Covering Large Complex Networks by Cliques—A Sparse Matrix Approach

W. M. Abdullah, S. Hossain, and M. A. Khan

Abstract The Edge Clique Cover (ECC) problem is concerned with covering edges of a graph with the minimum number of cliques, which is an NP-hard problem. This problem has many real-life applications, such as, in computational biology, food science, efficient representation of pairwise information, and so on. In this work we propose using a compact representation of network data based on sparse matrix data structures. Building upon an existing ECC heuristic due to Kellerman we proffer adding vertices during the clique-growing step of the algorithm in judiciously chosen degree-based orders. On a set of standard benchmark instances our ordered approach produced smaller sized clique cover compared to unordered processing.

Keywords Adjacency matrix · Clique cover · Intersection matrix · Vertex ordering · Sparse graph

1 Introduction

Identification of and computation with dense or otherwise highly connected subgraphs are two of the kernel operations arising in areas as diverse as sparse matrix determination and complex network analysis [1, 6, 9]. Identification of special interest groups or characterization of information propagation are examples of frequently performed operations in social networks [8]. Efficient representation of network data is critical to addressing algorithmic challenges in the analysis of massive data sets using graph theoretic abstractions. In this paper, we propose sparse matrix data

W. M. Abdullah (✉) · S. Hossain
University of Lethbridge, Alberta, Canada
e-mail: w.abdullah@uleth.ca

S. Hossain
e-mail: shahadat.hossain@uleth.ca

M. A. Khan
Inbridge, Alberta, Canada
e-mail: muhammad@inbridgeinc.com

D. M. Kilgour et al. (eds.), *Recent Developments in Mathematical, Statistical and Computational Sciences*, Springer Proceedings in Mathematics & Statistics 343,
https://doi.org/10.1007/978-3-030-63591-6_11

structures to enable compact representation of graph data and use an existing sparse matrix framework [5] to design efficient algorithms for the ECC problem.

Let $G = (V, E)$ be an undirected connected graph with $|V| = n$ vertices and $|E| = m$ edges. A clique is a subset of vertices such that every pair of distinct vertices are connected by an edge in the induced (by the subset of vertices) subgraph. An edge clique cover of size k in graph G is a decomposition of set E 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 can be specified by the set of edges E with each edge being a clique. The problem of finding a clique cover with minimum number of cliques (and many variants thereof) is known to be NP-hard [7].

In the literature, the ECC problem and its variants have been extensively investigated from theoretical perspectives and have found applications in disparate areas. In [3], the authors describe a branch-and-bound approach to determine sparse Jacobian matrices. Given the sparsity pattern of the Jacobian, the problem is to find a partition of the columns into structurally orthogonal column groups of smallest cardinality. Blanchette et al. [15] study the protein complex identification problem from computational biology, where the problem is to identify overlapping protein complexes in protein-protein interaction networks. When modelled as a graph problem, the goal is to decompose the network into a smallest collection of cliques. Several polynomial time algorithms have been proposed in the paper for graphs with bounded tree-width. In sensory science, a seemingly unrelated application area, a frequently occurring task is concerned with the concise representation of pairwise interaction of products with many attributes [14, 16]. This pairwise information can be given in a tabular form called "compact letter display". The challenge is to minimize redundant information. It has been shown that this problem can be posed as a variant of the ECC problem [16].

Many heuristics have been proposed in the literature to approximately solve ECC problem while there are only few exact methods which are usually limited to solving small instance sizes. A recent approach is described by Gramm et al. in [10], where they introduce and analyze data reduction techniques to shrink the instance size without sacrificing the optimal solution. The main idea is that with small enough instance sizes, exact algorithms may become feasible.

In this paper we propose a compact representation of network data based on sparse matrix data structures [3] and provide an improved algorithm based on an existing heuristic for finding clique cover. Our approach is based on the simple but critical observation that for a sparse matrix $A \in \mathbb{R}^{m \times n}$, the column intersection graph of A is isomorphic to the adjacency graph of $A^\top A$, and that the row intersection graph of A is isomorphic to the adjacency graph of $AA^\top$ [5]. Consequently, the subset of columns corresponding to nonzero entries in row i induces a clique in the adjacency graph of $A^\top A$, and the subset of rows corresponding to nonzero entries in column j induces a clique in the adjacency graph of $AA^\top$. Note that, matrices $A^\top A$ and $AA^\top$ are most likely dense even if matrix A is sparse. In this work, we exploit the 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 existing sparse matrix computational framework to solve graph problems [5]. This duality between graphs and sparse matrices has also been exploited where the graph algorithms are expressed in the language of sparse linear algebra [1, 4]. However, they use adjacency matrix representation which is different from our intersection matrix representation.

