Vol.2011-DBS-153 No.21
2011/11/3

グラフの木分割を用いたコミュニティ発見手法

付興艶† 岩井原瑞穂†

†早稲田大学大学院情報生産システム研究科
福岡県北九州市若松区ひびきの２－７

E-mail: †xinyanfu@fuji.waseda.jp, †iwaihara@waseda.jp

ABSTRACT

A number of researches in graph mining have been devoted in the discovery of communities. In this report, we study a query-dependent variant of the community-detection problem based on the tree decomposition methodology. In the tree decomposition of a graph, each tree node is labeled with a set of graph vertices, and each clique in the graph is always contained by a tree node.

Also, tree nodes containing an identical graph vertex are always connected. Using such properties, we discuss an algorithm that finds a community that contains a given query node. By pre-computing tree decomposition, the task of finding a community can be reduced. We show the experimental results on community detection over real data sets.

Keyword: graph mining, community detection, tree decomposition

1. Introduction

Graphs are one of the most ubiquitous data representations, and they find applications in a wide range of areas including physics, biology, social sciences, and information technology. With the increasing availability of very large networks, there is a need for designing efficient algorithms for graph mining and for discovering latent structures in graphs.

Discovering communities in graphs and social networks has drawn a large amount of attention in recent years [1, 2, 3, 4, 7]. It has been one of the most well-studied problems in graph mining. The general notion of community structure in complex networks was first pointed out in the physics literature by Girvan and Newman [4], and refers to the fact that nodes in many real networks appear to group in subgraphs in which the density of internal connections is larger than the connections with the rest of nodes in the network.

Community detection is a popular topic in a wide range of areas. Most of the work focused on the scenario where communities need to be discovered in an a-priori manner, with only reference to the input graph. However, in many application scenarios, we are interested in discovering the community defined by a given set of nodes, where nodes may represent users of interest, or given criteria.

In this paper, we study a query-dependent variant of the community detection problem. Our problem formulation takes as input an undirected graph G (V, E), and a set of query vertices, and the task is to find a dense connected subgraph of G that contains the query vertices. We use the tree decomposition technique to preprocess the graph G to improve the performance in exploring communities from various query vertices.

Note that from now on, a node in the graph G is referred to as a *vertex*, and a node in a tree decomposition is referred to as a *tree node* or simply a *node*.

In graph theory, a tree decomposition is a mapping of a graph into a tree that can be used to speed up solving certain problems on the original graph. The treewidth measures the number of graph vertices mapped onto any tree node in an optimal tree decomposition. It is NP-hard to

determine the treewidth of a graph. Many instances of NP-hard problems on graphs can be solved in polynomial time if their treewidth is bounded by a constant [18, 19, 20].

Tree decomposition has been used for a number of applications, like combinatorial optimization problems, expert systems, computational biology, etc. Utilizing tree decomposition for shortest path problem in large graphs is shown in [5].

In the tree decomposition of a graph, each tree node contains a set of graph vertices, and each clique in the graph is always contained by a tree node. Also, each vertex in the graph induces a connected subtree in the decomposed tree, and the tree nodes belonging to a community C is also a subtree. In the subtree containing a community C, each tree node includes multiple vertices which also appear in community C. We call the number of vertices which appear in community C of a tree node *community width*. The community width can be used to measure the density of relationship between a tree node and a community. We can terminate exploring the tree decomposition if the community width decreases to a predefined level. This allows us to search communities locally on the tree decomposition.

The rest of the paper is organized as follows. In Section 2 we explain related work, and in Section 3 we explain an existing tree decomposition algorithm we use. In Section 4, we show our algorithm for searching communities over tree decomposition. In Section 5 we present experimental results, and Section 6 concludes the paper and discusses the future work.

