# BRICS – Efficient Techniques for Estimating the Farness-Centrality in Parallel

Sai Charan Regunta, Sai Harsh Tondomker and Kishore Kothapalli

Center for Security, Theory, and Algorithmic Research (CSTAR),
International Institute of Information Technology, Hyderabad
Gachibowli, Hyderabad, India 500 032.
Email: {{saicharan.regunta,sai.harsh}@research.},kkishore@}iiit.ac.in

***Abstract*—In this paper, we study scalable parallel algorithms for estimating the farness-centrality value of the nodes in a given undirected and connected graph. Our algorithms consider approaches that are more suitable for sparse graphs. To this end, we propose four optimization techniques based on removing redundant nodes, removing identical nodes, removing chain nodes, and making use of decomposition based on the biconnected components of the input graph.**

**We test our techniques on a collection of real-world graphs for the time taken and the average error percentage. We further analyze the applicability of our techniques on various classes of real-world graphs. We suggest why certain techniques work better on certain classes of graphs.**

## I. Introduction

Centrality metrics offer a numerical perspective on the relative importance on individual nodes and/or edges of a graph. These metrics have applications to many problems in social network analysis, epidemiology, and electrical network analysis [21]. Bavels [3] initiated the study of structural centrality in the context of social graphs. Popular notions of centrality metrics include the betweenness-centraltity, degree centrality, Katz centrality, and farness-centrality [23], [20], [3]. Computing such metrics on a variety of modern parallel architectures continues to attract research attention [12], [19]. Since these metrics usually take a large time to compute especially on large graphs, there have been recent attempts to also obtain approximate solutions or estimates to the metrics [20], [6], [5].

In this paper, we study the farness metric defined on the nodes of a graph. The farness-centrality of a node $v$ in a connected undirected graphs $G$, denoted farness$(v)$is defined as the sum of the shortest distances from node $v$ to all other nodes in the graph. In symbols [24], farness$(v) = \sum_{w \in V, w \neq v} d(v, w)$ where $d(v, w)$ indicates the distance of a shortest path from $v$ to $w$ in $G$. (The inverse of this quantity is referred to as the closeness-centrality of node $v$.)

From the definition of farness centrality, it is easy to note that the metric can be computed exactly for each node $v$ by performing a single-source-shortest-path computation from $v$. However, as the size of the networks of interest, such as social networks, is often of the magnitude of millions of vertices, exact computation faces scalability challenges. Therefore, one is interested in computing variants of the farness centrality such as estimates of closeness centrality [12], the top $k$ values of closeness centrality given a graph $G$ [22], the 1-median of a graph $G$, and improving closeness centrality of a node [8].

Building on ideas from sampling techniques of Indyk and Thorup [17], [26], in a recent paper, Cohen et al. [6] show that estimates to the closeness centrality of nodes in a graph can be obtained by combining two popular techniques: sampling and pivoting. It provides an adaptive error estimator but does not define the actual error/deviation of estimated values from the actual centrality values.

In this paper, we study efficient parallel algorithms for obtaining estimates to the farness centrality of all nodes in a given graph. To the best of our knowledge, this is the first work in parallel algorithms for estimating farness centrality. Our work is aimed at real-world graphs that have discernible characteristics such as being sparse, bi-connectivity. Such algorithms that aim to make use of the characteristics of the input are reported also by Banerjee et al. [4] for computing shortest paths, Chaitanya et al. [23] for computing betweenness-centrality values, and Dutta et al. [9].

Our algorithms aim to address key questions on scalable estimation of the closeness centrality of nodes in a graph. We introduce four key optimizations to this end. Our first optimization targets identical nodes and removes such nodes from the computation. Two nodes are deemed identical if they have the same set of neighbors. The second optimization technique that we introduce targets redundant nodes. A node is said to be redundant if **no** shortest paths ever pass through that node, except as the end points of the path. Our next optimization targets chain nodes: nodes of degree two that lie in a maximal path. Computations with respect to these nodes can use information from computations at the end points of the chain in a post-processing step. Our final optimization involves using the decomposition of a graph into its biconnected components. It is know that biconnected components can aid in shortest path problems by providing good mechanisms to extend the shortest paths across pairs of vertices in in one biconnected component to shortest paths for pairs of vertices across distinct biconnected components.

We use the above techniques to estimate the farness-centrality values of nodes in a given graph in parallel. On top of these techniques, our algorithms use sampling based estimation

wherein breadth-first traversals from a set of sampled nodes are used to obtain the exact or an estimate to the shortest path distance between pairs of vertices. We notice that using our techniques allows us to obtain the exact distance on more pairs of vertices and also in a smaller time compared to a random sampling based approach.

A summary of our results is given below.

- We introduce four optimization techniques to address scalability challenges in estimating the farness-centrality values of nodes in a large real-world graph in parallel. Our techniques include using a biconnected components based decomposition (B), removing redundant nodes (R), removing identical nodes (I), removing chain nodes (C), and sampling (S). Using these techniques allows us to estimate the lengths of various shortest paths.
- Experimental results on a variety of graph classes indicates that our techniques result in much better estimates than sampling nodes randomly. Further, Figure 4(b) shows that 20% sample nodes are sufficient for our approach (BRICS) to give nearly better estimates and running time than a simple random sampling using 30% of the nodes.
- We also analyze on the applicability of each of the techniques that we introduce to various classes of graphs and their structural properties.

### A. Related Work

Using the structural properties of graphs to (re)design parallel algorithms for graphs is a research area that is catching rapid attention in recent years. This approach has been used by Hong et al. [16] to detect the strongly connected components of a directed real-world graph. Similar principles have been used in the works of [14], [27], [25], [13].

Another popular technique to address scalability challenges in parallel graph algorithms is use decompose the graph into smaller subgraph. The widely used graph decomposition technique Metis [18] decomposes a graph into a specified number $k$ of subgraphs such that the number of edges that cross any partition is minimized. This decomposition is can then be used parallel graph algorithms [11]. However, for path-based problems such as calculating or estimating farness-centrality, decomposition via Metis may not be ideal. The likely presence of cycles that go across the partitions induced by a Metis decomposition hamper efforts to find exact distances or estimates to shortest paths.

