldots, C_k$ such that $C_i, i = 1, 2, \ldots, k$ induces a clique in G and each edge $\{u, v\} \in E$ is included in some C_i. A trivial clique cover with $k = m, |E| = m$ can be specified by the set of edges E with each edge being a clique. Finding a clique cover with the minimum number of cliques (and many variants thereof) is known to be an NP-hard problem [16].

In 1973, Bron and Kerbosch [2] proposed an algorithm to find all maximal cliques of a given graph. That algorithm uses a branch-and-bound technique. The algorithm is made more efficient by cutting off branches of the search tree that will not lead to new cliques at a very early stage. Etsuji Tomita et al. [22] presented a depth-first search algorithm for generating all maximal cliques of an undirected graph, in which pruning methods are employed as in the Bron–Kerbosch algorithm.

Many algorithms have been proposed in the literature to solve the ECC problem approximately. At the same time, there are only a few exact methods that are usually limited to solving small instance sizes. A recent heuristics approach is described by Conte et al. [4] to find an edge clique cover in $O(m\Delta)$ time, where m is the number of edges and Δ is the highest degree of any vertex in the graph.

In this paper, we use a compact representation of network data based on sparse matrix data structures [11] and provide an improved algorithm motivated by the works of Bron et al. [2], and E. Tomita et al. [22] for finding clique covers. In [1], we used a similar compact representation of network data. In that paper, we employ a "vertex-centric" approach where a vertex, in some judiciously chosen order, together with its edges incident on a partially constructed clique cover, is considered for inclusion in an existing clique. The preliminary implementation produced smaller-sized clique covers when compared with the method of [9] on a set of test instances. While the vertex-centric ECC algorithm frequently produced smaller clique covers compared with other methods, the high memory footprint of the method made it less scalable on very large problem instances. In this paper, we propose an "edge-centric" minECC method. Our method is characterized by a significantly reduced memory footprint and exhibits very good scalability when applied to extremely large synthetic and real-life network instances.

Our approach is based on the simple but critical observation that for a sparse matrix $A \in \mathbb{R}^{m \times n}$, the row intersection graph of A is isomorphic to the adjacency graph of $AA^\top$, and that the column intersection graph of A is isomorphic to the adjacency graph of $A^\top A$ [11]. Therefore, the subset of rows corresponding to

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

nonzero entries in column j induces a clique in the adjacency graph of $AA^\top$, and the subset of columns corresponding to nonzero entries in row i induces a clique in the adjacency graph of $A^\top A$. Note that matrices $A^\top A$ and $AA^\top$ are most likely dense even if matrix A is sparse. We exploit the close connection between sparse matrices and graphs in the reverse direction. We show that given a graph (or network), we can define a sparse matrix, *intersection matrix*, such that graph algorithms of interest can be expressed in terms of the associated intersection matrix. This structural reduction enables us to use the existing sparse matrix computational framework to solve graph problems [11]. This duality between graphs and sparse matrices has also been exploited where the graph algorithms are expressed in the language of sparse linear algebra [14, 15]. However, they use adjacency matrix representation which is different from our intersection matrix representation.

The paper is organized as follows. In Section 2, we consider representations of sparse graph data and introduce the notion of intersection representation and cast the minECC problem as a matrix compression problem. Section 3 presents the new edge-centric minECC algorithm. An important ingredient of our algorithm is to select edges incident on the vertex being processed in specific orders. The details of the implementation steps are described, followed by the presentation of the ECC algorithm. The section ends with a discussion on the computational complexity of the algorithm. Section 4 contains results from elaborate numerical experiments. We choose 5 different sets of network data consisting of real-world network and synthetic instances. Finally, the paper is concluded in Section 5.

2 Compact Representation and Edge Clique Cover

For efficient computer implementation of many important graph operations, representing graphs using adjacency matrix or adjacency lists is inefficient. Adjacency matrix stored as a two-dimensional array is costly for sparse graphs, and typical adjacency list implementations employ pointers where indirect access leads to poor cache utilization [19]. The intersection matrix representation that we propose below enables an efficient representation of pairwise information and allows us to utilize the computational framework DSJM to implement the new ECC algorithm.

