Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2025.0429000

ABCD: Algorithm for Balanced Component Discovery in Signed Networks

MUHIEDDINE SHEBARO[1], (Member, IEEE) and Jelena Tešić[2],(Senior Member, IEEE)
[1]Independent Researcher, San Marcos, TX, USA (e-mail: shebaro.m@gmail.com)
[2]Department of Computer Science, Texas State University, San Marcos, TX, USA (e-mail: jtesic@txstate.edu)

Corresponding author: Muhieddine Shebaro (e-mail: shebaro.m@gmail.com).

ABSTRACT The largest balanced element in signed graphs plays a vital role in helping researchers understand the fundamental structure of the graph, as it reveals valuable information about the complex relationships between vertices in the network. The challenge is an NP-hard problem; there is no current baseline to evaluate state-of-the-art signed graphs derived from real networks. A scalable state-of-the-art approach for the maximum balanced subgraph detection in the network of *any* size is proposed. This approach finds the largest balanced subgraph by considering only the top K balanced states with the lowest frustration index. The ABCD method efficiently selects the largest possible subset from an extensive signed network with millions of vertices and edges, and the size of the discovered subset is double that of the state-of-the-art in a similar time frame.

INDEX TERMS balanced subgraph, balanced states, frustration index, and signed graphs.

I. INTRODUCTION

SIGNED networks allow for unsigned and negative weights in the graph-based representation. It is a graph where each edge between nodes is assigned a positive or negative sign. Negative weights represent antagonistic relationships and effectively model conflicting opinions between vertices [1]. Balance theory represents a theory of changes in attitudes [2]: people's attitudes evolve in networks so that friends of a friend will likely become friends, and so will enemies of an enemy [2]. Heider established the foundation for social balance theory [3], and Harary established the mathematical foundation for signed graphs and introduced the k-way balance [4], [5]. The solutions to the tasks to predict edge sentiment, to recommend content and products, or to identify unusual trends have had balanced theory at their core [7], [8], [9], [6].

Motivation: The task of the most extensive balanced subgraph discovery has applications in portfolio system's economic risk management [10], computational and operational research [11], community analysis and structure [12], computational biology to model balanced interactions between genes and proteins [13] and social network analysis [14]. The vertices that are part of the maximum balanced subgraph Σ' of Σ may not necessarily have a high degree of centrality among them. By locating the largest balanced subgraphs, we can simplify the system into sub-systems with balanced interactions and eliminate inconsistencies regarding unbalanced cycles. The largest balanced subgraph is the largest possible subset of the signed network that satisfies the balance theory (every cycle has an even number of negative edges). This subgraph doesn't have to be a clique, where a clique is a subset of nodes in the graph where every node has an edge to every other node in that subset.

Finding the largest balanced subgraph is a well-known NP-hard problem [15], and existing solutions do not scale to real-world graphs [1]. This paper proposes an algorithm for balanced component discovery (ABCD) in signed graphs. The approach builds on the scalable discovery of fundamental cycles in [16] and utilizes the graph's vertex density distribution and stable states to minimize the number of vertices removed from the balanced subgraph. Section II explains the notations, definitions, and theorems behind the signed graph balancing and the algorithm for the scalable graph cycle-basis computation of the underlying unsigned graph G of Σ. Section IV introduces the novel ABCD algorithm and the implementation details. We use the edge sign switching technique using a fundamental cycle basis discovery method to *search* for the maximum balanced subgraph. In Section V, we analyze the complexity of our algorithm. Section VI compares the proposed method to the state-of-the-art method proposed in [17]. The TIMBAL method achieved the highest vertex cardinality (number of vertices in the largest balanced subgraph) across all signed graphs, among other baselines in the literature [17]. We evaluate our algorithm on the

Konect and Amazon signed graphs in Subsection VI-A and Subsection VI-B, respectively. Next, we perform an ablation study on our proposed algorithm in Subsection VI-C and we test our method on synthetic graphs with random signs and varying sparsity levels in Subsection VI-D. In Section VII, we summarize our findings.

Problem Definition: It is to find the largest balanced component $\Sigma^G, |\Sigma^G| = g$ in *any* size signed graph $\Sigma, |\Sigma| = n$ is in Equation 1.

$$\Sigma^G \subseteq \Sigma \wedge Fr(\Sigma^G) = 0 \wedge \arg\max_{g \leq n} \Sigma^G \implies \Sigma^G \quad (1)$$

where $Fr(\Sigma^G)$ is the frustration of balanced subgraph Σ^G defined in [18]. The frustration is the level of imbalance (number of fundamental cycles with an odd number of negative signs) found in the network.

Goal: It is to find a subgraph in a signed network with an even number of negative edges along each fundamental cycle, and its size (node cardinality) is as large as possible.

TABLE 1. Summary of notations and their meanings.

Notation	Meaning
Σ/ G	signed network/ its underlying graph
Σ^G	subgraph
T/ I	spanning tree/ # of spanning trees
v/ V_x/ V / $\|V\|$	a node/ set of some vertices/ set of all vertices/number of nodes
E_x/E/ $\|E\|$	a set of edges/ set of all edges in Σ / number of edges
e/ e^+ / e^-	an edge/ positive edge/ negative edge
Σ'	*any* balanced state of Σ
Σ_i	i^{th} nearest balanced state Σ
$Fr(\Sigma_i)$	frustration from Σ to Σ_i
(U, W)	Harary bipartition sets, $\|U\| \geq \|W\|$
$\mathcal{C}_\Sigma$	the frustration cloud set of Σ
$\|\mathcal{C}_\Sigma\|$	the size of the frustration cloud.
K	# of nearest balanced states w lowest frustration index, $K \leq \|\mathcal{C}_\Sigma\|$
H	binary array of size$\|V\|$ to separate the vertices into Harary bipartitions where $H[v] = 1$ if v is in set U, and 0 if v is in set W
$\mathcal{H}_\Sigma$	container to save the collection of H array for the top-K balanced states with the lowest frustration
$\mathcal{E}_\Sigma$	container of Σ for storing a set of edges in each element that switched signs during balancing
$\mathcal{F}_\Sigma$	K size array of the number of edge sign switches of top K nearest balanced states.
$\mathcal{V}_i$	set of vertices to remove from graph i
$\mathcal{ABCD}$	a set containing K subgraphs after $V/\mathcal{V}_i$

Contributions: We introduce the ABCD method that identifies the largest balanced subgraph in a signed network. Our approach begins by considering only the top K balanced states with the lowest frustration index. For each state, we identify the edges whose signs have been switched. Then, by leveraging one of three handling criteria, ABCD selects an appropriate node for removal along each switched edge. This selection is designed to minimize node removal while breaking the unbalanced cycle associated with that edge, which preserves as many nodes as possible. Finally, ABCD chooses the largest balanced subgraph among the residues corresponding to each of the K balanced states. Our experimental results demonstrate that our approach consistently discovers a larger balanced subgraph compared to the baseline proposed in [17].

Unlike TIMBAL, which relies on eigenvalue computations and subsampling to scale to large graphs, ABCD leverages scalable fundamental cycle discovery together with the extraction of the top-K stable states with the lowest frustration. This unique approach enables ABCD to reduce the number of vertex removals along the edges causing imbalance as well as the number of vertices lost in the process to obtain significantly larger balanced subgraphs than that of the spectral-based baseline.

II. PRELIMINARIES AND DEFINITIONS

In this section, we define the fundamental cycle basis and relevant signed graph network terms and outline the theorems and corollaries for the proposed ABCD approach. Table 1 outlines the meaning of the notations used when describing the algorithm steps in the paper

A. BASIC CONCEPTS IN SIGNED GRAPHS

Definition 2.1: **Signed graph** $\Sigma = (G, \sigma)$ consists of underlying unsigned graph G and an edge signing function $\sigma : e \to \{+1, -1\}$ [41]. Each edge $e \in E$ is either positive (e^+) or negative (e^-). The sign of a **subgraph** is the product of its edge signs. **Path** is a sequence of distinct edges $|E|$ that connect a sequence of distinct vertices $|V|$ in a graph. **Connected graph** has a path that joins any two vertices. **Cycle** is a path that begins and ends at the same node.

Definition 2.2: Graph Σ^G is a **subgraph** of a graph Σ if **all** edges and vertices of Σ^G are contained in Σ.

Definition 2.3: **Balanced Signed graph** is a signed graph where every cycle is positive. The **Frustration Index** of a signed graph is the minimum number of candidate edges whose sign needs to be switched for the graph to reach a balanced state [19].

We next define cycles and fundamental cycle bases, which will be used in our algorithm.

B. CYCLE-RELATED CONCEPTS

Definition 2.4: **Cycle Basis** is a set of simple cycles that forms a basis of the cycle space [20].

Definition 2.5: A **Cut vertex** is a vertex whose removal increases the number of connected components within the network.

Definition 2.6: For the underlying graph G, let T be the spanning tree of G, and let an edge e be an edge in G between vertices x and y that is *NOT* in the spanning tree T. Since the spanning tree spans all vertices, a unique path in T between vertices x and y does not include e.

