Incremental Distributed Algorithms for Game-theoretic Betweenness Centralities in Dynamic Graphs

Yefei Wang Qiang-Sheng Hua Wenjie Gao Hai Jin

National Engineering Research Center for Big Data Technology and System, Service Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract. Maintaining game-theoretic betweenness centralities in highly dynamic networks is challenging due to the high computational cost of recalculating it from scratch. This paper presents distributed incremental algorithms in the classic CONGEST model for maintaining Shapley- and semi-value-based betweenness centralities. By addressing the challenges of parallel traversal congestion and communication overhead, we propose incremental algorithms with round complexities of $O(D^G(\mathcal{A}_\mathcal{B}^{max}+\mathcal{D}_\mathcal{B}^{max})+|F_{Batch}|+|\text{Batch}|)$ for multi-edge updates. Here, D^G, $\mathcal{A}_\mathcal{B}^{\max}$ and $\mathcal{D}_\mathcal{B}^{\max}$ denote the diameter of the graph, the maximum number of articulation points, and the maximum diameter of the biconnected components, respectively. $|F_{\text{Batch}}|$ and $|\text{Batch}|$ represent the number of affected vertices resulting from insertions and the number of inserted edges, respectively. Experimental results demonstrate that the proposed multi-edge incremental algorithm achieves speedup factors of up to 7× and 16× compared to the single-edge incremental algorithm and the static algorithm, respectively.

Keywords: Distributed algorithms · Dynamic graphs · Network centrality

1 INTRODUCTION

Game-theoretic betweenness centralities[1] rooted in cooperative game theory—such as Shapley-value and semi-value based measures—model node interactions by capturing cooperative behaviors. Compared with ordinary betweenness centrality, they more accurately identify critical nodes under multi-node failure scenarios, thereby optimizing network stability and resource allocation.

To illustrate the advantage of game-theoretic betweenness centralities, we present a toy example depicted in Figure 1. Every edge is labeled with its direction and weight. Since a vertex's ordinary betweenness centrality can be decomposed into dependencies contributed by different source–destination pairs, we

This project is funded in part by the National Science and Technology Major Project (Grant No. 2022ZD0115301).

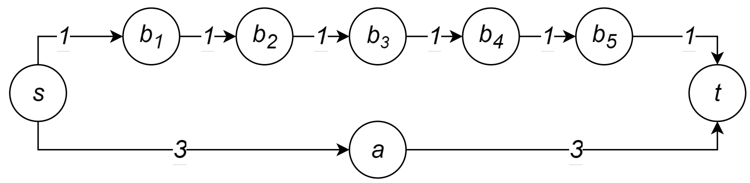

Fig. 1. Toy example

focus, for simplicity, on the single pair (s, t). In this setting, vertex a and each vertex b_i obtain an ordinary betweenness centrality of $1/2$. Yet it is evident that their importance for sustaining communication between s and t is not equal: if multiple vertices b_i fail, s and t can still communicate as long as a remains operational; however, if a fails, the failure of any single b_i will disconnect s and t. Thus, in scenarios where multiple nodes may fail simultaneously, ordinary betweenness centrality fails to distinguish their varying importance. Game-theoretic betweenness centralities address this limitation by drawing on cooperative game theory: the group betweenness centrality is used as the value function for different coalitions, providing a principled way to assess vertex importance under multi-node failures. Specifically, the semi-value-based betweenness centrality of vertex v is defined as follows:

$$bc_{SM}(v) = \sum_{U \subset V \setminus \{v\}} p_{|U|} \left(bc(U \cup \{v\}) - bc(U) \right)$$

where $p_{|U|}$ denotes the weight assigned to all subsets of size $|U|$ and satisfies $\sum_{0 \le k \le |V|-1} p_k \binom{|V|-1}{k} = 1$, with $|V|$ being the number of vertices in the graph, and $bc(U)$ is the group betweenness centrality of subset U. Intuitively, $bc(U \cup \{v\}) - bc(U)$ measures the marginal contribution of vertex v to the betweenness centrality of coalition $U \cup \{v\}$. By taking a weighted sum of these marginal contributions over all coalitions, the semi-value-based betweenness centrality provides a principled basis for prioritizing node protection in multi-node failure scenarios.

On the other hand, real-world networks are inherently dynamic, with relationships between nodes continuously evolving as nodes are added or removed. As a result, recalculating game-theoretic betweenness centralities from scratch becomes computationally expensive. For unweighted graphs $G = \{V, E\}$, the complexity of computing semi-value-based betweenness centrality is $O(|V|^4)$ [2]. Although distributed algorithms can calculate this in $O(|V|)$ rounds [3], this remains impractical for large-scale networks. Thus, efficient incremental algorithms are crucial for updating game-theoretic betweenness centralities in real time.

