Elsevier Editorial System(tm) for
Information Sciences

Manuscript Draft

Manuscript Number:

Title: Compact structure for sparse undirected graphs based on a clique-graph partition

Article Type: Full length article

Keywords: graph compression;
clustering;
compact data structures;
social networks analysis;
maximal cliques

Corresponding Author: Mrs. Cecilia Hernandez,

Corresponding Author's Institution: University of Concepcion

First Author: Felipe Glaria

Order of Authors: Felipe Glaria; Cecilia Hernandez; Susana Ladra; Gonzalo Navarro; Lilian Salinas

Abstract: Compressing real-world graphs may have many benefits, such as, improving or enabling visualization in small memory devices, graph query processing, and mining algorithms. In this paper, we propose a compact representation for real
sparse and clustered undirected graphs based on using vertex redundancy of their maximal cliques and representing their edges implicitly. Our approach first lists all maximal cliques using a fast algorithm and defines a clique-graph based on its maximal cliques. Then, we define an effective heuristic for finding a clique-graph partition that avoids the construction of the clique-graph. Finally, we use this partition to define a compact representation of the input graph. Our experimental evaluation shows that our approach is competitive with the state-of-the-art compression efficiency, access times for neighbor queries, and recovers all maximal cliques much faster than using the original graph.

Suggested Reviewers: Yinghong Ma
yinghongma71@163.com

Cover Letter

Department of Computer Science
University of Concepcion
Chile
Tel: +56-41-2203568

July 19, 2019

Executive Editor
Information Sciences Journal

Manuscript "Compact structure for sparse undirected graphs based on a clique-graph partition" by Felipe Glaria, Cecilia Hernandez, Susana Ladra, Gonzalo Navarro, and Lilian Salinas.

Dear Editor,

I am please to enclosed the manuscript titled "Compact structure for sparse undirected graphs based on a clique-graph partition" by Felipe Glaria, Cecilia Hernandez, Susana Ladra, Gonzalo Navarro, and Lilian Salinas, for your consideration to publication in *Information Sciencies Journal*.

The submitted manuscript is within the scope of the Journal since it proposes a novel computational approach that includes modeling, developing, evaluation and analysis for a compact representation of large undirected graphs.

The proposed approach is based on effective heuristics for partitining large and sparse undirected graphs represented by its maximal cliques. Such partitioning is used to define a compact data structure consisting of two integer sequences, a byte sequence and a bitmap. We compare our best alternatives with the state-of-the-art compression techniques and our compact representation provides competitive compression efficiency and random access time to adjacent neighbors of any vertex. In addition, it supports the recovery of all graph maximal cliques.

The submitted manuscript is not under review by any other journal or conference, nor has it been published or accepted for publication at another journal or conference. All authors have read the final manuscript version.

Thank you for your consideration. I look forward to hearing from you.

Sincerely,

Cecilia Hernandez, Ph. D.
Department of Computer Science
University of Concepcion, Chile.

Enclosures:
1) Manuscript.

Highlights

- A novel compact representation for sparse and clustered undirected graphs based on maximal cliques.
- The representation supports queries for accessing adjacent neighbors for any vertex in the graph.
- The proposed algorithm first list all maximal cliques of an input graph, then it defines effective heuristics to build a partition of cliques, which it is represented compactly using two integer sequences, a byte sequence and a bitmap.
- The compression efficiency and random query access times are competitive with the state-of-the-art compression techniques.
- The compact representation also enables the recovery of all maximal cliques between 1.9 and 27 times faster than from the original graph.

Compact structure for sparse undirected graphs based on a clique-graph partition

Felipe Glaria[a], Cecilia Hernández[a,*], Susana Ladra[b], Gonzalo Navarro[c,d], Lilian Salinas[a]

[a] *Department of Computer Science, University of Concepcion, Concepción, Chile*
[b] *Universidade da Coruña, CITIC, A Coruña, Spain*
[c] *IMFD — Millennium Institute for Foundational Research on Data, Chile*
[d] *Department of Computer Science, University of Chile, Santiago, Chile*

Abstract

Compressing real-world graphs may have many benefits, such as, improving or enabling visualization in small memory devices, graph query processing, and mining algorithms. In this paper, we propose a compact representation for real sparse and clustered undirected graphs based on using vertex redundancy of their maximal cliques and representing their edges implicitly. Our approach first lists all maximal cliques using a fast algorithm and defines a clique-graph based on its maximal cliques. Then, we define an effective heuristic for finding a clique-graph partition that avoids the construction of the clique-graph. Finally, we use this partition to define a compact representation of the input graph. Our experimental evaluation shows that our approach is competitive with the state-of-the-art compression efficiency, access times for neighbor queries, and recovers all maximal cliques much faster than using the original graph.

Keywords: graph compression, clustering, compact data structures, social networks analysis, maximal cliques

*Corresponding author
Email addresses: felipeglaria@udec.cl (Felipe Glaria), cecihernandez@udec.cl (Cecilia Hernández), susana.ladra@udc.es (Susana Ladra), gnavarro@dcc.uchile.cl (Gonzalo Navarro), lilisalinas@udec.cl (Lilian Salinas)

1. Introduction

A wide variety of real systems that are modeled by graphs, including communication, transit, Web, social, and biological networks. The process of discovering relevant information from graphs is usually referred to as graph mining. This is usually a time-consuming task, especially with the current trend of data size growth. The main challenges are triggered by different aspects, such as the data volume itself, the data complexity (i.e., many relationships among the data), and application needs [27]. Several schemes have been proposed for analyzing graphs aiming at understanding the properties and patterns found in them to serve different application purposes. Some known applications include recommendation systems [20], graph compression [5, 10, 28], measuring the relative relevance of network actors [30, 11], and network visualization [24, 8]. Recent works on graph mining postulate that dense patterns are prominent and describe different dense substructures such as maximal cliques [18, 7], communities [16], and others [14, 29, 19, 10]. Such substructures have been used for improving network analysis, graph compression [10, 4] and visualization [24, 8].

Given the resource consumption imposed by large graphs, the research community has proposed graph compression formats supporting basic navigation queries directly over the compressed structure without the need of decompression. This approach enables the simulation of any graph algorithm in main memory using much less space than a plain representation. Even though these compressed structures are usually slower than uncompressed representations, they are still attractive in devices of limited memory, such as tablets or cell phones, or they provide faster access than incurring in I/O costs. Most compression methods are based on exploiting patterns that provide compression opportunities, such as locality and similarity of adjacency lists [2], sparseness and clustering of the adjacency matrix [4], node ordering algorithms [2], and representing dense patterns more compactly [21, 10].