Definition 2.7: **The fundamental cycle** is any cycle that is built using path in T plus edge e in graph G [18].

Corollary 2.1: All the cycles formed by combining a path in the tree and a single edge outside the tree create a fundamental cycle basis from a spanning tree. Thus, the underlying un-

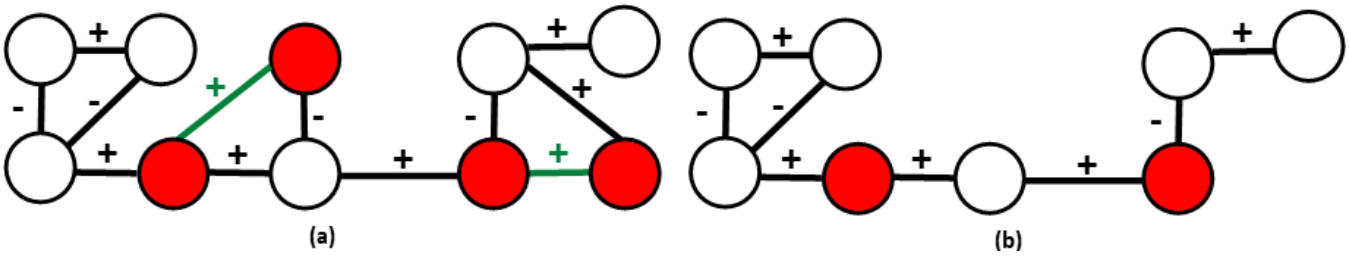

FIGURE 1. **(a): The unbalanced signed network. Green edges are the candidate edges causing imbalance, and red vertices are the candidate vertices. (b)The maximum balanced signed subgraph is obtained after deleting one candidate vertex along each edge.**

signed graph G with $|V|$ vertices and $|E|$ edges has precisely $|V| - |E| + 1$ fundamental cycles [16].
We now formalize structural balance in terms of frustration and near-balanced states.

C. BALANCE AND FRUSTRATION

Definition 2.8: The balanced states are **near-balanced** if and only if the original graph requires a minimum number of edge sign switches to reach a balanced state. We label the stable states of Σ as Σ_i, where $i \in [1, |\mathcal{F}_\Sigma|]$. $|\mathcal{F}_\Sigma|$ is the the size of the frustration cloud in [18].
Definition 2.9: The frustration cloud is the collection of nearest balanced states generated using I for the unbalanced signed network. For example, if $I = 4$, then 4 nearest balanced states will be generated using the efficient fundamental cycle basis discovery method (graphB+).
Definition 2.10: An **imbalanced edge** as an edge that contributes to structural instability and violates the balance theory. For example, suppose individuals A and B are friends (+), individuals B and C are friends (+), and individuals A and C are enemies (-). In that case, we can say that the edge between A and C is imbalanced because it creates an imbalanced triad.
Lemma 2.1: If a signed subgraph Σ^G is balanced, the following are equivalent [4]:

1) Σ^G is balanced. (All cycles are positive.)
2) For every vertex pair (v_i, v_j) in Σ', all (v_i, v_j)-paths have the same sign.
3) $Fr(\Sigma^G) = 0$.
4) There exists a bipartition of the vertex set into sets U and W such that an edge is negative if, and only if, it has one vertex in U and one in W. The bipartition (U,W) is called the *Harary-bipartition*. Note the sets so that U always contains a more significant or equal number of vertices than W.

III. RELATED WORK

Finding the stable subgraph in a signed graph is known to be an NP-hard problem. Gülpinar et al. proposed the GGMZ algorithm. GGMZ computes the input graph's minimum spanning tree, then selects the subset of vertices so that all the edges crossing that subset are inverted to create positive edges, and the result is the set of vertices disconnected by negative edges. The system's overall complexity is $O(|V|^3)$ if V is a set of vertices [21] Poljak and Daniel Turzík showed that any signed graph that has $|V|$ vertices and $|E|$ edges contains a balanced subgraph with at least $0.5|E|+0.25(|V|-1)$ edges [22]. Crowston et al. [23] proposed discovering of a balanced subgraph of size $0.5|E|+0.25(|V|-1+k)$ where k is the parameter. They reduced data by finding small separators and a novel gadget construction scheme. Figueiredo et al. introduced a polyhedral-based branch-and-cut algorithm to find the largest subgraph [24]. Then, they proposed GRASP, an improved algorithm version with the pre-processing and heuristic methods [11]. The GRASP algorithm randomly selects a subset of vertices. It greedily adds vertices that maximize the number of edges connecting them to the current subset while keeping the size of the subset balanced [11]. The EIGEN algorithm [25] works by first computing the eigenvectors of the Laplacian matrix of the graph. Using the dominant eigenvector of the adjacency matrix, it then partitions the graph into two disjoint sets. The partition is made by setting a threshold value for the eigenvector and assigning each vertex to one of the two sets based on whether its value in the eigenvector is above or below the threshold. The algorithm then recursively applies this partitioning process on each of the two sets. Sharma et al. proposed a heuristic that deletes edges from the graph associated with the smallest eigenvalues in the Laplacian matrix of the graph until a maximum balanced subgraph is obtained [26]. Ordozgoiti et al. introduced the most scalable version of the algorithm to date. TIMBAL is an acronym for *trimming iteratively to maximize balance* two-stage method approach, where the first stage removes vertices and the second one restores them as long as it does not cause imbalance [17]. Both algorithms rely on signed spectral theory. These approaches do not scale to the large signed graphs as they rely on the costly eigenvalue computation ($O(|E|^2)$), and their performance decreases due to the spectral pollution in eigenvalue computation [27]. TIMBAL proposed a novel bound for perturbations of the graph Laplacian preprocessing techniques to scale the processing for large graphs. They evaluate the scalability of the proposed work on graphs over 30 million edges by artificially implanting balanced subgraphs of a specific size and recovering them [17]. Shang [28] proposes a continuous-time model for social networks that represents friendship and hostility through directed signed graphs. The study shows that, from almost any initial state, the network evolves into at most four factions. While under mild initial conditions, structural balance with at most two

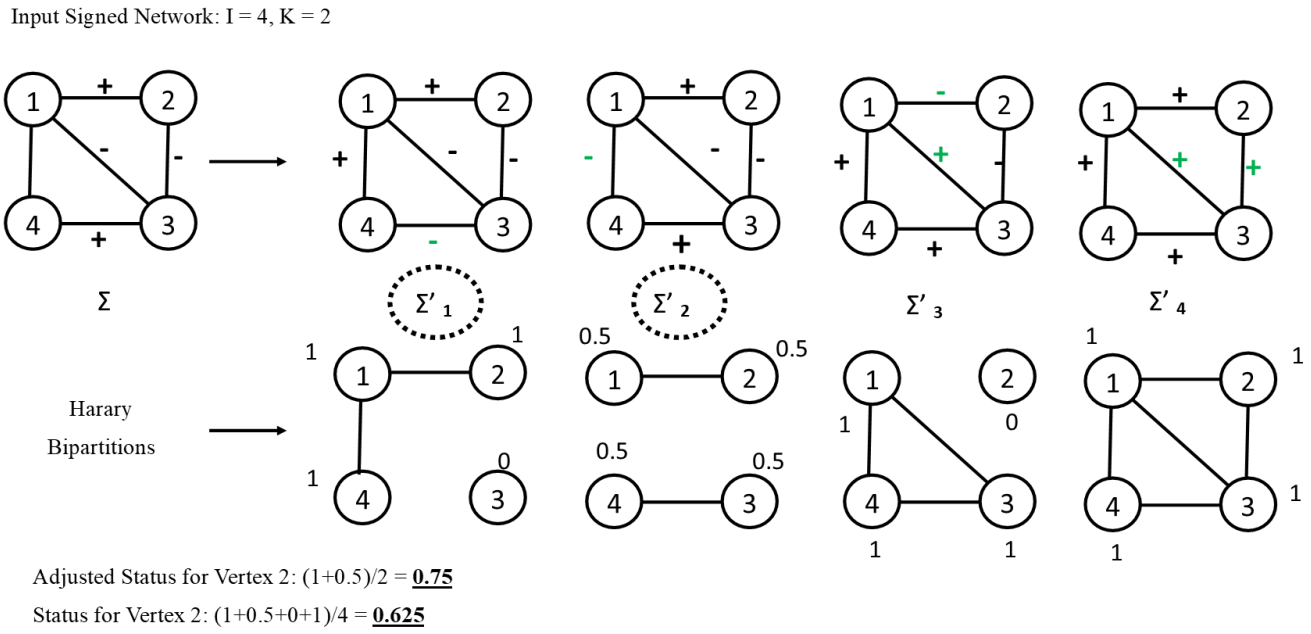

FIGURE 2. **Example of Harary bipartition, adjusted status, and status on a small graph with 4 nodes and 5 edges. With $I = 4$, four nearest balanced states are generated. Among these, the first two states with lowest frustration ($K = 2$) are selected. Harary bipartition is obtained by deleting negative edges from each balanced state. Adjusted status is computed over K balanced states, while the status is computed over all I states. The green edge signs indicate flipped signs.**

factions can be achieved.