2.1 Intersection Representation

We require some preliminary definitions. The *adjacency graph* associated with a symmetric matrix $A \in \mathbb{R}^{n\times n}$ is an undirected graph $G = (V, E)$ in which for each column or row k of A there is a vertex $v_k \in V$ and $A(i, j) \neq 0, i \neq j$ if and only if $\{v_i, v_j\} \in E$.

Let $G = (V, E)$ be an undirected and connected graph without self-loops or multiple edges between a pair of vertices. The adjacency matrix $A(G) \equiv A \in$

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

$\{0,1\}^{|V|\times|V|}$ associated with graph G is defined as,

$$A(i,j) = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E, i \neq j \\ 0 & \text{otherwise} \end{cases}$$

We now introduce the intersection representation, an enabling and efficient representation of pairwise information. The *intersection representation* of graph G is a matrix $X \in \{0,1\}^{k\times n}$ in which for each vertex v_j of G there is a column j in X and $\{v_i, v_j\} \in E$ if and only if there is a row l for which $X(l,i) = 1$ and $X(l,j) = 1$. A special case is obtained for $k = m$. Then, the rows of X can be uniquely labeled by the edge list sorted by vertex labels. Therefore, matrix $X \in \{0,1\}^{m\times n}$ can be viewed as an assignment to each vertex a subset of m labels such that there is an edge between vertices i and j if and only if the inner product of the columns i and j is 1. Since the input graph is unweighted, the edges are simply ordered pairs and can be sorted in $O(m)$ time. Unlike the adjacency matrix, which is unique (up to fixed labeling of the vertices) for graph G, there can be more than one *intersection matrix* representation associated with graph G [1]. We exploit this flexibility to store a graph in a structured and space-efficient form.

Let $X \in \{0,1\}^{m\times n}$ be the intersection matrix as defined above associated with a graph $G = (V, E)$. Consider the product $B = X^\top X$.

Theorem 1. *The adjacency graph of matrix B is isomorphic to graph G. [1]*

Theorem 1 establishes the desired connection between a graph and its sparse matrix representation. The following result follows directly from Theorem 1.

Corollary 1. *The diagonal entry $B(i,i)$ where $B = X^\top X$ and X is the intersection matrix of graph G, is the degree $d(v_i)$ of vertex $v_i \in V, i = 1,\ldots,n$ of graph $G = (V, E)$. [1]*

Intersection matrix X defined above represents an edge clique cover of cardinality m for graph G. Each edge $\{v_i, v_j\}$ constitutes a clique of size 2. In the intersection matrix X, edge $e_l = \{v_i, v_j\}$ is represented by row l with $X(l,i) = X(l,j) = 1$ and other entries in the row being zero. In general, column indices j' in row l where $X(l,j') = 1$ constitutes a clique on vertices $v_{j'}$ of graph G. Thus the minECC problem can be cast as a *matrix compression* problem.

minECC Matrix Problem. Given $X \in \{0,1\}^{m\times n}$ determine $X' \in \{0,1\}^{k\times n}$ with k minimized such that the intersection graphs of X and X' are isomorphic.

3 An Edge-Centric minECC Algorithm

The algorithm that we propose for the ECC problem is motivated by the maximal clique algorithm due to Bron et al. [2], and E. Tomita et al. [22]. For ease of presentation, we discuss the algorithm in graph-theoretic terms. However, our computer implementation uses a sparse matrix framework of DSJM [11], and all computations are expressed in terms of intersection matrices.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

3.1 Selection of Uncovered Edges

An edge $\{u, v\} \in E$ is said to be *covered* if both of its incident vertices have been included in some clique; otherwise the edge is *uncovered*. In our algorithm, we select an uncovered edge $\{u, v\}$ and try to construct a maximal clique, C, containing the edge. The algorithm selects vertices and edges in a prespecified order during the clique construction process. Note that it may or may not be possible to include additional uncovered edges while building a clique after selecting an uncovered edge. This subsection will give details on how the algorithm selects an uncovered edge.

