Finding Dense Components in Large-Scale Network Using Randomized Binary Search Tree

Trinh Anh Phuc[1*], Pham Dang Hai[1], Phan Thi Thuy Dung[2]

[1]*Hanoi University of Science and Technology, No. 1, Dai Co Viet, Hai Ba Trung, Hanoi, Vietnam*

[2]*Viettel High Technology Industries Corporation, 40th floor, 72 Landmark Keangnam, Hanoi, Vietnam*

Received: October 05, 2019; Accepted: November 12, 2020

Abstract

Given a simple undirected graph G=(V, E), the density of a subgraph on vertex set S is defined as a ratio between the number of edges |E(S)| and the number of vertices |S|, where E(S) is the set of edges induced by vertices in S. Finding the maximum density subgraph has become an intense study in recent years, especially in the social network era. Being based on a greedy algorithm that connects with a suitable graph data structure, we have reduced its time complexity by using a randomized binary search tree, also called treap. We make the complexity analysis in both time and memory requirements, including computational experiments in large scale real networks.

Keywords: Dense subgraphs, greedy algorithm, randomized binary search tree, graph data structures, large scale real networks.

1. Introduction

The problem of finding high connected subgraphs has been intensively interested in [1-3]. The paper [3] shows that the *Densest k-subgraph* problem remains NP-hard. However, its variance *Densest subgraph* problem can be solved by polynomial time. The algorithm proposed in [2,3] gives us a natural greedy idea of solving the Densest subgraph problem and reveals an interesting result for not only proving theoretical aspects but also practical ones.

Many applications have been seen in a series of papers as [4-7]. In the paper [6], the author presented an algorithm for identifying *hubs* and *authorities* among potential web pages being relevant to a specified query. They said that a set of hubs and authorities is highly connected in comparison to the rest of the graph representing web pages. In [4-5], the authors want to extract, classify and inference dense communities in the web whose communities are considered as dense subgraphs. The definition of graph density provided by [8] will be used in our paper.

1.1. Definitions

Let $G=(V, E)$ be a simple undirected graph where V is a set of vertices and E is a set of edges. Let S be a subset of V and $E(S)$ be the edges induced by S, that means $E(\mathrm{S}) = \{ij \in E : i \in S, j \in S\}$. We get the density definition of the subset S.

Definition 1: *Let S be a subset of V, the density of the subset S is defined as*

$$d(\mathrm{S}) = \frac{|\,\mathrm{E(S)}\,|}{|\,\mathrm{S}\,|}$$

Following the handshaking lemma, $2d(S)$ is the mean degree of the subgraph induced by S.

Definition 2: *The density of the graph G is defined as*

$$d(\mathrm{G}) = \max_{S \subseteq V} \{d(\mathrm{S})\}$$

Hence, 2d(G) is the maximum average degree overall subgraphs.

1.2. Problem and solution

Determining a subgraph S satisfying both definitions 2 and 1 mentioned above is the Densest subgraph problem. The paper [9] showed that this problem is totally equivalent to the integer linear programming solved approximatively by the simplex algorithm [10]. Nevertheless, our intent of the algorithm gets some critical situations in large scale real networks. Firstly, the number of vertices requires much memory space to store $O(|V|^2)$ parameters. Secondly, a polynomial time algorithm seems not sufficient to run quickly in large scale real networks which usually have thousands or even millions of vertices and edges. In addition to the provision of

* Corresponding author: Tel.: (+84) 942.227.941
Email: phucta@soict.hust.edu.vn

another more attractive solution, the paper [9] also proposes the following greedy algorithm