Note that the Maximizing balance via edge deletions (MBED) task is *different* from the task of discovering the maximal balanced subgraph. The MBED task requires the target community and the budget as input, and the goal is to remove edges to make that input community as close to being balanced as possible [26]. Discovering the maximal balanced subgraph *does not require* community and budget specifications. Moreover, for MBED, we minimize the number of edges along the cycles that are causing an imbalance, and the goal is to make the entire graph as close to balanced as possible by removing or flipping the fewest edges. On the other hand, for discovering the maximal balanced subgraph, we aim to extract the largest possible subgraph that is perfectly balanced, as we are not trying to fix the whole graph; we find the biggest chunk that already satisfies the balance theory.

Preserving connectivity when deleting vertices is critical to maximizing the chances of obtaining a large, balanced subgraph. Kleinberg et al. [29] considered a model for tracking the network connectivity under vertex or edge deletions, focusing on detecting (ϵ, k)-failures. These failures occur when an adversary deletes up to k network elements, each at least an ϵ fraction of the network, becoming disconnected. A set of vertices is called an (ϵ, k)-detection set if, for any $\frac{1}{\epsilon}$-failure, some two vertices in that set of vertices can no longer communicate. The authors show that for an adversary that can delete $k\lambda$ edges, the random sampling approach can detect a set of size $O(\frac{k}{\epsilon} log \frac{1}{\epsilon})$, and polynomial time is required to detect the (ϵ, k) set of minimum size with the proposed approach [29].

Deng et al. [30] studied the effect of vertex deletion on the network structure by proposing an evolving network model. They revealed that as the intensity of vertex deletions increases, the network's degree distribution shifts from a scale-free to an exponential form and that vertex deletions generally decrease the network's clustering coefficient. On the other hand, the problem opposite to preserving connectivity upon vertex deletion is called the Critical vertex Detection Problem [31]. This problem has garnered significant interest recently, and the goal is to identify a set of vertices whose removal most effectively disrupts network connectivity based on specific connectivity metrics. In addition, the problem of finding various subgraphs has been intensively studied in the random graph setting [33], [32].

IV. METHODOLOGY

The **Algorithm for the Balanced Component Discovery (ABCD)** approach removes the minimal number of vertices by removing one vertex per imbalanced edge in a signed graph. We find a candidate set of imbalanced edges for the given signed graph by implementing a fast balancing algorithm proposed in [16], [18]. Section IV-A outlines the ABCD Phase 1 approach of collecting multiple candidate edge sets for deletion. Section IV-B describes three different Algorithms for the Balanced Component Discovery (ABCD) approaches to approximate connectivity in the vertex removal procedure: Degree-based (ABCDD), Harary-based (ABCDH), and Status-based (ABCDS). We title Phase 1 as "Top K Nearest Balanced States Extraction and Candi-

date Edge Identification'' and Phase 2 as ''Candidate Vertex Purging and Largest Subgraph Retrieval. Fig. 1 demonstrates an example execution of our algorithm where the balancing algorithm identifies the candidate edges causing imbalance (green) [18]. One of the red vertices along these candidate edges has been removed based on handling criteria. These candidate edges are selected based on a backbone algorithm, which is graphB+ [16] that generates a random spanning-tree (can be Breadth-first Search or Depth-First Search), then the edges that are not part of the spanning-tree are chosen to be candidate edges if they switch signs during balancing. Balancing occurs when graph B+ traverses each fundamental cycle and switches the non-tree edges' sign if the corresponding cycle has an odd number of negative signs. Fig. 2 shows an illustration of the Harary bipartition, adjusted status, status of a small graph of 4 nodes and 5 edges. With $I = 4$, 4 nearest balanced states are generated. With $K = 2$, the states Σ'_1 and Σ'_2 with the lowest frustration (one sign switched in each) will be selected for further processing. Harary bipartition is formed by deleting the negative edges after obtaining each balanced state. To compute the adjusted status and (vanilla) status, we assign 1 score to a vertex if it is in the larger partition, 0.5 if both partitions have equal size, and 0 if the vertex is in smaller partition. Adjusted status of a vertex is defined as the normalized sum of scores if the vertex is in the larger partition over K balanced states unlike the status where it is over I balanced states.

A. ABCD PHASE 1

ABCD Phase 1 generates the nearest balanced states, retrieves the top-K states with the lowest frustration, and identifies the candidate edges causing an imbalance of each K state by comparing the edge signs before and after the balancing process. I is the number of iterations we run the algorithm, and the upper bound on how many optimal balanced states we discover in the process. The Algorithm 1 outlines the Phase 1 steps in detail. We also get the placement of each vertex in the Harary subsets of the vertices along each candidate edge of K states. Essentially, we (1) discover the fundamental cycle bases for each of the I spanning trees; (2) for each of the cycles in the basis, count the number of cycles that contain an odd number of negative edges; and (3) keep only the K, $K << I$ balanced states out of I accessed that have the smallest number of fundamental cycles with an odd number of negative edges (imbalanced cycles).

B. ABCD PHASE 2

The ABCD Phase 2 employs an innovative vertex deletion approach for all K discovered balanced states to minimize the number of vertices removed along the edges that switched signs (causing imbalance) from the graph, which increases the vertex cardinality of the largest balanced subgraph. A graph is said to be connected if there is a path between any two vertices. Removing a vertex can potentially disconnect the graph, thus significantly reducing the size of the largest subgraph. Studying the connectivity before and after the removal of vertices gives us insights into the critical points that maintain the graph's connectivity. In graph theory, a bridge, cut-edge, or cut arc is an edge whose deletion increases the graph's number of connected components. Removing the non-bridge vertices instead of bridge vertices increases the chances of graph connectivity in the vertex removal process. Detecting bridges takes $O(|V| + |E|)$ if we use the efficient Tarjan's algorithm [34]. This approach is too costly for $Fr(\Sigma)$ times in the edge deletion process and prohibitive for large graphs. The total complexity will be $O(Fr(\Sigma) * (|V| + |E|))$. In real graphs, $|E| > |V|$, so we approximate the complexity as $O(Fr(\Sigma) * (E))$, which is too expensive for large graphs.

Here, we propose three efficient approximations of connectivity that rely on the scale-free characteristic of the prominent real graphs [36], [35]. Those graphs have a degree distribution that follows a power law, at least asymptotically. For example, in Table 2, WikiPolitics and WikiConflict have a relatively large max degree of 20,153 and 10,715, respectively, where the median and average degrees are much lower. Recent interest in heavy-tailed degree distribution in social, biological,

Algorithm 1 ABCD Phase 1

```
Fetch signed graph Σ, number of iterations I, and integer K that determines the stable states with the lowest frustration index to keep
Generate set of I spanning trees T of Σ
Create empty sets F_Σ, E_Σ, and H_Σ
Initialize c = 0
for i = 0; i++; i < I do
    Create empty set M_i
    for edges e, e ∈ Σ \ T_i do
        if fundamental cycle T_i ∪ e is negative then
            Add edge e to M_i
        end if
    end for
    if |F_Σ| < K then
        F_Σ[c] = |M_i|
        E_Σ[c] = M_i
        Fetch H_i by executing Algorithm 4 with inputs Σ and M_i
        H_Σ[c]=H_i
        c = c + 1
    else
        Get index l such that F_Σ[l] is the largest
        if F_Σ[l] < |M_i| then
            Delete F_Σ[l], E_Σ[c], and H_Σ[c]
            F_Σ[l] = |M_i|
            E_Σ[l] = M_i
            Fetch H_i by executing Algorithm 4 with inputs Σ and M_i
            H_Σ[l]=H_i
        end if
    end if
end for
Return F_Σ, E_Σ, and H_Σ
```

Algorithm 2 ABCD Phase 2

Input Σ, $\mathcal{E}_\Sigma$, K, $\mathcal{H}_\Sigma$, and integer *app* (1 for ABCDD, 2 for ABCDH, 3 for ABCDS)
Compute the adjusted status $\mathcal{O}_\Sigma$ (as in Algorithm 5) and the degree for each vertex (degree array)
Fetch the sum of neighborhood degree *nei* by executing Algorithm 3
for $i = 0$; i++; $i < K$ **do**
  Initialize empty set $\mathcal{V}_\rangle = \emptyset$
  for edges e, $e \in \mathcal{E}_\Sigma[i]$ **do**
    if any of the vertices along $e \in \mathcal{V}_i$ **then**
      Skip iteration
    end if
    if *app*==2 **then**
      if e is positive **then**
        Append the vertex of index q along e that has a lower sum of neighborhood degrees $nei[q]$ to set $\mathcal{V}_i$
      else
        Append the vertex of index w along e where $\mathcal{H}_\Sigma[i][w] = 0$ to set $\mathcal{V}_i$
      end if
    else if *app*==3 **then**
      Fetch the adjusted status of both vertices of index q and w along edge e
      if $\mathcal{O}_\Sigma[q] < \mathcal{O}_\Sigma[w]$ **then**
        Append the vertex of index q along e to set $\mathcal{V}_i$
      else
        Append the vertex of index w along e to set $\mathcal{V}_i$
      end if
    else
      Fetch the degree of both vertices of index q and w along edge e
      if degree$[q]$ < degree$[w]$ **then**
        Append the vertex of index q along e to set $\mathcal{V}_i$
      else
        Append the vertex of index w along e to set $\mathcal{V}_i$
      end if
    end if
  end for
  Create Σ'_i as $(V \setminus \mathcal{V}_i, E \setminus \mathcal{E}_\Sigma[i])$
  Find the largest connected component of Σ'_i as Σ''_i
  Push Σ''_i to $\mathcal{ABCD}$
end for
Find the largest $\Sigma''_i \in \mathcal{ABCD}, i \in [1, K]$ and return it.

and economic networks shows that a few power users connect directly or indirectly to most vertices. A large number of vertices connect to a few power users [37], and recently analyzed real signed graphs exhibit scale-free behavior [38].