2. Related Work

Many different algorithms, coming from different fields such as physics, statistics, data mining, have been proposed to detect communities in complex networks [11, 12, 13, 14, 15, 9]. One of the most famous algorithms is by Newman and Girvan [9], which divides a graph in separated clusters of vertices, which includes the removal of the edges depending on their betweenness values. By iteratively cutting the edge with the greatest betweenness value, it uses the Network Modularity Q to obtain an optimized division of the network with $O(m^3)$ time complexity[16]. The concept of modularity Q:

$$Q = \sum_r (e_{rr} - a_r^2)$$

where e_{rr} are the fraction of links that connect two vertices inside the community r, a_r the fraction of links that have one or both vertices inside of the community r, and the sum extends to all communities r in a given network.

A community in a graph is a group of vertices having a higher density of edges within them, and a lower density of edges between groups. Because there is no universal agreement on the concept of density, so this definition of community is not clear. But a more formal definition has been introduced in [21, 22]. Let C is a subgraph of graph G, S_C is the graph G removed subgraph C. The number of edges in C is denoted as

interEdge = sum(edges in C)

The number of edges between C and S_C is denoted as

outerEdge = sum(edges between C and S_C)

The rate *d* between interEdge and outerEdge is denoted as

$$d = \frac{\text{interEdge}}{\text{outerEdge}}$$

The subgraph C is a community if $d>1$, and the bigger of *d* the stronger sense of C.

Recently, the problem of searching community based on given query nodes is also studied. Mauro Sozio and Aristides Gionis proposed a measure of density based on minimum degree and suggested pruning nodes based on their distance to the query nodes [23].

3. Tree Decomposition

3.1. Definition and properties

An undirected graph G = (V, E) consists of the vertex set V = {0, 1, … , n-1} and the edge set E⊆V×V. Let n=|V| be the number of vertices and m=|E| be the number of edges. In this paper, we consider only undirected graphs, where ∀u, v∈V: (u, v)∈E ⇔ (v, u) ∈E holds. Moreover, we assume that the edges are not labeled.

Definition 1 (Tree Decomposition). A tree decomposition of G = (V,E), denoted as T_G, is a pair ({X_i | i ∈ I}, T), where { X_i | i ∈ I} is a collection of subsets of V and T = (I,F) is a tree with node set I and edge set F, such that:

1. $\bigcup_{i\in I} X_i = V$.

2. for every (u, v) ∈ E, there is i ∈ I, s.t. u, v ∈ Xi.

3. for all v ∈ V , the set {i | v ∈ X_i } induces a subtree of T.

A tree decomposition consists of a set of tree nodes, where each node contains a set of vertices in V. It is required that every vertex in V should occur in at least one tree node (condition 1), and for every edge in E, both vertices of the edge should occur together in at least one tree node (condition 2). The third condition is usually referred to as the connectedness condition, which requires that given a vertex v in the graph, all the tree nodes which contain v should be connected.

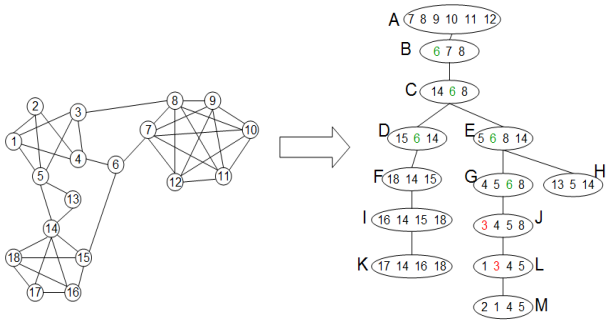
Given a tree decomposition T_G , we denote its root as R.

Given any graph G, there may exist many tree decompositions which fulfill all the conditions in Definition 1. However, we are interested in those tree decompositions with smaller node sizes. We call the cardinality of a node the *width* of the node.

Definition 2 (width, treewidth). Let G = (V, E) be a graph.

- The width of a tree decomposition ({X_i | i ∈ I}, T) is defined as max{| X_i | | i ∈ I }.
- The treewidth of G is the minimal width of all tree decompositions of G. It is denoted as tw(G) or simply tw.

Figure 1 shows a graph G (18 vertices) and a possible tree decomposition of G. The width of the shown tree decomposition is 6.