Some common techniques used for discovering knowledge in large graphs include ranking and clustering. Ranking usually computes a score for each

vertex of the system based on a ranking function. Examples of ranking functions are PageRank [3] and HITS [15]. Such functions allow the comparison and sorting by score of the vertices in the graph. On the other hand, clustering is used for grouping the vertices by a distance or similarity function so that similar or close vertices are assigned to the same cluster, and dissimilar or distant vertices are grouped in different clusters. There are many clustering algorithms with different goals and different distance and similarity functions. Some good surveys related to clustering algorithms are available [31, 26].

Although there are many different types of real-world graphs of interest, in this work we aim at processing highly clustered and sparse graphs. Clustered graphs contain vertices grouped in highly connected subgraphs. Such graphs have high cluster coefficient and transitivity [25] measures. Sparse graphs usually expose low degeneracy, and are often found in real graphs [7].

In this paper we propose a compact data structure for clustered sparse undirected graphs that exploits the cliques to represent the edges implicitly. Further, it also exploits the vertex redundancy of the cliques, by partitioning them into components that share many vertices. This structure enables neighbor queries as well as queries for recovering all the maximal cliques.

Our structure is built on a partition of the clique-graph, where each node is a maximal clique in the original graph. We use an effective heuristic for finding the maximal cliques, as well as a clique-graph partition based on ranking and clustering, which avoids the construction of the clique-graph. We then define a compact representation of this partitioned clique-graph.

Our experimental evaluation shows that our compressed graph representation is competitive with the state-of-the-art compression efficiency for large real graphs, sometimes using less space of the smallest previous representation, and sometimes reducing the access time for neighbor queries. As discussed, in a context of limited memory or steep memory hierarchies, using less space is more valuable than being significantly faster in the same memory level. Our representation can also recover all the maximal cliques much faster than computing them from the original graph.

2. Proposed method

In this section we describe our method for compressing real sparse undirected graphs using a compact data structure that takes advantage of the vertex redundancy of the graph represented by its maximal cliques.

Our compression method includes three steps. The first step (*cliques listing*) consists in listing all maximal cliques of size at least two of the input graph. For the second step (*clique-graph partitioning*), we define a clique-graph based on the maximal cliques of the graph and a partitioning clustering heuristic for the clique-graph representation without building the clique-graph. In the last step (*compact graph representation*), we define a compact data structure based on symbol and bit sequences for the clique-graph partitions found during the second step.

2.1. Cliques listing

Let $G(V, E)$ be a graph where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. For any vertex $u \in V$, let $N(u) = \{v \in V | (u, v) \in E\}$ be its neighborhood. A clique is a complete subgraph of $G(V, E)$, that is, where every pair of distinct vertices is connected by an edge. A maximal clique is a clique that cannot be extended by including an additional vertex. All maximal cliques with minimum clique size of 2 constitute the *edge clique cover* of $G(V, E)$. We obtain all maximal cliques of a graph using the fast algorithm proposed by Eppstein and Strash [7].

2.2. Clique-graph partitioning

Let $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ be the collection of all maximal cliques of an undirected graph $G(V, E)$. We define a graph of cliques, or clique-graph, such that each *vertex* in this graph is a maximal clique and there is an *edge* between two vertices if the node intersection between two cliques is nonempty. The formal definition for a clique-graph follows.

Definition 1. Clique-graph

Given a graph $G = (V, E)$ and $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ the collection of maximal cliques that cover G, the clique-graph $CG_c = (V_c, E_c)$ of G is defined as

1. $V_c = \mathcal{C}$
2. $\forall c, c' \in \mathcal{C}, (c, c') \in E_c \iff c \cap c' \neq 0$

We define the problem of finding a clique-graph partition as follows:

Problem 1. Find a graph partition in the clique-graph CG_c.
Given a clique-graph $CG_c = (V_c, E_c)$, output a partition $\mathcal{CP} = \{cp_1, cp_2, \ldots, cp_M\}$ of V_c, aiming at maximizing shared nodes of V within the partitions and minimizing shared nodes among partitions.

We observe that in Problem 1 it is not possible to share maximal cliques among different partitions, but it is possible to have a subset of vertices of the original graph G belonging to different partitions in the clique-graph partition.

We aim at finding a clique-graph partition that takes advantage of the vertex redundancy in maximal cliques: The basic goal of applying a partitioning clustering method is to group maximal cliques that share many vertices, leaving in different partitions the maximal cliques that share none or only a few vertices.

The problem of finding a clique-graph partition has been addressed recently [17]. A naive approach to find a clique-graph partition is to first evaluate the similarity between all maximal cliques in CG_c using a method such as SimRank [32], and then apply a clustering algorithm such as spectral clustering [22] or hierarchical clustering [13]. However, evaluating similarity between maximal cliques is time-consuming for large graphs. In addition, using spectral clustering often yields separate clusters composed by low-degree vertices [17]. The new method [17] introduces a balancing factor to avoid this problem, however, as discussed by the authors, such solution is very time-consuming ($O(n^3)$), which makes it applicable on very small graphs only.

Instead, we propose an effective heuristic for finding a clique-graph partition, which is based on using a ranking function without the need of building the clique-graph.

2.2.1. Algorithm for finding a clique-graph partition

In this section we present our partitioning clustering algorithm for a clique-graph, which is effective for a compact representation of the input graph G.

Our approach first defines a ranking function for each vertex in G as presented in Definition 2. A ranking function assigns a score to each vertex based on some properties of the clique-graph. We consider ranking functions based on the number and sizes of the maximal cliques where a vertex in G is found.

The clustering heuristic is given in Algorithm 1. The output of the ranking computation are the arrays D and R (line 1). Array D contains a list with the clique ids where each vertex in G is found, and array R contains a score for each vertex in G. The time complexity of the ranking computation includes first passing through all vertices of G in the maximal clique collection $\mathcal{C}$, and then sorting R from higher to lower score. The total time complexity is $O(L \log L)$, where $L = \sum_{c_i \in \mathcal{C}} |c_i|$ (i.e., all vertices in all maximal cliques).

Next, we create a bit array Z of size $|\mathcal{C}|$ and set each bit to 0. We then go through the array R in decreasing score order and, for each vertex u, we get from $D[u]$ the clique ids where u is found and add each clique id to the current partition ($cpid$) iff $Z[id] = 0$. If id is added to the current partition we also mark it in $Z[id]$. If the current partition, $cpid$, has at least one clique id, the partition is added to the collection $\mathcal{CP}$. The time complexity of this step is $O(|\mathcal{C}| + |V|)$. The algorithm finally returns the clique-graph partition in the collection $\mathcal{CP}$, where each partition is represented as a set of clique ids.