Vertex Ordering. We recall that $d(v)$ denotes the degree of vertex v in graph $G = (V, E)$. Let $Vertex\_Order$ be a list of vertices of graph G using one of the ordering schemes below.

- **Largest-Degree Order (LDO) (see [12]):** Order the vertices such that $\{d(v_i), i = 1, \ldots, n\}$ is nonincreasing.
- **Degeneracy Order (DGO) (see [7, 21]):** Let $V' \subseteq V$ be a subset of vertices of G. The subgraph induced by V' is denoted by $G[V']$. Assume the vertices $V' = \{v_n, v_{n-1}, \ldots, v_{i+1}\}$ have already been ordered. The i^{th} vertex in DGO is an unordered vertex u such that $d(u)$ is minimum in $G[V \setminus V']$ where, $G[V \setminus V']$ is the graph obtained from G by removing the vertices of set V' from V.
- **Incidence-Degree Order (IDO) (see [3]):** Assume that the first $k - 1$ vertices $\{v_1 \ldots, v_{k-1}\}$ in incidence-degree order have been determined. Choose v_k from among the unordered vertices that has maximum degree in the subgraph induced by $\{v_1, \ldots, v_k\}$.

Edge Ordering. After the vertices have been ordered using one of the above schemes, the algorithm proceeds to choose a vertex in that specific order, which has at least one uncovered incident edge. If there is more than one uncovered edge incident on the vertex being processed, the order in which the edges are processed (i.e., to include in a clique) is as follows. Place all the edges $\{u, v\}$ before $\{p, q\}$ in an ordered edge list, $Edge\_Order$, such that vertex u or vertex v is ordered before vertices p and q in $Vertex\_Order$ list.

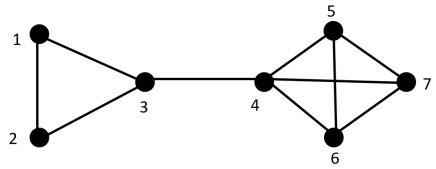

Fig. 1. An example of an undirected graph.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

Figure 1 shows an undirected graph. $\{4, 3, 5, 6, 7, 1, 2\}$ would be the list with LDO. The edge list induced by the $Vertex\_Order$ will have the following form.

$$Edge\_Order = \Big\{\{4,3\},\{4,5\},\{4,6\},\{4,7\},\{3,1\},\{3,2\},\{5,6\},\{5,7\},\{6,7\},\{1,2\}\Big\}$$

Edge Selection. We select an edge to $\{u, v\} \in E$ to include in a new clique if $\{u, v\}$ is uncovered and ordered before all uncovered edges in $Edge\_Order$. The clique that gets constructed with edge $\{u, v\}$ may cover other uncovered edges that are further down the list.

We consider three variants of edge selection for our algorithm, denoted by L, D, and I.

- **L**: In this variant, the set of vertices are ordered using the Largest-Degree Ordering (LDO) scheme. We select a vertex u in that order and then return all the uncovered edges of the form $\{u, v\}$.
- **D**: All the vertices are ordered using Degeneracy Ordering (DGO) scheme. Select a vertex u in that order, and then return all the uncovered edges of the form $\{u, v\}$.
- **I**: Finally, this variant orders the set of vertices using the Incidence-Degree Ordering (IDO) scheme. We select a vertex u in that order and return all the uncovered edges $\{u, v\}$.

3.2 The Algorithm

Let $E_{\mathcal{P}} = \{e_1, \ldots, e_{i-1}\}$ be the set of edges that have been assigned to one or more cliques $\{C_1, \ldots, C_{k-1}\}$ and let $e_i = \{v_j, v_{j'}\}$ be the edge currently being processed according to the ordered edge list. Denote by

$$W = \{v_l \mid \{v_j, v_l\}, \{v_{j'}, v_l\} \in E\}$$

the set of common neighbors of v_j and $v_{j'}$.

The complete algorithm is presented below.

