Compact structure for social graphs based on clique-graph partition

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

[a] *Computer Science Department, University of Concepcion, Concepción, Chile*
[b] *Computer Science Department, University of Coruña, A. Coruña, Spain*
[c] *Computer Science Department, University of Chile, Santiago, Chile*

Abstract

This template helps you to create a properly formatted LaTeX manuscript.

Keywords: `elsarticle.cls`, LaTeX, Elsevier, template

2010 MSC: 00-01, 99-00

1. Introduction

There is a wide variety of real systems that are modeled by graphs, such as communication, Web, social, and biological networks. The process of discovering relevant information from graphs is usually referred as graph mining, which is usually a difficult task, especially with the current trend of data size growth. The main challenges are triggered by different aspects, such as the data volume itself, data complexity (i.e. many relationships among the data), and application needs [1]. 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 recomendation systems [], finding redundancy for graph compression [], measuring the relative

[⋆]This document is the results of the research project funded by the National Science Foundation.

[⋆⋆]The second title footnote which is a longer text matter to fill through the whole text width and overflow into another line in the footnotes area of the first page.

[*]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) relevance of network actors [], and network visualization []. Recent works on graph mining believe that dense patterns are prominent and describe different dense substructures such as maximal cliques [2, 3], community detection [4], and dense bipartite patterns [? ? ?]. Such substructures have been used for improving network analysis, graph compression and visualization.

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 [?], sparseness and clustering of the adjacency matrix [?], node ordering algorithms [?], and representing dense patterns more compactly [? ? ?].

Some common techiques 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[] and HITS[]. 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 disimilar 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 [? ?].

Although, there are many different types of real graphs, in this work we aim at processing highly clustered and sparse graphs. Highly clustered graphs contain vertices grouped in highly connected subgraphs. Sparse graphs usually expose low arboricity and degeneracy, and are often found in real graphs [3].

In this paper, we suggest that maximal cliques, ranking, clustering and compression techniques can be used for fast basic and mining query resolution for large graphs. We propose a compact data structure for large sparse graphs that exploits the vertex redundancy of the maximal cliques of the graph and represent its edges implicitly. This structure enables neighbor queries as well as queries for recovering maximal cliques. In order to define the compact structure we define a clique-graph representation of the input graph and propose an effective heuristic for finding a clique-graph partition that is based on combining ranking with clustering. Our experimental evaluation shows that our approach greatly improves the state-of-the-art compression efficiency for large real graphs and it provides competitive access times for neighbor and recover all maximal cliques faster than all listing maximal cliques over the original graph. In fact listing maximal cliques on the compressed graph is much faster than listing them from the original graph.

2. Related work

3. Proposed method

In this section, we describe our method for compressing real sparse graphs using compact data structures taking advantage of the vertex redundancy of the graph represented by its maximal cliques.

Our compression method includes three steps. The first step consists of listing all maximal cliques of size at least two of the input graph. We obtain all maximal cliques of a graph using the fast algorithm proposed by Eppstein and Strash [3]. 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 of the 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.

3.1. Clique-graph partitioning

Let be $G(V,E)$ 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 $N(u)$ be its neighborhood $\{v \in V | (u,v) \in E\}$. A clique is a complete subgraph in $G(V,E)$, and a maximal clique is a clique that can not be extended by including an additional vertex. All maximal cliques with minimum clique size of 2 is the *edge clique cover* of $G(V,E)$.

Let the collection $\mathcal{C}$ of all maximal cliques c_1,c_2,c_3,...c_N be the maximal cliques of an undirected graph $G(V,E)$. We observe that we can define a graph of cliques, where 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 not empty. The formal definition for a clique-graph is given next.

Definition 1. Clique-graph
Given a graph $G = (V,E)$ and $\mathcal{C} = \{c_1, c_2, ..., c_N\}$ the collection of size N of maximal cliques that cover G, let $CG_c = (V_c, E_c)$ be a clique-graph where

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

We define the problem aiming at 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 clique-graph partition $\mathcal{CP} = \{cp_1, cp_2, ..., cp_M\}$ of $CG_c(V_c, E_c)$, with $M \geq 1$, i.e. such that
(1) $\bigcup_{i \in \mathcal{CP}} cp_i = CG_c$,
(2) $cp_i \cap cp_j = 0$ for $i \neq j$, and
(3) any $cp_i \in \mathcal{CP}$ is a subgraph in $CG_c(V_c, E_c)$ induced by the subset of vertices in cp_i.