Definition 2. Ranking function

Given a graph G and its clique collection $\mathcal{C}$, a ranking function is a function $r : V \to \mathbb{R}_{>0}$ that gives a rank score for each vertex $u \in V$.

We define ranking functions for each vertex based on the number and sizes of the maximal cliques where the vertex is found. We first define the set $C(u)$ for the vertex $u \in V$ as $C(u) = \{c \in \mathcal{C} | u \in c\}$, and then consider the following possible ranking functions.

Algorithm 1 Clique-graph partition algorithm.

Require: graph $G = (V, E)$, maximal clique collection $\mathcal{C}$, ranking function $r(u)$

Ensure: Returns clique-graph partition $\mathcal{CP}$

```
(D, R) ← computeRanking(r, C)
Initialize bit array Z of size |C| and set each bit to 0
for u ∈ V in decreasing order of score in R[u] do
    cpid ← ∅
    for id ∈ D[u] s.t. Z[id] = 0 do
        Z[id] ← 1
        cpid ← cpid ∪ {id}
    end for
    if cpid ≠ ∅ then
        CP ← CP : cpid
    end if
end for
return CP
```

$$r_f(u) = |C(u)| \tag{1}$$

$$r_c(u) = \sum_{c \in C(u)} |c| \tag{2}$$

$$r_r(u) = \frac{r_c(u)}{r_f(u)} \tag{3}$$

2.3. Compact graph representation

We now describe the compact data structure for representing G using the clique-graph partition $\mathcal{CP}$ obtained in the previous step. We consider compact data structures for symbol and bit sequences with support for *rank*, *select*, and *access* operations, where $rank_v(B, i)$ is the number of occurrences of bit/symbol v in $B[1, i]$, $select_v(B, j)$ is the j-th occurrence of bit/symbol v in B, and $access(B, k)$ is the bit/symbol at position k in B.

Our graph representation is composed of two symbol sequences X and Y, a bitmap B and a byte sequence BB. For each partition $cp_p \in \mathcal{CP}$, we create a sequence X_p that stores the vertices of G that are in any of the cliques of cp_p. We then concatenate the sequences X_p corresponding to all partitions $cp_p \in \mathcal{CP}$ to obtain sequence X, and use bitmap B to mark the starting position of each partition in X. In addition, we use a byte sequence BB to register, for each vertex in X, the maximal cliques where it participates within the corresponding partition. More concretely, for each partition cp_p we create a matrix of bits BB_p where each row represents a vertex u in the partition in the same order as in X_p, and we mark with 1 the columns corresponding to the cliques where u participates. We then convert each bit matrix BB_p into a byte-aligned sequence and concatenate all sequences BB_p to create the byte sequence BB. If there is only one maximal clique in a partition in cp_p, then BB stores no bytes for that partition. Finally, sequence Y stores the position in BB where each partition BB_p starts. We formalize our data structure next.

Definition 3. Compact representation of $G(V, E)$

Let $\mathcal{CP} = \{cp_1, \ldots, cp_M\}$, $cp_p \in \mathcal{CP}$, and $cp_p = \{c_1, ..., c_{m_p}\}$, where m_p is the number of maximal cliques in partition cp_p. Let $bpu_p = \lceil \frac{m_p}{8} \rceil$ be the number of bytes for BB_p, where BB_p is empty if $bpu_p = 0$ (M' partitions with $bpu_p > 0$). We then define X, B, BB, and Y as follows:

$$X_p = \{u \in c, c \in cp_p\} = \{u_1, ...u_{|X_p|}\} \tag{4}$$

$$B_p = 1 : 0^{|X_p|-1} \tag{5}$$

$$BB_p = bb_p[1..|X_p|][1..bpu_p] \text{ if } m_p > 1, \text{ empty otherwise} \tag{6}$$

$$bb_p[i][j] = \sum_{k=1}^{8} 2^{k-1} \cdot (u_i \in c_{8(j-1)+k})$$

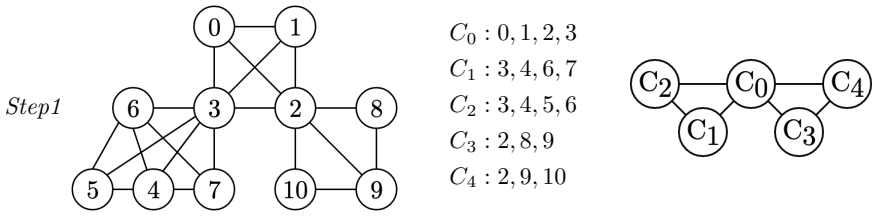
$$X = X_1 \cdots X_M, B = B_1 \cdots B_M, BB = BB_1 \cdots BB_{M'} \tag{7}$$

$$Y[p] = |X_{p-1}| \cdot bpu_{p-1} + Y[p-1], \qquad Y[1] = 0 \tag{8}$$

Figure 1 shows an example of the three steps of the creation of our data structure. The first step finds five maximal cliques in the 11-vertex input graph. Maximal cliques and the clique-graph are also displayed in the top-center of the figure. The second step uses Algorithm 1 to compute the clique-graph partition. The figure shows the content of the ranking array R using the function r_r and the resulting clique-graph partition stored in $\mathcal{CP}$. As seen in the example, we locate partitions with more than one maximal clique first in X to reduce the space of Y. The first partition in $\mathcal{CP}$ contains two maximal cliques C_1 and C_2, the second partition contains C_3 and C_4 and the final partition contains only maximal clique C_0. The third step builds the compact representation and the figure shows the content for X, B, BB and Y. Array X contains the vertices in all partitions in $\mathcal{CP}$. The first vertex in each partition in X is marked in bitmap B with a bit set to 1. For this example, BB contains bytes for the first two partitions. Since the partitions have two cliques, one byte per vertex is enough for representing the clique ids where each node participates; thus, BB contains five bytes in the first partition and four bytes in the second. Given that in the first partition, vertices 3, 4 and 6 participate in both cliques C_1 and C_2, their corresponding byte at BB encodes "11" as a byte with value 3; vertex 7 is included only in clique C_1, thus its byte encodes "01" as a byte with value 1; finally, vertex 5 only in clique C_2, thus its corresponding byte at BB encode "10" as the byte value 2. Similar encoding is performed in the second partition. Finally, sequence Y indicates the starting position of each partition in BB.