EO-ECC ($Edge\_Order$)
Input: $Edge\_Order$, set of edges in a predefined order using schemes L, D, or I

$k \leftarrow 0$ ▷ Number of cliques
for $index = 1$ to m **do** ▷ m is the number of edges
  $\{u, v\} \leftarrow Edge\_Order[index]$
  if $\{u, v\}$ is uncovered **then**
    $W \leftarrow FindCommonNeighbors(u, v)$
    if $W = \emptyset$ **then**
      $k++$
      $C_k \leftarrow \{u, v\}$
      Mark $\{u, v\}$ as covered
    else
      $k++$
      $C_k \leftarrow \{u, v\}$
      Mark $\{u, v\}$ as covered
      while $W \neq \emptyset$ **do**
        let t be a vertex in W
        $W \leftarrow W \setminus t$

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

17: **if** $\{t, s\} \in E$ for each $s \in C_k$ **then**
18: Mark $\{t, s\}$ as covered
19: $C_k \leftarrow C_k \cup \{t\}$
20: $FindCommonNeighbors(W, FindNeighbors(t))$
21: **return** $C_1, C_2, ..., C_k$

3.3 Discussion

In this subsection, we analyze algorithm EO-ECC to derive it's asymptotic running time. The two kernel operations used in the algorithm are "FindCommonNeighbors" and "FindNeighbors." The *FindCommonNeighbors* operation merges two sorted lists (of integers) and computes the intersection of the lists. The list (of vertices) that this operation returns after each call has at least one fewer vertices. Thus, to construct a clique C_i, the total cost would be $(\frac{\rho_i(\rho_i-1)}{2})$, where $|C_i| = \rho_i$. Let, $C = \{C_1, C_2, \ldots, C_k\}$ be a clique cover returned by the algorithm EO-ECC. Then the total cost of calling *FindCommonNeighbors* for the algorithm would be $O(\sum_{i=1}^{k} \frac{\rho_i(\rho_i-1)}{2})$. The operation *FindNeighbors* in algorithm EO-ECC computes the neighbors set of vertex $v \in V$ [12]. In line 20, *FindNeighbors* operation is used to compute the neighbors of a vertex. Since an uncovered edge gets covered only once, the total cost of *FindNeighbors* operation is at most $O(m)$. Thus, the overall running time of algorithm EO-ECC is $O(m + \sum_{i=1}^{k} \frac{\rho_i(\rho_i-1)}{2})$. The following result follows immediately from the above running time expression.

Theorem 2. *If the input graph G is triangle-free, then the algorithm EO-ECC runs in $O(m)$ time.*

4 Numerical Testing

In this section, we provide results from numerical experiments on selected test instances. 10th Discrete Mathematics and Theoretical Computer Science (DIMACS10) instances and Stanford Network Analysis Platform (SNAP) instances are obtained from the University of Florida Sparse Matrix Collection [5]. (`SNAP`) is a collection of more than 50 large network datasets containing large number of nodes and edges including social networks, web graphs, road networks, internet networks, citation networks, collaboration networks, and communication networks [17]. We also experiment with synthetic graph instances. We generated 182 Erdös-Rényi and Small-World instances using the Stanford Network Analysis Project (SNAP) [18] instance generator. The number of edges of these generated graphs is varied from 800 to 72 million.

The experiments were performed using a PC with 3.4GHz Intel Xeon CPU, with 8 GB RAM running Linux. The implementation language was *C++* and the code was compiled using $-O2$ optimization flag with a *g++* version 4.4.7 compiler. We employed the High-Performance Computing system (Graham cluster) at Compute Canada for large instances that could not be handled by the PC.

In what follows, we refer to the vertex-centric ECC algorithm from [1] as Vertex Ordered Edge Clique Cover (`VO-ECC`). We also refer to the ECC algorithm due to Conte et al. as (`Conte-Method`). Finally, the edge-centric minECC algorithm of this paper is identified as Edge Ordered Edge Clique Cover (`EO-ECC`). `EO-ECC` has three variants associated with the three different edge ordering schemes `D`, `L`, and `I`. They are: `EO-ECC-D`, `EO-ECC-L`, and `EO-ECC-I` respectively. In these results, m denotes the

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

number of edges, n denotes the number of vertices of the graph; $|C|$ denotes the number of cliques in the cover, and t is the time in seconds to get the cover. In the presented tables, the smallest cardinality clique cover is marked in **bold**.