(a) (b)

Figure 1: A graph G (a) and a tree decomposition of G (b)

Induced subtree. Let G = (V, E) be a graph and T_G = ({X_i | i ∈ I}, T) be a tree decomposition of G. Due to the third condition in Definition 1, for any vertex v in V there exist an *induced subtree* of T_G in which every node contains v. We denote the induced subtree as T_v. Further, we denote the root of T_v as r_v and its corresponding tree node as X_{rv} . For example, X_B, X_C, X_D, X_E and X_G constitute the nodes of the induced subtree of vertex 6 in Figure 1(b), because vertex 6 occurs precisely in these five nodes. Since X_B is the root of the induced subtree, we have r_6 = B.

Using the tree decomposition concept, and using the property of decomposed trees, the efficiency of the proposed approach for the shortest path problem was showed. Inspired by the above paper, we use the tree decomposition concept to design an efficient algorithm for community detection.

Our approach on the community detection problem

contains two steps:

Step 1: Decompose the given graph G, and create a tree as an indexing.

Step2: Given a set of query nodes in the graph G, we find a subgraph of G that contains the query nodes and satisfies specified community criteria.

Since the same tree decomposition can be reused for different query nodes, Step 1 can be preprocessed. Thus only Step 2 needs to be evaluated when answering a query.

3.2. Algorithm for tree decomposition

In this subsection, we explain the tree decomposition algorithm proposed by Fang Wei[5]. First, compose the decomposed nodes to a tree and create an index save the location of every vertex in the tree. The algorithm consists of three steps (see Algorithm 1).

Algorithm 1 tree_decomposition(G)

Input: G = (V, E)

Output: T_G and index of vertices.

1: graph_reduction(G); {output the vertex stack S}

2: tree_construction(S,G); {output the tree decomposition T_G}

3: index_construction(S, T_G); {output the index I_T for every vertex in T_G}

Virtual Edge. Let $v_1, v_2 \in V$, and $(v_1, v_2) \notin E$, but in the process of computation, we add (v_1, v_2) to E for the need of composition process, then we call edge (v_1, v_2) as virtual edge. The opposite of the virtual edge is called a real edge.

The main idea of Wei's tree decompostion algorithm is to reduce the graph by removing the vertices one by one from the graph, and at the same time push the removed vertices into a stack, so that later on the tree can be constructed with the information from the stack. Each time we identify a vertex which has a minimum degree at the time. We first check whether all its neighbors form a clique, if not, we add the missing edges as virtual edges to construct a clique. Then the node v, its neighbors and the real edge between them are pushed into the stack, which is followed by the deletion of v and the corresponding edges in the graph.

The program will terminate if the remaining graph becomes a clique by real edges and virtual edges. Then push the remaining clique's vertices and virtual edge into the stack.

After the reduction process, a tree decomposition can be constructed as follows: (1) At first we pop the top of stack S, and obtain a set of vertices and edges, assign the set as the tree node of the tree root R. The node size of the root is the greatest among all nodes in the tree decomposition. (2) Let X_c be the set of vertices $\{v, v_1,\ldots,v_n\}$ which is popped up from the top of S. Here v is the removed vertex and $\{ v_1,\ldots,v_n \}$ are the neighbors of v which form a clique. Locate the parent node X_p which contains $\{v_1,\ldots,v_n\}$, and add X_c to X_p as a child node. This process proceeds until S is empty.

For efficient access to the tree decomposition, we create a hash table which stores the root of each vertex induced subtree in the T_G .

4. Searching Community from Given Vertices

In this section, we introduce our community detection algorithm utilizing a decomposed tree. In a decomposed tree, the vertex list and edge list between them are stored at each tree node. We traverse one vertex induced subtree to obtain all information about the vertex and its neighbors. If we further traverse neighbors of the induced subtree, we can obtain more information around the original vertex. So under certain conditions, we can obtain a community (also called subgraph) containing the given vertex without traversing the entire tree decomposition.