We observe that condition (2) in the Problem 1 establishes that it is not possible to have two o more maximal cliques in different partitions, but it is possible to have a subset of vertices in graph $G(V,E)$ 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 would be to group maximal cliques that share many vertices in the same partition, and maximal cliques that share none or only a few vertices are in different partitions.

The problem of finding a clique-graph partition has been addressed recently [5]. 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 [**?**], and then apply a clustering algorithm such as spectral clustering [] or hierarchical clustering[]. However, to evaluate similarity between maximal cliques is time consuming since it requires to compute pairwise similarity between maximal cliques. In addition using spectral clustering often yield separate clusters composed by low degree vertices [5]. The method in [**?**] 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 only on very small graphs.

Therefore, 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.

3.1.1. Algorithm for estimating a clique-graph partition

In this section, we present our partitioning clustering algorithm for a clique-graph (Algorithm 1) that is effective for a compact representation of the input graph $G(V, E)$.

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

The clustering heuristic is given in Algorithm 1. The output of the ranking computation are the arrays D and R (Algorithm 1-line 1). The D array contains a list with the clique ids where each vertex in G is found, and the R contains a score for each vertex in $G(V, E)$. 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 $N = |\mathcal{C}|$ and set each bit to 0. Then, we go through the R array and for each vertex u, we get the clique ids where u is found from $D[u]$ and we add the clique id to the current partition ($cpid$) only if $Z[id] = 0$. 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 concatenated to the collection $\mathcal{CP}$ and continue processing vertices in R. The time complexity of this step is $O(N + V)$. The algorithm finally returns the clique-graph partition in the collection $\mathcal{CP}$, where each partition contains a set of clique ids included in the corresponding partition.

We consider the following ranking functions that takes into account the number and sizes of maximal cliques where each vertex in $G(V, E)$ participates in.

Algorithm 1 Clique-graph partition algorithm.

```
Require: $\mathcal{C}$ maximal clique collection ($N = |\mathcal{C}|$), ranking function $r(u)$
Ensure: Returns clique-graph partition collection $\mathcal{CP}$
1: $(D, R) \leftarrow computeRanking(r, \mathcal{C})$ (array D and R, $\forall u \in V$ )
2: Initialize bit array $Z$ of size $N$ and set each bit to 0
3: for ($u \in R$) do
4:     $cpid \leftarrow \emptyset$
5:     for ($id \in D[u]$ and $D[u] = 0$) do
6:         $Z[id] \leftarrow 1$
7:         $cpid \leftarrow cpid \cup \{id\}$
8:     end for
9:     if ($cpid \neq \emptyset$) then
10:        $\mathcal{CP} \leftarrow \mathcal{CP} : cpid$
11:    end if
12: end for
13: return $\mathcal{CP}$
```

Definition 2. Ranking function

Given a graph G and its clique-graph CG_c, a ranking function is a function $r : V \rightarrow \mathbb{R}_{>0}$ gives the 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 we considered the following ranking functions.

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

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

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

3.2. Compact graph representation

In this section we describe the compact data structure for representing $G(V, E)$ using the clique-graph partition $\mathcal{CP}$ obtained by the clique-graph partitioning in the second step and described in Section 3.1.1. We considered compact data structures based on symbol and bit sequences with support for *rank()*, *select()*, and *access()* operations.

This compact structure uses two symbol sequences (X and Y), a bitmap B and a byte sequence BB. Sequence X stores the vertices in $G(V, E)$ that are in the induced clique-graph obtained by all maximal cliques in each partition in $\mathcal{CP}$. Bitmap B marks the separation between each partition in X, the byte sequence BB register for each vertex in X the maximal cliques where it participates. As observed in eq.(6), $BB_p \in BB$ is a matrix of bytes, where each row represents a vertex u in the partition $X_p \in X$, and the columns corresponds to the bytes used for the vertices in X_p to mark the cliques where they participates in the partition. Finally the Y sequence stores the position in the BB sequence where each partition BB_p starts. If there is only one maximal clique in a partition in $\mathcal{CP}$, BB_p stores no bytes for the partition.