For every set of edges we save in the $\mathcal{E}_\Sigma$, their cardinality is the frustration of that stable state and saved in $\mathcal{F}_\Sigma$ set in Algorithm 1 and it is the frustration measure for that nearest balanced state. Thus, the number of vertices erased is bound to be smaller than or equal to $\mathcal{F}_\Sigma[i]$ for the i^{th} nearest balanced state saved as an output of the Algorithm 1. Consider the upper bound on the frustration and define it as in Equation 2 from the fundamental cycle balancing theory:

$$\forall i, i \in [1, K] \rightarrow \mathcal{F}_\Sigma[i] \leq |E| - |V| + 1 \tag{2}$$

This equation stems directly from the Corollary 2.1. The maximum number of fundamental cycles is $|E| - |V| + 1$. In the worst-case scenario for finding the largest balanced subgraph, the graph requires disconnecting each of the $|E| - |V| + 1$ cycles, with none of the edges removed along these cycles sharing vertices. Thus, the upper bound on the number of vertices ABCD removes in Phase 2 in Algorithm 2 is $|E| - |V| + 1$. Note that for the large scale-free graphs, $|E|$ and $|V|$ are within the order of magnitude. The number of the fundamental cycles is much smaller than the number of edges, as illustrated in Table 4.

Algorithm 2 summarizes the Phase 2 steps for each stable state. First, we initialize an empty set $\mathcal{V}_i$ to track vertices that should not be deleted along candidate edges (edges that flip signs). If either vertex is already in $\mathcal{V}_i$, we skip that edge and proceed to the next one. We add the vertex to $\mathcal{V}_i$ based on one of the following three criteria, depending on the selection of the *app* parameter: (1) ABCDD: Subsection IV-B1, *app* = 1; (2) ABCDH: Subsection IV-B2, *app* = 2; (3) ABCDS: Subsection IV-B3, *app* = 3. We repeat this step K times, $\forall i$, $i \in [1, K]$. The outcome of Algorithm 2 is the set V_i of vertices to be removed for each of the K stable states. Each V_i is then used to obtain the graph Σ''_i, defined as the largest connected component of $(V \setminus \mathcal{V}i, E \setminus \mathcal{E}_\Sigma[i])$. The algorithm stores each K Σ''_i in $\mathcal{ABCD}$, and the largest $Sigma'_i$ will be returned as the outcome for this task.

In the next three subsections, we will describe the three approaches in this phase that are efficient approximations for connectivity modeling. Fig. 4 displays three versions of the ABCD Phase 2 step-by-step on a sample signed graph with seven vertices and ten edges, introduced at the top. Phase 2 starts by retrieving Harary bipartitions from Phase 1 and then calculating neighborhood sum (green), degree (blue), and adjusted status (red). Unbalanced fundamental cycle edges are shown in orange, while candidate edges are in green. At the phase's end, one candidate vertex is marked for deletion and removed from the figure; the red vertices are candidate vertices that remain. The largest balanced subgraph has the most vertices among the three produced subgraphs.

1) ABCDD: Algorithm for Balanced Component Discovery Degree

The algorithm for Balanced Component Discovery Degree (ABCDD) models the graph's overall connectivity as the degree. The degree of a vertex (the number of edges connected) can impact the graph's connectivity. Removing high-degree vertices removes more edges from the graph, which can introduce smaller connected components, specifically for scale-free graphs. Our goal is exactly the opposite. The approach eliminates the vertex with a smaller (neighborhood) degree

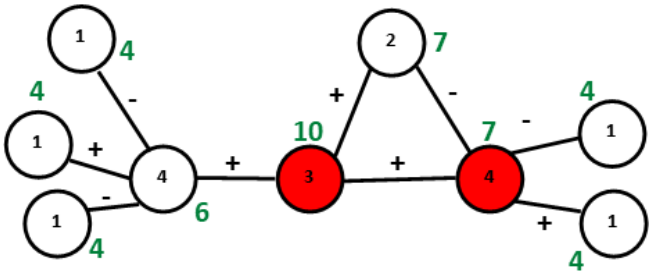

FIGURE 3. Degree (black, in node) vs. Sum of Neighborhood Degrees (green, next to the node) computation. The sum of neighborhood degrees labels the red vertices connected by a positive link that are candidates for deletion based on the ABCDD criteria.

Algorithm 3 Computation of the sum of neighborhood degree

```
Input signed graph Σ, degree array, and |V|
Declare and initialize array neighborhood_degree = []
for q = 0; q++; q < |V| do
    Declare and initialize integer sum = 0
    for every neighbor of index nei connected to vertex of
    index q via an edge do
        sum += degree[nei]
    end for
    neighborhood_degree[q]=sum
end for
Return array neighborhood_degree
```

out of the two. Algorithm 3 outlines the computation of the sum of neighborhood degrees measure. We also observe that performing the process three times, where in each iteration, we set degree equal to neighborhood_degree and then compute a new neighborhood_degree using the updated degree, generally enhances the results. Fig. 3 demonstrates how the sum of neighborhood degree can be better in certain cases than the degree as a criterion for purging vertices. The two red vertices in the image indicate that the balancing algorithm has labeled the edge and that its sign needs to be switched for the graph to achieve a complete balancing state. The degree of the left vertex is 3, and the right vertex is 4. The neighborhood degree of the left vertex is 10 (neighbors of a neighbor), and the neighborhood degree of the right vertex is 7. We chose the vertex on the right to delete and the vertex on the left to keep (if we used degree as a criterion to delete vertices, we wouldn't have obtained the largest balanced subgraph in this particular case). For the following experiments, we use degree and not the sum of neighborhood degrees as a handling criterion for purging vertices.

2) ABCDH: Algorithm for Balanced Component Discovery Degree Harary

The stable states to Σ, defined in Def. 2.8, do not have the same sets of candidate edges for balancing. The balancing can be achieved in multiple ways, as explained in detail in [18]. The proposed balancing algorithm identifies the stable states and the exact edges to remove to balance the remaining subgraph. Algorithm 4 changes the signs of those edges and creates sets U and W for each stable state. Each of the sets is balanced and has positive connectivity among its vertices. If the original edge was negative, now it is positive, and both vertices are either in U or W. Note that most real networks have 20% of opposing edges compared to 80% of positive edges [18]. Thus, for the deletion criteria in the ABCDH, there is less chance that the edge will end up in one of the bipartitions. The majority case is now if the original edge was positive, and now it is negative; one vertex is in U, and another is in W.

Algorithm 4 Harary Algorithm

```
Input Σ, set of edges that should flip their signs |E|.
Create zero vector H of dimension |V|
for edge e ∈ M do
    switch edge sign in Σ: e^- → e^+; e^+ → e^-
end for
Cut all the negative edges to create Harary bi-partitions U
and W so that |U| > |W|
for every vertex of index q ∈ Σ do
    if v ∈ U, H[q] = 1
end for
Return H
```

The Algorithm for Balanced Component Discovery Degree Harary (ABCDH) approximates the connectivity by placing the vertex from the smaller set W in the candidate deletion set $\mathcal{V}_\rangle$ if the edge is negative. If the original edge is negative, the balancing algorithm will switch to positive and place both vertices in the same Harary partition. In that case, we resort to the ABCDD baseline, where the sum of neighborhood degrees determines which vertices to delete. If both have the same sum of neighborhood degrees, choose a random vertex along that edge to discard. Note that the *neighborhood* degree is computed for all vertices in the signed graph *once* and reused for computation.