2.3.1. Query algorithms

Finally, we describe how the main queries are solved using our compact data structure. Algorithm 2 displays a sequential algorithm that retrieves the input graph G in a single pass. The time complexity of the sequential algorithm is $O(M' \times |X_p|^2 \times bpu_p + (M - M') \times |X_p|^2)$. Algorithm 3 computes neighbor queries for any vertex $u \in V$. We consider the neighbors of a vertex v as all the adjacent vertices to $v \in G$. As G is an undirected graph, the sum of all the lengths of all adjacency lists is $2|E|$, since if (u, v) is an undirected edge, then u



Step2

$u \in G$	0	1	3	4	5	6	7	2	8	9	10
R	4.0	4.0	4.0	4.0	4.0	4.0	4.0	3.3	3.0	3.0	3.0

$\mathcal{CP}$:	C_1	C_2	C_3	C_4	C_0

Step3

X:	**3**	4	5	6	7	**2**	8	9	10	**0**	1	2	3	
B:	1	0	0	0	0	1	0	0	0	1	0	0	0	1
BB:	**3**	3	2	3	1	**3**	1	3	2					
Y:	0	5												

Figure 1: Example of our compression method. The first step enumerates all maximal cliques. The second step computes the clique-graph partition. Finally, the last step (bottom part of the figure) obtains the compact representation using sequences X, B, BB, and Y.

appears in v's adjacency list and viceversa. The time complexity for retrieving the neighbors of a random vertex is $O(M' \times |X_p| \times bpu_p + (M - M') \times |X_p|)$. We also present Algorithm 4, which retrieves the maximal cliques of G in time $O(M' \times |X_p| \times bpu_p + (M - M') \times |X_p|)$.

The sequential algorithm goes through each partition of the compact representation, retrieving all edges in that partition and storing it in the output graph. If a partition in X contains only one clique, then the edges are all the vertex pairs in the clique. Otherwise, two vertices are neighbors if they participate in the same clique. To compute this, the algorithm checks if the *bitwise and* between the bitarrays of BB_p corresponding to those vertices is nonzero. To retrieve the neighbors for a given vertex u, the algorithm looks up the vertex u in all the partitions in X and then retrieves the corresponding neighbors in each partition. The algorithm for retrieving the maximal cliques also navigates through the partitions in sequence and gets all the cliques in each partition.

Algorithm 2 Retrieve the complete list of edges E.

Require: X, B, BB, Y

Ensure: Returns the edges E

```
Initialize empty edge list E
for p = 1 to rank_1(B, |B|) do
    s ← select_1(B, p)
    e ← select_1(B, p + 1)
    bpu_p ← (Y[p+1] − Y[p]) / (e − s)
    X_p ← X[s..e]
    for j = 0 to |X_p| − 1 do
        for k = j + 1 to |X_p| − 1 do
            if bpu_p = 0 then
                Insert (unoriented) edge (X_p[j], X_p[k]) into E
            else
                for b = 1 to bpu_p do
                    if BB_p[bpu_p · j + b] & BB_p[bpu_p · k + b] ≠ 0 then
                        Insert (unoriented) edge (X_p[j], X_p[k]) into E
                        break
                    end if
                end for
            end if
        end for
    end for
end for
return E
```

3. Experimental evaluation

We ran a number of experiments to study and tune our method, and to compare it with the state-of-the-art algorithms for compressing graphs, the including last version of WebGraph (WG, version 3.6.1) [2], the graph compression by BFS from Apostolico and Drovandi (AD) [1], and the k^2-tree [4]. We measure

Algorithm 3 Retrieve neighbors $N(u)$ of vertex $u \in V$.

Require: u, X, B, BB, Y

Ensure: Returns $N(u)$

```
Initialize empty list N(u)
occur ← rank_u(X, |X|)
for i = 1 to occur do
    u_p ← select_u(X, i)
    p ← rank_1(B, u_p)
    s ← select_1(B, p)
    e ← select_1(B, p + 1)
    bpu_p ← (Y[p+1] − Y[p]) / (e − s)
    X_p ← X[s..e]
    for j = 0 to |X_p| − 1 do
        if X_p[j] ≠ u then
            if bpu_p = 0 then
                Insert X_p[j] into N(u)
            else
                for b = 1 to bpu_p do
                    if BB_p[bpu_p · j + b] & BB_p[bpu_p · (u_p − s) + b] ≠ 0 then
                        Insert X_p[j] into N(u)
                        break
                    end if
                end for
            end if
        end if
    end for
end for
return N(u)
```

Algorithm 4 Retrieves the maximal cliques of $G(V, E)$.

Require: X, B, BB, Y

Ensure: Returns collection of maximal cliques CC

```
CC ← ∅
for p = 1 to rank_1(B, |B|) do
    s ← select_1(B, p)
    e ← select_1(B, p + 1)
    bpu_p ← (Y[p+1] − Y[p]) / (e − s)
    X_p ← X[s..e]
    if bpu_p = 0 then
        CC ← CC ∪ {X_p[e..s]}
    else
        for j = 0 to |X_p| − 1 do
            cluster ← 0
            for b = 1 to bpu_p do
                cluster ← cluster + 1
                for k = 1 to 8 do
                    if BB_p[bpu_p · j + b][k]=1 then
                        Insert vertex X_p[j] to C[cluster]
                    end if
                end for
            end for
        end for
        CC ← CC ∪ {C[1], C[2], ..., C[cluster]}
    end if
end for
return CC
```

compression efficiency in bits per edge (*bpe*).

For our approach, we use succinct data structures to represent the sequences of symbols and bits. Concretely, we use compact data structures based on the wavelet matrix [6] for X and Y, and compressed bitmaps [23] for B. Such representations are available in the sdsl-lite library[1] [9]. For the byte sequence BB we use plain Huffman compression [12].

We used the sparse social network graphs dblp2010 and dblp2011 from WebGraph[2], snapdblp and snapamazon from SNAP[3], markastro and markcondmat from QuickCliques[4], citation networks coPapersDBLP and coPapersCiteseer from DIMACS[5]. Table 1 shows the main statistics of the graphs. When comparing with techniques that encode directed graphs, we duplicated the edges of our graphs, which are all undirected. Such are the cases of AD and WG, which work on directed graphs and are able to retrieve out-neighbors of any vertex. On the other hand, k^2-tree can efficiently retrieve both in- and out-neighbors of the graphs. Thus, it is possible to compactly represent the undirected graphs without replicating the edges, requiring computing both the successor and predecessor queries of a vertex to be able to retrieve all its neighbors.

All the experiments ran on a machine with an Intel i7-7500U CPU @ 2.70GHz, 12 GB RAM, and 4 MB cache. We used g++ version 8.2.1 compiler with optimization O3.

[1] https://github.com/simongog/sdsl-lite

[2] Datasets dblp-2010 and dblp-2011, available at `http://law.di.unimi.it/datasets.php`.