Next, we define our compact data structure based on X, Y, B and BB.

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

Given $\mathcal{CP} = \{cp_0, ..., cp_{M-1}\}$, $cp_p \in \mathcal{CP}$, and $cp_p = \{c_0, ..., c_{m_r-1}\}$. We considered $bpu_p = \lceil \frac{m_r-1}{8} \rceil$ as the number of bytes per each vertex u in X_p, and then we define X_p, B_p, BB_p, Y_p as follows:

$$X_p = \{u \in c | c \in cp_p\} = \{u_0, ...u_{|X_p|-1}\} \tag{4}$$

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

$$BB_p = bb[|X_p|][bpu_p] \tag{6}$$

$$bb[i][j] = \begin{cases} \sum_{k=0}^{7} 2^k (u_i \in c_{8j+k}), & bpu_p \neq 0 \\ \emptyset, & otherwise \end{cases}$$

$$Y_p = |X_{p-1}| \times bpu_{p-1} + Y_{p-1}, \qquad Y_0 = 0 \tag{7}$$

Figure **??** shows an example of the three steps. In the first step we list all maximal cliques of size at least 2 using quickcliques[3]. Maximal cliques are displayed for the graph in the example, as sets, as seen in Figure **??**-*step1*. In the second step, Figure **??** shows the clique-graph and the ranking array R using the ranking functions as explained in Algorithm 1. Figure **??** shows the ranking scores for the function r_f, r_c and r_r. We also present the values for the sequences X, B, BB and Y using the r_r ranking function.

3.2.1. Query algorithms

In this section, we describe the query algorithms that can process the compact data structure. The Algorithm 2 displays the sequential algorithm, which allows us to retrieve the original graph $G(V, E)$ in a single pass. Algorithm 3 presents the algorithm for computing neighbor queries for any vertex $u \in G(V, E)$. We also present the Algorithm 4 which retrieves the maximal cliques of $G(V, E)$.

The sequential algorithm goes through each partition of the compact representation and retrieves all edges in that partition and stores it in the output graph. If a partition in X only contains a clique then the edges are all vertex pairs in the clique. However, if a partition contains more than one clique, any pair of vertices are neighbors if they participate in the same clique. To compute this, the algorithm checks if the *logical and* between any pair of bytes in BB_p of the corresponging vertices is true. To retrieve the neighbors for a given vertex u, the random algorithm requires to look up the vertex u in all the partitions in X and then retrieve the corresponing neighbors in each partition. The algorithm for retrieving the maximal cliques also goes through the partitons in sequence and gets all the cliques present in each partition.

4. Experimental evaluation

We compare our algorithms with the state-of-the-art approaches for compressing graphs including last vertion WebGraph (WG) [?], AD [?], and k2tree [?]. We executed all experiments on a machine with an Intel i7 2.70GHz CPU and 12GB RAM, and implemented the algorithms in g++ 8.2.1 compiler with optimization O3.

We considered clustered and sparse graphs dblp2010, and dblp2011 from WebGraph `http://law.di.unimi.it/datasets.php`, snapdblp, and snapamazon from SNAP [?], markastro and markcondmat from QuickCliques `http://www.dcs.gla.ac.uk/~pat/jchoco/clique/enumeration/quick-cliques/doc/`, and ca-coauthors from Network repository [6] (`http://networkrepository.com`). Table 4 shows that main statistics of the graphs. This table also displays the graph degeneracy and the average and maximum degree.

We also considered succint representation for sequences of symbols based on the wavelet matrix [?], Raman, Raman and Rao compressed bitmaps [?], and Hu-Tacker byte sequences [?]. Such representations are available in the sdsl-lite library [?].