Addressing this challenge, the goal of this paper is to design efficient incremental algorithms within the CONGEST model, a classical distributed communication model where each edge transmits a message of size $O(\log n)$ bits per round. Table 1 summarizes the computational complexities of the Shapley value-

Table 1. A summary of the complexities of SPBC and SMBC

	Centralized	Distributed (round complexity)		
Centrality	Static	Static	Single-Edge Insertion(SEI)	Multi-Edge Insertion(MEI)
SPBC	$O(\lvert V\rvert^3)$ [2]	$O(\lvert V\rvert)$ [3]	$O(D^G(\mathcal{A}_{\mathcal{B}}^{max}+\mathcal{D}_{\mathcal{B}}^{max})$ $+\lvert F_{Batch}\rvert)$ **(this article)**	$O(D^G(\mathcal{A}_{\mathcal{B}}^{max}+\mathcal{D}_{\mathcal{B}}^{max})$ $+\lvert F_{Batch}\rvert+\lvert\text{Batch}\rvert)$ **(this article)**
SMBC	$O(\lvert V\rvert^4)$ [2]			

based (SPBC) and semi-value-based betweenness centrality (SMBC) algorithms for undirected, unweighted graphs.

2 PRELIMINARIES

In this section, we introduce the relevant notations, the system model, and the problem definition.

We consider an undirected, unweighted graph $G(V^G, E^G)$, where V^G and E^G represent the vertex and edge sets, respectively. The graph's diameter is denoted as D^G. G can be decomposed into biconnected components, each connected by articulation points. For a biconnected component B, G_a^B denotes the subgraph of G that is connected through articulation point a in B. When the context clearly identifies the biconnected component, the superscript is omitted.

For brevity, we omit the definitions of betweenness centrality and group betweenness centrality. In the various versions of betweenness centrality, the *dependency* of s on v is defined as the sum of v's betweenness centrality where s is either the source or the destination. This is denoted by $\delta_s[v]$, and it reflects the degree to which s depends on v. Similarly, $\delta_G[v]$ denotes the dependency of vertex v within graph G.

Shapley value is a fundamental method for fairly allocating values in co-operative games. However, it may not fully capture complex relationships or constraints in some practical scenarios, which led to the semi-value.

Definition 1 (Semi-value). *Given a game (A, v), where A is the player set and v is the characteristic function, the semi-value of player $i \in A$ is given by*

$$semi-value(i) = \sum_{S \subset A \setminus \{i\}} p_{|S|}(v[S \cup \{i\}] - v[S])$$

where $p_{|S|}$ denotes the weight assigned to a coalition of size $|S|$. When the weight function is set to $\frac{1}{|A|\binom{|A|-1}{|S|}}$, all subsets are assigned equal weights, and the semi-value becomes the Shapley value.

Assuming that each vertex in the graph represents a player and the set of vertices corresponds to the set of players, we then can derive the semi-value-based betweenness centrality by treating the group betweenness centrality as the characteristic function defined on subsets of players.

Definition 2 (Semi-value-based betweenness centrality). *For a vertex $v \in V^G$, the semi-value-based betweenness centrality $bc_{SM}[v]$ is defined as*

$$bc_{SM}[v] = \sum_{U \subset V \setminus \{v\}} p_{|U|}(bc[U \cup \{v\}] - bc[U])$$

Here, $bc[U]$ is the group betweenness centrality of U. Formulas for Shapley value-based betweenness centrality can be derived, but are omitted due to space constraints.

Directly computing $bc_{SM}[v]$ is infeasible due to the exponential time complexity of calculating betweenness centrality for all subsets U. To address this, Szczepanski et al. [2] proposed an alternative method based on analyzing the marginal contributions of vertex v, which enables computing $bc_{SM}[v]$ and $bc_{SP}[v]$ in polynomial time.

$$bc_{SM}[v] = \sum_{k \in [0,n-1]}^{k} P_k(\sum_{\substack{s,t \in V \setminus \{v\} \\ n-d_s[t] \leq k}}^{s,t} \frac{\sigma_{st}[v]}{\sigma_{st}} f_{SM}(d_s[t], k) + \sum_{\substack{s \in V \setminus \{v\} \\ n-d_s[v] \leq k}}^{s} g_{SM}(d_s[v], k))$$

$$bc_{SP}[v] = \sum_{s,t \in V \setminus \{v\}}^{s,t} \frac{\sigma_{st}[v]}{\sigma_{st}} f_{SP}(d_s[t]) + \sum_{s \in V \setminus \{v\}}^{s} g_{SP}(d_s[v])$$

where $f_{SM}(d,k) = \frac{(n-d)!(n-k-1)!}{(n-d-k)!(n-1)!}$, $g_{SM}(d,k) = f_{SM}(d,k) + \frac{k-n+1}{n-1}$, $f_{SP}(d) = \frac{1}{d}$, $g_{SP}(d) = \frac{2-d}{2d}$. We work in the CONGEST model: each round, every node can send an $O(logn)$-bit message to each neighbor.

Table 2 lists the notations used in the subsequent pseudocode and descriptions of this paper.