Input: $G = (V, E)$
Output: $d(G)$
$G_0 = (V_0, E_0) \leftarrow G = (V, E)$, $d(G) \leftarrow \frac{|E|}{|V|}$;
for $i \leftarrow 0$ *to* $|V| - 1$ **do**
  if i_{min} *be a vertex of minimum degree* **then**
    for *each* $i_{min}j \in E$ **do**
      remove $E_{i+1} \leftarrow E_i - \{i_{min}j\}$;
      drop j's degree by 1 ;
    end
    remove $V_{i+1} \leftarrow V_i - \{i_{min}\}$;
  end
  if $\frac{|E_{i+1}|}{|V_{i+1}|} > d(G)$ **then**
    $d(G) \leftarrow \frac{|E_{i+1}|}{|V_{i+1}|}$
  end
end
return $d(G)$

Algorithm 1 Densest-Sub-Graph

In our paper, section 2 details a graph data structure using treap which is a new adaptative version for solving the Densest subgraph problem associated with the greedy idea [1-3, 9]. Section 3 represents a complexity analysis of both time and memory requirements corresponding to this graph data structures. Section 4 gives a series of experiments based on large network datasets [11]. Section 5 summarizes our results and gives some future research works.

2. Graph data structures

A simple undirected graph G=(V, E) is used to represent our network data. Without loss of generality, we suppose each vertex of G to be labeled by an identical letter. Each vertex corresponds to a node in our network data and each edge represents a connection between two nodes (see Figure 1).

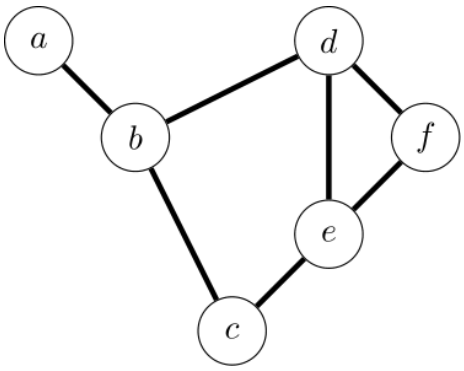

Fig. 1. A simple undirected graph G=(V,E) with $|V|$=6 and $|E|$=7. Each vertex is labeled by an identical letter.

2.1. Adjacency list using treap

We noticed the comment of [12], which real-world networks are usually sparse in, i.e. E=O(|V|), to choose an appropriate data structure to represent our network data. Therefore, the adjacency-list in [6] replaced the conventional adjacency-matrix in the representation of our undirected graph G=(V,E). Recall that the adjacency-list representation of G requires only O($|V|$+2$|E|$) memory space, whereas the adjacency-matrix representation needs much O($|V|^2$) memory space (see Figure 2).

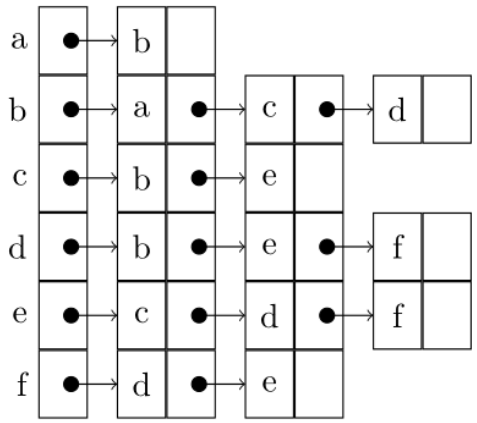

Fig. 2. Undirected graph G=(V,E) in Figure 1 is represented by an adjacency-list data structure.

For the adjacency-list data structure, the authors in [13] suggest an implementation consisting of two components (see Figure 2). The first component is an array indexed by vertex label, sometimes is called by a vertex header containing $|V|$ pointers. These pointers point to head nodes of single-linked list, usually are named first. The second component is just a set of single-linked lists representing 2$|E|$ edges. The relationship between two sets of vertices and edges is therefore represented by an ensemble of first pointers.

Even we get an advantage in the choice of adjacency-list graph representation about memory complexity. Nevertheless, there are some other inconveniences. The adjacency-list data structure does not give us a mechanism of sorting vertices by degree. While we want to remove a specified edge $i_{min}j$ in the previous algorithm 1, a traversal of the whole list of adjacency vertex could take at least O($|V|$) time. We have to update the degree of a vertex j after rearranging remained vertices by degree for the next iteration without knowledge of its new position. Thus, it is necessary to modify the header component of the adjacency list, more precisely, we use a randomized search tree – treap - as a new header component (see Figure 3).