We found that our approach works well for graphs that have a number of

Algorithm 2 Sequential algorithm, it retrieve $G(V, E)$.

```
Require: $X$, $B$, $BB$, $Y$
Ensure: Returns $G(V, E)$
  Initialize empty graph $G$
  for ($i = 1$ to $rank_B(1, |B|)$) do
    $bpu_p \leftarrow \frac{Y_p[i] - Y_p[i-1]}{select_B(1, i+1) - select_B(1, i)}$
    for ($j = 0$ to $|X_p|$ and $bpu_p = 0$) do
      for ($k = j + 1$ to $|X_p|$ and $u \neq X_p[k]$) do
        Add edges $(X_p[j], X_p[k])$ and $(X_p[k], X_p[j])$ to $G$
      end for
    end for
    for ($j = 0$ to $|X_p|$ and $bpu_p \neq 0$) do
      for ($k = j + 1$ to $|X_p|$) do
        for ($b = 0$ to $bpu_p$) do
          if ($BB_p[bpu_p * j + b]$ and $BB_p[bpu_p * k + b]$) then
            Add edges $(X_p[j], X_p[k])$ and $(X_p[j], X_p[k])$ to $G$
            $break$
          end if
        end for
      end for
    end for
  end for
  return $G$
```

Algorithm 3 Algorithm that retrieves the neighbors any $u \in G$ $N(u)$.

```
Require: $u$, $X$, $B$, $BB$, $Y$
Ensure: Returns $N(u)$
  Initialize empty graph $N(u)$
  for ($i = 1$ to $rank_X(u, |X|)$) do
    $u_p s \leftarrow select_X(u, i)$
    $p \leftarrow rank_B(1, u_p + 1) - 1$
    $bpu_p \leftarrow \frac{Y_p[p+1] - Y_p[p]}{select_B(1, p+2) - select_B(1, p+1)}$
    for ($j = 0$ to $|X_p|$ and $bpu_p = 0$) do
      Add $X_p[j] \neq u$ to $N(u)$
    end for
    for ($j = 0$ to $|X_p|$ and $bpu_p \neq 0$) do
      for ($b = 0$ to $bpu_p$) do
        if ($BB_p[bpu_p * j + b]$ and $BB_p[bpu_p * u_p + b]$) then
          Add $X_p[j] \neq u$ to $N(u)$
          $break$
        end if
      end for
    end for
  end for
  return $N(u)$
```

Algorithm 4 Algorithm that retrieves $G(V, E)$ maximal cliques.

```
Require: $X$, $B$, $BB$, $Y$
Ensure: Returns collection of maximal cliques $CC$
  $CC \leftarrow \emptyset$
  for ($i = 1$ to $rank_B(1, |B|)$) do
    $s \leftarrow select_B(1, i)$
    $e \leftarrow select_B(1, i + 1)$
    $bpu_p \leftarrow \frac{Y_p[i] - Y_p[i-1]}{e-s}$
    if $bpu_p \neq 0$ then
      $CC \leftarrow CC : X_p[e..s]$
      continue
    end if
    for ($j = 0$ to $|X_p|$ and $bpu_p \neq 0$) do
      $currentbit \leftarrow 0$
      for ($b = 0$ to $bpu_p$) do
        for ($k = 0$ to $7$) do
          if ($BB_p[bpu_p * j + b][k]$) then
            Add vertex $X_p[j]$ to $C[currentbit]$
          end if
        end for
      end for
    end for
    $CC \leftarrow CC : C$
  end for
  return $CC$
```

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$degree_{avg}$	$degree_{max}$	degeneracy
markastro	16,706	242,502	14.51	360	56
markcondmat	40,421	351,386	8.69	278	29
snapdblp	317,080	2,099,732	6.62	2,752	113
snapamazon	403,394	4,886,816	12.11	343	10
dblp2010	326,186	1,615,400	4.95	238	74
dblp2011	986,324	6,707,236	6.80	979	118
ca-coauthors	540,486	30,491,458	56.41	3,299	336

Table 1: Graph main statistics.

maximal cliques that is proportional al number of vertices of the graph.

We compare our compressed structure (CliqueCDS) with the state-of-the-art compression techniques such as WebGraph (WG) [], k2tree [], and Apostolico and Drovandi (AD) [].

First, we present in Table 4 the execution time for listing all maximal cliques (t_{lg}) and the time for exucuting the clique-graph partitioning heuristic (t_{cg}). Then, building the compressed representation takes the time given by t_b. In addition, we present the execution time that takes listing the maximal cliques using our best compressed representation of the graph. As observed this time is faster that listing the maximal cliques from the original graph.

4.1. Analysis for ranking functions

In this section, we analyze the ranking functions (r_r, r_c and r_f) given in Definition 2 to choose the best option for our compressed representation. Table 4.1 shows the cummulative distribution for the number of bytes required in BB for each vertex in all partitions using the three ranking functions. This parameter is important for the access time for retrieving edges of the original graph.