Table 2. NOTATIONS

Notation	Definition
D^G	The diameter of G
$\mathcal{B}^G$	The set of biconnected components of G
G_i^B	The subgraph connected through articulation point a in B
A^G	The set of articulation points of G
$d_s[v]$	The distance from s to v
$\delta_s[v] \backslash \delta_G[v]$	The dependency of $s \backslash G$ on v
σ_{st}	The number of the shortest paths from s to t
$\sigma_{st}[v]$	The number of the shortest paths from s to t passing through v
$f_{SM}(h,k), g_{SM}(h,k)$	Auxiliary functions of SMBC
$f_{SP}(h), g_{SP}(h)$	Auxiliary functions of SPBC

3 RELATED WORK

Cooperative game theory has been applied to network vertex centrality, starting with [1]. Szczepanski et al. [4] defined Shapley value-based betweenness centrality and expanded it to weighted and unweighted graphs with a polynomial-time algorithm [2]. Tarkowski et al. [5] further extended these measures to other game-theoretic centralities, like the Banzhaf index. Wang et al.[3] proposed static algorithms on the CONGEST model for Shapley-based betweenness centrality and semi-value-based betweenness centrality, with a round complexity of $O(n)$. They also proved the lower bound of the round complexity for calculating SPBC and SMBC in the CONGEST model, demonstrating that their algorithm is near-optimal.

Although algorithms for game-theoretic betweenness centralities in dynamic graphs are not yet fully developed, dynamic algorithms for betweenness centrality have been extensively studied. Jamour et al. [6] introduced the iCentral algorithm to update betweenness centrality after single-edge changes. The algorithm identifies affected vertices, computes their dependencies, and updates centrality by adjusting these dependencies. The key idea is to partition the betweenness centrality change based on whether the source and destination vertices are in the same biconnected component as v, and handle each component separately. iCENTRAL is restricted to undirected graphs, whereas Pons et al. [7] extended it to directed graphs. Shukla et al. [8] further generalized the iCENTRAL to handle multiple edge updates, introducing redundant nodes and redundant chains to reduce the number of nodes that must be traversed.

4 CHALLENGES AND MITIGATION STRATEGIES

The single-edge incremental algorithm in this paper is based on the iCentral algorithm [6]. However, the iCentral algorithm involves a large number of parallel tasks, which can lead to congestion when these tasks are executed simultaneously in the CONGEST model.

Moreover, compared to the ordinary betweenness centrality, the game-theoretic betweenness centralities require the distances between a specified vertex and other vertices. However, the iCentral algorithm does not maintain this information.

Both the iCentral algorithm and the multi-edge version [8] rely on biconnected components, which must be updated whenever the graph undergoes changes. In centralized environments, these components can be recalculated from scratch without significantly affecting the overall time complexity of centrality maintenance. However, in the CONGEST model, this recalculation approach requires a linear number of rounds, making it prohibitively expensive for centrality maintenance, especially when dealing with multi-edge insertions, where obtaining updated biconnected components is non-trivial. Therefore, a biconnected component maintenance algorithm with low-round complexity is crucial for efficient operation within the CONGEST model.

To address the three aforementioned challenges—congestion arising from parallel tasks, the absence of distance information, and the high cost of updating biconnected components—we introduce the following solutions. First, congestion is mitigated through a scheduling mechanism that serializes otherwise conflicting communications. Second, a distance-list procedure maintains, for every vertex, its up-to-date shortest-path distances to all others, thereby guaranteeing the correctness of game-theoretic betweenness centrality. Third, a low-round-complexity routine for the maintenance of biconnected components—adapted from the shared-memory algorithm of Haryan et al. [9]—ensures efficient graph-structural updates, particularly under multi-edge insertions, while curbing overall round complexity.

5 SINGLE-EDGE INCREMENTAL ALGORITHM

This section presents `SEI-SPBC` and `SEI-SMBC` for incrementally updating game-theoretic betweenness centralities for single-edge insertions.

Algorithm 1 `SEI-SPBC`(G, e)

Input: Graph $G(V,E)$ each vertex v in graph G knows $bc^{G}_{SP}[v]$ and new edge $e =< p, q >$
Output: each vertex v in the new graph G' knows $bc^{G'}_{SP}[v]$ after inserting e
$G' \leftarrow G \cup \{e\}$
$\mathcal{B}^{G'} \leftarrow$ `SEI-UB`(G')
$B_e \leftarrow B^{G'}_e/e$
$\{DL_a\} \leftarrow$ `Distancelist`(G') for $a \in A_{G'}$
Perform `BFS`(p) and `BFS`(q)
Mark s as an affected vertex and $S \leftarrow S \cup \{s\}$ if $d_p(s) \neq d_q(s)$ for $s \in G$
`BrandesSPBC`$(G, B_e, S, +)$
`BrandesSPBC`$(G', B^{G'}_e, S, -)$
return $bc^{G'}_{SP}[v]$

`SEI-SPBC` is presented in Algorithm 1. Before the update, each vertex must be aware of its old SPBC, $|V|$, f_{SP} and g_{SP}, and which biconnected component it belongs to. The biconnected components can be obtained through a Depth-First Search. Upon the insertion of edge e into the original graph G, resulting in the updated graph G', the maintenance of the biconnected components in G' is achieved through the `SEI-UB`. To capture the distance information, we define a data structure: for each component, each articulation point a maintains a dictionary DL_a, where v denotes the number of vertices in G^B_a at distance k from a. This structure facilitates the computation of f_{SP} and g_{SP}.