We organize the rest of the paper in the following way. In Sect. 2, we present our main theoretical result that allows us to pose the ECC problem as a matrix determination problem. This is followed by a brief description of the clique-cover heuristic of [11]. Next, we describe algorithms for preprocessing the vertices according to their degree in the graph. Results from numerical experiments on a standard collection of test instances are provided in Sect. 3. Finally, the paper is concluded in Sect. 4.

2 Compact Representation and Edge Clique Cover

Classical data structures adjacency matrix (full matrix storage) and adjacency list for representing graphs are inadequate for efficient computer implementation of many important graph operations. Adjacency matrix is costly for sparse graphs and typical adjacency list implementations employ pointers where indirect access leads to poor cache utilization. In a typical adjacency list implementation of undirected graphs, each edge is represented twice. An alternative adjacency list representation of undirected labelled graph avoids this redundancy by storing each edge only once, where the edges incident on each vertex are stored in sorted order of vertex labels [17]. The intersection matrix representation below enables efficient representation of pairwise information where the edges are implicit. Moreover, it allows us to utilize computational framework DSJM to implement the ECC algorithms.

2.1 Intersection Matrix

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

Let $G = (V, E)$ be an undirected and connected graph without self-loops or multiple edges between a pair of vertices. The adjacency matrix $A(G) \equiv A \in \{0, 1\}^{|V|\times|V|}$ associated with graph G is defined as,

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

Unlike the adjacency matrix which is unique (up to a fixed labeling of the vertices) for graph G, there can be more than one *(column) intersection matrix* associated with graph G. We exploit this flexibility to store a graph in a structured and space-efficient form using an intersection matrix. Let the edges in E be labelled $e_1, \ldots, e_{|E|}$. An intersection matrix associated with graph $G = (V, E)$ where $|V| = n$ and $|E| = m$, is a matrix $C \in \{0, 1\}^{m \times n}$ where for edge $e_k = \{v_i, v_j\}, k = 1, \ldots, m$ we have $C(k, i) = C(k, j) = 1$, and all other entries of matrix C are zero.

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

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

Proof Consider an arbitrary edge $e_k = \{v_i, v_j\}$ of graph G. By construction, row k of the intersection matrix C has $C(k, i) = C(k, j) = 1$ and $C(k, l) = 0$ for $l \notin \{i, j\}$. Since there are no multiple edges in G, there is one and only one such row k corresponding to edge e_k. Element $B(i, j)$ is the inner product of column vectors i and j of matrix C. The inner product is 1 if and only if $C(k, i) = C(k, j) = 1$. Thus, e_k is in E if and only if $B(i, j) = 1$ implying that it is an edge connecting vertices v_i and v_j of the adjacency graph of matrix B. This proves the theorem. □

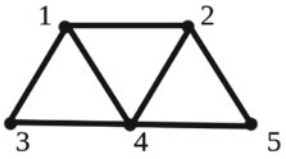
Theorem 1 establishes the desired connection between a graph and its sparse matrix representation. For a vertex $v \in V$ we define by $N_v = \{w \in V \mid \{v, w\} \in E\}$ the set of its neighbors. The *degree* of a vertex v, denoted $d(v)$, is the cardinality of set N_v. The following result follows directly from Theorem 1.

Corollary 1 *The diagonal entry $B(i, i)$ where $B = C^\top C$ and C 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 $G = (V, E)$.*

Intersection matrix C 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 C, the clique (edge) is represented by row k with $C(k, i) = C(k, j) = 1$ and other entries in the row being zero. In general, column indices l in row k where $C(k, l) = 1$ constitutes a clique on vertices v_l of graph G. Thus the ECC problem can be cast as a matrix compression problem.