Table 1. Test Results (Number of cliques) for SNAP instances.

Graph			$\|C\|$		
Name	m	n	VO-ECC using [1]	Conte-Method using [4]	EO-ECC
p2p-Gnutella04	39994	10878	38474	38491	**38449**
p2p-Gnutella24	65369	26518	63726	63725	**63689**
p2p-Gnutella25	54705	22687	53368	53367	**53347**
p2p-Gnutella30	88328	36682	85823	85822	**85717**
ca-GrQc	14496	5242	3777	3753	**3717**
as-735	13895	7716	8985	**8938**	10130
Wiki-Vote	103689	8297	42914	**39393**	51145
Oregon-1	23409	11492	15631	**15491**	15527
ca-HepTh	25998	9877	9663	9270	**9162**

Table 1 displays the size of clique covers returned by three algorithms: the edge-centric algorithm (`EO-ECC`), the vertex-centric algorithm (`VO-ECC`) discussed in [1] and algorithm (`Conte-Method`) discussed in [4]. `Conte-Method` randomly selects an edge and attempts to build a clique around the selected edge. As the table illustrates, `EO-ECC` produces smaller cardinality edge clique cover than `VO-ECC` except for two instances. On the other hand, it outperforms `Conte-Method` on six out of nine instances.

Table 2. Test Results (number of cliques) for DIMACS10 matrices.

Graph			Number of cliques			
Name	m	n	Conte-Method	EO-ECC-D	EO-ECC-L	EO-ECC-I
chesapeake	170	39	**75**	76	**75**	76
delaunay_n10	3056	1024	1250	**1233**	1275	1241
delaunay_n11	6127	2048	2485	**2449**	2544	2481
delaunay_n12	12264	4096	4993	**4906**	5095	4939
delaunay_n13	24547	8192	9989	**9881**	10211	9920
delaunay_n14	49122	16384	19974	**19672**	20435	19855
delaunay_n15	98274	32768	39923	**39501**	40876	39782
delaunay_n16	196575	65536	79933	**78792**	81528	79445
delaunay_n17	393176	131072	159900	**157792**	163321	158851
delaunay_n18	786396	262144	319776	**315684**	326741	317987
com-DBLP	1049866	317080	238854	237713	**237685**	**237685**
belgium_osm	1549970	1441295	1545183	1545183	1545183	1545183
delaunay_n19	1572823	524288	639349	**631354**	653383	635877
delaunay_n20	3145686	1048576	1279101	**1262843**	1307080	1271229
delaunay_n21	6291408	2097152	2557828	**2525301**	2613106	2542333

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

Test results for the selected test instances from group `DIMACS10` are reported in Table 2. For comparison, we show the results of `Conte-Method`, `EO-ECC-D`, `EO-ECC-L`, and `EO-ECC-I`. On twelve out of fifteen instances, `EO-ECC-D` gives the least number of cliques to cover all the edges of the given graph. On the graph named `com-DBLP` `EO-ECC-L` and `EO-ECC-I` produce smaller cardinality covers. Overall, `EO-ECC` emerges as the clear winner over `Conte-Method` in terms of the size of the clique covers.

Besides DIMACS10 selected instances, we compare these algorithms on 182 generated instances where the number of edges is varied from 800 to 7.2×10^7. Using SNAP tool [18], we generated 72 "Small-world" and 110 "Erdös-Rényi" graphs. `EO-ECC` produces smaller (on 47.3%instances) or equal (on 52.7% instances) cardinality clique covers compared with `Conte-Method`.

Rodrigues [20] used different graph instances to evaluate their edge clique cover algorithms. The well-known instances to evaluate edge clique cover problem are from the application "compact letter display" [10]. On thirteen out of fourteen instances, `Conte-Method` [4] gives optimum results. Both Rodrigues's algorithm and our `EO-ECC` give optimum results for all the instances.

The performance comparison between `Conte-Method` and `EO-ECC` is shown in Figure 2. We compare the time required to find edge clique cover for the given graph.