Next, we present the impact of using the ranking functions in the compressed representation of the graph. We present in Table 4.1(top-left) the maximum

Dataset	$\lvert V \rvert$	$N = \lvert \mathcal{C} \rvert$	t_{lg} (s)	t_{cg} (s)	t_b (s)	t_{lcg} (s)
markastro	16,706	15,724	0.18	0.28	0.46	0.10
markcondmat	40,421	34,274	0.28	0.40	0.68	0.18
snapdblp	317,080	257,551	1.68	2.30	3.98	1.12
snapamazon	403,394	1,023,572	5.93	8.44	14.37	4.12
dblp2010	326,186	196,434	1.12	1.46	2.58	0.76
dblp2011	986,324	806,320	5.58	7.30	12.88	3.55
ca-coauthors	540,486	139,340	17.96	3.44	21.40	1.60

Table 2: Listing maximal cliques and building compact data structure. Number of maximal cliques, listing time in seconds (t_{lg}), computing clique-graph partition (t_{cg}), and total time building compact structure using clique-graph partitioning ($t_b = t_{lg} + t_{cg}$). Las column (t_{lcg}) displays the running time for listing all maximall cliques from the compact data structure.

number of bytes per vertex in X required in BB. We observe that the r_r ranking function requires fewer bytes than r_c and r_f, where the maximum number of bytes is similar between them. Table 4.1(top-right) shows the space in bits distribution of the compressed representation of the graph using the three ranking functions. We see that the total space does not change greatly between the ranking function, where the space of X and Y are bigger than for r_c an dr_f, but BB is smaller, while the space of B is very small in all cases. Then, we present in Table 4.1(bottom-left) the space in $bpe = Bits/|E|$, bits per edge, of the compressed representation using the ranking functions. We observe that the best space is achieved by the r_f function, second r_c and then r_r, although the difference is not very high. In addition, we measure the random access time for retriving neighbors of any vertex $u \in G$ in Table 4.1(bottom-right). We observe that using the function r_r is much faster than r_f and r_c. Then, considering space and access times we agree that the best ranking for the compressed representation is r_r.

Finally, we compare our best compressed representation with the state-of-the-art compression techniques. Table 4.1 shows the space and time requirements for retrieving neighbors for $1,000,000$ random vertices in G. We observe