On the other hand, using a decomposition based on the biconnected components of a graph for path based problems is gaining recent attention. Some of the recent works that make use of such a decomposition include [10], [23]. Biconnected decomposition of graph is also used in incremental betweenness centrality [19].

Some of the recent works that are closest to spirit as the work presented in this paper is that of Garg and Kothapalli [13] and Sariyuce et al. [25]. In [13], Garg and Kothapalli propose optimizations based on identifying identical and redundant nodes, identifying chain nodes, and also use a decomposition based on the strongly connected components of a directed graph to improve the calculation of the pagerank of nodes in a given graph. Sariyuce et al. [25] show various optimizations that can help in speeding up various centrality computations in the sequential computing model.

There are various uses of random sampling method, some among them are estimating the closeness-centrality values for all nodes [12], computing top-$k$ closeness-centrality values [22], and finding the approximate 1-median (a node with maximum closeness centrality) in a network [26].

## II. Basic Approaches

In this section, we review basic ideas from Cohen et al. [6] for estimating the farness-centrality of the nodes in a graph. In the following, and the algorithms in the the rest of the paper, we use farness$(v)$ to denote the estimate to the farness-centrality of node $v$.

### A. Random Sampling Approach

A simple way to estimate the farness in a graph $G = (V, E)$ is by using a sampling based technique as shown in Algorithm 1 [6]. In this approach, we first identify a set $S$ of $k$ nodes chosen uniformly at random from $V$. For nodes $u, v$ of $G$, let $d(u, v)$ denote the shortest distance between $u$ to $v$. (Since we assume that $G$ is undirected, $d(u, v) = d(v, u)$.) Subsequently, we run BFS from each node in $S$ as the source and compute $d(v, w)$ for every $v \in S$ and $w \in V$. To reduce the memory complexity needed in the computation of algorithm 1, we can use an array farness, initialized farnessfor all $u \in V$ to 0. While running BFS from $v \in S$, add $d(v, u)$ to farness$[u]$ and return the sum of distances from start node to all other vertices. With this approach now we require only $O(n)$ space (instead of $O(nk)$) and the computation can be performed in $O(k(m+n))$ time. The farness for $v \in S$ is exactly computed, but for $u \in S \backslash V$, farness$[u]$ has the sum of distance of $k$ nodes instead of $n$ vertices.

**Algorithm 1:** Algorithm to implement Random Sampling

```
1 Function RandomSampling(G, k):
2     Choose s_i ∈ V(G) u.a.r such that
        S = {s_1, s_2, ...s_k}
3     for each vertex u ∈ S do in parallel
4         runBFS(u, G)
5         farness[u] = Σ_{v∈V} d(u, v)
6     for each vertex u ∈ V(G)/S do
7         farness[u] = Σ_{v∈S} d(u, v)
```

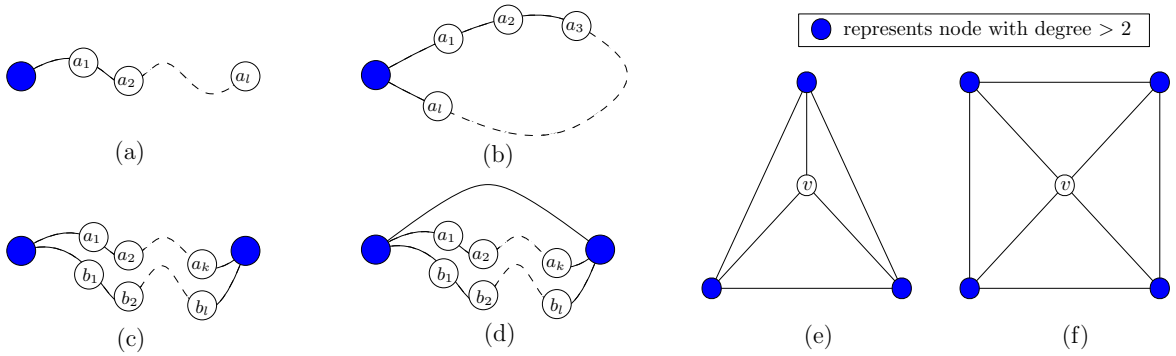



Fig. 1. Figures (a), (b), (c) and, (d) show redundant chains nodes, and figures (e), and (f) shows redundant 3D,4D nodes respectively.

A limitation of this approach however is that as the set of nodes in $S$ are chosen uniformly at random, these nodes do not offer any insights into how the shortest paths in the graph behave. So, estimates obtained out of using the nodes in $S$ tend to be error-prone.

## III. Our Approach

In this section we explain the algorithmic techniques which helps us address scalability and quality issues in estimating farness centrality. The **BRICS** framework stands for **B**iconnected components based decomposition, **R**edundant nodes, **I**dentical nodes, **C**hain nodes, and **S**ampling. Removal of identical nodes saves the computation required for nodes whose neighbour list is same. Chains and Redundant nodes reduce the graph size and a decomposition based on biconnected components decomposes the graph $G$ into smaller sub-graphs which are the bi-connected components of $G$. Finally, sampling within each biconnected component allows us to arrive at good estimates to the farness centrality of nodes.

In our approach, we first form a reduced graph from the input graph by preprocessing for redundant nodes, identical nodes, and chain nodes. The reduced graph thus obtained is decomposed into its biconnected components. From each bi-connected component $B_i$, we identify a set $S_i$ of $k_i$ nodes chosen uniformly at random among nodes in $B_i$. The set of smapled nodes $S$ is set to $\cup_i S_i$, and $k := \sum_i k_i$ is the total number of samples nodes. Subsequently, we run BFS from each node in $S_i$ as the source and compute $d(v, w)$ for every $v \in S_i$ and $w \in B_i$. Notice that in our approach the BFS from a sampled node $s \in S_i$ is limited to the bi-connected component $B_i$. Here $d[v]$ denotes the distance between vertex $v$ and the source of a BFS Tree, where source can be any node from Sampled Nodes. We briefly describe each of these techniques in the following sections.

### A. Identical Nodes

Two nodes $u, v \in V$ are identical nodes if there have the same set of neighbors. As $u$, $v$ has the same neighbor list the BFS tree constructed with either of $u$ or $v$ as the source node is identical. This implies that $u$ and $v$ will have the same farness value i.e farness[$u$] = farness[$v$] [24]. A group of identical nodes will all have the same farness value. Hence, for every group of identical nodes, we can consider only one of the nodes in the group as the *representative* node and remove the remaining nodes of the corresponding group from the graph. For every such node $u$ removed, we store its representative. By hashing the neighbour list of each node, we can find all the groups of identical nodes.

