Triangle Counting in Networks Using a Multi-Level Branching Technique

Jungeun Kim, Minseo Kang, Sungsu Lim, Jae-Gil Lee*
Department of Knowledge Service Engineering
KAIST
Daejeon, Republic of Korea
Email: {je_kim, minseo, ssungssu, jaegil}@kaist.ac.kr

***Abstract*—Counting triangles in networks is a fundamental problem in network science. In addition, because we are forced to manage very large real-world networks, current triangle counting algorithms naturally require a distributed computing system. In this paper, we propose a distributed triangle counting algorithm based on both the vertex-centric and node-iterator models and using the multi-level branching technique. *Multi-level branching* is a method that constructs an ordered graph structure based on levels. This method not only facilitates an efficient triangle counting process, but also guarantees the computational integrity of each split in the distributed triangle counting process. First, we describe a *level-based* triangle counting algorithm based on both the vertex-centric model and the node-iterator algorithm. Then, we develop a distributed implementation of the proposed algorithm using GraphChi. The main advantages of the proposed algorithm are that the execution is simple yet effective, and thus its parallelization is efficient. Experiments on real-world networks verify its performance, particularly, its near-linear parallelization scalability.**

I. Introduction

One of the main issues that arise in mining massive graphs is counting of triangles. Traditional graph-mining approaches use a set of nodes and edges. However, whereas nodes and edges do not give us much information about structure, a triangle is regarded as a basic module, subgraph, clique, or motif that represents the small-scale structure of a graph. Another key feature is the triadic closure, which means that friends of friends are more likely to become friends, so that there are many triangles of connections in social networks [8]. In addition, the problem of triangle counting has recently received much more attention in the data mining community [1], [2]. There are numerous applications in network analysis related to these problems; for example, detecting spam or fake users, and uncovering the hidden structure in networks [6], [5], [7]. Moreover, it is highly related to the problem of computing the clustering coefficient [8], which measures the degree to which nodes in a graph tend to cluster together.

In this paper, we focus on the problem of counting triangles in large-scale graphs and propose a new triangle counting algorithm along with its distributed implementation using GraphChi [4]. The key idea underlying our algorithm is construction of an ordered graph structure based on the level from a given graph, called the *multi-level branching technique*. We then present the *level-based* triangle counting algorithm and prove its properties and advantages.

$*$ Jae-Gil Lee is the corresponding author.

II. Preliminaries

In this section, we introduce a well-known node-iterator algorithm based on an ordering of nodes. Consider that there is a strict partial order for the set of nodes such that for all u, v, and w in V, not $u \prec u$ (irreflexivity), if $u \prec v$ and $v \prec w$ then $u \prec w$ (transivity), and if $u \prec v$ then $v \prec u$ (asymmetry) holds. Based on the order, we label a number $n(v)$ to each node v where $n(v) = |\{u \in V | u \prec v\}|$. Then, $\{n(v)|v \in V\} = \{0, 1, \ldots, n-1\}$. Subsequently, we define a node-iterator algorithm with the same order in Algorithm 1.

Algorithm 1 Node Iterator

```
INPUT: An undirected unweighted simple graph $\mathcal{G} = (V, E)$;
OUTPUT: The list of triangles in $\mathcal{G}$;
 1: $L \leftarrow \{\}$;
 2: for each $v \in V$ do
 3:     for each $u \in \Gamma(v)$ and $u \succ v$ do
 4:         for each $w \in \Gamma(v)$ and $w \succ u$ do
 5:             if $\{u, w\} \in E$ then
 6:                 $L \leftarrow L \cup \{(v, u, w)\}$;
 7:             end if
 8:         end for
 9:     end for
10: end for
11: return $L$;
```

In the next section, we develop the node-iterator algorithm by proposing a technique for ordering nodes. Our proposed technique gives not only an order but also levels; consequently, it possesses advantages for developing a parallelization algorithm.

III. Proposed Algorithm

Table I summarizes the notations used throughout this paper.