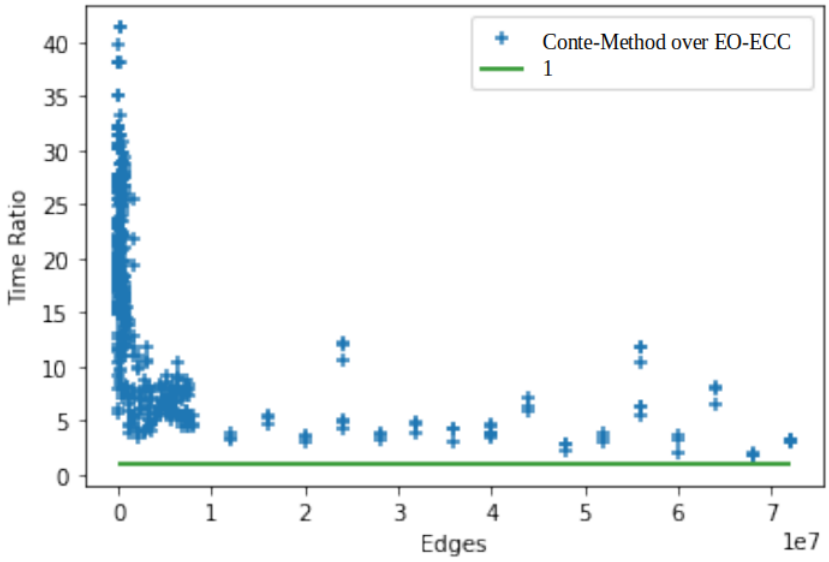

Fig. 2. Ratio between the time used by and `EO-ECC` for each graph, as a function of the number of the edges (y-axis is in log-scale).

We use fifteen DIMACS10 instances and 182 Erdos-Renyi and Small-World instances. In the figure, a cross mark represents the ratio between the time needed by and `EO-ECC`, as a function of the number of the edges. The green line at height 10^0 means that `Conte-Method` took the same time as `EO-ECC` to process the corresponding graph, and a cross mark at height 10^1 means that `Conte-Method` was ten times slower.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

As the figure clearly demonstrates, `EO-ECC` is always faster than `Conte-Method`, and more than 40 times faster on some of the test instances.

Table 3. Graph processing rate (number of edges processed per sec).

Group	Total instances	Largest rate	Smallest rate	Average rate
DIMACS10	15	$2.7E6$	$3.0E5$	$1.7E6$
SNAP	9	$2.5E6$	$6.2E4$	$1.5E6$
Erdos-Renyi	110	$2.0E6$	$1.2E5$	$8.9E5$
Small World	72	$1.7E6$	$4.3E5$	$1.1E6$

The graph processing rate is one of the quality assessment metrics for an algorithm. We report the processing rate of our algorithm for a selection of real-world (DIMACS10, SNAP) and synthetically generated (Erdös-Rényi, Small World) graphs in Table 3. Table 3 shows the largest rate, the smallest rate, and the average rate for each set of graph instances. On DIMACS10 instances, the algorithm performs the best, while on Erdös-Rényi instances, the algorithm is not as efficient. This can be explained by the structural properties of graphs. Real-life and Small World synthetic instances display a power-law degree distribution resulting in a large proportion of vertices with very small degrees. Thus, the set intersection operation in our algorithm can be very efficient on those types of graphs.