4.1. Community criteria

There are many definitions of community proposed in various fields. The community we searched need to fulfill two conditions. First, the community must contains the given query vertices. Secondly, vertices in a community have high density links between them. There are a number of ways to define community density. In this paper we introduce a simple condition that can be tested over a tree decomposition. Before introduce our condition, we will introduce two related definitions firstly.

Definition 3 (Node Width, Community Width). Let X_i be a tree node, and C be a community.

- Node Width: The number of vertices in node X_i , denoted as $| X_i |$.
- Community width: The community width $cw(X_i)$ of a community C is the number of the vertices that appear both in C and X_i.

Apparently $cw(X_i) <= | X_i |$ holds. Then by the above definitions, we introduce two parameters: (1) minimum community width *m* and (2) minimum linkage k.

Suppose that we have an original community C_o and a candidate new vertex v. If v and C_o are connected by more than k edges, v is added to community C_o. Or if the degree of v is less than k in the graph G and all the neighbors of v are in C_o, v is added to community C_o.

Suppose that we have an original subtree T_o which contains original community C_o , and a candidate new tree node t. If community width cw(t) on community C_o is no less than minimum community width *m*, t is added to subtree T_o. The new T_o becomes the new boundary of expanding the community. Additionally, other candidate conditions can be used to add new tree nodes to T_o. For example, cw(t) is more than r% of the maximum community width in T_o. But in this paper, we only discuss the minimum community width.

4.2. Community detection from a Single query vertex

Firstly, we will introduce our approach to detect community from a single query vertex. When a single query vertex v is given, use the index I_T to obtain the location of T_v's root r_v on the decomposed tree T_G , and traverse the T_G starting from r_v . If vertices satisfying the community criteria are found, add them to the community. The algorithm consists of three steps (see Algorithm 2).

Algorithm 2 community detection for single (T_G, I_T, v)

Input: decomposed tree T_G, index I_T, query vertex v

Output: community C.

1: initial_community(T_G, I_T, v); {output the initial community C_i}

2: add_node(T_G, C_i); {output the new traversal tree T_c }

3: traversal(T_G , T_c, C_i); {repeat 2-3, output the new community C }

The first step of community detection is to obtain a set of vertices which are directly connected to query vertex v. Subtree that T_v is induced by vertex v. Then traverse T_v and obtain the initial community $C = \{v' \mid v' \in X_i , X_i \in T_v , (v,v') \in E\}$.

The second step is to expand T_v . In Step 1, we traversed T_v and obtained the initial community C. Now we let $T_c = T_v$, check the nodes $\{X_i\}$ around T_c, compute $cw(X_i)$ on community C. While $cw(X_i)$ is no less than *m*(*m* is aninteger parameter for minimum community width), add X_i to T_c. Repeatedly check $\{X_i\}$ around T_c until the community width becomes less than *m*. Return the resulting T_c as output.

The last step of community detection is to traverse T_c. We visit each node X_i in T_c. We check each vertex {v'} contained in X_i and find v' that is not in C. If (1) v' and C are connected by more than k edges, or (2) the degree of v'

is less than k and all the neighbors of v' are in C, then add v' to C. The traversal ends when all nodes of T_c are visited.

Repeat Step 2 and Step 3, until no more new nodes are added to T_c. When Algorithm 2 terminates, we obtain the latest C as community.

5. Experiments

In this section we evaluate our approach for community detection on real social graphs. All tests are executed on an Intel(R) Core 2 Duo 3.0 GHz CPU, and 4 GB of main memory. All algorithm are implemented in JAVA with JUNG (Java Universal Network Graph) Framework 2.0 .

First, we decompose the graph and obtain a tree decomposition. We test a variety of real graph data including biological networks (PPI, Yea and Homo), social networks (Pfei, Geom, Erdos, Dutch, and Eva), information networks (Cal and Epa), DBLP, Arxiv and American College Football dataset. Except American College Football dataset [4] all the graph data are provided by the authors of [5]. The tree decomposition results are shown in Table 1.