TABLE I: Summary of the notation.

Notation	Description
$\mathcal{G} = (V, E)$	a given graph
d_v	the degree of v in V
$n(v)$	the labeled number of v in V
$\Gamma(v)$	the set of neighbors of v in V
L_i	the set of nodes belonging to level i
CL_i	the union of L_i and L_{i+1}
Q_i	the set of nodes that have not yet had their levels after i iterations

Our observations indicate that there can be three types of triangles in parallel algorithms for counting triangles: a triangle can belong to one split, two splits, or three splits together.

Therefore, in order to count triangles precisely, all triple splits have to be checked. For instance, Figure 1 shows that three types of triangles can exist in any triple of splits of the set of nodes.

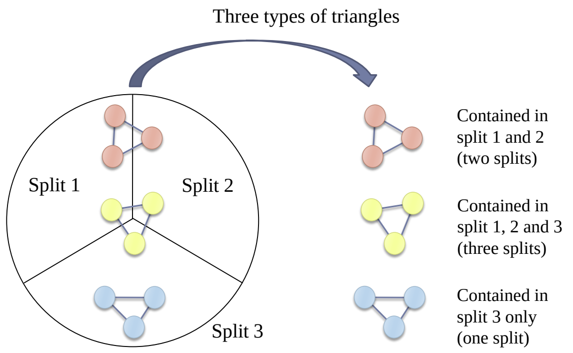

Three types of triangles
Split 1
Split 2
Split 3
Contained in split 1 and 2 (two splits)
Contained in split 1, 2 and 3 (three splits)
Contained in split 3 only (one split)

Fig. 1: Three types of triangles in parallel algorithms.

A. Multi-level branching

We propose a new technique for splitting a set of nodes that enables reduction of the number of checking processes among the splits. The main idea is to construct a structure of the node set that can be identified by the level and only two sequential levels are closely related to each other. Our goal is to design an algorithm that runs from the higher level to the lower level in sequential order, while we only need to look at the current level and the next level in each iteration, *i.e., two levels not three splits*. Thus, we introduce a simple observation of triangles and then present our technique.

By the definition of a triangle, every two nodes in any triangle are joined by an edge. This implies that two nodes connected by 2-hop cannot be adjacent to each other in a triangle. Motivated by this simple observation, we discover that the multi-level structure obtained by the following algorithm bestows many advantages.

Proposed technique for constructing a multi-level structure is described as follows. Firstly, we select the node with the highest degree as the root (with a tie-breaking rule). Let $L_0 = \{v\}$, and let its neighbors form a level L_1. Next, among the remaining nodes, let the nodes that can be reached by an edge in L_1 form a level set L_2. Perform this procedure iteratively to give a sequence of level sets as a partition of the node set. For instance, in Figure 2, node 1 is the root node and the level sets are constructed deterministically by Algorithm 2.

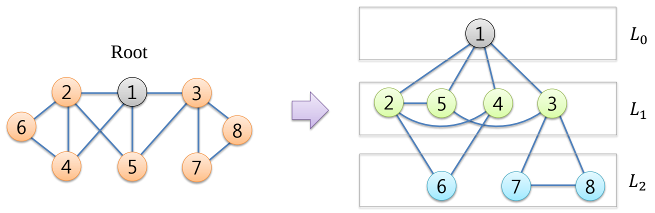


Fig. 2: The multi-level branching technique.

Note that $L_i \in V \backslash Q_{i-1} = \cup_{j=0}^{i-1} L_j$ (from Lines 6 and 8), $L_i \cap L_j = \emptyset$ for all $j = 0, \ldots, i-1$, and for all $i \geq 1$. Thus, all the L_i's are disjoint and the level sets form a partition of V. Also, the following Theorem 1 guarantees that Line 6 can be replaced by $L_i \leftarrow \{c \in Q_{i-1} \mid \exists p \in L_{i-1}$ and $\{p, c\} \in L_{i-1}\}$. Thus, there is no edge joining two non-sequential level sets.