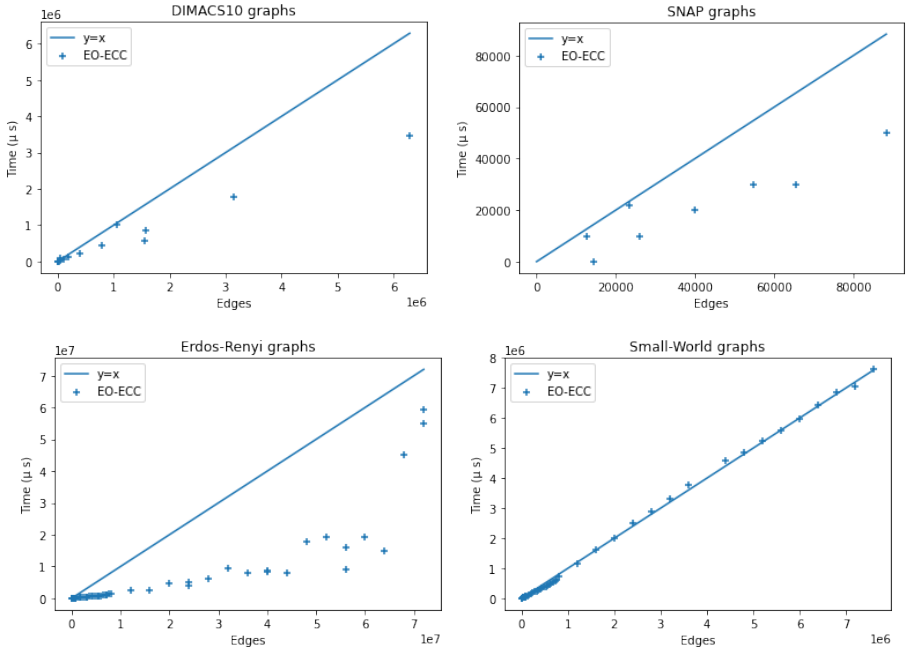

Fig. 3. Runtime to find clique cover using `EO-ECC`.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

Finally, in Figure 3, we demonstrate the superior scalability of our algorithm. The figure plots the time used to compute clique covers by `EO-ECC`, where the time is a function of the number of edges in the graph. We report the time in microseconds. A dot (x, y) states that the graph has x edges, and the algorithm spent y microseconds to finish the computation. The figure also displays the line $y = x$ for comparison with the actual running time. On each of the four sets of test instances, the running time shows a linear relationship with the number of edges, demonstrating that the running time of `EO-ECC` is linear in practice.

5 Conclusion

In this work, we have proposed a compact representation of network data. The edge clique cover problem is recast as a sparse matrix determination problem. The notion of *intersection matrix* provides a unified framework that facilitates the compact representation of graph data and efficient implementation of graph algorithms. The adjacency matrix representation of a graph can potentially have many nonzero entries since it is the product of an intersection matrix with its transpose. We have compared our results concerning the clique cover size and runtime with the current state-of-the-art algorithm for minECC [4]. Our algorithm achieves significantly smaller clique covers on the vast majority of the test instances and never returns a clique cover that is larger than the `Conte-Method` [4]. It is also significantly faster than the `Conte-Method`. `EO-ECC` algorithm runs in linear time, which allowed us to process extremely large graphs, both real-life and generated instances. Finally, our algorithm is highly scalable on large problem instances, while the algorithm of `Conte-Method` does not terminate on instances containing 7×10^7 or more edges within a reasonable amount of time.

A less well-studied but related problem, known as the *Assignment Minimum Edge Clique Cover* arising in computational statistics, is to minimize the number of individual assignments of vertices to cliques It is not always possible to find assignment-minimum clique coverings by searching through those that are edge-clique-minimum. Ennis et al. [6] presented a post-processing method with an existing ECC algorithm to solve this problem. However, their backtracking algorithm becomes costly for large graphs, especially when they have many maximal cliques. Our edge-centric method can be easily adapted, via a post-processing step, to assignment minimum cover calculation. This research is currently being carried out. Results from preliminary computational experiments with a new linear-time post-processing scheme are favourable.

Acknowledgments This research was supported in part by NSERC Discovery Grant (Individual), and the AITF Graduate Student Scholarship. A part of our computations were performed on Compute Canada HPC system (http://www.computecanada.ca), and we gratefully acknowledge their support.