3) ABCDS: Algorithm for Balanced Component Discovery Status

Algorithm 5 Adjusted Status Computation

```
Input Σ, H_Σ, and K
Initialize zero array O of size |V|
for i = 0; i++; i < K do
    for every vertex of index x ∈ Σ do
        O_Σ[x] += H_Σ[i][x]
    end for
end for
for every vertex of index x ∈ Σ do
    O_Σ[x] /= K
end for
Return O
```

The Algorithm for Balanced Component Discovery Degree Status (ABCDS) approximates the connectivity by placing the vertex with the smaller adjusted status measure of the

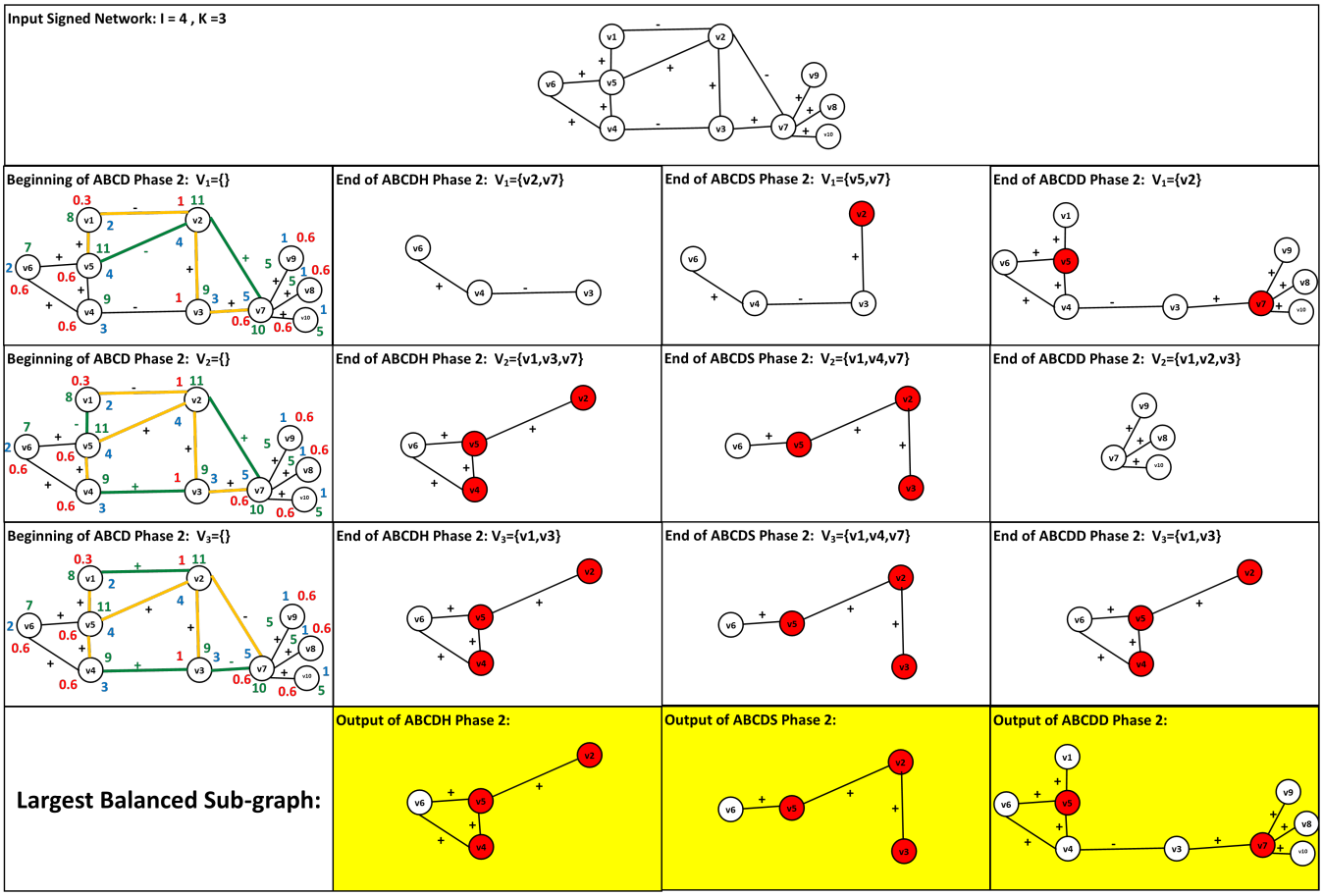

FIGURE 4. **The ABCD algorithm applied to a sample signed graph with ten vertices and thirteen edges. For connectivity approximations, we compute the Harary bipartition in ABCD Phase 1 (Algorithm 1) and compute the sum of neighborhood (green), degree (blue), and adjusted status (red) at the beginning of Phase 2 (Algorithm 1) once for the entire graph. Orange edges are the edges of the unbalanced fundamental cycles. Green edges are the candidate edges. The result of Phase 2 for all three connectivity approximations is the subgraph defined by black and red vertices.**

two vertices along an edge that switched sign in the candidate deletion set $\mathcal{V}_\rangle$. The ABCDS uses the adjusted status measure from [18] to determine which vertex to delete. The adjusted status computation takes all K binary $\mathcal{H}_\Sigma$ vectors that Algorithm 1 created and sums all K $\mathcal{H}_\Sigma$ element-wise, and divides each element in this array by K to define the adjusted status of the vertex. The logic behind using this adjusted status using K balanced states' Harary bipartition is that it is more robust than using a single Harary bipartition, as shown in [18], which captures a node's importance, and it is easy to compute. Algorithm 5 outlines the steps for computing this adjusted status using the Harary bipartition binary vectors. If the statuses of both vertices are the same, choose a random vertex along that edge to discard.

The criteria for choosing which variant to use are all hinged upon the structure of the signed network, and there is no one-size-fits-all approach for identifying the largest balanced subgraph. For instance, in Fig. 3, the red node on the right would be better to purge because it is connected to fewer nodes, and we used the sum of neighborhood degree to obtain the largest balanced subgraph. In that case, the ABCDH variant is better to use in this scenario since it utilizes the sum of neighborhood degrees to purge vertices. On the other hand, if the red left node was only connected to the right red and the shared neighbor with degree 2. Then, both red nodes would have a sum of neighborhood degree of 6, which impedes the selection process of which node to drop. In that scenario, using the degree as a measure to drop the node carrying fewer vertices is suitable. Because the left red node has a degree of 2, which is less than that of the right red node, which is 4, we drop the left red node, which is the correct action. Thus, the ABCDD variant is better since it utilizes the degree measure.

V. COMPLEXITY ANALYSIS

In ABCD Phase 1, balancing the signed network for all vertices takes $O(|E| * log(|V|) * d_a)$ where d_a is the average degree of a vertex as analyzed in [16]. The time complexity to count the number of edge sign switches that occurred after is O($|E|$). ABCD Phase 2 takes O($K * (|E| * log(|V|) + |E|)$) because we have to loop over every top-K balanced state. For each state, we loop over candidate edges, insert vertices to be kept in a set that takes $log(|V|)$ (assuming the set data structure in C++ is a red-black tree), and reread O($|E|$) to reread each top-K balanced state without the candidate

TABLE 2. Konect plus Twitter Ref. and PPI properties. Avg, Median, and Max refer to the degree.

Signed Graph	# vertices	# edges	# cycles	Density	# of Triads	Avg	Median	Max	% of e^-
Highland	16	58	43	0.483	68	7.25	7.5	10	50
CrisisInCloister	18	126	145	0.82	479	14	14	17	42.06
ProLeague	16	120	105	1.0	560	15.0	15	15	10.83
DutchCollege	32	422	391	0.85	3,343	26.37	28	31	0.47
Congress	219	521	303	0.021	212	4.71	3	33	20.34
PPI	3,058	11,860	8,803	< 0.01	3,837	3.87	2	55	32.5
BitcoinAlpha	3,775	14,120	10,346	< 0.01	22,153	7.48	2	511	8.39
BitcoinOTC	5,875	21,489	15,615	0.01	33,493	7.31	2	795	13.57
Chess	7,115	55,779	48,665	< 0.01	108,584	15.67	7	181	24.15
TwitterReferendum	10,864	251,396	240,533	< 0.01	3,120,811	46.28	12	2,784	5.08
SlashdotZoo	79,116	467,731	388,616	< 0.01	537,997	11.82	2	2,534	25.16
Epinions	119,130	704,267	585,138	< 0.01	4,910,009	11.82	2	3,558	16.82
WikiElec	7,066	100,667	93,602	< 0.01	607,279	28.49	4	1,065	21.94
WikiConflict	113,123	2,025,910	1,912,788	< 0.01	13,852,201	35.81	4	20,153	62.33
WikiPolitics	137,740	715,334	577,595	< 0.01	2,978,021	10.38	2	10,715	11.98

vertices and detect the largest connected component using union-find. Hence, the total time complexity of the algorithm is $O(I * (|E| * log(|V|) * d_a) + K * (|E| * log(|V|))$. For each of the Phase 2 vertex deletion criteria, we add the following:

ABCDD: Computing the degrees of each vertex and reading the graph takes $O(|E|)$, assuming Harary bipartitions are not computed. So the total beginning-to-end complexity of ABCDD is

$O(I * (|E| * log(|V|) * d_a) + K * (|E| * log(|V|))$.

ABCDH: Obtaining the Harary bipartition takes $O(|V| * |E|)$ because we use the Bellman-Ford algorithm to compute distances on the detected connected components. In case the number of balanced states retrieved is less than K, the time complexity for storing the state and Harary bipartition takes $O(|E|)$. On the other hand, when the number of stable states exceeds K, finding the balanced state with the largest edge sign switches and replacing it with a state of lower frustration takes $O(|E|)$ in total as well. Moreover, computing the degrees of each vertex and reading the graph takes $O(|E|)$. Computing the sum of the neighborhood degrees of each vertex uses the same procedure as calculating the degrees, but it's repeated three times. Hence, $O(|E|)$ is added. So, the total beginning-to-end complexity of ABCDH is

$O(I * (|E| * log(|V|) * d_a) + K * (|E| * log(|V|) + |V| * |E|)$.