Algorithm 2 Multi-Level Branching

```
INPUT: An undirected unweighted simple graph $\mathcal{G} = (V, E)$;
OUTPUT: A partition of $V$, $(L_0, L_1, \ldots, L_k)$ for some $k \leq n$;

 1: $v = argmax_v d_v$ (with a tie-braking rule);
 2: $L_0 \leftarrow \{v\}$;
 3: $Q_0 \leftarrow V \backslash L_0$;
 4: $i = 1$;
 5: while $Q_{i-1} \neq \emptyset$ do
 6:     $L_i \leftarrow \{c \in Q_{i-1} \mid \exists p \in V \backslash Q_{i-1}$ and $\{p, c\} \in E\}$;
 7:     /* $L_i$ is a set of nodes that is reachable by an edge from
        $L_{i-1}$ but have not yet had their levels. */
 8:     $Q_i \leftarrow Q_{i-1} \backslash L_i$;
 9:     $i \leftarrow i + 1$;
10: end while
11: return The level sets $L_i$'s;
```

Theorem *1:* For each $i \geq 1$, $L_i = \{c \in Q_{i-1} \mid \exists p \in L_{i-1}$ and $\{p, c\} \in E\}$ holds.

Proof: From Line 6 of Algorithm 2, $L_i = \{c \in Q_{i-1} \mid \exists p \in V \backslash Q_{i-1}$ and $\{p, c\} \in E\}$. Thus, it suffices to show that if there is an edge $\{p, c\}$ with $c \in L_i$ and $p \in V \backslash Q_{i-1}$ then $p \in L_{i-1}$. Assume to the contrary that $p \notin L_{i-1}$, then $p \in V \backslash Q_{i-1} \backslash L_{i-1} = \cup_{j=0}^{i-1} L_j - L_{i-1} = \cup_{j=0}^{i-2} L_j$. Therefore, p is contained in some L_j and $V \backslash Q_j$, $j = 0, \ldots, i-2$. By the definition of the level set, $\{p, c\} \in E$ implies that $c \in L_{j+1}$ and $j + 1 < i$. It contradicts to the assumptions. Hence, $p \in L_{i-1}$ holds and we get the desired result. ■

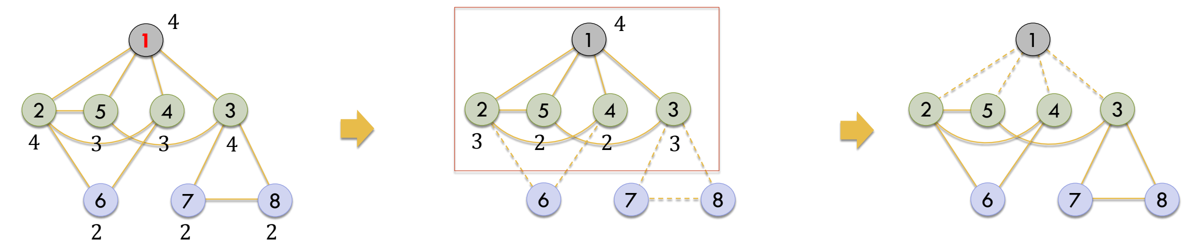
The theoretical meaning of the level is that a node v at a level i is equivalent if v is reachable from the root node using i sequentially-connected edges, i.e., a path of length i. Further, Theorem 2 states that a node has its unique level (if a given graph $\mathcal{G}$ is connected[1]). Therefore, the resulting level sets form a partition of the set of nodes.

Theorem *2:* For each node $v \in V$, a node v is contained in L_i if and only if the shortest-path distance between the root node and v is i.

