Information Processing Letters 125 (2017) 35–40

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A linear time algorithm for maximal clique enumeration in large sparse graphs

Ting Yu, Mengchi Liu *

State Key Lab of Software Engineering, Wuhan University, Hubei, PR China

ARTICLE INFO

Article history:
Received 15 March 2016
Received in revised form 12 April 2017
Accepted 11 May 2017
Available online 19 May 2017
Communicated by R. Uehara

Keywords:
Graph algorithms
Maximal clique enumeration
Data structures
Computational complexity
Linear time algorithms

ABSTRACT

A maximal clique is one of the most fundamental dense substructures in an undirected graph, and maximal clique enumeration (MCE) plays an essential role in densely connected subgraphs discovering. Existing algorithms of maximal clique enumeration employ recursive iteration of adjacent nodes as guiding thought, which incurs high time complexity. In this paper, we propose a linear time algorithm, CM-Constructor (Candidate Map Constructor), for maximal clique enumeration in large sparse graphs which generates a novel data structure called candidate map as result. A candidate map holds not only all maximal cliques of an undirected graph but also some non-maximal cliques that can be easily discarded via an inverted clique tree. To the best of our knowledge, CM-Constructor is the first algorithm to tackle maximal clique enumeration problem utilizing linear procedure. It generates all maximal cliques without duplications for an undirected graph $G = (V, E)$ within $O(|E|)$ time.

1. Introduction

A clique is a subgraph with no less than two vertices in which every vertex connects to each other. A maximal clique is a clique that is not subset of any other cliques. Maximal clique enumeration, also known as maximal clique finding [1,2], is a classical fundamental issue in the field of graph theory that has been investigated for decades and has several variations [3,4]. Maximal clique enumeration is to mine all the maximal cliques in an undirected graph. For example, from the graph in Fig. 1, we can find 8 maximal cliques: $\{a, b, c, d\}$, $\{b, c, d, e\}$, $\{c, e, f\}$, $\{e, g, l\}$, $\{e, k\}$, $\{g, h, i\}$, $\{g, i, l\}$ and $\{i, j\}$. The problem has numerous applications in diverse domains such as data mining on web graphs, social networks, and biological networks. Despite numerous approaches concentrating on

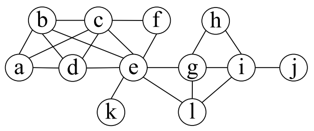

Fig. 1. An example graph.

MCE proposed, most algorithms employ recursive iteration procedure to exhaustive list and estimate all possible cliques.

The first expansion-based algorithm for generating all the maximal cliques has been proposed by Bron and Kerbosch [5]. The algorithm utilizes depth-first search strategy and maintains three disjoint vertex sets R, P, and X, where R is a candidate clique; P holds all the shared neighbors of the vertices in R; X recording the visited common neighbors of R is used to hint R as a non-maximal clique. Within a recursive call of the procedure, whenever P is not empty, the algorithm chooses a vertex v from P to construct new candidate clique $R' = R \cup \{v\}$, new common

* Corresponding author.
E-mail addresses: yuting@whu.edu.cn (T. Yu), mengchi@scs.carleton.ca (M. Liu).

http://dx.doi.org/10.1016/j.ipl.2017.05.005
0020-0190/© 2017 Elsevier B.V. All rights reserved.

neighbors $P' = P \cap N_v$ (N_v is the neighbor set of v) and new visited common neighbors $X' = X \cap N_v$, and recursively calls the procedure with R', P' and X'. After the call, the chosen vertex v is moved from P to X.

Since then, many algorithms have been proposed to improve Bron–Kerbosch algorithm [6–11]. Those optimized algorithms try to cut down the execution time via various strategies such as choosing pivot instead of random vertex from P [7], building degeneracy order for graph [9] and parallelizing the algorithm utilizing parallel framework MapReduce [12,2,13,11,14]. However, the overall idea of using recursive iteration procedure based on depth-first search algorithm has not been changed.

Furthermore, another serial of approaches adopting shrinkage-based algorithm have been studied [15,16]. Those approaches also utilize recursive iteration procedure to iterate objective graph in a top-down fashion, which is opposite to Bron–Kerbosch algorithm.