ABCDS: The total time complexity is similar to ABCDH as we sum all K $\mathcal{H}_\Sigma$ element-wise, and divide each element in this array by K, which takes $O(K * |V|)$ as we have precomputed the Harary bipartitions. However, it won't be a dominant term, so the total beginning-to-end complexity of ABCDS is the same as ABCDH, which is

$O(I * (|E| * log(|V|) * d_a) + K * (|E| * log(|V|) + |V| * |E|)$.

The time complexity for TIMBAL [17] is $O(|E|+|V|^2+|E|+|V|+|V|*\Delta)$ where Δ is the maximum degree. This time complexity is less than that of the proposed method. However, our method is still scalable, and achieving a larger balanced subgraph is critical, which justifies the additional computational cost. Also, the time complexity of our proposed method is highly dependent on the parameters I and K as they offer a compromise between the size of the balanced subgraph found and the execution time. We substantiate this by running the ABCDH variant with lower values of I and K, and the following section shows that the execution time significantly drops with only a slight decrease in the vertex cardinality of the balanced subgraph in some signed networks.

VI. PROOF OF CONCEPT

All real-world benchmark graphs have one significant connected component, and the implementation of the algorithms is in C++. The algorithm identifies the largest connected component (LCC). It applies the ABCD algorithm to the largest connected component. The implementation treats edges without signs as positive edges in the fundamental cycle. ABCD Phase 1 (Algorithm 1) implementation builds on [16] and has recently shown that the breadth-first search sampling of the spanning trees captures the diversity of the frustration cloud [18]. We adopt the breadth-first search method for sampling spanning trees in Algorithm 1. Algorithm 2 implements ABCD Phase 2, and the final set of largest connected components per stable states is returned in the $\mathcal{ABCD}$ set, and the winner is the largest among them.

Baseline: The TIMBAL approach has reached the highest cardinality of the subgraphs in various datasets and is a de facto state-of-the-art [17]. The input parameters of TIMBAL [17] are set as follows for all subsample_flag=False, samples=4 based on the paper implementation. The parameter max_removals=1 is set for small graphs (under 1000 vertices) and to max_removals=100 for the rest of the signed networks. We set avg_size=20 for datasets of several vertices less than 80,000, and subsample_flag=True, samples = 1000, avg_size = 200, max_removals=100 for datasets with the number of vertices greater than 80,000. TIMBAL is a non-deterministic algorithm, and we run it 5 and 10 times for Konect data to get the maximum vertex cardinality.

The *subsample_flag* boolean flag tells TIMBAL whether to perform subsampling of the network. When set to **False**, the algorithm runs on the entire graph without dividing it into smaller pieces. When **True**, it will use a subsampling approach to extract multiple subgraphs and then work with those. This is useful for very large networks where handling the entire graph might be computationally expensive.

TABLE 3. Comparison of TIMBAL and ABCD $l = 5000$ runs on Konect plus TwitterReferendum and PPI signed graphs. The numbers between the parentheses in the ABCDH column are the subgraph # vertices and time for the faster version of ABCDH with different parameters. t stands for time.

	TIMBAL 5 runs		TIMBAL 10 runs		ABCDH		ABCDS		ABCDD	
Konect Dataset	subgraph # vertices	Run t (s)	subgraph # vertices	Run t (s)	subgraph # vertices	Run t (s)	subgraph # vertices	Run t (s)	subgraph # vertices	Run t (s)
Highland	13	0.1	13	0.2	13 (13)	0.9 (0.18)	12	1.07	12	0.98
CrisisInCloister	8	0.25	8	0.5	8 (8)	1.28 (0.25)	**9**	1.8	7	1.5
ProLeague	10	0.1	10	0.2	10 (10)	1.40 (0.28)	10	1.6	7	1.32
DutchCollege	29	0.1	29	0.2	**30** (30)	14 (2.5)	**30**	14.1	**30**	14.98
Congress	207	0.85	207	1.9	207 (207)	7.79 (1.44)	206	2.38	**210**	9.03
PPI	900	78.45	900	156.9	2,073 (2,033)	97.59 (18.8)	2,038	99.48	**2,149**	105.46
BitcoinAlpha	3,014	5.57	3,081	11.4	3,146 (3,107)	55.68 (14.79)	**3,154**	231.19	3,077	221.64
BitcoinOTC	4,250	10.15	4,349	20.3	**4,910** (4,890)	92.29 (23.48)	4,814	350.12	4,880	361.59
Chess	2,230	15.25	2,320	30.5	**2,551** (2,554)	174.67 (40.72)	2,230	440.98	2,199	443.27
TwitterReferendum	9,021	27,6	9,110	55.2	9,438 (9,263)	851.94 (231.29)	**9,453**	4397.77	8,622	3479.59
SlashdotZoo	39,905	259.4	40,123	518.8	43,544 (43,219)	1870.20 (381.24)	**45,250**	4530.95	41,290	4426.75
Epinions	73,433	774.9	74,106	1549.8	74,843 (74,522)	3272.87 (632.39)	**78,126**	2406	74,555	2385.04
WikiElec	**3,758**	18.95	**3,856**	37.9	3,506 (3,506)	3033.08 (73.43)	3,528	831.78	3,142	805.73
WikiConflict	56,768	539.25	56,768	1078.5	54,476 (53,549)	8332.22 (1979.049)	57,748	8317.99	**60,606**	8344.92
WikiPolitics	67,009	602.65	69,050	1205.3	63,584 (63,584)	3478.08 (672.077)	65,730	2790.39	**69,520**	2779.16

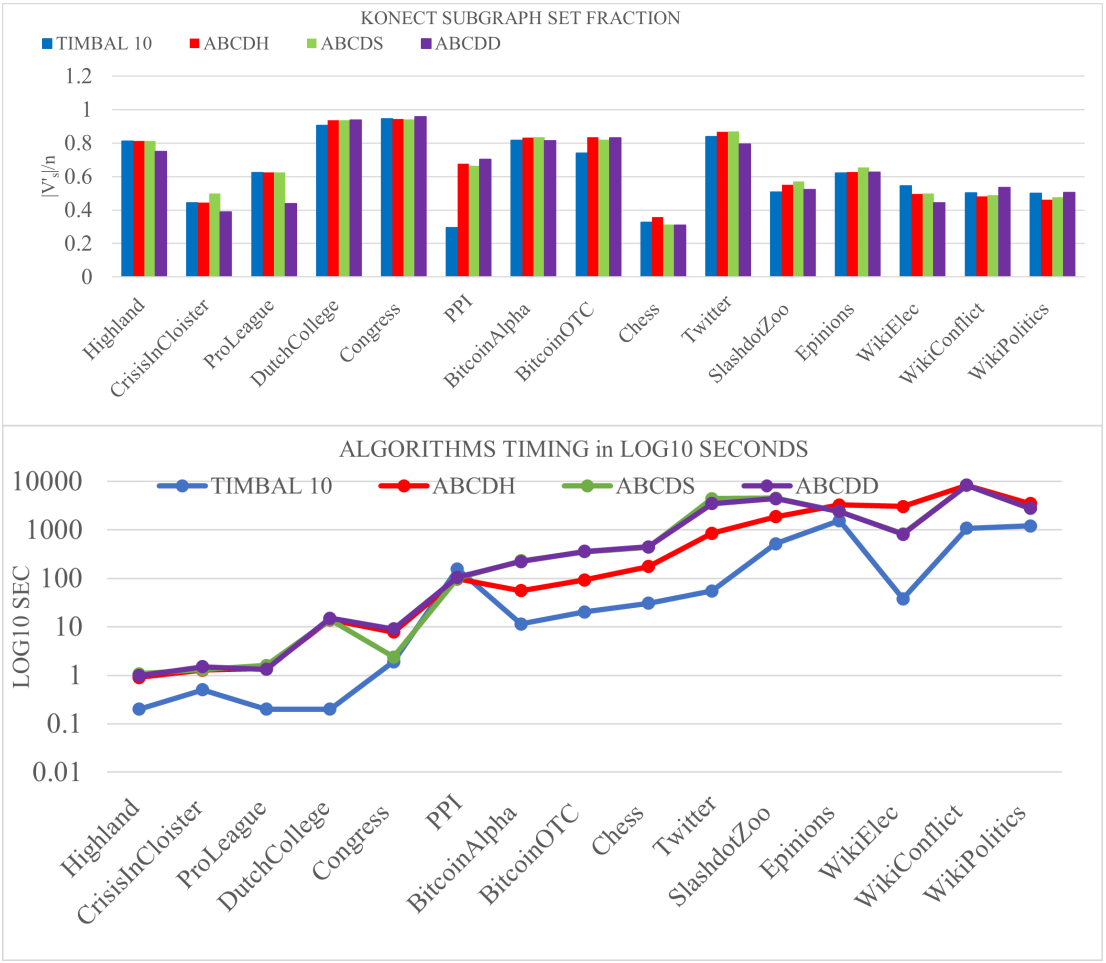

FIGURE 5. **ABCD and TIMBAL performance comparison for Konect benchmark in terms of subset graph fractions (left) and algorithmic timing (right).**

The parameter *samples* is the number of randomly sampled graphs, and it must be a factor of the number of threads used. The *max_removal* parameter sets the maximum number of removed vertices per iteration. The *avg_size* parameter is the approximate desired size of the randomly sampled graph.