Proof: We prove it by mathematical induction. (i) base case: if $i = 0$ or $i = 1$, the statement holds by the definition. (ii) inductive hypothesis for some $i \geq 1$: for all $i' \leq i$, we assume that if a node $v \in V$ is contained in $L_{i'}$ then the shortest-path distance between a root node and v is i'. If a node u is contained in L_{i+1}, then there is a node $w \in L_i$ where $\{w, u\} \in E$. Thus, the shortest-path distance between a root node and u is at most $i + 1$. By the inductive hypothesis, the shortest-path distance between a root node and u is larger than i. Thus, the distance should be $i + 1$. By (i) and (ii), the statement is true. ■

Let us not look at why this technique can reduce the number and types of triangles in a given graph. Theorem 3 proves that if three nodes form a triangle, then they are contained in either one level set or two level sets. Further, we can easily check because, if they participate in two level sets, then the difference between them should be one. These

[1]In this paper, we assume that a given graph is connected. As a preprocessing step, we first separate the graph into its connected components, then apply the algorithm to each component.



Step 1:
Select a highest degree root node

Step 2:
List triangles in the induced subgraph of $\Gamma(1)$

Step3:
Ignore already seen edges

Fig. 3: An example of the counting algorithm with a multi-level structure.

observations lead to the property that we only need to check every two sequential level sets in order to count the triangles in the graph, instead of every three splits of the set of nodes.

Theorem *3:* If three nodes form a triangle, then those nodes participate in at most two level sets.

Proof: Suppose not. Then, three nodes have different levels. Thus, there are two nodes with the difference at least 2. Since every two nodes in a triangle are adjacent, there difference have to at most 1. Contradiction. ■

B. Triangle listing with the multi-level structure

Let us not look at how we list the triangles with the level sets. We apply the node-iterator process to the multi-level structure. For the set of nodes, we construct a lexicographical order, as in Definition 1, using both the level sets and the degrees, which is used by the node-iterator-type algorithms when they select a node at each step.

Definition *1:* (Ordering of the nodes) $a \prec b$ if and only if (i) the level of a is less than that of b or (ii) the levels are the same but the degree of a is less than that of b (with a tie-breaking rule).

Two main advantages of our proposed algorithm are computational complexity and parallelization. The former is achieved from the properties of the node-iterator because it can be applied to a hybrid algorithm with the other algorithm, e.g., fast matrix multiplication. The latter is from the use of the multi-level structure. As seen before in this section, this structure can be a powerful tool if it executes in parallel.

Algorithm 3 Triangle Listing with the Multi-Level Structure

```
INPUT: An undirected unweighted simple graph $\mathcal{G} = (V, E)$;
OUTPUT: The list of triangles in $\mathcal{G}$;
1: Run Algorithm 2 (Multi-Level Branching);
2: Construct an ordering of the nodes using Definition 1;
3: Run Algorithm 1 (Node Iterator);
4: return The list of triangles $L$;
```

Example *1:* To clarify the explanation, we here demonstrate use of Algorithm 3 for a toy graph in Figure 3.

- Step 1: The graph has eight nodes and three levels. First, we select node 1 as the root node because it has the largest degree. (We use an arbitrary tie-breaking rule.)
- Step 2: For the top level, node 1 is marked as number 1, and then the nodes in the next level are marked in a degree-descending order, and so on.
- Step 3: For each iteration of the node-iterator algorithm, we select the nodes in ascending order according to the marks. At the first iteration, we construct the induced subgraph $G[\Gamma(i)]$ of $\Gamma(1)$, where $\Gamma(i)$ is the union of node i and its neighbors. Then, we list the triangles that contain node i according to the order of nodes. For the counting problem, we can calculate the number of triangles that contain node i in $G[\Gamma(i)]$ using the degrees of all nodes of $G[\Gamma(i)]$ (see Theorem 4). Then, we ignore (for simplicity, we delete) already seen edges and select the next node, until there is no remaining node. Another important action is to delete all nodes of degree 1 in each step since they cannot participate in any triangle.

Theorem *4:* For the induced subgraph $G[\Gamma(i)]$, let $d(j)$ be the degree of each node $j \in \Gamma(i)$. Then, the number of triangles that contain node i is given by $(\sum_{j\in\Gamma(i)} d(j) - d(i))/2$.