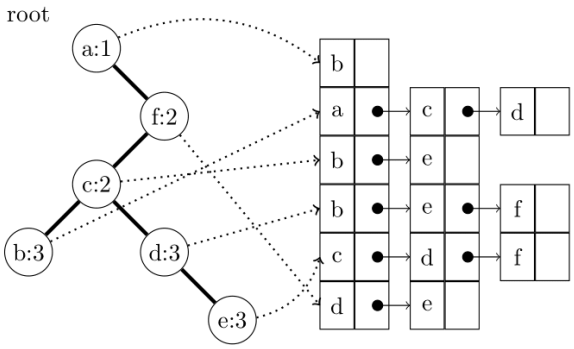

Fig. 3. The adjacency-list representation of G=(V,E) in Figure 2 using treap as a new header. Each treap node consists of a label, degree, and the first of fields. Two labels and degree are encircled in the node. Every first pointer is drawn by a dotted arrow.

Input: $G = (V, E)$
Output: $d(G)$
1 $T^{(0)} \leftarrow G$;
2 $T^{(0)}$.num_vertices$\leftarrow |V|$;
3 $T^{(0)}$.num_edges$\leftarrow |E|$;
4 $d(G) \leftarrow \frac{|E|}{|V|}$;
5 **for** $i \leftarrow 0$ *to* $|V| - 1$ **do**
6 **if** $i_{min} \leftarrow T^{(i)}$.*root.label* **then**
7 **for** *each* j *from root.first* **do**
8 removes j from root.first ;
9 drops i_{min}'s degree by 1 ;
10 drops j's degree by 1 ;
11 updates j's position in $T^{(i)}$;
12 $T^{(i+1)}$.num_edges $\leftarrow T^{(i)}$.num_edges - 1 ;
13 **end**
14 removes root's node $T^{(i+1)} \leftarrow T^{(i)} - \{i_{min}\}$;
15 $T^{(i+1)}$.num_vertices $\leftarrow T^{(i)}$.num_vertices - 1 ;
16 **end**
17 **if** $\frac{T^{(i+1)}.num\_edges}{T^{(i+1)}.num\_vertices} > d(G)$ **then**
18 $d(G) \leftarrow \frac{T^{(i+1)}.\text{num\_edges}}{T^{(i+1)}.\text{num\_vertices}}$
19 **end**
20 **end**
21 **return** $d(G)$

Algorithm 2 Densest-Subgraph-with-Treap.

Treaps are randomized search trees where each node has a key and an associated random priority. Their nodes are organized so that the keys appear in in-order traversal and the priorities appear in max-heap-order. The paper [14] showed that insertion, deletion and searching operators of treap can be implemented in $O(\log|V|)$ expected time. However, we need some modifications of treap to apply it as a new header in our adjacency-list data structure. The first modification is to add a first pointer field into each node of the treap; this pointer usually points to the linked-list of adjacency vertices. The second modification is to replace respectively key and priority by label and degree fields. Instead of using max-heap order, we decided to arrange our treap nodes in min-heap order. A global node T attached to *num_edges*, *num_vertices* and root pointer fields is also implemented. The *num_edges* and *num_vertices* fields are used to get current numbers of vertices and edges. The root field points to the head node of treap that means a container of G=(V,E) representation. The greedy algorithm associated with our new graph data structure is detailed as follow

3. Complexity analysis

3.1. Space complexity

The adjacency-list representation of G=(V,E) requires only $O(|V|+2|E|)$ memory space. We need more space to store degree field for each vertex in the header of adjacency-list representation. Accurately, this header requires O(2|V|) memory space for all vertices and always O(2|E|) memory space for all edges in graph G. The graph data structure using treap costs O(2|V|+2|E|) space complexity.