**Fact III.1.** *Two nodes $u$, $v \in V$ are identical, if both $u$ and $v$ have same parent in all BFS trees in $G$ with the source of BFS in $V - \{u, v\}$.*

Another set of identical nodes we consider corresponds to nodes in chains with identical end points. Two chains of nodes $c_1 = u, a_1 ..., a_k, v$ and $c_2 = u, b_1, ..., b_\ell, v$ are known as *identical Chains* if they have same endpoints and they are of equal length as shown in Fig. 1(c) where $k = \ell$ and nodes lie in those chains are known as identical chain nodes. Here farness[$a_i$] = farness[$b_i$] $\forall i \in [1, \ell]$ because any shortest path length from $x$ to $a_i$ is same as $x$ to $b_i$ where $x \in V \setminus \{c_1, c_2\}$

**Fact III.2.** *Let $i \subseteq IN$ where $IN$ is a set of nodes which are either Identical nodes or Identical chain nodes, then closeness centrality for all the nodes in $I$ will be same and they all lie in the same BiCC.*

### B. Chain Nodes

Our preliminary experiments shows that real world graphs have a significant number of nodes of degree one and two. (See also Table I). Such low degree nodes lead to formation of chains $C = (u, a_1, a_2, ..., a_\ell, v)$ where all the nodes between $u, v$ have degree two. There are 4-types of chains as shown in Fig. 1(a)–(d). We label each kind of chain as follows.

- Type-1: A chain with one of the end point as 1-degree node as shown in Fig. 1(a) which is a redundant chain.
- Type-2: A chain where the two end points are same which forms a cycle as shown in Fig. 1(b) is a redundant chain.
- Type-3: if $k > \ell$ as shown in Fig. 1(c), then $c_1$ is known as redundant chain and $c_2$ is non-redundant chain where $c_1 = u, a_1 ..., a_k, v$ and $c_2 = u, b_1, ..., b_\ell, v$. In this case for simplicity let us assume we have only two paths

between $u$ and $v$ that is via $c_1$ and $c_2$ if we want a shortest path from $u$ to $v$ we will always take the path via $c_2$ which means all the 2-degree nodes in $c_1$ could be removed safely from the graph $G$ and in Fig. 1(d) all the chains are redundant as there is a direct edge between two vertices.

- Type-4: All identical chains (explained above) are redundant leaving one identical chain as non-redundant.

It is clear that all redundant chains can be removed safely from the graph $G$ without making $G$ disconnected. However, since the above classification is not non-overlapping, we have to ensure that the type of chain is identified as only belonging to one of the four types. Whenever a chain is removed from input graph or the reduced graph, it has to be stored in some data-structure so as to set the farness values in post processing. To further optimize, as we know that all the shortest paths pass through $u$ in Type-1 chains, so contribution of all nodes can be calculated directly to increment farness$[w]$ by $(|c_i| - 1) \times (d[u] + |c_i|/2)$, where $c_i$ is the i$^{\text{th}}$ chain of Type-1, $|c_i|$ says number of nodes in chain $c_i$ and $d[u]$ is distance from node $u$ to the source vertex $w$. Similarly, farness of nodes in chains of Type-1 can be estimated using Fact III.4.

**Algorithm 2:** Get chain nodes contribution

**for** *each chain* $u \rightsquigarrow v \in C$ **do**
  // Let $u \rightsquigarrow v : u - a_1 - a_2 - ... - a_\ell - v$
  $a_0 = u$; $a_{\ell+1} = v$
  **if** $\deg v = 1$ **then**
    **for** $\forall a_i$ **do**
      $d[a_i] = d[a_{i-1}] + 1$ where $i \in [1, \ell + 1]$
  **else**
    **if** $|d[u] - d[v]| \geq \ell$ **then**
      **for** $\forall a_i$ **do**
        $d[a_i] = d[a_{i-1}] + 1$ where $i \in [1, \ell]$
    **else**
      **if** $d[u] < d[v]$ **then**
        $(u, v) = (v, u)$// swap $u$ and $v$
      $\ell_m = d[u] - d[v] + (\ell - d[u] + d[v])/2$
      **for** $\forall a_i$ **do**
        $d[a_i] = d[a_{i-1}] + 1$ where $i \in [1, \ell_m]$
      **for** $\forall a_i$ **do**
        $d[a_i] = d[a_{i+1}] + 1$ where $i \in [l, \ell_m]$

**Fact III.3.** *Let $c \in C$ be a chain with end points as $u$ and $v$ such that* $\deg(u) > 2$ *and* $\deg(v) = 1$ *then farness$[w]$ = farness$[u] + d(u, v) \times |S|$.*

**Fact III.4.** *Let $v$ be one degree node then farness[$v$] = farness[neighbour of $v$] + $|S|$.*

**Fact III.5.** *Let $c$ be a chain $\in C$ with end points as $u$ and $v$ then it is not necessary that both the vertices lie in same BiCC.*

### C. Redundant 3 and 4 degree nodes

While removing nodes of degree 1 and 2 seems to be simple in the context of centrality computations, see also [25], we proceed with such techniques by also considering cases where we can remove nodes of a higher degree too. In particular, we notice that real-world graphs have nodes of degree 3 and 4 that are also redundant. We say $RN$ is set of redundant nodes and a node $v \in RN$ of degree 3 is redundant if the three neighbours of $v$ are mutually neighbors of each other as shown in Fig. 1(e). In such a case, no shortest paths go through $v$ except for those that start or end at $v$. Similarly we say a node $v$ of degree 4 is redundant if every neighbour of $v$ is connected with at least 2 other neighbours of $v$ as shown in Fig. 1(f). In this case all the shortest paths go through $v$ can also go through neighbours of $v$ which means removing $v$ will not affect graph connectivity and doesn't disturb any shortest path length. Therefore, we can ignore $v$ in the computations of farness centrality.