References

1. Abdullah, W.M., Hossain, S., Khan, M.A.: Covering large complex networks by cliques—a sparse matrix approach. In: Kilgour, D.M., Kunze, H., Makarov, R., Melnik, R., Wang, X. (eds.) Recent Developments in Mathematical, Statistical and Computational Sciences. pp. 117–127. Springer International Publishing, Cham (2021)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

2. Bron, C., Kerbosch, J.: Algorithm 457: Finding all cliques of an undirected graph. Commun. ACM **16**(9), 575–577 (Sep 1973). https://doi.org/10.1145/362342.362367, https://doi.org/10.1145/362342.362367
3. Coleman, T.F., Moré, J.J.: Estimation of sparse jacobian matrices and graph coloring blems. SIAM journal on Numerical Analysis **20**(1), 187–209 (1983)
4. Conte, A., Grossi, R., Marino, A.: Large-scale clique cover of real-world networks. Information and Computation **270**, 104464 (2020)
5. Davis, T., Hu, Y.: Suitesparse matrix collection. https://sparse.tamu.edu/, accessed: 2019-10-02
6. Ennis, J., Ennis, D.: Efficient Representation of Pairwise Sensory Information. IFPress **15**(3), 3–4 (2012)
7. Eppstein, D., Strash, D.: Listing all maximal cliques in large sparse real-world graphs. In: International Symposium on Experimental Algorithms. pp. 364–375. Springer (2011)
8. Erdös, P., Goodman, A.W., Pósa, L.: The representation of a graph by set intersections. Canadian Journal of Mathematics **18**, 106–112 (1966)
9. Gramm, J., Guo, J., Huffner, F., Niedermeier, R.: Data reduction, Exact and Heuristic Algorithms for Clique Cover. Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX), SIAM pp. 86–94 (2006)
10. Gramm, J., Guo, J., Huffner, F., Niedermeier, R., Piepho, H., Schmid, R.: Algorithms for Compact Letter Displays: Comparison and Evaluation. Computational Statistics & Data Analysis **52**, 725–736 (2007)
11. Hasan, M., Hossain, S., Khan, A.I., Mithila, N.H., Suny, A.H.: DSJM: A Software Toolkit for Direct Determination of Sparse Jacobian Matrices. In: G.M. Greuel, T. Koch, P. Paule, A. Sommese and Editors. ICMS2016. Springer International Publishing Switzerland pp. 425–434 (2016)
12. Hossain, S., Khan, A.I.: Exact Coloring of Sparse Matrices. In: D.M. Kilgour et al. (eds.) Recent Advances in Mathematical and Statistical Methods. Springer Proceedings in Mathematics and Statistics, Springer Nature Switzerland AG **259**, 23–36 (2018)
13. Hossain, S., Suny, A.H.: Determination of Large Sparse Derivative Matrices: Structural: Orthogonality and Structural Degeneracy. In: B. Randerath, H. Roglin, B. Peis, O. Schaudt, R. Schrader, F. Vallentin and V. Weil. 15th Cologne-Twente Workshop on Graphs & Combinatorial Optimization, Cologne, Germany pp. 83–87 (2017)
14. Kepner, J., Gilbert, J.: Graph Algorithms in the Language of Linear Algebra, Society for Industrial and Applied Mathematics. Philadelphia, PA, USA (2011)
15. Kepner, J., Jananthan, H.: Mathematics of big data: Spreadsheets, databases, matrices, and graphs. MIT Press (2018)
16. Kou, L., Stockmeyer, L., Wong, C.: Covering edges by cliques with regard to keyword conflicts and intersection graphs. Communications of the ACM **21**(2), 135–139 (1978)
17. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014), accessed: 2019-10-02
18. Leskovec, J., Sosič, R.: Snap: A general-purpose network analysis and graph-mining library. ACM Transactions on Intelligent Systems and Technology (TIST) **8**(1), 1 (2016)
19. Park, J.S., Penner, M., Prasanna, V.K.: Optimizing graph algorithms for improved cache performance. IEEE Transactions on parallel and distributed systems **15**(9), 769–782 (2004)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43

20. Rodrigues, M.O.: Fast constructive and improvement heuristics for edge clique covering. Discrete Optimization **39**, 100628 (2021)
21. Rossi, R.A., Gleich, D.F., Gebremedhin, A.H., Patwary, M.M.A.: Fast maximum clique algorithms for large graphs. In: Proceedings of the 23rd International Conference on World Wide Web. pp. 365–366 (2014)
22. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical computer science **363**(1), 28–42 (2006)
23. Wasserman, S., Faust, K.: Social network analysis: Methods and applications. Cambridge university press (1994)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_43