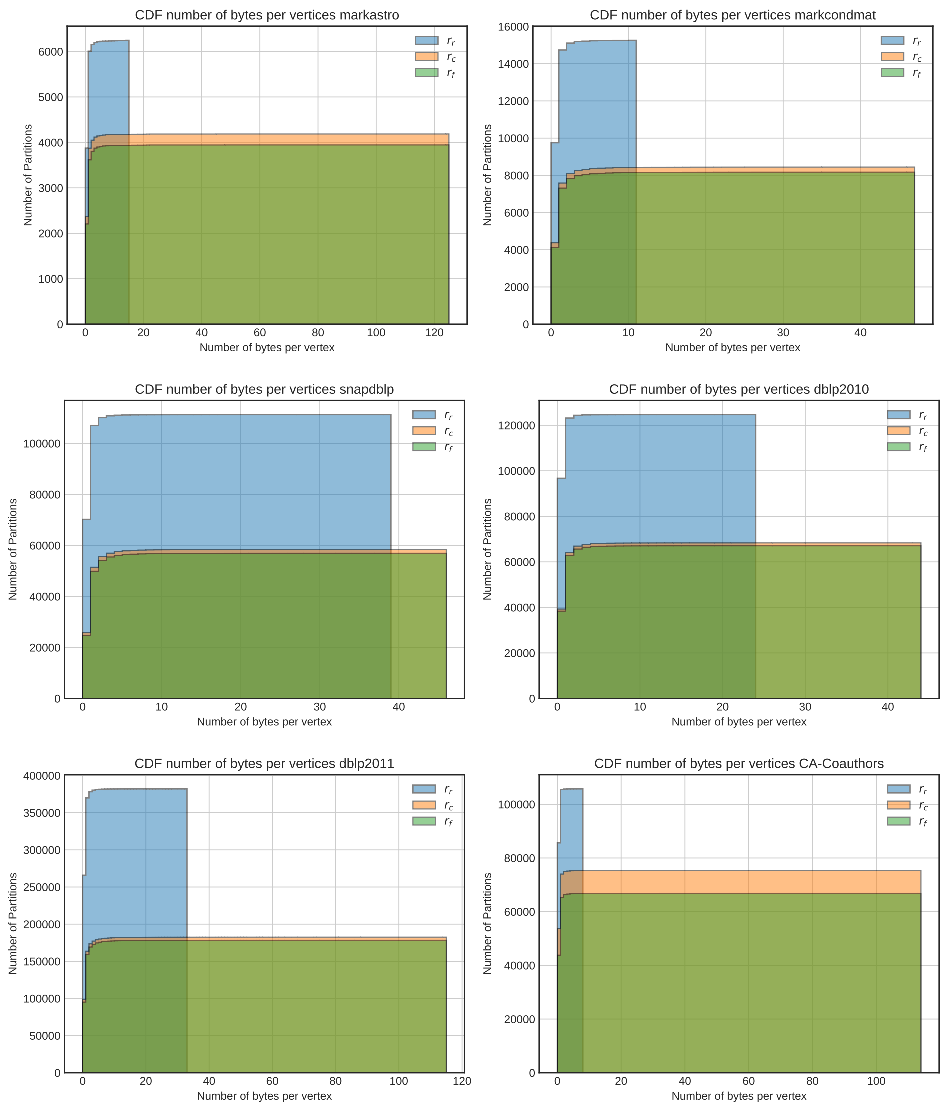

Table 3: Bytes per vertex for all partitions given by its Cummulative Distribution Function (CDF).

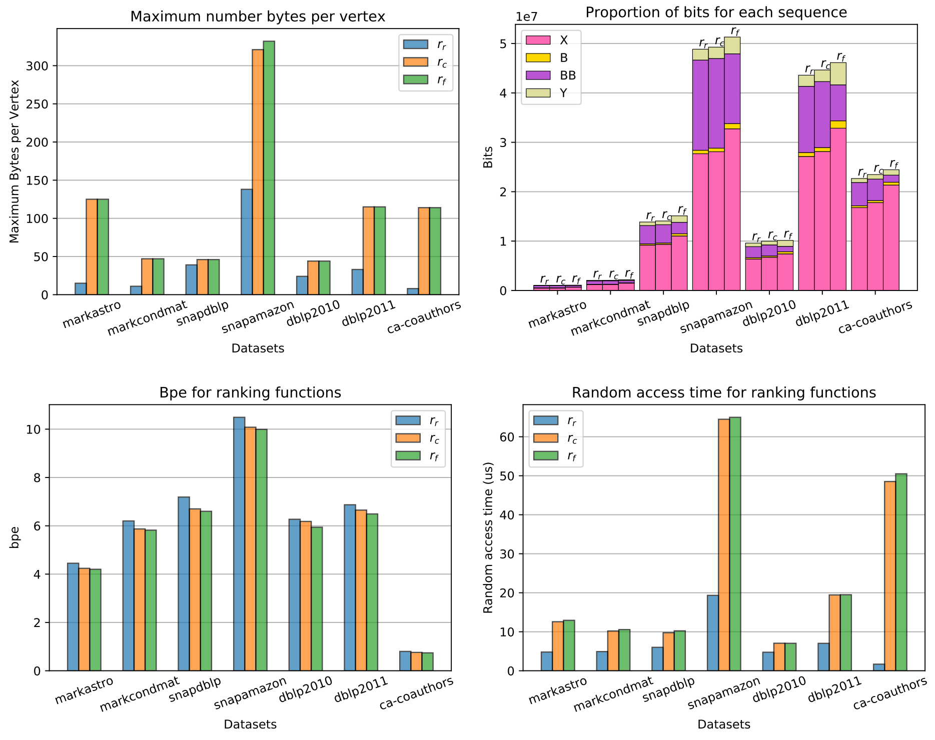

Table 4: Maximum bytes per vertex in partitions (left) and Bpe for different rank funcitons (right).

that our representation provides much better compression in all datasets and the access time is competitive.