**Algorithm 3:** Get redundant nodes contribution

**for** *each* $u \in RN$ **do**
  $d[u] = \min(d[v]\ \forall v \in \text{neighbour}[u]) + 1$

**Fact III.6.** *Let $u \in RN$ then $x \in BiCC_i\ \forall x \in neighbour(u)$*

**Fact III.7.** *Let $u \in RN$ then $u$ will not be a part of any shortest path if $u$ is neither source nor destination.*

Let $(v_1, v_2, v_3...v_k)$ be a set of redundant nodes then according to Facts III.6 and III.7, we can remove all the redundant nodes by storing their neighbours for post processing. Later on by using this information about neighbours we can estimate farness$[v_i]$ and contribution of $v_i$ in farness of sample nodes. Algorithm 3 explains how all the contribution of redundant nodes are taken.

**Algorithm 4:** Procedure for the Cumulative Technique

**Data:** Graph $G$ = ($V$, $E$) and $k$
**Result:** Estimated farness value of $v \in V$
/* find Identical Nodes */
$IN$ = getIdenticalNodes($G$)
$G_R$ = $G$ - $IN$
/* find 1,2-Degree Redundant Nodes */
$CN$ = getRedundantChains($G_R$)
$G_R$ = $G_R$ - $CN$
/* find 3,4-Degree Redundant Nodes */
$RN$ = getRedundantNodes($G_R$)
$G_R$ = $G_R$ - $RN$
/* Construct Block Cut-Vertex Tree */
$BCT$=constructBCT($G_R$)
farness = processBCT($BCT$, $IN$, $CN$, $RN$, $k$)

### D. Putting Together Everything

Starting from the input graph $G$, we apply the optimizations mentioned in the earlier sections to arrive at the reduced graph

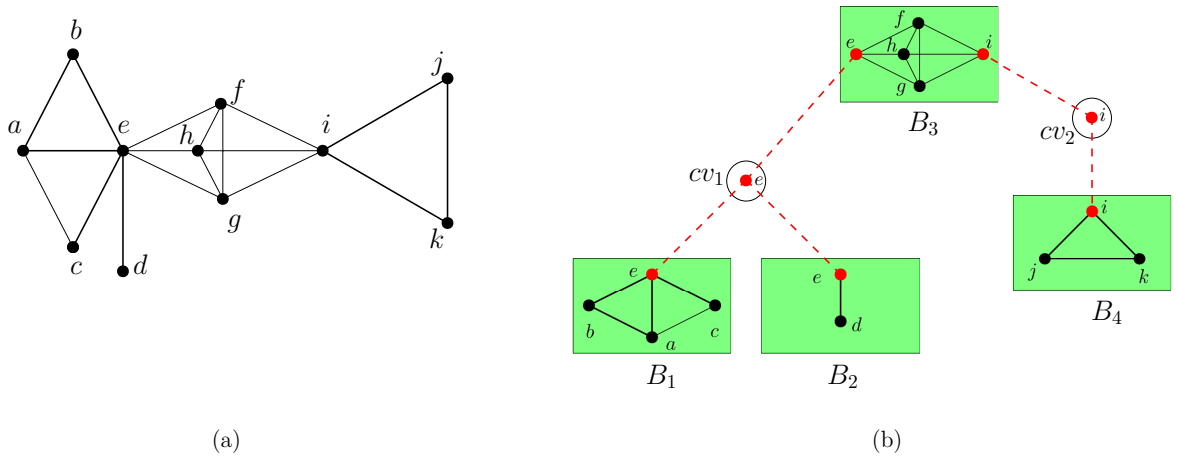



Fig. 2. Block Cut-Vertex Tree representation of Graph in (a) is shown in (b)

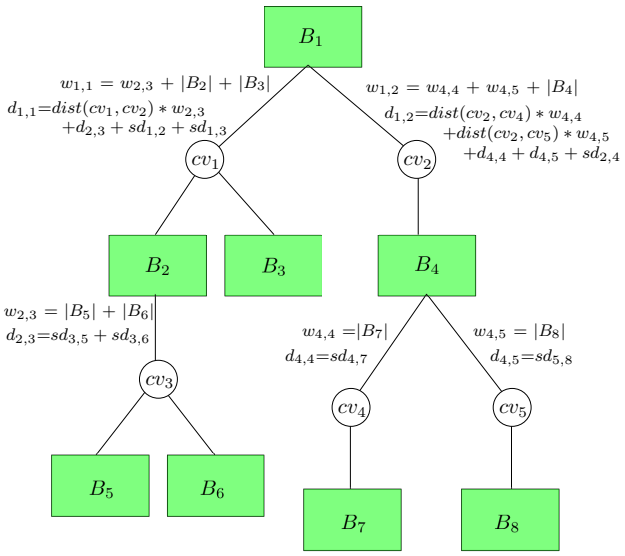



Fig. 3. Bottom up traversal $w_{x,y}, d_{x,y}$ are weight and dCarry where x,y are blockID and cutVetexID respectively. $sd_{i,j}$ is the sum of distances from rep. node $i$ to all nodes in block $j$, $dist(u, v)$ is the distance from vertex $u$ to vertex $v$ in Graph

$G_R$. We then apply the Bi-Connected Component decomposition on the reduced graph $G_R$ and construct the Block Cut-Vertex Tree (BCT)[15], where the nodes of the BCT are the bi-connected components of $G_R$ and cut-vertices of $G_R$. Two nodes $B_i$, $B_j$ in the BCT are connected with a common node called a cut point. A small illustration is given in Fig. 2. The two major benefits of BCT representation is that the shortest paths from nodes of $G_R$ in one BiCC to nodes in another BiCC go through some cut point which implies that BFS runs can be restricted to one block instead of a complete reduced graph $G_R$ and the complete contribution of one block node can be computed from the cut point.

Algorithm 4 explains the reduction techniques used for cumulative approach and the order in which they are used. As indicated, we apply the removal of identical nodes (*IN*) followed by the removal of chain nodes (*CN*), followed by the removal of redundant 3D and 4D nodes (*RN*), and then construction of BCT. Algorithm 5 explains how nodes are processed in the BCT. Here, each (BiCC) block has four types of nodes.

- *Cut Vertices* : Articulation points in Block Cut-Vertex Tree.
- *Sampled Nodes* : Non-Articulation points chosen for sampling.
- *Non-Sampled Nodes* : Non-Articulation points which are not sampled.
- *Removed Vertices* : Removed vertices in previous reduction techniques.