3.2. Time complexity

In algorithm 2, $T^{(i)}$ denotes our graph data structure using treap at i^{th} iteration. In line 6 of algorithm 2, the i_{min} vertex is always located at the root of the treap and the arrangement of nodes is in min-heap order of degree, takes only O(1) time. A for-loop between lines 7-13 has exactly deg(i_{min}) iterations, always be minuscule, since i_{min} is the minimum degree vertex of $T^{(i)}$. The cost of this loop is calculated as follows:

$$deg(i_{min}) \text{x} O(h^{(i)})$$

where $h^{(i)}$ is the height of treap$T^{(i)}$. Because of the decrease in vertex size of treap after iteration, the expected height of this treap is logarithmic of the current *num_vertices*. If we suppose this deg(i_{min}) is constantly minuscule over graph that means deg(i_{min}) becomes $O(1)$ then the previous cost can be expressed

$$\log(|V|)+\log(|V|-1)+\cdots+\log(1)=\log(|V|!)$$

Using Stirling's approximation to second order,

$$\log(|V|!) \approx |V| \log |V| - |V| + \frac{1}{2}\log(2\Pi |V|)$$

Again, two consecutive lines 14-15 in algorithm 2 take a constant time $O(1)$. In brief, algorithm 2 requires asymptotically $O(|V|\log|V|)$ time complexity. It is necessary to notice that the time complexity is based on the strong hypothesis of the constant of minimum degree vertex.

4. Computational experiments

We decided to implement a series of four graph data structures whose headers are respectively treap [15], heap [16], AVL tree [17] and array [13]. All of them are encoded in the C programming language.

All experiments were carried out on a laptop machine with Intel Dual Core 1.6 Ghz processor and 4 GB RAM under Ubuntu running 64-bit version. We used gcc to compiler our code with -O3 flag optimization option.

Our experimental network datasets are curated by [14] which consists of social networks (Facebook, Google+, Twitter, etc...), networks with ground-truth communities (Amazon product, Youtube social network, etc...), networks of communications (Email, Wikipediatalk, etc...), networks of co-purchasing product at Amazon, networks of road networks. We selected carefully eleven instances detailed in Table 1 for our computational experiments. They are all really large-scale networks containing more than seven thousand vertices to more than one million vertices.

For each instance, we report the following information

- Elapsed times in second, time counter provided by C programming library, corresponding to our four different graph data structures.
- Density of the graph *d(G)* is estimated by the four different graph data structures.

The performance results of these four graph data structures are all displayed in Table 2 in which a bold time in second marks the fastest running time of graph data structure for each instance and a subtraction symbol - indicates that the algorithm associated with a particular graph data structure was unable to run on that network instance due to time limitation, i.e. more than 17000 seconds. The graph data structure using treap outperforms almost the large-scale network instances, in comparison to other variants of graph data structure in term of time complexity, except for the BerkStand instance that reflects the densest network *d(G)* among eleven instances. The graph density *d(G)* estimated by algorithm 2 is very competitive, it exhibits high performance on six network instances. Notice that the greedy algorithm does not provide an optimal solution $d(G)^*$. Hence, the results of the density graph varied for different runs on the same network instances.

Table 1. A brief description of Stanford's large real networks. The space complexity cost is $O(|V|+2|E|)$ since the number of edges represents in twice.

Large scale networks	|V|	2|E|
wiki-vote	7115	103689
email-Enroll	36692	367662
soc-Epinions1	75875	508837
soc-Slashdot	82144	549202
email-EuAll	265214	420045
amazon0601	403394	3387388
BerkStand	685230	7600595
NotreDame	325729	1497134
roadNet-PA	1088092	3083796
roadNet-TX	1379917	3843320
roadNet-CA	1965206	5533214

Table 2. The results of the greedy algorithm based on the four different graph data structures are evaluated on the Stanford's large scale networks.