5. Conclusions

References

[1] B. Shao, H. Wang, Y. Xiao, Managing and mining large graphs: systems and implementations, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2012, pp. 589–592.

[2] K. Makino, T. Uno, New algorithms for enumerating all maximal cliques, in: Scandinavian Workshop on Algorithm Theory, Springer, 2004, pp. 260–272.

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

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

[5] Z. Lu, J. Wahlström, A. Nehorai, Community detection in complex networks via clique conductance, Scientific reports 8 (1) (2018) 5982.

[6] R. A. Rossi, N. K. Ahmed, The network data repository with interactive graph analytics and visualization, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
URL `http://networkrepository.com`

[7] R. Feynman, F. Vernon Jr., The theory of a general quantum system interacting with a linear dissipative system, Annals of Physics 24 (1963) 118–173.

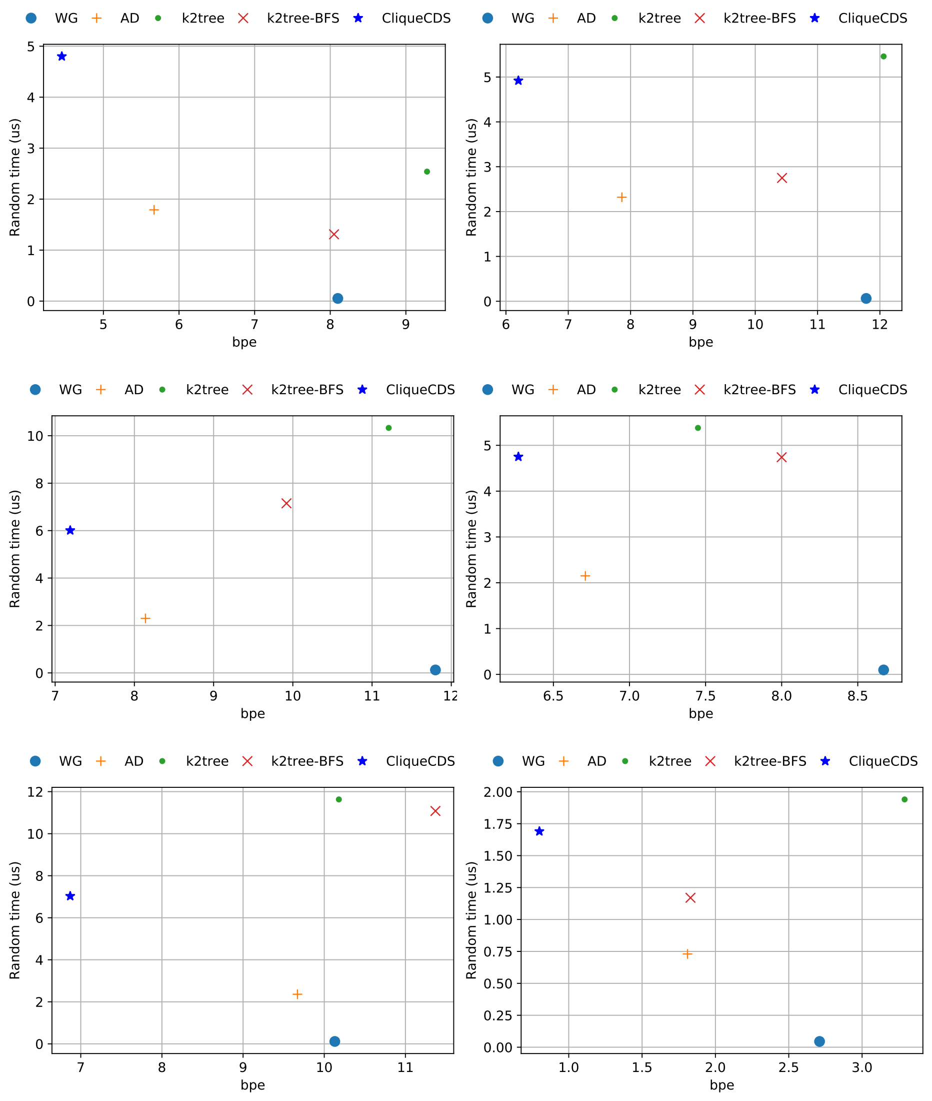

Table 5: Random access time to retrieve neighbors for 1 million of random vertices in G.