[3] Datasets com-DBLP and amazon0601, available at `https://snap.stanford.edu/data/`. For amazon0601, we use the undirected version available at `http://www.dcs.gla.ac.uk/~pat/jchoco/clique/enumeration/data/snap/snap-amazon0601`.

[4] Datasets marknewman-astro and marknewman-condmat, available at `http://www.dcs.gla.ac.uk/~pat/jchoco/clique/enumeration/quick-cliques/`.

[5] Datasets coPapersDBLP and coPapersCiteseer, available at `https://www.cc.gatech.edu/dimacs10/archive/coauthor.shtml`

Table 1: Main properties of the datasets used in the experiments.

Dataset	$\|V\|$	$\|E\|$	$degree_{avg}$	$degree_{max}$
markastro	16,706	242,502	14.51	360
markcondmat	40,421	351,386	8.69	278
snapdblp	317,080	2,099,732	6.62	2,752
snapamazon	403,394	4,886,816	12.11	343
dblp2010	326,186	1,615,400	4.95	238
dblp2011	986,324	6,707,236	6.80	979
coPapersDBLP	540,486	30,491,458	56.41	3,299
coPapersCiteseer	434,101	32,073,440	73.88	1,188

3.1. Analysis of ranking functions

We analyze the ranking functions (r_r, r_c, and r_f) given in Definition 2 to choose the best options for our compressed representation. Figure 2 shows the cumulative distribution for the number of bytes required in BB for each vertex in all partitions using the three ranking functions. This parameter can be important for compression efficiency and the access time when retrieving adjacent vertices of a node of the original graph. As seen in the figure, in all cases the r_r function yields fewer bytes in BB than r_c and r_f.

Next, we present the impact of using the ranking functions in the compressed representation of the graph. Figure 3(top-left) shows the *Average* number of bytes per vertex in X required by BB. We observe that the r_r ranking function requires fewer bytes than r_c and r_f. Figure 3(top-right) shows the space in bits required by each component of the compressed representation of the graph using the three ranking functions. We see that the total space required does not differ substantially among ranking functions. This suggests that the compression techniques we use are effective and reduce the impact of the ranking functions in the final compression. The space required by X and Y is bigger for r_r compared to those for for r_c and r_f. In contrast, BB is smaller, whereas the space of B is very small in all cases. Figure 3(bottom-left) shows the same space, now

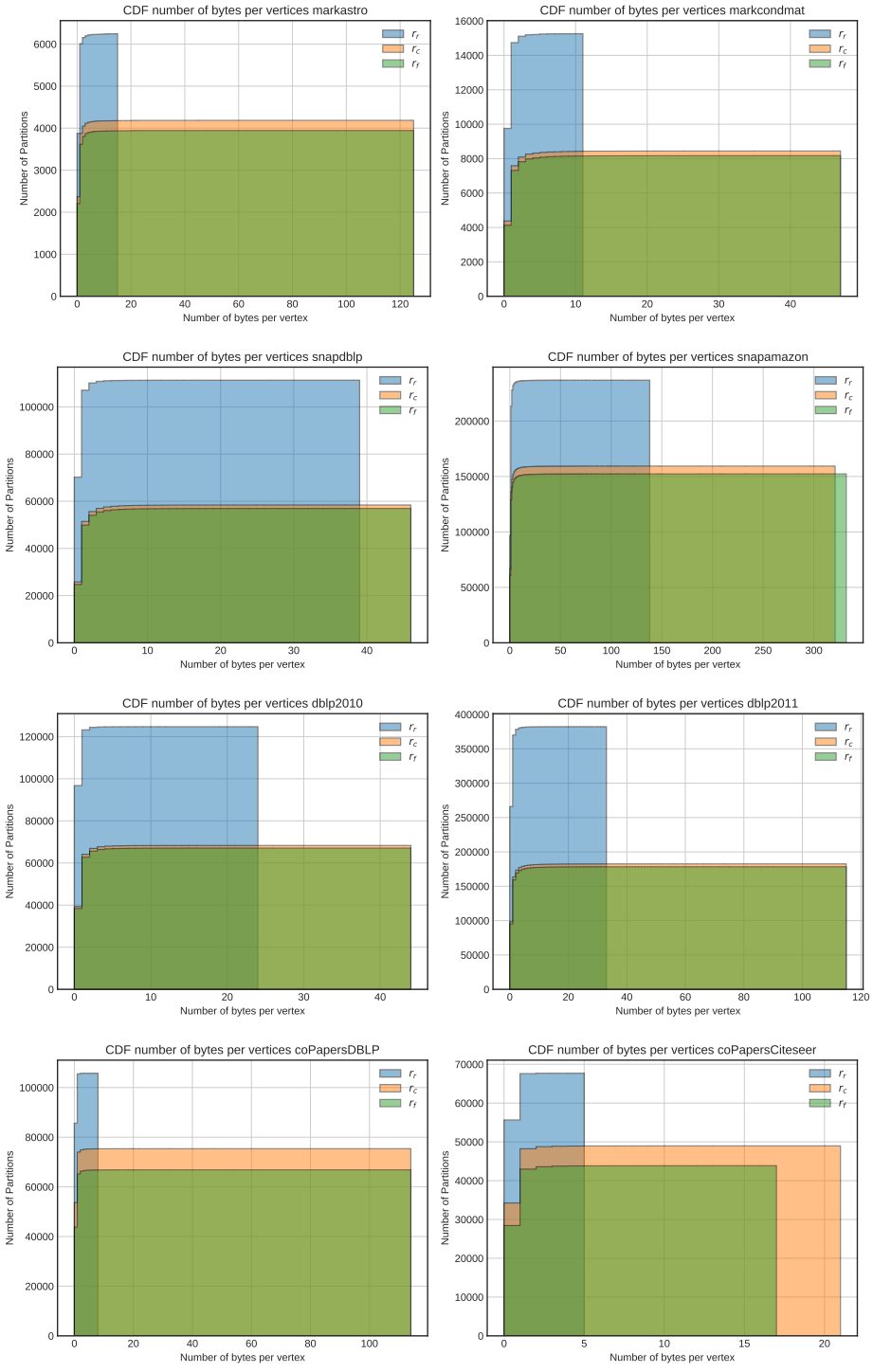


Figure 2: Bytes per vertex for all partitions given by its Cumulative Distribution Function.