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

Figure 1a displays a graph on 5 vertices. Figure 1b depicts an intersection matrix representing an edge clique cover of cardinality 7 (number of edges). In the figure a dark dot represents numerical value 1 while a blank entry is a zero. The intersection matrix in Fig. 1c corresponds to an edge clique cover with three cliques. This is also the minimum clique cover for the given graph. To verify that it represents a clique cover, we examine each row of the matrix. Row 1 has dots in columns 1, 3, 4 representing the clique on vertices 1, 3, 4. Row 2 represents the clique on vertices 2, 4, 5 and the remaining edge is covered by row 3.



(a) $G = (V, E)$

	1	2	3	4	5
1	●	●			
2	●		●		
3	●			●	
4		●		●	
5		●			●
6			●	●	
7				●	●

(b) Intersection matrix of cardinality 7

	1	2	3	4	5
1	●		●	●	
2		●		●	●
3	●	●			

(c) Intersection matrix of cardinality 3

Fig. 1 ECC as a sparse matrix problem

2.2 A Heuristic for Clique Cover

The heuristic algorithm that we have implemented for the ECC problem is based on an algorithm due to Kellerman [11]. For ease of presentation we discuss the algorithm in graph theoretic terms. However, our computer implementation uses sparse matrix framework of DSJM [5] and all computations are expressed in terms of intersection matrices.

There is a close connection between the clique cover of a graph $G = (V, E)$ and the coloring of vertices of the complement graph $\bar{G} = (V, \bar{E})$ where $\bar{E} = \{\{u, v\} \mid \{u, v\} \notin E\}$. In the classical graph coloring problem, vertices of the graph are partitioned into subsets (colors) such that pair of vertices connected by an edge are in different subsets. The optimization version asks for the partition with smallest number of subsets. It is well-known that the greedy coloring heuristic is sensitive to the order in which the vertices are processed (see [3]). Consider an optimal coloring of graph G and order the vertices in nondecreasing color index. It is not difficult to see that the greedy heuristic on graph G with the given order of the vertices produces optimal coloring. We experimentally verify that the ECC heuristic is sensitive to the ordering in which the vertices are processed. We employ three vertex ordering algorithms from the literature: Largest-first order (LFO), Smallest-Last Order (SLO), and Incidence-degree Order (IDO) prior to applying the heuristic [11]. We recall that $d(v) = |N_v|$ denotes the degree of vertex v in graph $G = (V, E)$.

- **(LFO)** Order the vertices such that $\{d(v_i), i = 1, \ldots, n\}$ is nonincreasing.
- **(SLO)** Assume that the last $n - k$ vertices $\{v_{k+1} \ldots, v_n\}$ in smallest-last order have been determined. The kth vertex in the order is an unordered vertex whose degree in the subgraph induced by

$$V \setminus \{v_{k+1}, \ldots, v_n\}$$

is minimum.

- **(IDO)** 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

$$\{v_1, \ldots, v_k\}$$

Next, we present the algorithm for the ECC problem.

Let the vertices of graph $G = (V, E)$ be ordered in one of SLO, LFO, and IDO: $v_1, \ldots, v_n$. Also, let $V_{\mathcal{P}} = \{v_1, \ldots, v_{i-1}\}$ denote the vertices that have been assigned to one or more cliques $\{C_1, \ldots, C_{k-1}\}$ and v_i be the vertex currently being processed. Denote by set

$$W = \{v_j \mid j < i \text{ and } \{v_i, v_j\} \in E\}$$

the neighbors of v_i in $V_{\mathcal{P}}$. The task is to assign v_i to one or more of the existing cliques (or create a new clique) such that each edge incident on v_i that connects to a vertex in $V_{\mathcal{P}}$ is covered by a clique. There are three possibilities:

Case I. W is empty: Create a new clique $C_k = \{v_i\}$

Case II. W is not empty:

Case a. There is a clique $C_l, l \in \{1, \ldots, k-1\}$ such that $W = C_l$: add v_i to C_l

Case b. There is no such clique:

i. If $C_l \subset W$ for some l, add v_i to C_l together with uncovered edges from $V_{\mathcal{P}}$. Update W by removing edges that got covered.

ii. If there are uncovered edges after step II(b(i)) create a new clique from an existing clique and add v_i and the incident edges until all the edges of W are covered.