The four steps involved in implementation of processing nodes for estimating farness centrality in BCT are as follows

Step 1: All the removed vertices except few redundant chains lie in the same BiCC from Facts III.2, III.5 and III.6. We process removed vertices to the blocks they belongs in BCT leaving those chains with endpoints in different BiCCs.

Step 2: Now, every block has their own cut vertices, removed vertices and for getting sampled and non-sampeld nodes we need to choose $k\%$ of nodes in random from that block which includes cut vertices. Now, run BFS in parallel for each sample nodes in BiCCs.

Step 3: The contribution of distances to each node in one block from all nodes in every other block can be computed using a bottom-up and top-down traversal approach, explained in Algorithm. 6. Each block has at least one *representative node* which is an articulation point in the BCT. Each *representative node* in a block has a subtree in the BCT. Distance contribution to the nodes in a block come from the nodes in all its subtrees trough their respective *representative node*. Each *representative node* has two parameters namely *weight* and *dCarry*, where *weight* is the number of nodes in its subtree and *dCarry* is the sum of distances from all nodes in that subtree to the *representative*

**Algorithm 5:** Processing in Block Cut-Vertex Tree

```
1  Function processBCT(BCT, IN, CN, RN, k):
2      Step:1 Process removed vertices in G \ G_R among Blocks
3      processRemovedVertices(BCT, IN, CN, RN)
4      Step:2 Parallelise Block computations
5      for each block B_i ∈ B do in parallel
6          initialize(S_i, Φ)
7          for each cut vertex c_j ∈ B_i do
8              append(S_i, c_j)
9          k_i = [k * |B_i| / |G_R|] − |S_i|
10         append( S_i, {k_i vertices choosen u.a.r in B_i} )
11         Run BFS from s_j and store all the distances
12         for each sampled node s_j ∈ S_i do
13             BFS(B_i, s_j)
14     Step:3 Compute Contributions across Blocks
15     ComputeContributions(BCT) // Refer Algorithm 6
16     Step:4 Parallelise Computing farness values
17     for each block B_i ∈ B do in parallel
18         Compute farness using contributions of nodes ∈ B_i and ∉ B_i from Line 13,15 respectively
19         for each vertex v ∈ V(B_i) do
20             Compute the farness value of v
21         for each removed vertex r_j ∈ B_i do
22             Compute the farness value of r_j
```

**Algorithm 6:** Computing Contributions across Blocks

**Data:** BCT and sum of distances from cut vertices to blocks ($sd$)
**Result:** Arrays of weight and dCarry

```
1  Function ComputeContributions(BCT):
2      Initialize weight and dCarry arrays with 0's
       /* Bottom Up Traversal                          */
3      for each cut vertex cv ∈ reverse(BFSOrder) do
4          pb = parent[cv]
5          for each child block cb of cv do
6              weight[pb][cv] += |cb|
7              dCarry[pb][cv] += sd[cv][cb]
8              for each child rep. node rn of cb do
9                  weight[pb][cv] += weight[cb][rn]
10                 dCarry[pb][cv] += dCarry[cb][rn] +
                     (weight[cb][rn] * dist(cv,rn))
       /* Top Down Traversal                           */
11     for each cut vertex cv ∈ BFSOrder do
12         pb = parent[cv]
13         for each child block cb of cv do
14             weight[cb][cv] += |pb|
15             dCarry[cb][cv] += sd[cv][pb]
16             for each child rep. node rn of pb do
17                 weight[pb][cv] += weight[pb][rn]
18                 dCarry[pb][cv] += dCarry[pb][rn] +
                     (weight[pb][rn] * dist(cv,rn))
19             dCarry[cb][cv] += dCarry[pb][parent[pb]]
```

*node*.

To populate the contributions, we start with leaves in the BCT and traverse till the root (bottom-up), accumulating the distance contributions of all blocks it visits as shown in Fig. 3(a) by using sum of distances from *representative node* $cv_i$ to all nodes in block $B_j$ $sd_{i,j}$ through Step-2.

Once reach the root node in BCT, traverse from root to all the leaves (top-down), by using information from bottom-up traversal as shown in Fig. 3(b).

Step 4: Using the weight and dCarry of all *representative nodes* BiCC has farness values of all nodes in that BiCC are computed which was shown in Lines 17 - 22 of Algorithm 5.

## IV. Experimental Results and Analysis

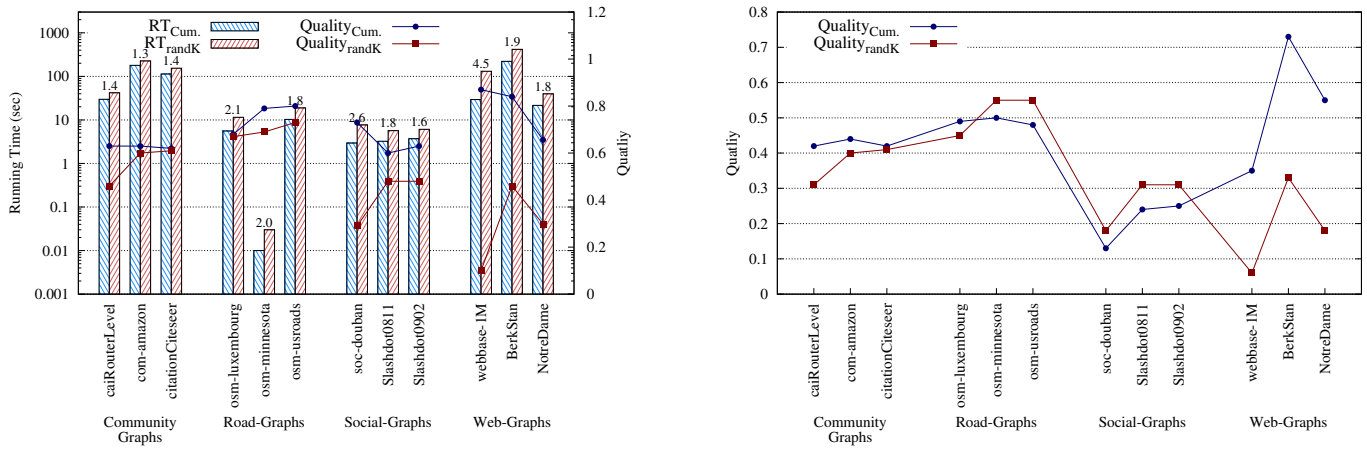
### A. Experimental Platform

We implement our algorithms in C++ using `g++` version 5.4.0 compiler with -O3 optimization level and -fopenmp flag with openmpi version 3.1, we use Xeon(R) CPU E5-2640 v4 processor with 128GB RAM as the experimental platform to test our results, this processor is from the Intel family, which is based on x86_64 architecture and it has 20 cores with each core running at 2.40 GHz and with active hyper threading it can support 40 logical threads. The memory hierarchy has 3 level cache system of Intel(R) Xeon(R) CPU E5-2640 v4, L1 cache size is 32 KB per core, L2 is 256 KB per core and a shared L3 is 25.6 MB.