Graph	Vertices	Edges	Nodes of Tree	Root Vertices	Root Edges
PPI	1458	1948	1409	50	19
Pfei	1738	1876	1709	30	5
Yeast	2284	6646	2016	269	997
Dutch	3621	4310	3509	113	31
Geom	3621	9461	3452	170	1030
Epa	4253	8897	3986	268	773
Eva	4475	4652	4463	13	7
DBLP	592983	591981	589704	3280	1390
ACFD*	115	616	67	49	96
Cal	5925	15770	5582	344	703
Erdos	6927	11850	6745	183	902
Homo	7020	19811	6266	755	3247
Arxiv	27400	352021	18890	8511	150444

* ACFD = American College Football dataset

Table 1: Statistics of real graphs and its decomposed tree

From Table 1 we can see that the number of tree nodes is almost equals to the number of vertices in graph G, and little less than the number of vertices. The largest node of a tree is often the root.

Now we evaluate our community detection algorithm on the decomposed tree. After the tree decomposition is done, we can find communities from different query nodes while reusing the same tree decomposition. We also evaluate on the results by varying linkage parameter k and minimum community width parameter m. In a tree node, the linkage between candidate vertex and community C is always less than the community width on C. So we can assume that m ≧k.

We show a discovered community from American College Football dataset by query vertex 76, and various parameter m and k. It is showed in Table 2.

m	*k*	community size	Rate *d*
2	2	33	1.245
3	2	28	1.403
4	2	27	1.351
5	2	16	1.038
3	3	16	1.229
4	3	15	1.244
5	3	12	0.268
4	4	15	1.244
5	4	12	0.268
5	5	12	0.268

Table 2: results of community detection for query vertex 76

From Table 2, we can see that when m = 3 and k = 2, the rate d (introduced in section 2) is the biggest. So the community containing 28 vertices is the optimal result. We compared the community with the ones found by Cluset-Newman-Moore community detection algorithm, which is downloaded from SNAP (Stanford Network Analysis Package) [6]. The size of the community which contains query vertex 76 is 27, so we will compare the

result with parameters m = 3 and k = 2 and the result of Cluset-Newman-Moore [16].

Figure 2 depicts the community discovered by our algorithm. The pink vertices are in the community we discovered from query vertex 76 (in the red circle), and the green ones are not in the community. Figure 3 is the result of Cluset-Newman-Moore. Each color expresses one community. Query vertex 76 is in the community colored in yellow, and vertex 76 is surrounded by the red circle.