Before introducing Algorithm 2, we first clarify the definition of the BCT (Block-Cutpoint Tree) and other related notations. In the BCT of graph G, each articulation point and the biconnected component in G correspond to a

Algorithm 2 `Distancelist(G)`

1: **Input:** Graph $G(V, E)$
2: **Output:** DL_a: Dictionary of distances for articulation point a
3: Exchange distance lists among articulation points within each connected component
4: Aggregate distance lists along the block-cutpoint tree [11] at the root component
5: Exchange distance lists among articulation points within the root component
6: Send aggregated distance lists back to other components along the block-cutpoint tree
    return DL_a

node, with the node corresponding to an articulation point being connected to the nodes corresponding to the biconnected components it affects. In this paper, for each BCT, we randomly select one biconnected component as the root biconnected component, denoted as B_{root}. For a biconnected component B, if there exists an articulation point of B on the reachable path from B to B_{root}, this articulation point is defined as the primary articulation point of B. In a BCT T, for a node a_{bct} corresponding to an articulation point a in G and a node B_{bct} corresponding to a biconnected component B, we use $T(a_{\text{bct}})$ and $T(B_{\text{bct}})$ to denote the subtrees of T rooted at a_{bct} and B_{bct}, respectively. Additionally, $\overline{T}(a_{\text{bct}}) = T \setminus T(a_{\text{bct}})$ represents the part of T excluding $T(a_{\text{bct}})$, and the definition of $\overline{T}(B_{\text{bct}})$ is similar.

The `Distancelist` procedure computes DL_a for each articulation point (Algorithm 2). The algorithm initially performs a bottom-up traversal along T to aggregate distance lists. During this process, each primary articulation point a collects the distance list of its subtree $T(a_{\text{bct}})$ and transmits it upward for further iteration. Specifically, for leaf biconnected components, i.e., those with only one articulation point other than B_{root}, a BFS is executed within the component, rooted at the unique articulation point, to collect the distance list of the component. Note that this articulation point is also its primary articulation point. For non-leaf biconnected components, after the distance lists of the subtrees of all articulation points other than the primary one have been collected, a similar BFS is performed, rooted at the primary articulation point, and then the distance lists of the other articulation points are collected along the BFS tree. It should be noted that the distance lists can be merged when collecting information from other articulation points, so the information collection can be sent in a pipelined manner without causing congestion. At the end of this stage, all articulation points a will have knowledge of the distance lists of all vertices in their subtrees, i.e., how many vertices in the subtree are at a distance k from a.

Subsequently, within each B, the distance lists collected in the previous stage are exchanged, including B_{root}. Specifically, all articulation points broadcast their distance lists to other articulation points. Distance messages are sent in batches, with each batch sending path messages of distance k, where $k \in [1, D^G]$. Since this process is executed within B, all operations can be performed in parallel. At this point, for B_{root}, all articulation points have collected the complete distance lists of T. For the remaining B, all articulation points still need to obtain the distance lists in $\overline{T}(B_{\text{bct}})$.

Finally, a top-down traversal is performed along T. Specifically, the articulation point a of B_{root} has obtained the distance list of $\overline{T}(a_{\text{bct}})$ in the previous stage. Suppose there are p biconnected components $B^1, B^2, \ldots, B^p$ with a as their primary articulation point. Then a can obtain $\overline{T}(B^i_{\text{bct}})$ based on $\overline{T}(a_{\text{bct}}), T(B^1_{\text{bct}}), \ldots, T(B^{i-1}_{\text{bct}}), T(B^{i+1}_{\text{bct}}), \ldots, T(B^p_{\text{bct}})$ and broadcast this path information to all other articulation points within B_i. For the remaining articulation points in B_i, taking a' as an example, after obtaining the distance list of $\overline{T}(B^i_{\text{bct}})$. After completing the distance list of $\overline{T}(a'_{\text{bct}})$, the same operation as for the articulation point a is performed, continuing the iteration in $T(a'_{\text{bct}})$, thereby completing the traversal. Ultimately, all articulation points will have obtained the complete distance lists.

Once all articulation points of a component have correctly updated their dictionaries, according to the definition of biconnected components, every vertex within that component can determine the number of vertices in the entire graph that are at distance k from itself, $k \in [1, D^G]$. This information will be used in Algorithm 3. The `BFS` in Algorithm 1 is a Breadth-First Search that determines distances from the endpoints of the inserted edge and marks vertices with differing distances as affected vertices. These vertices signify changes in their BFS trees, affecting shortest paths originating or terminating at them.

The `BrandesSPBC` procedure (Algorithm 3) is inspired by the classic Brandes algorithm, executing a forward BFS and a backward BFS from each affected source vertex within the biconnected component. The `MSBFS` procedure, a distributed version of multi-source BFS [10], adapts to the CONGEST model with low-round complexity.

Lines 6-13 of Algorithm 3 correspond to the reverse phase. It is important to note that this component uses the $Schedule_B$ from the `MSBFS` procedure to mitigate congestion problems. Finally, lines 14-21 encompass the local statistics phase. Returning to Algorithm 1, at line 9, the positive factor implies that a portion of the old dependencies is removed from the original SPBC. The first execution of `BrandesSPBC` on G eliminates the old dependencies. Similarly, the negative factor at line 10 implies the addition of the correspondingnew dependencies. The second execution of `BrandesSPBC` on G' incorporates the new dependencies. Lemma 1 proves the correctness of the `SEI-SPBC`.