Large scale networks	Treap		Heap		AVL		Array	
	d(G)	time	*d(G)*	time	*d(G)*	time	*d(G)*	time
wiki-vote	46.275	**0.308**	46.126	1.116	46.275	0.882	46.278	1.114
email-Enroll	37.344	**2.003**	36.563	10.125	37.344	21.612	37.344	14.798
soc-Epinions1	60.253	**17.931**	59.987	42.121	60.253	208.95	60.252	58.806
soc-Slashdot	42.804	**20.199**	41.337	63.582	42.804	261.242	42.804	90.396
email-EuAll	33.588	**212.039**	33.493	287.185	33.589	2029.695	33.586	354.692
amazon0601	7.012	**309.936**	6.893	1192.771	7.012	780.924	7.497	805.591
BerkStand	103.405	130.063	103.405	**85.113**	103.405	900.629	103.405	2094.88
NotreDame	79.645	**266.371**	79.157	631.392	79.645	1277.523	79.645	1161.997
roadNet-PA	1.64	**1679.421**	1.417	2079.127	1.652	2048.168	-	-
roadNet-TX	1.691	**183.029**	1.393	1667.592	1.718	212.696	-	-
roadNet-CA	1.659	**781.205**	1.407	1489.01	-	-	-	-

5. Conclusion

We have introduced the Densest-Subgraph-With-Treap algorithm for finding dense subgraph in large networks based on the greedy idea [1-3, 9] of removing iteratively the minimum degree vertex. After fixing this greedy idea, the graph data structures were taken into account to reduce the complexity of solving the densest subgraph problem.

There are several notices that reflect invariants of the greedy idea [1-3, 9] to find the densest component in graph. In spite of the theoretical proofs showed the 2-approximation algorithm, we got certainly different results on the same input graph performed by different adjacency-list graph headers (treap, heap, AVL or array).

References

[1] Yuichi Asahiro and Kazuo Iwama. Finding dense subgraphs. In John Staples, Peter Eades, Naoki Katoh and Alistair Moffat, editors, Algorithms and Computations, Springer Berlin Heidelberg (1995) 102-111.

[2] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily finding a dense subgraph. J. Algorithms, volume 34 (2000) 203-221.

[3] Uriel Feige and Michael Seltser. On the densest k-subgraph problem. Algorithmica, 29:2001 (1997).

[4] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense communities in the web. In Proceedings of the 16th International Conference on World Wide Web, WWW '07,New York, NY, USA (2007) 461-470.

[5] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In Proceedings of the 31st International Conference on Very Large Data Bases, VLDB '05,VLDB Endowment (2005) 721-732.

[6] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment . J. ACM, volume 46 (1999) 604-632.

[7] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for emerging cyber-communities. In Proceedings of the 8th International Conference on WWW, New York, NY, Elsevier North-Holland, Inc (1999)1481-1493.

[8] V. Vinay Ravi Kannan. Analyzing the structure of large graphs. In manuscrit, NY, NewYork,USA, Elsevier North-Holland, Inc (1999).

[9] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization, APPROX '00, London, UK, UK, Springer-Verlag (2000) 84-95.

[10] George B. Dantzig. A history of scientific computing. Chapter Origins of the Simplex Method ACM, New York, NY, USA (1990) 141-151.

[11] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection (2014).

[12] David Eppstein and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. Lecture Notes in Computer Science, Springer, volume 6630 (2011)364-375.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein.Introduction to Algorithms.The MIT Press, 3rd edition (2009).

[14] Cecilia R. Seidel, Raimund, Aragon. Randomized search trees. Algorithmica, (4/5) volume 16 (1996) 464-497.

[15] Guy E. Blelloch and Margaret Reid-Miller. Fast set operations using treaps. In Proceeding of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '98, New York, NY, USA (1998) 16-26.

[16] J. W. J. Williams. Algorithm 232: Heapsort. Communications of the ACM, volume 7(6), (1964) 347-348.

[17] Evgenii Adelson-Velsky, Georgy, Landis. An algorithm for the organization of information. In Proceedings of the USSR Academy of Sciences, volume 146 (1962) 263-266.