### B. Dataset

We experiment with four different classes of real-world graphs namely web graphs, social graphs, community networks, and road networks. The important characteristics of these graphs are listed in Table I. In Table I the column BiCC indicates the information regarding bi-connected components and $\#, Max, Avg$ says Number of bi-connected components, Number of nodes in largest bi-connected component, Average number of nodes in a bi-connected component of the corresponding graph respectively, In column Identical, Nodes and Ch.Nodes indicates number of identical nodes and identical chain nodes respectively, columns of *Redundant* and *Chain* Nodes indicates number of redundant nodes of degree 3 and 4, and the number of nodes in chainss in the corresponding graph respectively.

Each graph is made simple undirected, unweighted, and



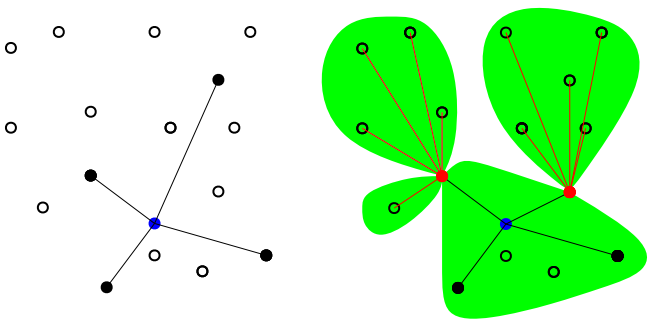

(a) (b)

Fig. 4. Performance of Graphs on (a) 40% Sampling Rate for both approaches, (b) 20%,30% Sampling Rates for cumulative and random sampling approach respectively.

connected by removing self loops, multiple edges. Directed edges are converted to undirected edges and if the graph is disconnected, we added few edges to make it connected. All the graphs are from the dataset of University of Florida Sparse Matrix collection [2] and SNAP database[1].

### C. Results

We first briefly describe speedup and quality followed by analysis of graph classes in which we explain how our optimization is working for each graph class.



(a) (b)

Fig. 5. (a) Random Sampling and (b) BiCC Sampling Approaches

*1) Speedup and Quality:* We are interested in finding the time taken by our approaches along with approximation ratio which says how our estimated values are closer to actual values. The approximation ratio $AR(v)$ of node $v$ is calculated as the ratio between its farness_actual$(v)$ and farness_estimated$(v)$ values defined as follows:

$$AR(v) = \frac{\text{farness\_estimated}(v)}{\text{farness\_actual}(v)}$$

where farness_actual$(v)$ and farness_estimated$(v)$ are the actual and estimated farness values of a node $v$ respectively. The average approximate ratio ($AR$) which is termed as Quality and calculated as follows

$$\text{Quality} = \frac{\sum_{v \in V} AR(v)}{n}$$

Speedup is the ratio of time taken by random sampling to that of our algorithms, all the algorithm are run with 40 threads on the machine as mentioned in Section IV-A.

Fig. 4 is generated by taking 20% sample nodes for Cumulative and 30% sample nodes for random sampling, which is comparison of Random sampling vs Cumulative method across graph classes for quality and speedup. In Fig. 4 the value on top of histogram is speed up. On the X-axis the graphs are arranged according to the classes as listed in Table I. On an average Cumulative provides better quality than random sampling for all graph classes, when it comes to speedup on average we get **2.73** for web graphs, **2.0x** for social networks, **1.36x** for community graphs and **1.96x** for road networks. This speed up is observed by taking 40% of sample nodes from reduce graph $G$. We expect speedup will increase as we increase the number of sample nodes.

*2) Analysis of Graph Classes:* To study how speedup and quality is affected by our techniques on various graph classes, we configured our algorithm in 3 different ways, C+R says we reduce graph by applying chain reduction and then followed by removal of redundant 3 and 4 degree nodes, I+C+R says reduction of graph by removing identical nodes followed by C+R and Cumulative approach which includes partitioning of reduced graph into its Bi-connected components. We observe that quality is affected only when we partition graph using its BiCC's because of it's nature that the shortest paths from nodes of $G$ in one BiCC to nodes in other BiCCs go through some cut point so the contribution of one BiCC could be taken via it's cut point as shown in Fig. 5 which result in better quality, for rest of the techniques, quality remains unaffected. Coming to speedup, we study in detail below.

| Graph name | $\|V\|$ | $\|E\|$ | Identical | | Redundant | Chain | BiCC | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Nodes | Ch.Nodes | Nodes | Nodes | # | Max | Avg |
| **Web Graphs** | | | | | | | | | |
| web-NotreDame | 325728 | 1082486 | 202K | 22K | 6K | 200K | 168K | 134K | 3 |
| web-BerkStan | 685230 | 6650145 | 228K | 22K | 35K | 106K | 63K | 490K | 12 |
| webbase-1M | 1000005 | 2108301 | 874K | 45K | 12.1K | 875K | 53K | 342K | 9 |
| **Social Graphs** | | | | | | | | | |
| soc-Slashdot081106 | 77360 | 469180 | 15K | 294 | 284 | 40K | 30K | 47K | 4 |
| soc-Slashdot090216 | 82168 | 504230 | 18K | 1.2K | 328 | 41K | 30K | 52K | 4 |
| soc-douban | 131580 | 828255 | 96K | 581 | 90 | 125K | 103K | 51K | 3 |
| **Community Networks** | | | | | | | | | |
| caidaRouterLevel | 192244 | 609373 | 48K | 12K | 3K | 5K | 45K | 132K | 5 |
| com-citationCiteseer | 268495 | 1156647 | 18K | 2K | 3K | 72K | 45K | 220K | 7 |
| com-amazon | 334863 | 925872 | 28K | 3.5K | 25K | 66K | 36K | 281K | 10 |
| **Road Networks** | | | | | | | | | |
| osm-minnesota | 2642 | 3304 | 8 | 4 | 0 | 1534 | 143 | 2.4K | 20 |
| osm-luxembourg | 114599 | 119666 | 257 | 119 | 0 | 101K | 24K | 88K | 6 |
| usroads | 29164 | 284142 | 9 | 2 | 6 | 2186 | 173 | 126K | 747 |