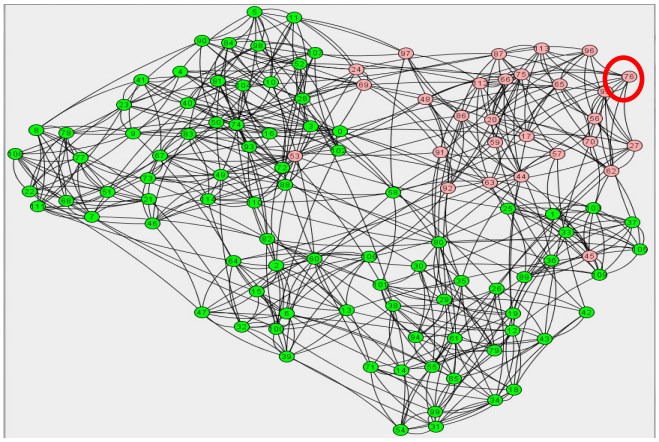


Figure 2: American College Football dataset for query vertex 76

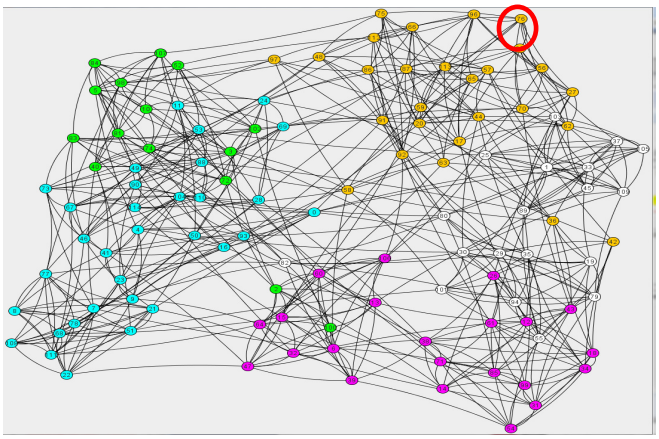


Figure 3: Cluset-Newman-Moore for American College Football dataset

Vertex list of our result: 17 20 24 27 44 45 48 53 56 57 59 62 63 65 66 69 70 75 76 86 87 91 92 95 96 97 112 113;

Vertex list of Cluset-Newman-Moore result: 17 20 27 36 42 44 48 56 57 58 59 62 63 65 66 70 75 76 86 87 91 92 95 96 97 112 113.

The similarity rate between the two results is 88.89%.

6. Conclusion and Future Work

In this paper, we introduced an approach for community detection based on tree decomposition. We showed an algorithm for single-query vertex community detection depended on the decomposed tree of the graph.

In future, a number of issues will be done for this approach, for example, multiple query vertices algorithm, evaluation of this approach, complexity analysis, and how to determine the parameters k and m.

References

[1] Y. Dourisboure, F. Geraci, and M. Pellegrini. Extraction and classification of dense communities in the web. In WWW, 2007.

[2] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee. Self-organization and identification of web communities. Computer, 35(3):66–71, 2002.

[3] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In VLDB, 2005.

[4] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. Proceedings of the National Academy of Sciences of the USA, 99(12):7821–7826, 2002.

[5] Fang Wei. TEDI: Efficient Shortest Path Query Answering on Graphs. In SIGMOD'10, June6-11, 2010, Indianapolis, Indiana, USA.

[6] http://snap.stanford.edu/snap/index.html

[7] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In WWW, 2008.

[8] P. Pons and M. Latapy. Computing communities in large networks using random walks. In ISCIS2005, pages 284{293, 2005.

[9] A. Clauset, M. Newman, and C. Moore. Finding community structure in very large networks. Physical Review E, 70(066111), 2004.

[10] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. De¯ning and identifying communities in networks. PNAS, 101(9):2658{2663, March 2004.

[11] John E. Hopcroft, Omar Khan, Brian Kulis, and Bart Selman. Natural communities in large linked networks. In Proc. International Conference on Knowledge Discovery and Data Mining (KDD'03),

pages 541–546, 2003.

[12] M. E. J. Newman. Fast algorithm for detecting community structure in networks. Physical Review E, 69:066133, 2004.

[13] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. Proc. Natl. Acad.Sci. USA (PNAS'04), 101(9):2658–2663, 2004.

[14] A. Arenas and A. Diaz-Guilera. Synchronization and modularity in complex networks. European Physical Journal ST, 143:19–25, 2007.

[15] S. Lozano, J. Duch, and A. Arenas. Analysis of large social datasets by community detection. European Physical Journal ST, 143:257–259, 2007.

[16] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical Review E, 69(026113), 2004.

[17] http://www.facebook.com/

[18] W. Cook and P. D. Seymour. An algorithm for the ring-routing problem. in: Bellcore Techn. Memorandum, Bellcore (1994).

[19] W. Cook and P. D. Seymour. Tour merging via branch decomposition. Inform. J. Comput., 15, No. 3, 233–248 (2003).

[20] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. van Hoesel. Treewidth: Computational experiments. in: Zib-Report-38, Berlin (2001).

[21] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. Proc. Natl. Acad.Sci. USA (PNAS'04), 101(9): 2658–2663, 2004.

[22] Clara Pizzuti. Overlapped Community Detection in Complex Networks. GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

[23] Mauro Sozio and Aristides Gionis. The Community-search Problem and How to Plan a Successful Cocktail Party. KDD'10, July 25–28, 2010, Washington, DC, USA.