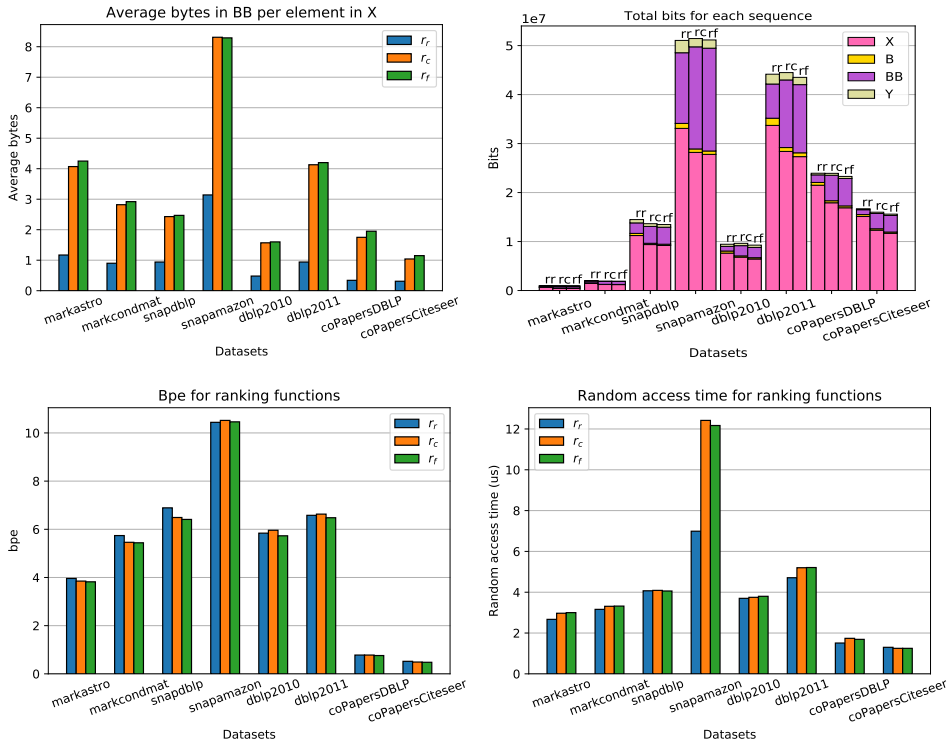


Figure 3: Average bytes per vertex in partitions (top-left), total bits per sequence (top-right), bpe (bottom-left) and random access time in microseconds (bottom-right) when applying different rank functions over several datasets.

measured in bits per edge (*bpe*). In general, the best space is achieved by the r_f function, followed by r_c and finally r_r, although the differences are small.

Figure 3(bottom-right) measures the random access time, that is, retrieving all the adjacent neighbors of an arbitrary vertex $u \in G$. The compressed representations using function r_r are similar to their counterparts using r_f and r_c, with the exception of the snapamazon graph, where r_r is faster. Considering the trade-off between space and access time and that r_c and r_f are similar, we present our compression and access time results using ranking functions r_r and r_f.

Table 2: Execution time for building the compact data structure for different datasets. We include the number of vertices ($|V|$), number of maximal cliques (N), time for listing maximal cliques (t_{lg}), time for computing clique-graph partition (t_{cg}), and total time building compact structure using clique-graph partitioning ($t_b = t_{lg} + t_{cg}$). The last column (t_{lcg}) displays the running time for listing all maximal cliques from the compact data structure. We report execution times in seconds (s).

Dataset	$\|V\|$	$N = \|\mathcal{C}\|$	t_{lg} (s)	t_{cg} (s)	t_b (s)	t_{lcg} (s)
markastro	16,706	15,724	0.18	0.28	0.46	0.05
markcondmat	40,421	34,274	0.28	0.40	0.68	0.11
snapdblp	317,080	257,551	1.68	2.30	3.98	0.86
snapamazon	403,394	1,023,572	5.93	8.44	14.37	3.01
dblp2010	326,186	196,434	1.12	1.46	2.58	0.57
dblp2011	986,324	806,320	5.58	7.30	12.88	2.77
coPapersDBLP	540,486	139,340	17.96	3.44	21.40	1.39
coPapersCiteseer	434,102	86,303	26.70	4.70	31.40	0.96

3.2. Compression performance

We first measure the time required for building the compact data structure for all datasets. Table 2 shows the results. We first include the execution time for listing all maximal cliques (t_{lg}) and the time for executing the clique-graph partitioning heuristic (t_{cg}). The sum of these two times, which is the total construction time, is given by t_b. The last column gives the time required by our compact data structure for listing the maximal cliques (column t_{lcg}), using Algorithm 4. For these experiments we show only the results with r_r since the difference is not significant with r_f. As observed, this time is much lower than that of listing the maximal cliques from the original graph, especially in the two largest graphs.

Table 3 shows the number of maximal cliques, degeneracy, cluster coefficient and transitivity [25] of the graphs. Cluster coefficient is a measure that indicates the degree of how nodes tend to group in a graph. Graphs with highly connected nodes have high clustering coefficient. Transitivity measures the fraction of pairs

Table 3: Number of maximal cliques, degeneracy (degen.), cluster coefficient (CCoeff.), Transitivity (Tr.), and bits per edge (bpe_r) and (bpe_f) of our representation using the ranking function r_r and r_f.

Dataset	$N = \|\mathcal{C}\|$	degen.	CCoeff.	Tr	bpe_r	bpe_f
markastro	15,724	56	0.66	0.42	3.96	3.82
markcondmat	34,274	29	0.64	0.24	5.74	5.44
snapdblp	257,551	113	0.63	0.30	6.89	6.41
snapamazon	1,023,572	10	0.41	0.16	10.44	10.46
dblp2010	196,434	74	0.61	0.39	5.85	5.73
dblp2011	806,320	118	0.63	0.20	6.58	6.48
coPapersDBLP	139,340	336	0.80	0.65	0.78	0.76
coPapersCiteseer	86,303	844	0.82	0.77	0.52	0.48

of vertices that are connected through a third vertex they have in common, then it also indicates how well connected are nodes in a network. In addition, the last two columns show the compression ratio obtained by our proposal (in bpe). In fact, we can see that the best compression (fewer bits per edge) is achieved in graphs coPapersDBLP and coPapersCiteseer, which have the highest cluster coefficient and largest transitivity; the worst compression is achieved by snapamazon, having the lowest cluster coefficient and transitivity. According to the results, we suggest that our approach works well for graphs that have a number of maximal cliques that is proportional to the number of vertices of the graph, and are sparse and clustered as measuring degeneracy, cluster coefficient and transitivity metrics.

3.3. Comparison with the state of the art

We now compare our proposal with several state-of-the-art methods. More concretely, we consider WebGraph representation (WG) [2], the graph compression by BFS from Apostolico and Drovandi (AD) [1], and the k^2-tree [4]. For WebGraph, we consider where WG with support for random access. For the k^2-tree, we consider the input undirected graph with two variants: k2treeU,

Table 4: Compression space (in bpe) of our structure and other state-of-the-art methods.