TABLE I
LIST OF GRAPHS WE USED IN OUR EXPERIMENTS

*a) Web Graphs:* We experiment with three web graphs. We observe that on average web-graphs have a significant number of BiCC's. However, the BiCCs have sizes that have a large tail – a large number of BiCCs have a very small size. This results in an overhead in processing these BiCCs during Algorithm 6.

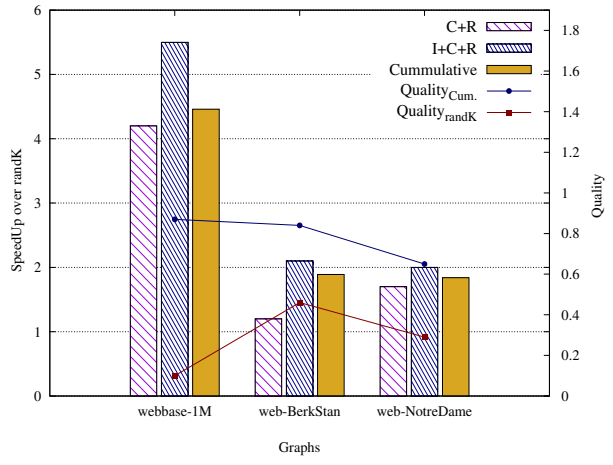



Fig. 6. Relative Speedup of optimizations on Web graphs

On average these graphs have 44% of identical nodes (I), 54% of nodes of degree 1 and 2 (C) and 2.4% of redundant nodes (R). We expect that all the optimizations, except for using the BiCC based decomposition, will improve the performance. The impact of each of the optimizations on web-graphs is shown in Figure 6. As can be seen applying the BiCC based decomposition results in a small decrease in the overall speedup compared to not using the BiCC based decomposition.

*b) Social Networks:* We experiment on three graphs in the social network class. These of graphs have a large number of nodes with degree one and two and do not have a large number of nodes that are 3 or 4 degree redundant on average. So we avoid redundant nodes removal optimization. Identical nodes are 38% on average so we apply this optimization. As it has good number of BiCC components we apply the BCC based optimization.

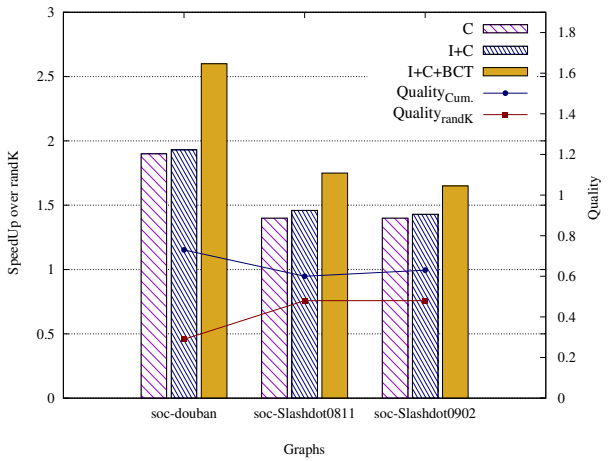



Fig. 7. Relative Speedup of optimizations on Social graphs

After applying I+C optimization, the number of bi-connected components reduces to hundreds on an average and largest component has 72% of nodes of the reduced graphs which implies that the distribution of nodes in BiCC's are skewed. As a result performance is effected but better quality than random sampling is observed, as shown in Fig. 7.

*c) Community Networks:* In the class of community networks, we consider three graphs. These graphs have a good number of bi-connected components but on average, one of the bi-connected component covers 80% of nodes which does not benefit the speedup yet helps in getting slightly better quality than random sampling. These graphs have a moderate number of identical nodes, redundant nodes, and nodes of degree one and two.

So we can apply all the optimization methods I+C+R to obtain a reduced graph on collaboration network followed by BiCCs.. As a result the number of BiCC's come down to of 4K but the largest component covers 78% of nodes in the reduced graph on average. Figure 8 shows the relative speedup and quality achieved by our techniques on community graphs.

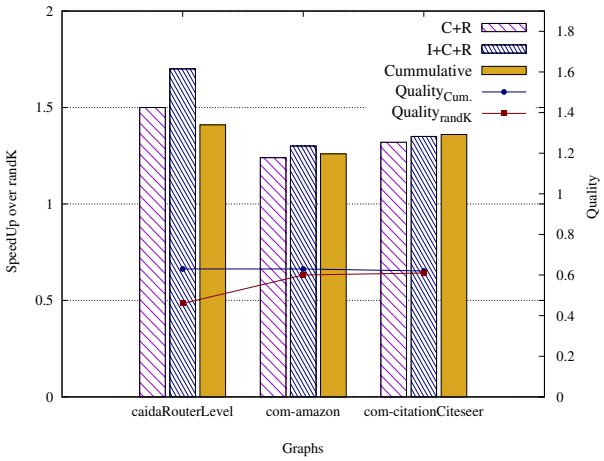



Fig. 8. Relative Speedup of optimizations on Community networks

*d) Road Networks:* Now we move to our final graph class road networks. These networks by nature have 70-85% of nodes with degree one and two. Therefore, when we apply the chain optimization, we get a good speedup. Further, these networks has few BiCC's and the largest BiCC covers in excess of 90% of nodes on average. For this reason, applying the BiCC based technique slows down the estimation and does not guarantee higher quality. These graphs have small number of identical and redundant nodes thereby implying that the advantage gained by identifying them is offset by the preprocessing time.