Lemma 1. *Algorithm 1 is capable of accurately computing the SPBC for each vertex.*

Proof. After identifying the affected source vertices, according to iCentral, the changes in centrality are confined to $B_e^{G'}$, and for each vertex $v \in B_e^{G'}$, the pairs (s,t) that need to be updated can be divided into three categories. The removal and addition of dependencies for these three categories of pairs are executed, referring to lines 9 and 10 of Algorithm 1.

For pairs (s,t) where both s and t belong to $B_e^{G'}$, the shortest path between the two vertices lies within $B_e^{G'}$. Therefore, the forward BFS and backward BFS

Algorithm 3 `BrandesSPBC(G,B,S,factor)`

1: Initialize $\delta_s[v], \delta_{G_s}[v]$ for $v \in B, s \in S$
2: $\{Schedule_B, P_s[v],\ d_s[v], \delta_s[v]\} \leftarrow \texttt{MSBFS}(S, B)$
3: **if** s and $v \in \mathcal{A}^G$ **then**
4:     **for** $i \leftarrow [0, |DL_s|],\ j \leftarrow [0, |DL_v|]$ **do**
5:         $\delta_{G_s}[v] = \delta_{G_s}[v] + DL_s[i] * DL_v[j] * f_{SP}(i + d_s[v] + j)$
****The vertices in B send messages follow the reverse $Schedule_B$****
6: **if** v is scheduled to send msg_s to $u \in P_s[v]$ **then**
7:     $msg_s \leftarrow \{\delta_s[v], d_s[v], \sigma_s[v]\}$
8:     **if** $s \in \mathcal{A}^G$ **then**
9:         $msg_s \leftarrow msg_s \cup \{\delta_{G_s}[v]\}$
10: **if** u receive msg_s from v **then**
11:     $\delta_s[u] = \delta_s[u] + \frac{\sigma_s[u]}{\sigma_s[v]} \times (f_{SP}(d_s[v]) + \delta_s[v])$
12:     **if** $s \in \mathcal{A}^G$ **then**
13:         $\delta_{G_s}[u] = \delta_{G_s}[u] + \delta_{G_s}[v] \times \frac{\sigma_s[u]}{\sigma_s[v]}$
****Message transmission complete****
14: **for** affected vertex $s \in S$ **do**
15:     **if** $v \neq s$ **then**
16:         $bc_{SP}^{G'}[v] = bc_{SP}^{G'}[v] + factor * \frac{\delta_s[v]}{2} + factor * g_{SP}(d_s[v])$
17:     **if** $s \in \mathcal{A}^G$ **then**
18:         $bc_{SP}^{G'}[v] = bc_{SP}^{G'}[v] + factor * \delta_s[v] \times |G_s|$
19:         **for** $i \leftarrow [0, |DL_s|]$ **do**
20:             $bc_{SP}^{G'}[v] = bc_{SP}^{G'}[v] + factor * DL_s[i] * g_{SP}(i + d_s[v])$
21:     $bc_{SP}^{G'}[v] = bc_{SP}^{G'}[v] + factor * \frac{\delta_{G_s}[v]}{2}$

in Algorithm 3 can correctly compute the dependencies, consistent with the static algorithm [3].

For pairs (s,t) where one of s and t belongs to $B_e^{G'}$ and the other does not, without loss of generality, assume $s \in B_e^{G'}$ and $t \notin B_e^{G'}$. The shortest path between the two vertices passes through the articulation point a of $B_e^{G'}$. According to the definition of an articulation point, $\sigma_{st} = \sigma_{sa} \cdot \sigma_{at}$ and $\sigma_{st}[v] = \sigma_{sa} \cdot \sigma_{at}[v]$. The dependency of the pair (s,t) on v can be computed via the pair (a,t), both of which belong to $B_e^{G'}$. Moreover, if the pair (s,t) requires an update, then all pairs formed by the vertices in $G_a^{B_e^{G'}}$ and the vertex t need to be updated. According to the definition of biconnected components, this requires line 18 of Algorithm 3. On the other hand, the auxiliary function g corresponding to the source vertices in $G_a^{B_e^{G'}}$ also needs to be computed, with the help of the distance lists we previously obtained, referring to line 20 of Algorithm 3.

For pairs (s,t) where neither s nor t belongs to $B_e^{G'}$, assume the shortest path passes through the articulation points a_i and a_j of $B_e^{G'}$. If the pair (s,t) requires an update, then all pairs formed by the vertices in $G_{a_i}^{B_e^{G'}}$ and the vertices in $G_{a_j}^{B_e^{G'}}$ need to be updated, and they have different values of the auxiliary function f based on their distances, which are initialized in lines 4 and 5. Similar to the previous case, since the shortest path between s and t must pass through a_i and

a_j, the dependency of the pair (s,t) on v can be computed via the pair (a_i, a_j), both of which belong to $B_e^{G'}$, referring to line 13 of Algorithm 3.