The complete algorithm is presented below.

VertexOrderedECC $(W, list)$

```
1:  k ← 0                                          ▷ Number of cliques
2:  for index = 1 to N do                          ▷ N denotes the number of vertices
3:      i ← list[index]                            ▷ list contains the vertices in a predefined order
4:      if W = ∅ then                              ▷ W ← {j | j < i and {i, j} ∈ E}
5:          k ← k + 1
6:          C_k ← {i}                              ▷ C_k denotes k^th clique
7:      else
8:          U ← ∅                                  ▷ Contains neighbours of i, which are in the cliques
9:          for l = 1 to k do
10:             if C_l ⊆ W then
11:                 C_l ← C_l ∪ {i}
12:                 U ← U ∪ C_l
13:                 if U = W then
```

14: break
15: $W \leftarrow W \setminus U$
16: **while** $W \neq \emptyset$ **do**
17: $Max \leftarrow \emptyset$
18: $MIN_l \leftarrow 0$
19: **for** $l = 1$ to k **do**
20: **if** $|Max| < |(C_l \cap W)|$ **then**
21: $Max \leftarrow (C_l \cap W)$
22: $MIN_l \leftarrow l$
23: $l \leftarrow MIN_l$
24: $k \leftarrow k + 1$
25: $C_k \leftarrow (C_l \cap W) \cup \{i\}$
26: $W \leftarrow W \setminus C_l$
27: **return** $C_1, C_2, ..., C_k$

We argue that the cliques $C_1, C_2, ..., C_k$ returned by the algorithm constitutes an edge clique cover for the input graph G.

The main **for**-loop (line 2) reads the next vertex (i) from the ordered list of vertices and tries to include it in one of the existing cliques, or creates new clique(s) with vertex i included. If vertex i has no neighbor ($W = \emptyset$) in $V_{\mathcal{P}}$, a new clique gets created (line 6). If the neighbor set W is not empty, the algorithm tries to identify existing cliques C_l that are subsets of W and assigns vertex i to each of them (lines 9 – 15, **Case 2. a.** and **Case 2. b. i.**). This step covers edges of the form $\{i, i'\}$ where $i' \in C_l$, $C_l \subset W$. Finally, the **while**-loop (line 16) covers the remaining edges (**Case II. b. ii.**) of the form $\{i, i'\}$ where $i' \in S$, $S = W \cap C'_l$, $l' \in \{1, 2, \ldots, l\}$ with $|S|$ maximum. The maximality on $|S|$ ensures that each newly created clique covers largest number of uncovered edges. For a graph $G = (V, E)$ each edge is a clique of size 2 so that set E constitute an (trivial) ECC. Therefore, each edge of input graph G eventually gets assigned to one of the cliques output by algorithm **VertexOrderedECC**.

The above discussion can be summarized in the following result.

Lemma 1 *The collection* $\{C_1, C_2, \ldots, C_k\}$ *computed by Algorithm* ***VertexOrderedECC*** *constitutes an ECC of graph* G.

3 Numerical Testing

In this section, we provide results from numerical experiments on selected test instances. The graph instances are chosen from standard benchmark collections that are used in the literature for ECC and closely related graph problems such as, graph coloring, graph partitioning, etc. The data set for the experiments is obtained from the University of Florida Sparse Matrix Collection [12]. Instances **chesapeake, delaunay_n10 to 13, as-22july06** are from "10th DIMACS Implementation Challenge" benchmark collection for graph clustering and graph partitioning. Instances **ca-GrQc, as-735, Wiki-Vote, p2p-Gnutella04, Oregon-1** are from "Stanford Net-

work Analysis Platform (SNAP)" collection. These instances represent social networks from variety of apllications. We also consider the data set for Compact Letter Displays used in [13]. The experiments were performed using a PC with 3.4G Hz Intel Xeon CPU, 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.