So we apply only the chain optimization on road networks and BiCC but the distribution of nodes across components is skewed because of which much improvement in quality is not observed but a decent speedup is achieved. Fig. 9 shows the impact of the optimization techniques applied on road networks.

## V. Conclusion

In this paper, we studied the gaps present in basic methods for estimating farness and proposed the BRICS framework w to provide better estimates of farness than the basic methods not only in quality but also in speedup. Our methods take the advantage of structures found in real-time graphs in reducing the processing time. Extension of this problem to dynamic setting is an interesting study.

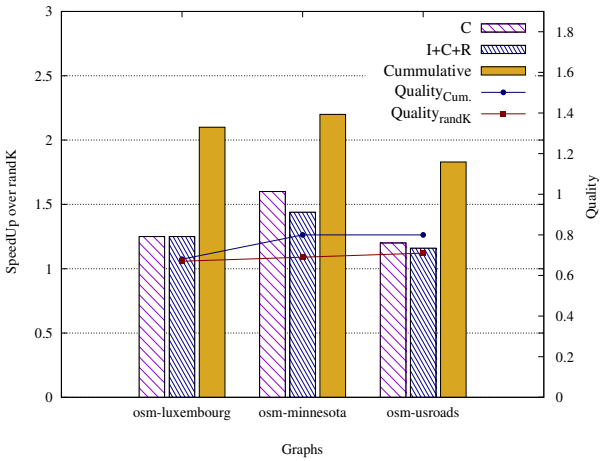



Fig. 9. Relative Speedup of optimizations on Road networks

## References

[1] Stanford Large Network Dataset Collection. https://snap.stanford.edu/data/.

[2] The University of Florida Sparse Matrix Collection. https://www.cise.ufl.edu/research/sparse/matrices/.

[3] A. Bavelas.: A mathematical model for small group structures. In Human Organization. 7:16–30(1948)

[4] D. S. Banerjee, A. Kumar, M. Chaitanya, S. Sharma, and K. K. Work efficient parallel algorithms for large graph exploration on emerging heterogeneous architectures. JPDC, 76:81–93, 2015.

[5] U. Brandes, C. Pich.: Centrality estimation in large networks. International Journal of Bifurcation and Chaos 17 (7), 2303–2318.

[6] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck.: Computing classic closeness centrality, at scale. In Proceedings of the 2nd ACM conference on Online social networks (COSN). 37–50(2014)

[7] P. Crescenzi, R. Grossi, M. Habib, L. Lanzi and A. Marino.: On computing the diameter of real-world undirected graphs. In Theoretical Computer Science 84–95(2013)

[8] P. Crescenzi, G. D'angelo, L. Severini, and Y. Velaj.: Greedily improving our own closeness centrality in a network. ACM Trans. Knowl. Discov. Data, 11(1):9:1–9:32, 2016

[9] D. Dutta, K. Kothapalli, G. Ramakrishna, R. S. Charan and T. S. Harsh.: An Efficient Ear Decomposition Algorithm. In 15th CTW on Graphs and Combinatorial Optimization, 65-68(2017)

[10] D. Dutta, M. Chaitanya, K. Kothapalli, D. Bera.: Applications of Ear Decomposition to Efficient Heterogeneous Algorithms for Shortest Path/Cycle Problems. Vol 8, no. 1, pp. 73-92, IJNC - 2018

[11] Hristo Djidjev, Guillaume Chapuis, Rumen Andonov , Sunil Thulasidasan and ominique Lavenier. All-Pairs Shortest Path algorithms for planar graph for GPU-accelerated clusters, J. Parallel Distrib. Comput., volume 85, pages:91–103, 2015.

[12] D. Eppstein and J. Wang.: Fast approximation of centrality. In SODA, pp. 228–229, 2001.

[13] P. Garg, K. Kothapalli. STIC-D: Algorithmic techniques for efficient parallel PageRank computation on Real-World Graphs. 17th International Conference on Distributed Computing and Networking (ICDCN - 2016)

[14] A. Gharaibeh, L. B. Costa, E. Santos-Neto, and M. Ripeanu. On graphs, GPUs, and blind dating: A workload to processor matchmaking quest. In Proc. of IEEE IPDPS (2013).

[15] Harary, Frank (1969), Graph Theory, Addison-Wesley, p. 36.

[16] S. Hong, N. C. Rodia, and K. Olukotun: On Fast Parallel Detection of Strongly Connected Components (SCC) in Small-World Graphs, in Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2013, p. 92.

[17] P. Indyk.: Sublinear time algorithms for metric space problems. In STOC. ACM, 1999

[18] G. Karypis, V. Kumar.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing. 20 (1): 359. (1999)

[19] Meghana Nasre, Matteo Pontecorvi, and Vijaya Ramachandran.: Betweenness centrality - incremental and faster. CoRR, abs/1311.2147, 2013.

[20] E. Nathan and D. A. Bader.: A Dynamic Algorithm for Updating Katz Centrality in Graphs. In ASONAM (2017)

[21] Newman, M.E.J. 2010. Networks: An Introduction. Oxford, UK: Oxford University Press.

[22] K. Okamoto, W. Chen, and X. Li.: Ranking of closeness centrality for large-scale social networks. In Proc. 2nd Annual International Workshop on Frontiers in Algorithmics, FAW. Springer-Verlag, 2008.

[23] C. Pachorkar, M. Chaitanya, K. Kothapalli and D. Bera.: Efficient Parallel Ear Decomposition of Graphs with Application to Betweenness-Centrality. In 23rd International Conference on High Performance Computing (HiPC) (2016)

[24] A. E. Sarıyüce, K. Kaya, E. Saule, and Ü. V. Çatalyürek.: Incremental algorithms for closeness centrality. In IEEE International Conference on BigData, 2013.

[25] Ahmet Erdem Sariyuce, Kamer Kaya, Erik Saule, Ümit V. Çatalyürek: Graph Manipulations for Fast Centrality Computation. TKDD 11(3): 26:1-26:25 (2017)

[26] M. Thorup.: Quick k-median, k-center, and facility location for sparse graphs. In ICALP . Springer-Verlag, 2001.

[27] X. Yang, S. Parthasarathy, and P. Sadayappan. Fast Sparse Matrix-Vector Multiplication on GPUs: Implications for Graph Mining, in Proc. VLDB Endowment, 4(4), pp: 231–242, 2011.