We now extend the `SEI-SPBC` algorithm by proposing an incremental method to maintain SMBC. The most straightforward approach, involves iteratively executing the `SEI-SPBC` algorithm n times [2]. However, this approach inevitably incurs prohibitively high round complexity. To address this issue, we introduce `SEI-SMBC`, which reduces the round complexity at the cost of increasing local computation. The `SEI-SMBC` algorithm is similar to Algorithm 1, with the key difference being the inclusion of an additional k-loop and a conditional check when computing the auxiliary functions. It should be noted that the k-loop is performed locally at each node, rather than forming loops within the network. The above optimization effectively reduces the round complexity and eliminates redundant traversals. Lemma 2 proves the correctness of the `SEI-SMBC` and Theorem 1 provides the round complexity of the single-edge incremental algorithms.

Lemma 2. *`SEI-SMBC` is capable of accurately computing the SMBC for each vertex, Furthermore, in contrast to Algorithm 1, `SEI-SMBC` does not require any additional communication.*

Proof. The core operation involves swapping the order of two summation loops for $bc_{SM}[v]$, namely the source vertex s loop and the k loop in the computation of the semi-value auxiliary function f. This exchange allows us to communicate solely within the s-loop. In the k-loop, since the nodes have already collected sufficient information, no communication is required, and the SMBC computation can be completed locally. Since the distinction between SPBC and SMBC lies solely in the auxiliary functions, once we resolve the high round complexity issue of the auxiliary functions for SMBC by swapping the order of the loops in the computation, and incur no additional communication, the remaining part of `SEI-SMBC` will also involve no extra communication, compared with Algorithm 1.

Theorem 1. *The round complexity of `SEI-SPBC` and `SEI-SMBC` are $O(D^G(\mathcal{A}_{\mathcal{B}}^{max} + \mathcal{D}_{\mathcal{B}}^{max}) + |F_{Batch}|)$.$\mathcal{A}_{\mathcal{B}}^{\max}$ and $\mathcal{D}_{\mathcal{B}}^{\max}$ denote the maximum number of articulation points and the maximum diameter of the biconnected components, respectively. $|F_{Batch}|$ represents the number of affected vertices resulting from insertion.*

Proof. In the case of the single-edge insertion, the maintenance of biconnected components can be completed in $O(D^G)$ rounds, for example, by using BFS. The bottom-up and top-down traversals in the `Distancelist` procedure require $O(D^G)$ rounds. The step of exchanging distance information is implemented via `MSBFS` procedure to avoid congestion. Since the size of the distance lists is $O(D^G)$, the number of rounds for the `Distancelist` procedure is $O(D^G(\mathcal{A}_{\mathcal{B}}^{\max} + \mathcal{D}_{\mathcal{B}}^{\max}))$. The round complexity of the `BrandesSPBC` procedure is dominated by the `MSBFS`(S, B) in line 2, which is $O(|F_{\text{Batch}}| + \mathcal{D}_{\mathcal{B}}^{\max})$, combining the maximum diameter of the biconnected components and the number of affected vertices. Compared with `SEI-SPBC`, the additional parts introduced by `SEI-SMBC` only increase local computation and do not affect the round complexity. The round complexities of the two algorithms remain the same.

6 MULTI-EDGE INCREMENTAL ALGORITHM

This section presents `MEI-SPBC` and `MEI-SMBC` for incrementally updating game-theoretic betweenness centralities for multi-edge insertions.

Haryan et al. [9] propose an incremental algorithm for shared-memory systems that builds a BFS tree T and, for every vertex v, maintains an auxiliary subgraph capturing the connectivity among v's children in T. A vertex u in this subgraph is **safe** if a non-tree edge joins u to an ancestor of v; a component is **safe** if it contains at least one safe vertex, and **unsafe** otherwise. Upon a new edge insertion, the algorithm updates the auxiliary subgraphs and identifies articulation points by the following rules: 1. w is the root of T and its subgraph has at least two components. 2. w is not the root and its subgraph contains at least one unsafe component. 3. w is adjacent to a bridge and has a degree of at least 2.

Algorithm 4 `MEI-UB`$(G, Batch)$

Input: Graph $G(V, E)$, set of new edge set Batch, each vertex v knows the biconnected component B_v^G it belongs
Output: Each vertex v knows the biconnected component $B_v^{G'}$ it belongs in the new graph G'
for each edge $e = \langle u, v\rangle \in Batch$ **do**
    Vertex u sends msg$(u, v, \langle u, v\rangle, +)$ upwards along T
    Vertex v sends msg$(v, u, \langle u, v\rangle, +)$ upwards along T
if Vertex w receives msg$(u, v, \langle u, v\rangle, +)$ and msg$(v, u, \langle u, v\rangle, +)$ **then**
    Mark w as a LCA vertex
    Update connectivity and propagate safety in the $subgraph_w$ held by w
    Vertex w transmits msg$(u, v, \langle u, v\rangle, -)$ and msg$(v, u, \langle u, v\rangle, -)$ along the path previously traversed by the message.
for each Vertex p receiving msg$(u, v, \langle u, v\rangle, -)$ from vertex q **do**
    Update the bridge flag of $\langle p, q\rangle$ if $\langle p, q\rangle$ is a bridge
    Mark q as a safe point
Identify articulation points and propagate within the biconnected components

Algorithm 4 is designed to update the biconnected components of a graph G following the insertion of edge set Batch. Each vertex u and v communicates connection update information by sending upward messages. Upon receiving messages from both u and v, a vertex w is designated as the least common ancestor(LCA) and updates the connectivity of the subgraph accordingly. Subsequently, vertex w propagates negative messages to update the bridge flags and propagate security information. Ultimately, the algorithm identifies articulation points and propagates updates within the biconnected components to ensure the consistency of the graph structure.