A short description of the data set for our experiments is as follows:

- **chesapeake:** Symmetric, undirected graph and contains 39 vertices and 170 edges.
- **delaunay_n10 to 13:** The graphs are symmetric and undirected. The minimum degree is 3 for all of them and the maximum degrees are 12, 13, 14 and 12 respectively.
- **as-22july06:** The graph is symmetric and undirected having maximum degree 2.4K and minimum degree 1.
- **ca-GrQc:** General Relativity and Quantum Cosmology network covers scientific collaboration between authors in this field. This graph contains an undirected edge from i to j, if author i co-authored a paper with author j.
- **as-735:** An autonomous system which represents a communication network of who-talks-to whom.
- **Wiki-Vote:** This data set contains voting data of Wikipedia till January 2008 where the contest was between volunteers to become one of the administrator. There is a directed edge from node i to node j if user i voted for user j.
- **p2p-Gnutella04:** A snapshoot of Gnutella peer-to-peer file sharing network on August 04, 2002. A directed graph where nodes represent hosts and edges represent connection between hosts.
- **Oregon-1:** Undirected graph where autonomous system peering information is inferred from Oregon route-views on May 26, 2001.
- **Triticale, winter wheat and oilseed rape yield trials:** These instances are from the application "compact letter display" [13] to test ECC algorithms.

Test results for the selected test instances from group DIMACS10 and SNAP are reported in Tables 1 and 2 respectively. Test results for Compact Letter Display are reported in Table 3. Here, N represents the number of vertices and M represents the number of edges of the graph. $|C|$ represents number of cliques required to cover all the edges.

For comparison we also show the ECC results where no specific vertex ordering is employed, in addition to ordering algorithms LFO, SLO, and IDO. Column labelled Natural reports the ECC result when the vertices are processed in the order they are specified in the data file. On `DIMACS10` instances, smallest last order gives the best result except for instance named `as-22july06`. On `SNAP` instances largest-first order is the overall winner. Note that on both sets of test instances ordered approach produces strictly better ECC compared with `Natural`. We remark that OCaml implementation from [2] fails (hangs) to run on `DIMACS10` and `SNAP` instances. As such no comparison of the ECC quality (size) can be made. Table 3 displays

Table 1 Test results for DIMACS10 matrices

Matrix			Natural	SLO	LFO	IDO
Name	N	M	$\|C\|$	$\|C\|$	$\|C\|$	$\|C\|$
chesapeake	39	170	90	**79**	83	80
delaunay_n10	1024	3056	1300	**1223**	1302	1268
delaunay_n11	2048	6127	2610	**2482**	2617	2527
delaunay_n12	4096	12264	5228	**4973**	5264	5061
delaunay_n13	8192	24547	10489	**9937**	10541	10121
as-22july06	22963	48436	34695	34772	**34568**	34666

Table 2 Test results for SNAP matrices

Matrix			Natural	SLO	LFO	IDO
Name	N	M	$\|C\|$	$\|C\|$	$\|C\|$	$\|C\|$
ca-GrQc	5242	14496	3791	3879	**3777**	3900
as-735	7716	13895	9055	9108	**8985**	9038
Wiki-Vote	7115	103689	43497	45530	**42482**	45491
p2p-Gnutella04	10876	39994	38475	**38474**	38475	**38474**
Oregon-1	11174	23409	15736	15807	**15631**	15857

Table 3 Test results for compact letter displays [13]

Graph			Degree ordered method	Insert-absorb	Clique-growing	Search tree
Name	N	M	$\|C\|$	$\|C\|$	$\|C\|$	$\|C\|$
Triticale 1	13	55	4	4	4	4
Triticale 2	17	86	5	5	5	5
Wheat 1	124	4847	50	56	50	49
Wheat 2	121	4706	48	50	48	48
Wheat 3	97	3559	32	39	32	31
Rapeseed 1	47	576	20	20	20	20
Rapeseed 2	57	1040	20	20	20	20
Rapeseed 3	64	1260	24	24	24	24
Rapeseed 4	62	1085	19	19	19	19
Rapeseed 5	64	1456	19	19	19	19
Rapeseed 6	70	1416	27	27	27	27
Rapeseed 7	74	1758	26	29	27	25
Rapeseed 8	59	1128	17	17	17	17
Rapeseed 9	76	1835	30	30	30	30

results using our degree ordered method and two other algorithms discussed in [13]. `Insert Absorb` and `Search Tree` require exponential running time while `Clique Growing` method is an improved implementation of the heuristic of [11]. `Search Tree` is an exact method that produces optimal ECC. `Degree Order Method` reports the best ECC of our implementation. It is evident from the table that our method produces optimal or near optimal (off by 1) ECC.