Setup: The operating system used for the experiments is Linux Ubuntu 20.04.3, running on the 11th Gen Intel (R) Core (TM) i9-11900 K @ 3.50GHz with 16 physical cores. It has one socket, two threads per core, and eight cores per socket. The architecture is X86_x64. Its driver version is 495.29.05, and the CUDA version is 11.5. The cache configuration is L1d: 384 KiB, L1i: 256 KiB, L2: 4 MiB, L3: 16 MiB. The CPU op is 32-bit and 64-bit.

Comparison: We evaluate three implementations of the ABCD algorithm (ABCDH, ABCDS, ABCDD) against TIMBAL using 14 Konect datasets, the Twitter Ref. and PPI signed graphs, and 17 Amazon datasets. The comparison considers runtime in seconds and the size of the largest produced subgraph.

ABCD algorithm parameters: $I = 5000$ for all, $K = 4000$ for $n < 100,000$, $K = 100$ for $100,000 < n < 300,000$, and $K = 20$ for $300,000 < n$ vertices. **ABCDH_Fast** is a faster version of **ABCDH** and the parameters are: $I = 1000$ for all, $K = 700$ for $n < 100,000$, $K = 100$ for $100,000 < n < 300,000$, and $K = 20$ for $300,000 < n$ vertices. For this faster version, we can also study the effect of decreasing the number of iterations on the overall speed and performance.

A. TIMBAL VS. ABCD FOR THE KONECT BENCHMARK

Konect signed graphs are from [36], and their characteristics are described in Table 2. *Highland* is the signed social network of tribes of the GahukuGama alliance structure of the Eastern Central Highlands of New Guinea, from Kenneth Read. *CrisisInCloister* is a directed network that contains ratings between monks related to a crisis in an abbey (or monastery) in New England (USA), which led to the departure of several of the monks. *ProLeague* are results of football games in Belgium from the Pro League in 2016/2017 in the form of a directed signed graph. Vertices are teams; each directed edge from A to B denotes that team A played at home against team B. The edge weights are the goal difference, and thus, favorable if the home team wins, negative if the away team wins, and zero for a draw. *DutchCollege* is a directed network that contains friendship ratings between 32

TABLE 4. Amazon ratings and reviews [35] mapped to signed graphs. The number of vertices, edges, and cycles reflects the number in the largest connected component of each dataset.

AMAZON Ratings	Input graph # ratings	Largest Connected Component # vertices	# edges	# cycles
Books	22,507,155	9,973,735	22,268,630	12,294,896
Electronics	7,824,482	4,523,296	7,734,582	3,211,287
Jewelry	5,748,920	3,796,967	5,484,633	1,687,667
TV	4,607,047	2,236,744	4,573,784	2,337,041
Vinyl	3,749,004	1,959,693	3,684,143	1,724,451
Outdoors	3,268,695	2,147,848	3,075,419	927,572
AndrApp	2,638,172	1,373,018	2,631,009	1,257,992
Games	2,252,771	1,489,764	2,142,593	652,830
Automoto	1,373,768	950,831	1,239,450	288,620
Garden	993,490	735,815	939,679	203,865
Baby	915,446	559,040	892,231	333,192
Music	836,006	525,522	702,584	177,063
Video	583,993	433,702	572,834	139,133
Instruments	500,176	355,507	457,140	101,634
Reviews	**# reviews**	**# vertices**	**# edges**	**# cycles**
Core Music	64,706	9,109	64,706	55,598
Core Video	37,126	6,815	37,126	30,312
Core Instrum	10,621	2,329	10,261	7,933

first-year university students (vertices) who mostly did not know each other before starting university. Students rate each other at seven different time points. An edge between two students shows how the reviewer rated the target, and the edge weights show how good their friendship is in the eye of the reviewer. The weight ranges from -1 for the risk of conflict to +3 for best friend. *Congress* is a signed network where vertices are politicians speaking in the United States Congress, and a directed edge denotes that a speaker mentions another speaker. In the *Chess* network, each vertex is a chess player, and a directed edge represents a game with the white player having an outgoing edge and the black player having an ingoing edge. The weight of the edge represents the outcome. *BitcoinAlpha* is a user-user trust/distrust network from the Bitcoin Alpha platform for Bitcoin trading. *BitcoinOTC* is a user-user trust/distrust network from the Bitcoin OTC platform for Bitcoin trading. *TwitterReferendum* captures data from Twitter concerning the 2016 Italian Referendum. Different stances between users signify a negative tie, while the same stances indicate a positive link [39].

WikiElec is the network of users from the English Wikipedia who voted for and against each other in admin elections. *SlashdotZoo* is the reply network of the technology website, Slashdot. Vertices are users, and edges are replies. The edges of *WikiConflict* represent positive and negative conflicts between users of the English Wikipedia. *WikiPolitics* is an undirected signed network that contains interactions between the users of the English Wikipedia who have edited pages about politics. Each interaction, such as text editing and votes, is valued positively or negatively. *Epinions* is the trust and distrust network of Epinions, an online product rating site. It incorporates individual users connected by directed trust and distrust links. *PPI* models the protein-protein interaction network [40].

The first benchmark consists of 14 signed graphs from the Konect repository [36](plus TwitterRef. and PPI) used in TIMBAL benchmark evaluations. Fig. 5 and Table 3 summarize baseline TIMBAL and proposed ABCD performance. The ABCD matched TIMBAL performance in two networks (Highland and Proleague). ABCD algorithm finds a more significant subset for 13 Konect datasets than TIMBAL. On the contrary, TIMBAL performs better on only one Konect signed graph (WikiElec). TIMBAL is faster than ABCD on smaller networks. For the largest graph in the collection, Epinions, ABCD takes double the time to recover the largest balanced subgraph. We recorded the maximum number of vertices obtained after 5 and 10 runs for TIMBAL, and only for one dataset did the repeated runs discover a more significant subset. Table 3 also summarizes the results of the ABCDH_Fast performance in parentheses in the ABCDH column. For this benchmark, ABCD algorithms outperform TIMBAL with comparable runtimes.

B. TIMBAL VS. ABCD FOR THE AMAZON BENCHMARK

Amazon benchmark consists of 17 signed graphs derived from the Amazon rating and review files [35]. The dataset contains product reviews and metadata from Amazon, spanning from May 1996 to July 2014. The rating score is mapped into an edge between the user and the product as follows: $(5, 4) \rightarrow e^+$, $3 \rightarrow e$ (no sign), and $(2, 1) \rightarrow e^-$ [35].

Table 4 summarizes the Amazon data used and the characteristics of the largest connected component of the graph. Fig. 6 (left) and Table 5 illustrate the subgraph size TIMBAL recovers (blue box) and the subgraph ABCD algorithm recovers (red box). Amazon data is extensive. The ABCD algorithm performs much better for millions of vertices than TIMBAL, especially the ABCDD version. One iteration of TIMBAL (blue line) takes as long as the entire ABCD algorithm (red line) for larger graphs. In this experiment, the ABCD algorithm has a superior runtime and performance regarding the graph size it discovers, as illustrated in Fig. 6 (right). TIMBAL's performance degrades with the graph size, and the discovered subgraphs are much smaller than what ABCD finds, as described in Fig. 6 (left).

C. ABCD ABLATION STUDY

First, we select the ABCDH version of ABCD and study the effect of the two parameters I and K on the balanced subgraph size. First, we set I to 1000 and vary K from 1 to 5 and then $K \in \{10, 20, 30, 40, 50\}$ on Epinions, SlashdotZoo, and TwitterReferendum, and Table 6 summarizes the results. The ABCD found a larger balanced subgraph of vertex cardinality 74,522 when increasing K to 10 for Epinions. When we vary the number of iterations I with $K = 5$, the results in Table 7 show a reduction in the size of the largest balanced subgraph found with a greater value of I. Thus, the vertex cardinality of the discovered subgraph increases when the K is larger. For $K = 100$, Table 8 summarizes the improvements as the size of the subgraph increases for comparable Table 7 performance. The execution timing also increases, and we use the size of

TABLE 5. Comparison between TIMBAL and ABCD on Amazon ratings and reviews [35] mapped to signed graphs. The time for ABCD is for $I = 5000$ iterations, and the time for TIMBAL is for all runs. N/A - TIMBAL DOES NOT COMPLETE WITHIN 48 hours. t stands for time.