Proof: For each triangle that contains node i, another two nodes are neighbors of i and are joining by an edge. Also, each edge that is not adjacent to i always a part of a triangle that contains i. Because, if two nodes $j_1, j_2 \in \Gamma(i)\backslash\{i\}$ and $\{j_1, j_2\} \in E$, then it belongs to a triangle since $j_1, j_2 \in \Gamma(i)$ implies that $\{i, j_1\} \in E$ and $\{i, j_2\} \in E$. For instance in Figure 3, for $G[\Gamma(1)]$, $\{2, 4\}$, $\{2, 5\}$ and $\{3, 5\}$ belong to triangles with a node i since 2, 3, 4 and 5 are neighbors of 1. Thus, the number of triangles that contain node i is $\frac{1}{2}\sum_{j\in\Gamma(i)\backslash\{i\}}(d(j) - 1) = (\sum_{j\in\Gamma(i)} d(j) - d(i))/2$. ■

C. Optimal splitting

In this section, we propose a means of parallelizing Algorithm 3. Note that the multi-level branching method guarantees that it suffices to check every two consecutive levels for triangle counting. This advantage leads to a simple parallel algorithm for counting triangles. Each split consists of a set of two consecutive levels while the union of all splits covers every two consecutive levels. Let CL be a list of all CL_i's, and $|CL_i|$ the size of CL_i that denotes the total number of nodes belonging to CL_i. We aim to decide on a list of splits $S = (S_1, S_2, \ldots, S_k)$, where k is the pre-assigned number of splits. An optimal method of splitting CL is achieved when the maximum difference in size between two S_j's is minimized. Once S is determined, each split is assigned to k-threads for parallel processing. This optimal splitting problem is similar to the bin packing problem [3] except that the number of splits is fixed and the capacity of the split may vary. The bin packing problem is known to be NP-hard [3], and heuristic algorithms are used. In this paper, we suggest Algorithm 4 for our optimal splitting problem based on greedy approximation.

IV. Experimental Evaluation

In this section, we present the experimental results for our proposed triangle counting algorithm. We used GraphChi's

Algorithm 4 Optimal Splitting

```
INPUT: A list of two consecutive levels $CL = (CL_1, CL_2, \ldots, CL_i)$, and the desired number of splits $k$;
OUTPUT: A list of splits $S = (S_1, S_2, \ldots, S_k)(k \leq i)$;
1: Compute $M = \frac{\sum_i |CL_i|}{k}$;
2: while $CL \neq \emptyset$ do
3:     for $j = 1, \ldots, k$ do
4:         Compute $\epsilon_j = M - |S_j|$;
5:     end for
6:     Move a largest $CL_i$ among the top $k$ largest $CL_i'$s into each $S_j$ in descending order of $\epsilon_j$;
7: end while
8: return A list of splits $S$;
```

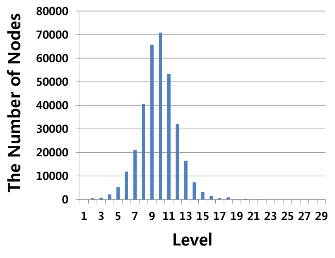
C++ version to implement our proposed method. All experiments were done on a 2.00 GHz quad-core CPU(Intel Xeon) with 14GB RAM, running CentOS 6.5.

A. Dataset

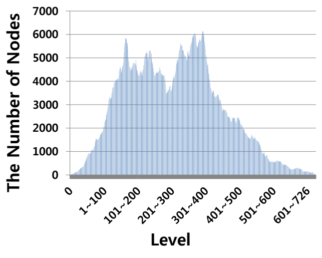
We used the Amazon co-purchasing network(com-Amazon) and the California road network(roadNet-CA) available at the Stanford Large Network Dataset Collection[2]. These networks range in size from 334,863 vertices and 1,957,027 edges(com-Amazon) to 1,957,027 vertices and 2,760,388 edges(roadNet-CA).