4 Conclusion

In this work, we have shown that the connection between large networks and their sparse matrix representation can be exploited to employ efficient techniques from sparse matrix determination literature in graph algorithms [18, 19]. The edge clique cover problem is recast as a sparse matrix determination problem. The notion of *intersection matrix* provides a unified framework that facilitates 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 shown that, similar to graph vertex coloring problem, the ECC problem is sensitive to ordering of the vertices.

Acknowledgements We thank referees for their many valuable suggestions that helped improve the paper. This research was partially supported by the Natural Sciences and Engineering Research Council (NSERC) under Discovery Grants Program.

References

1. Kepner, J., Gilbert, J.: Graph Algorithms in the Language of Linear Algebra. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2011)
2. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Data reduction and exact algorithms for clique cover. J. Exp. Algorithmics (JEA). **13**, 2–15 (2009)
3. Hossain, S., Khan, A.I.: Exact coloring of sparse matrices. In: Kilgour, D.M., et al. (eds.) Recent Advances in Mathematical and Statistical Methods. Springer Proceedings in Mathematics and Statistics, vol. 259, pp. 23–36. Springer Nature, Switzerland AG (2018)
4. Kepner, J., Jananthan, H.: Mathematics of Big Data: Spreadsheets, Databases, Matrices, and Graphs. MIT Press (2018)
5. 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: Greuel, G.M., Koch, T., Paule, P., Sommese, A. (eds.) ICMS2016, pp. 425–434. Springer International Publishing, Switzerland (2016)
6. Hossain, S., Suny, A.H.: Determination of large sparse derivative matrices: structural: orthogonality and structural degeneracy. In: Randerath, B., Röglin, H., Peis, B., Schaudt, O., Schrader, R., Vallentin, F., Weil, V. (eds.) 15th Cologne-Twente Workshop on Graphs & Combinatorial Optimizationpp, pp. 83–87. Cologne, Germany (2017)
7. Kou, L.T., Stockmeyer, L.J., Wong, C.K.: Covering edges by cliques with regard to keyword conflicts and intersection graphs. Commun. ACM. **21**(2), 135–139 (1978)

8. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press (1994)
9. James, O.: Contentment in graph theory: covering graphs with cliques. Indagationes Mathematicae (Proceedings), vol. 80, no. 5. North-Holland (1977)
10. Gramm, J., Guo, J., Hüffner, F. Niedermeier, R.: Data reduction, exact and heuristic algorithms for clique cover. In: Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX). pp. 86–94. SIAM (2006)
11. Kellerman, E.: Determination of keyword conflict. IBM Tech. Discl. Bull. **16**(2), 544–546 (1973)
12. SuiteSparse Matrix Collection. https://sparse.tamu.edu/. Accessed 02 Oct 2019
13. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R., Piepho, H., Schmid, R.: Algorithms for compact letter displays: comparison and evaluation. Comput. Stat. Data Anal. **52**, 725–736 (2007)
14. Nestrud, M.A., Ennis, J.M., Fayle, C.M., Ennis, D.M., Lawless, H.T.: Validating a graph theoretic screening approach to food item combinations. J. Sens. Stud. **26**(5), 331–338 (2011)
15. Blanchette, M., Kim, E., Vetta, A.: Clique cover on sparse networks. In: 2012 Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 93–102. Society for Industrial and Applied Mathematics, 2012 Jan 16
16. Ennis, J.M., Ennis, D.M.: Efficient representation of pairwise sensory information. IFPress **15**(3), 3–4 (2012)
17. Tinhofer, G.: Generating graphs uniformly at random. In: Tinhofer, G., Mayr, E., Noltemeier, H., Syslo, M.M. (eds.) Computational Graph Theory. Computing Supplementum, vol. 7, pp. 235–255. Springer, Vienna (1990)
18. Hossain, S., Steihaug, T.: Graph models and their efficient implementation for sparse Jacobian matrix determination. Discrete Appl. Math. **161**(12), 1747–1754 (2013)
19. Hossain, S., Steihaug, T.: Optimal direct determination of sparse Jacobian matrices. Optim. Methods Softw. (2012). https://doi.org/10.1080/10556788.2012.693927