Dataset	CCDSHf	CCDSHr	WG	AD	k2treeU	k2treeU-BFS
markastro	**3.82**	3.96	8.10	5.67	4.89	4.34
markcondmat	**5.44**	5.74	11.78	7.86	6.28	5.60
snapdblp	6.41	6.89	11.80	8.14	5.88	**5.23**
snapamazon	10.46	10.44	14.50	10.96	8.02	**6.38**
dblp2010	5.73	5.84	8.67	6.71	**4.23**	4.30
dblp2011	6.48	6.58	10.13	9.67	**5.48**	5.89
coPapersDBLP	**0.76**	0.78	2.71	1.81	1.67	0.94
coPapersCiteseer	0.48	0.52	1.79	0.85	1.21	**0.45**

which uses the original node ordering, and ktreeU-BFS, which uses a BFS node reordering. In the comparison, we use our variants achieving a good space/time trade-off, i.e., CCDSHr using r_r, and CCDSHf using r_f. Table 4 displays the compression efficiency of CCDSHr and CCDSHf compared to the state-of-the-art techniques. Our approach obtains the best space for three datasets, and second best, after k^2-tree, in the rest of the datasets.

In addition, we compare our representation with the state-of-the-art methods when retrieving the neighbors of random nodes. Figure 4 shows the space requirements and average access time for retrieving the neighbors for 10^6 random vertices in G. We observe that our representation provides competitive compression in most datasets, where compared to AD and WG uses always less space, yet in exchange our access time is higher. WG is the fastest but it also requires the most space. Compared to our closest competitor, with the exception of the snapamazon dataset, our structure is competitive with both k^2-tree variants in space and random access time. Our representation uses less space in three of the eight datasets, but it is slower. Also in other three datasets our structure uses more space, but it is faster.

Finally, Table 5 shows the time in seconds required to decompress the complete graph. In this case, our approach is outperformed by the state-of-the-art

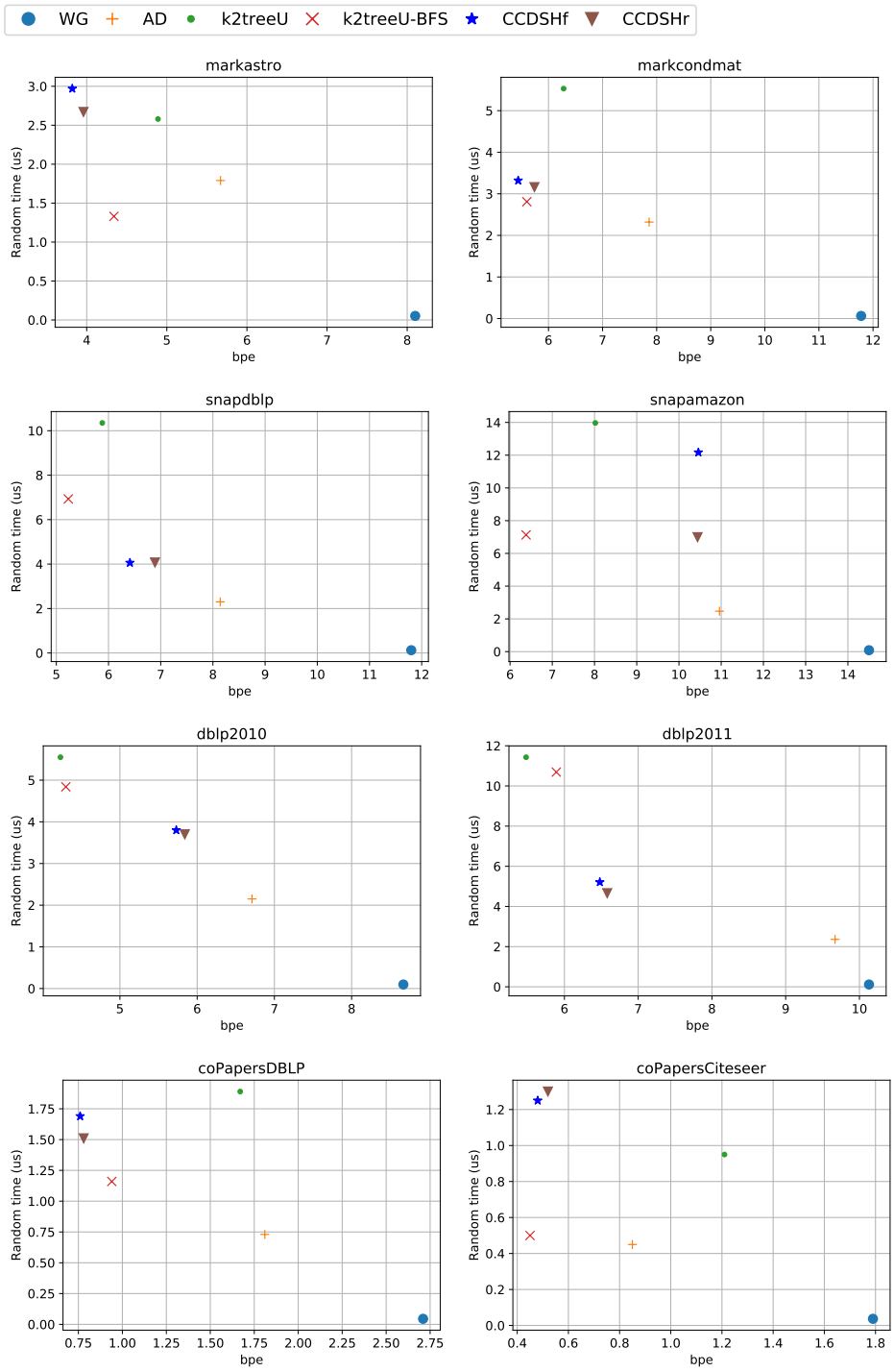


Figure 4: Random access time (in microseconds) to retrieve all the neighbors of a node.

Table 5: Decompression time in seconds of our structure, compared to state-of-the-art methods. WGs is WebGraph using sequential access, and we do not include AD, as it does not have a sequential access option.

Dataset	CCDSHf	CCDSHr	WGs	k2treeU	k2treeU-BFS
markastro	0.09	0.09	0.28	0.03	**0.02**
markcondmat	0.16	0.16	0.52	0.07	**0.04**
snapdblp	1.16	1.26	1.20	0.58	**0.35**
snapamazon	7.09	4.53	1.30	1.36	**1.13**
dblp2010	0.79	0.82	1.09	0.18	**0.16**
dblp2011	4.61	4.45	2.41	**1.1**	1.31
coPapersDBLP	5.68	5.81	1.59	1.45	**1.01**
coPapersCiteseer	4.62	5.46	1.56	1.33	**0.65**

methods, especially for coPapersDBLP and coPapersCiteseer, which are those where we obtain the highest compression ratio. This is an expected result, as we have prioritized compression ratio and random access time results, given that full decompression of the graph is expected to be infrequent.