B. Node distribution by level

First, we analyzed the node distribution by level. Figure 4a shows the node distribution of com-Amazon by level. The maximum level is 27 and the largest level set contains 70,793 nodes, approximately 21% of the total number of nodes. The number of level sets whose size is bigger than 10% of the largest level set is nine. Figure 4b shows the node distribution of roadNet-CA by level. The maximum level is 726 and the largest level set contains 6,089 nodes, approximately 0.3% of the total number of nodes. The number of level sets whose size is bigger than 10% of the largest level set is 542.



(a) com-Amazon.



(b) roadNet-CA.

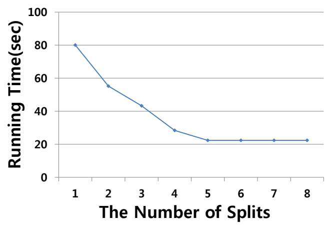
Fig. 4: Node distribution by level.

C. Scalability

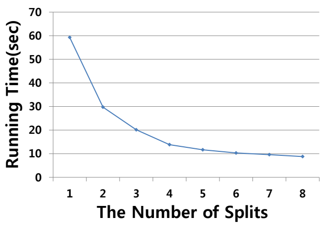
In order to demonstrate the scalability of the proposed method, we measured running time while varying the number of splits from one to eight. Figure 5a shows that running time decreased virtually linearly with increases in the number of splits up to five. Five splits are sufficient for the com-Amazon dataset, and dividing more splits is not effective. This result comes from the fact that the number of nodes in the largest CL is larger than 20% of the original dataset. Since CL cannot be divided, the largest CL still occupies one split even if the number of splits is above five.

Conversely, Figure 5b shows that running time decreased virtually linearly with increases in the number of splits up to eight. In this case, the largest CL takes only 0.6% of the original data set and the node distribution is relatively uniform across the level. Thus, running time decreased as the number of splits increased and this trend is expected to continue when the number of splits is much larger.

[2]http://snap.stanford.edu/data/



(a) com-Amazon.



(b) roadNet-CA.

Fig. 5: Running time.

V. Conclusions

In this paper, we proposed a distributed triangle counting algorithm based on both the vertex-centric and node-iterator models and using the multi-level branching technique, in order to support large-scale networks. The multi-level branching technique facilitates the development of an efficient parallel algorithm and reduces the number of checking processes for triangle counting because it guarantees the computational integrity of any combination of two sequential levels. The results of experiments conducted using real-world networks with millions of edges indicate near-linear scalability for our proposed algorithm.

VI. Acknowledgments

This work was supported by the ICT R&D program of MSIP/IITP. [1391104004, Development of Device Collaborative Giga-Level Smart Cloudlet Technology]

References

[1] X. Hu, Y. Tao, and C.-W. Chung. Massive graph triangulation. In *Proceedings of ACM SIGMOD '13*, pages 325–336, 2013.

[2] M. Jha, C. Seshadhri, and A. Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proceedings of ACM SIGKDD '13*, pages 589–597, 2013.

[3] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of FOCS '82*, pages 312–320, 1982.

[4] A. Kyrola, G. Blelloch, and C. Guestrin. GraphChi: large-scale graph computation on just a PC. In *Proceedings of OSDI '12*, pages 31–46, 2012.

[5] S. Lee, M. Ko, K. Han, and J.-G. Lee. On finding fine-granularity user communities by profile decomposition. In *Proceedings of ASONAM '12*, pages 631–639, 2012.

[6] S. Lim, S. Ryu, S. Kwon, K. Jung, and J.-G. Lee. LinkSCAN*: Overlapping community detection using the link-space transformation. In *Proceedings of ICDE '14*, pages 292–303, 2014.

[7] S. Moon, J.-G. Lee, and M. Kang. Scalable community detection from networks by computing edge betweenness on mapreduce. In *Proceedings of BigComp '14*, pages 145–148, 2014.

[8] D. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, June 1998.