This approach ensures that each vertex in G' is aware of its biconnected component, facilitating efficient computation of betweenness centralities in large-scale dynamic graphs.

The `MEI-SPBC` algorithm is similar to Algorithm 1. The differences are as follows: First, `MEI-SPBC` replaces `SEI-UB` with Algorithm 4. Second, when inserting the edge set Batch, it is necessary to perform BFS instances for each pair of endpoints simultaneously. Due to congestion constraints, the `MSBFS` procedure must be invoked. Moreover, `MEI-SMBC` can also be derived from `MEI-SPBC`. For brevity, we omit a detailed description. Below, we analyze its complexity and show that the connectivity maintenance part does not dominate its complexity, which remains comparable to that of single-edge updates. Theorem 2 provides the round complexity of the multi-edge incremental algorithms.

Theorem 2. *The round complexity of `MEI-SPBC` and `MEI-SMBC` are $O(D^G(\mathcal{A}_{\mathcal{B}}^{max}+\mathcal{D}_{\mathcal{B}}^{max}) + |F_{Batch}| + |Batch|)$. $\mathcal{A}_{\mathcal{B}}^{\max}$ and $\mathcal{D}_{\mathcal{B}}^{\max}$ denote the diameter of the graph, the maximum number of articulation points and the maximum diameter of the biconnected components, respectively. $|F_{Batch}|$ and $|Batch|$ represent the number of affected vertices resulting from insertion and the number of inserted edges, respectively.*

Proof. In Algorithm 4, the `msg` messages corresponding to all updated edges in lines 4 and 5 can be sent simultaneously. In the event of message congestion, the messages are sent in lexicographical order, resulting in a round complexity of $O(D^G + |\text{Batch}|)$ for Algorithm 4. On the other hand, as mentioned in the previous paragraph, line 7 of Algorithm 1 requires the `MSBFS` procedure in the case of multiple-edge insertions, which adds an additional $O(|\text{Batch}|)$ rounds. The other parts of the algorithm in the case of multiple-edge insertions are similar to Algorithm 1.

7 EXPERIMENTAL RESULTS

This section reports the experimental evaluation of the proposed algorithms on real-world graphs. All experiments were conducted on a 20-node cluster; each node was equipped with two 24-core, 3.0 GHz Intel Xeon Gold 6248R processors and 256 GB of DDR4 memory. The algorithms were implemented in the Gemini framework[12]. Since Gemini does not natively support dynamic graphs, we emulated incremental updates by randomly removing and re-inserting edges. We selected several real-world datasets from [13, 14], covering social, communication, and road networks; their detailed statistics are summarized in Table 3.

Table 3. Dataset

Graph	$\lvert V\rvert$	$\lvert E\rvert$	Graph	$\lvert V\rvert$	$\lvert E\rvert$
roadNet-PA(PA)	1.08M	1.54M	web-Stanford(ST)	281.90K	2.312M
roadNet-TX(TX)	1.37M	1.92M	web-Google(GG)	875.71K	5.105M
roadNet-CA(PA)	1.96M	2.76M	wiki-Talk(WK)	2.39M	5.02M

After randomly removing 10 edges from the graph, we pre-computed the SPBC and all auxiliary data on the resulting graph and then reinserted the edges.

This experiment was executed with three approaches: (i) a static recomputation of the SPBC; (ii) `SEI-SPBC`, which performs incremental updates edge-by-edge; and (iii) `MEI-SPBC`, which processes the edges in a single batch. The running time and number of iterations for each approach are reported in Table 4.

Table 4. Performance of the static algorithm and the incremental algorithms for SPBC.

	Static		`SEI-SPBC`		`MEI-SPBC`	
Graph	Round	Time(s)	Round	Time(s)	Round	Time(s)
wiki-Talk	6704278	1443800.73	7590205	430874.66	2407238	111097.93
road-CA	20698461	493276.30	11019734	245411.07	1295702	31129.25
road-PA	14630532	221301.74	5415652	103085.63	745211	16832.63
road-TX	14630532	264895.56	5214811	75409.95	791074	18276.79
web-Stanford	3354657	17673.51	6252892	64962.19	611092	6517.41
web-Google	3207498	107198.35	21847093	507614.39	2277607	50745.06

Table 4 shows the multi-edge incremental algorithm improves efficiency compared to both the static and single-edge algorithms. The maximum speedup over the static algorithm is 15.84, with a minimum of 13.00, indicating substantial reduction in execution time.

Compared to the single-edge incremental algorithm, the multi-edge incremental algorithm achieves a maximum speedup of 7.88 and a minimum of 4.13, further enhancing efficiency by reducing redundant traversals. The performance gain depends on the graph's structure; if single-edge updates affect disjoint sets, the speedup may be less, while batch processing in the multi-edge algorithm reduces unnecessary traversals.