4. Conclusions

We have introduced a new compact representation of real sparse and clustered undirected graphs based on clique-graph partitioning. We first list all maximal cliques of the input graph, then define a clique-graph (whose nodes are the cliques in the original graph), then obtain a partitioning of this clique-graph, and finally represent these clique-graph partitions using compact data structures.

Our technique depends on the quality of the clique-graph partitioning. We propose an algorithm that relies on a ranking function for the vertices of the graph, and works well for our approach. We define different ranking functions and analyze their behaviour, selecting the ones that obtain the best trade-offs between space and neighbor random access time. In addition, our compressed

structure supports the recovery of all maximal cliques between 1.9 and 27 times faster than using the original graph.

We compare our representation with the best-known techniques in the state of the art. Our structure is competitive in space and random access time, particularly for very clustered graphs. This trade-off can be very convenient when the significantly smaller space usage of our structure allows it reside in main memory while alternative representations must be deployed on disk.

As future lines of research, we will study alternative compact data structures and partitioning heuristics for improving the space/time tradeoffs. In addition, we will design more complex algorithms on top of this compact graph representation to perform different network analyses.

Acknowledgments

This research has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie [grant agreement No 690941]; from the Ministerio de Economía y Competitividad (PGE and ERDF) [grant numbers TIN2016-77158-C4-3-R]; from Xunta de Galicia (co-founded with ERDF) [grant numbers ED431C 2017/58; ED431G/01]; and from the Millennium Institute for Foundational Research on Data (IMFD), Chile.

References

[1] Apostolico, A. and Drovandi, G. (2009). Graph compression by BFS. *Algorithms*, 2(3):1031–1044.

[2] Boldi, P., Rosa, M., Santini, M., and Vigna, S. (2011). Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th International Conference on World Wide Web (WWW)*, pages 587–596. ACM.

[3] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117.

[4] Brisaboa, N. R., Ladra, S., and Navarro, G. (2014). Compact representation of web graphs with extended functionality. *Information Systems*, 39:152–174.

[5] Buehrer, G. and Chellapilla, K. (2008). A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 International Conference on Web Search and Data Mining (WSDM)*, pages 95–106. ACM.

[6] Claude, F., Navarro, G., and Ordóñez, A. (2015). The wavelet matrix: An efficient wavelet tree for large alphabets. *Information Systems*, 47:15–32.

[7] Eppstein, D., Löffler, M., and Strash, D. (2013). Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18:3–1.

[8] Fisher, D. (2016). Big data exploration requires collaboration between visualization and data infrastructures. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, page 16. ACM.

[9] Gog, S., Beller, T., Moffat, A., and Petri, M. (2014). From theory to practice: Plug and play with succinct data structures. In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA)*, pages 326–337.

[10] Hernández, C. and Navarro, G. (2014). Compressed representations for web and social graphs. *Knowledge and Information Systems*, 40(2):279–313.

[11] Huang, Z., Zheng, Y., Cheng, R., Sun, Y., Mamoulis, N., and Li, X. (2016). Meta structure: Computing relevance in large heterogeneous information networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1595–1604. ACM.

[12] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

[13] Irani, J., Pise, N., and Phatak, M. (2016). Clustering techniques and the similarity measures used in clustering: A survey. *International Journal of Computer Applications*, 134(7):9–14.

[14] Jiang, C., Coenen, F., and Zito, M. (2013). A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(1):75–105.

[15] Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

[16] Liu, Z. and Ma, Y. (2019). A divide and agglomerate algorithm for community detection in social networks. *Information Sciences*, 482:321–333.

[17] Lu, Z., Wahlström, J., and Nehorai, A. (2018). Community detection in complex networks via clique conductance. *Scientific Reports*, 8(1):5982.

[18] Makino, K. and Uno, T. (2004). New algorithms for enumerating all maximal cliques. In *Proceedings of the Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 260–272. Springer.

[19] Mitzenmacher, M., Pachocki, J., Peng, R., Tsourakakis, C., and Xu, S. C. (2015). Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 815–824. ACM.

[20] Moradi, P., Ahmadian, S., and Akhlaghian, F. (2015). An effective trust-based recommendation method using a novel graph clustering algorithm. *Physica A: Statistical Mechanics and its Applications*, 436:462–481.

[21] Nelson, M., Radhakrishnan, S., Chatterjee, A., and Sekharan, C. N. (2015). On compressing massive streaming graphs with quadtrees. In *Proceedings of the IEEE International Conference on Big Data (Big Data)*, pages 2409–2417. IEEE.

[22] Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856.

[23] Raman, R., Raman, V., and Rao, S. S. (2002). Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 233–242.

[24] Rossi, R. A. and Ahmed, N. K. (2015). The network data repository with interactive graph analytics and visualization. In *Proceedings of the 29th Conference on Artificial Intelligence (AAAI)*.

[25] Saramäki, J., Kivelä, M., Onnela, J.-P., Kaski, K., and Kertesz, J. (2007). Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75(2):027105.

[26] Saxena, A., Prasad, M., Gupta, A., Bharill, N., Patel, O. P., Tiwari, A., Er, M. J., Ding, W., and Lin, C.-T. (2017). A review of clustering techniques and developments. *Neurocomputing*, 267:664–681.

[27] Shao, B., Wang, H., and Xiao, Y. (2012). Managing and mining large graphs: systems and implementations. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 589–592.

[28] Stanley, N., Kwitt, R., Niethammer, M., and Mucha, P. J. (2018). Compressing networks with super nodes. *Scientific Reports*, 8(1):10892.

[29] Tsourakakis, C. (2015). The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*, pages 1122–1132.

[30] Wang, Ø., Bodd, N., Xing, C., Kvalheim, B., and Helvik, T. (2017). Enterprise graph search based on object and actor relationships. US Patent 9,542,440.

[31] Xu, D. and Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193.

[32] Yu, W., Lin, X., Zhang, W., Pei, J., and McCann, J. A. (2019). Simrank*: Effective and scalable pairwise similarity search based on graph topology. *The VLDB Journal*, pages 1–26.

LaTeX Source Files

Click here to download LaTeX Source Files: papersources.tar.gz

*Declaration of Interest Statement

Declaration of interests

☐ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Declarations of interest: none