The existing recursive algorithms are ubiquitously used in real world because recursive iteration procedure is not too complicated to be understood and realized in applications due to its intuitive nature. However, those algorithms are confronted with two problems with the sharply increasing of graph scale: (1) generating a large amount of unnecessary search paths during exhaustivity backtracking every candidate neighbor searching for candidate cliques; (2) excessive execution time wasted by set operations such as calculating intersection and difference of vertex sets during the infinity recursive iterations. The second problem is extremely striking in optimized Bron–Kerbosch algorithms for the reason that selecting a pivot vertex from P will gain several extra intersection operations in every recursive call [7,9].

To solve the problems, we propose in this paper a linear time algorithm for maximal clique enumeration for large sparse graphs. The contributions of the paper are as follows:

(1) A novel structure, called *candidate map* (CM), is designed to hold all the candidate cliques during the construction of maximal cliques. The candidate map reduces the overhead on set operations via representing cliques as ordered lists instead of sets.
(2) A fast algorithm, called *CM-Constructor*, is proposed. Unlike previous algorithms, CM-Constructor is the first algorithm, to the best of our knowledge, introducing linear procedure to the problem. It accomplishes the task via only one traversing of all the edges in a graph and constructs a candidate map as result recording all maximal cliques without duplications. For any large sparse graph, CM-Constructor is executed in linear time $O(|E|)$ which is related to the edge number $|E|$.
(3) An efficient tactic is designed to extract all maximal cliques and discard non-maximal cliques via one traversing of the candidate map based on a special structure: *inverted clique tree*.

2. Problem definition

Let $G = (V, E)$ be an undirected graph. In CM-Constructor algorithm, all the vertices in graph G are sorted in a predefined total order before finding maximal cliques. Every vertex $v_i \in V$ represents the i-th element in the order. See Table 1.

Table 1
Description of symbols.

Symbol	Description
v_i	The i-th vertex in the predefined total order
H_C	The smallest vertex in clique C
B_v	Clique body list of vertex v in G
N_v	The set of neighbors of a vertex v in G
N_v^s	Small neighbor list of a vertex v in G
N_v^b	Big neighbor list of a vertex v in G

Definition 2.1. For two vertices $v_i, v_j \in V$, $i \neq j$, we say v_i is *smaller* than v_j (denoted as $v_i < v_j$) if $i < j$. In the opposite, v_i is *bigger* than v_j (denoted as $v_i > v_j$) if $i > j$.

In a clique C of no less than two vertices, H_C (called *clique header*) is the smallest vertex in C. The clique C is also called H_C-header *clique*. Note that there can be two or more *v-header* cliques for a vertex v in G. If not specified, a clique is an ordered list of vertices in the remainder of this paper. Thus, H_C is the first vertex in clique C, and the rest vertex list in the clique (called *clique body*) is $C \setminus \{H_C\}$. The symbol B_v stands for clique body list, which is a list of clique body of *v-header* cliques.

Definition 2.2. A clique C is *sorted smaller* than vertex v_i if $H_C < v_i$. Similarly, a clique C_i is *sorted smaller* than clique C_j if $H_{C_i} < H_{C_j}$.

The set of neighbors of a vertex v in G is represented as $N_v = \{u \mid (u, v) \in E\}$. When loading the graph, the algorithm generates two neighbor lists N_v^s and N_v^b for each vertex v in the graph, where N_v^s is small neighbor list recording all the vertices in N_v that are smaller than v and N_v^b is big neighbor list holding all the vertices in N_v that are bigger than v. Thus, the number of vertices in the two lists are $|N_v^s|$ and $|N_v^b|$ respectively. Vertices in both N_v^s and N_v^b are sorted in the predefined total order.

3. Algorithm

The predefined total order could be set as an arbitrary order. In this paper, we employ alphabetical ascending order as the predefined total order for convenience. For the graph in Fig. 1, vertices are sorted as: $a < b < c < d < e < f < g < h < i < j < k < l$. Before constructing candidate map, the graph G is loaded to memory by a scan of edge set E to build vertex structure list for every vertex. The vertex structure list of graph in Fig. 1 is given in Table 2. Each vertex structure contains three fields: small neighbor list (SNL), vertex name and big neighbor list (BNL).

3.1. Candidate map

Definition 3.1. A clique C_i is a *v-header maximal clique*, if $H_{C_i} = v$ and there is no other clique C_j satisfies that $H_{C_j} = v$ and $C_i \subset C_j$.