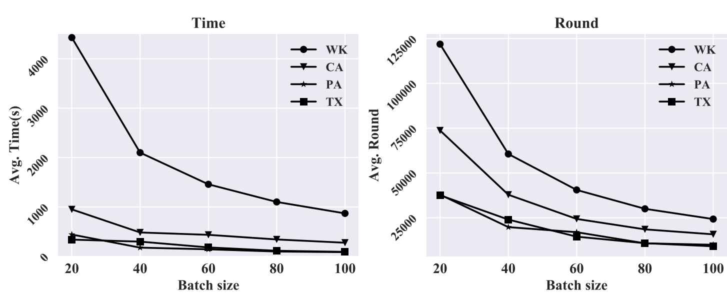

Fig. 2. Performance of `MEI-SPBC` across different batch sizes.

Incremental algorithms, whether for single or multi-edge insertions, are affected by the distribution of biconnected components in the graph. When the graph has many uniformly sized biconnected components, the incremental algorithm is more efficient. However, if there are only a few large components, the probability of update edges falling within these large components increases. This results in excessive traversal within the large components, causing the performance of the incremental algorithm to degrade to that of a static algorithm. As seen in the results for web-Google and web-Stanford, the incremental algorithm performs poorly, with `SEI-SPBC` even exceeding the execution time of the static algorithm.

We evaluated the scalability of `MEI-SPBC` with edge sets of sizes 20, 40, 60, 80, and 100, as shown in Figure 2. The x-axis represents batch size, and the y-axis shows the average time or rounds per inserted edge. `MEI-SPBC` scales efficiently with iterations, as processing speed improves with more edges, though it plateaus beyond a certain batch size. The algorithm demonstrates excellent scalability on wiki-Talk, but for road graphs, increasing batch size has little impact on processing speed.

8 CONCLUSION

This study explores the incremental algorithms for game-theoretic betweenness centralities in the CONGEST model. We introduce `SEI-SPBC`, a Shapley value-based algorithm for single-edge insertion that avoids congestion through efficient scheduling and includes a distance update procedure for accuracy. Building on this, we propose `SEI-SMBC`, a semi-value-based algorithm that reduces communication overhead at the cost of increased local computation, improving round complexity over general methods. Finally, we present a multi-edge insertion variant of `SEI-SMBC`, which updates biconnected components with low-round complexity, providing a scalable solution for large-scale network changes. Experimental results show that the multi-edge incremental algorithm achieves speedup factors of up to 7× and 16× compared to the single-edge and static algorithms, respectively.

When removing multiple edges, maintaining biconnected components in the CONGEST model requires regenerating components after destruction, which involves more complex procedures. Hence, this paper focuses solely on incremental algorithms. Future work will extend the algorithm to decremental algorithms with low-round complexity and optimize communication strategies to reduce congestion and costs in distributed environments.

References

1. B. N. Grofman and G. Owen, "A game theoretic approach to measuring degree of centrality in social networks," *Social Networks*, 4(3):213-224, 1982.
2. P. L. Szczepanski, T. P. Michalak, and T. Rahwan, "Efficient algorithms for game-theoretic betweenness centrality," *Artificial Intelligence*, 231:39-63, 2016.
3. Y. Wang, Q. Hua, W. Gao, and H. Jin, "Nearly optimal distributed algorithm for computing game-theoretic betweenness centralities," *Chinese Journal of Computers*, to appear, 2025.
4. P. L. Szczepanski, T. P. Michalak, and T. Rahwan, "A new approach to betweenness centrality based on the Shapley value," in *AAMAS 2012*, pp. 239–246, 2012.

5. M. K. Tarkowski, P. L. Szczepanski, T. P. Michalak, P. Harrenstein, and M. J. Wooldridge, “Efficient computation of semi-values for game-theoretic network centrality,” *J. of Artificial Intelligence Research*, 63:145-189, 2018.
6. F. T. Jamour, S. Skiadopoulos, and P. Kalnis, “Parallel algorithm for incremental betweenness centrality on large graphs,” *IEEE Trans. on Parallel and Distributed Systems*, 29(3):659-672, 2018.
7. R. G. Pons, “Space efficient incremental betweenness algorithm for directed graphs,” in *CIARP 2018*, pp. 262–270, 2018.
8. K. Shukla, S. C. Regunta, S. H. Tondomker, and K. Kothapalli, “Efficient parallel algorithms for betweenness- and closeness-centrality in dynamic graphs,” in *ICS ’20*, pp. 10:1–10:12, 2020.
9. C. A. Haryan, G. Ramakrishna, K. Kothapalli, and D. S. Banerjee, “Shared-memory parallel algorithms for fully dynamic maintenance of 2-connected components,” in *IPDPS 2022*, pp. 1195–1205, 2022.
10. C. Lenzen and D. Peleg, “Efficient distributed source detection with limited bandwidth,” in *PODC ’13*, pp. 375–382, 2013.
11. F. Harary, “Graph Theory,” *Addison-Wesley Publishing Company*, 1969.
12. X. Zhu, W. Chen, W. Zheng, and X. Ma, “Gemini: A computation-centric distributed graph processing system,” in *OSDI 2016*, pp. 301–316, 2016.
13. J. Leskovec and A. Krevl, “SNAP Datasets: Stanford large network dataset collection,” http://snap.stanford.edu/data.
14. T. A. Davis and Y. Hu, “The university of Florida sparse matrix collection,” *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.