AMAZON Ratings	TIMBAL 1 run # vertices	t (hr)	ABCDH # vertices	t (hr)	ABCDS # vertices	time (hr)	ABCDD # vertices	t (hr)
Books	N/A	N/A	7,085,285	32.5	6,265,058	32.97	**7,458,256**	32.85
Electronics	N/A	N/A	3,104,399	10.5	2,031,543	10.62	**3,689,985**	10.67
Jewelry	530,363	13.1	2,769,431	6	2,237,260	5.96	**2,949,384**	6.05
TV	891,106	3.16	1,579,760	4.76	1,299,795	4.84	**1,795,706**	4.84
Vinyl	612,700	3.2	1,452,496	3.61	1,358,541	3.70	**1,474,459**	3.68
Outdoors	683,846	3.53	1,640,544	3.14	1,400,498	3.16	**1,823,824**	3.17
AndrApp	437,740	1.4	977,536	3.4	636,566	3.42	**1,133,649**	3.45
Games	565,301	1.74	1,150,782	2.12	1,042,898	2.16	**1,261,748**	2.17
Automoto	140,711	3.61	744,474	1.15	685,805	1.17	**801,708**	1.18
Ratings	**# vertices**	**t (min)**	**# vertices**	**t (min)**	**# vertices**	**t (min)**	**# vertices**	**t (min)**
Baby	229,545	60	397,940	50	300,996	50.34	**468,446**	50.67
Music	351,124	53.7	451,320	36.7	428,561	37.53	**471,928**	37.15
Video	121,694	71.3	360,665	36.2	318,484	36.32	**401,236**	36.7
Instruments	97,486	30	285,233	24.4	273,250	24.55	**313,811**	24.89
Reviews	**# vertices**	**t (s)**	**# vertices**	**time (s)**	**# vertices**	**t (s)**	**# vertices**	**t (s)**
Core Music	4,193	30.3	**5,143**	200.4	4,963	548.64	3,595	527.18
Core Video	3,419	23.7	**3,934**	128.3	3,740	322.12	2,552	318.3
Core Instrum	**1,725**	19.1	1,559	36.9	1,535	110.27	1,272	98.31

TABLE 6. Ablation Study of ABCDH on several signed graphs with varying K with $I = 1000$.

Data	Epinions		SlashdotZoo		TwitterRef.	
K	# vertices	time(s)	# vertices	time(s)	# vertices	time(s)
1	72,417	448.21	41,919	286.24	8,965	147.3
2	72,895	449.65	43,171	287.61	9,255	147.76
3	73,664	454.02	43,189	288.09	9,255	148.25
4	73,664	453.52	43,189	289.34	9,255	149.16
5	73,664	454.73	43,189	289.95	9,255	150.53
10	74,522	467.15	43,189	294.67	9,255	155.18
20	74,522	487.90	43,189	304.84	9,263	163.93
30	74,522	507.6	43,189	315.96	9,263	172.45
40	74,522	523.80	43,189	324.53	9,263	181.23
50	74,522	544.65	43,189	335.62	9,263	189.6

TABLE 7. Ablation Study of ABCDH on several signed graphs with varying I with $K = 5$.

	Epinions		SlashdotZoo		Twitter Ref.	
$I \downarrow$	# vertices	time(s)	# vertices	time(s)	# vertices	time(s)
10	74,209	18.45	43,219	10.36	9,161	7.38
1000	73,664	455.1	43,189	292.57	9,255	160.59
2000	74,053	887.83	43,338	568.67	9,200	293.51
4000	73,717	1750.01	43,885	1130.36	9,243	593.98
5000	72,949	22190.82	43,154	1411.77	9,193	751.85

the frustration cloud K as a balancing barometer between the size of the subgraph found and execution time.

Next, as the number of iterations I for the larger $K = 100$ increases, the size of the discovered balanced subgraph also increases for the ABCD method. The execution time also increases. However, for more iterations, the algorithm generates more. We can also observe that the more stable states we generate (the greater the value of I), the greater we must increase K to capture the subgraph with the largest vertex cardinality.

TABLE 8. Ablation Study of ABCDH on several signed graphs with varying I with $K = 100$.

	Epinions		SlashdotZoo		Twitter Ref.	
$I \downarrow$	# vertices	time(s)	# vertices	time(s)	# vertices	time(s)
1000	74,522	634.55	43,219	384.42	9,263	232.84
2000	74,866	1075.73	43,338	665.4	9,276	374.54
3000	74,365	1500.92	43,683	952.35	9,262	521.62
4000	74,867	1943.77	43,885	1228.63	9,263	673.2
5000	74,843	2377.75	43,544	1515.50	9,228	830.33

The trends in Table 7 ($K = 5$) and Table 8 ($K = 100$) appear contrad

D. EFFECT OF SPARSITY ON ABCD

We generate random signed Albert-László Barabási graphs [42] using the SigNet package [43] with varying sparsity values $p \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. We set the number of nodes to 100, the number of communities to 4, and the noise value to 0.2. We select to run the ABCDS variant with $I = 5000$ and $K = 4000$, and the results are in Fig. 7. As expected, the sparsity value does not matter in terms of finding a larger balanced subgraph, as it all comes down to the assigned edge signs of the signed network. This is evidenced by the fact that # vertices of the largest balanced subgraph found increases arbitrarily when the sparsity value reaches 1.0. In addition, as the signed network gets denser, there will be more fundamental cycles, and the computational cost and execution time of traversing every cycle by our algorithm for balancing and flipping signs increase. However, real-world signed graphs are typically sparse, resulting in significantly reduced computational demands and execution times, as seen in Table 2.

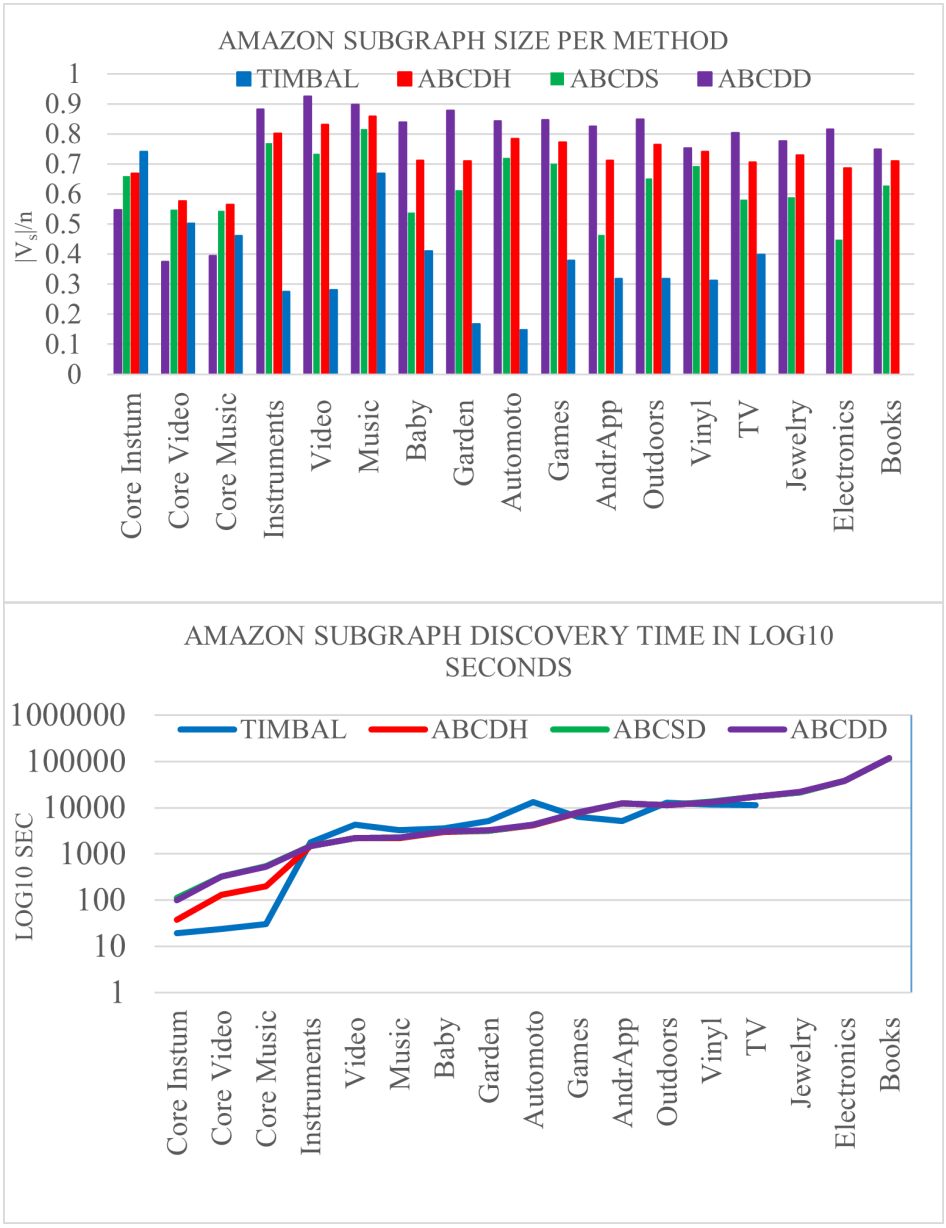

FIGURE 6. ABCD and TIMBAL performance and running time comparison for Amazon data.

VII. CONCLUSION

Finding maximum balanced subgraphs is a fundamental problem in graph theory with significant practical applications. While the situation is computationally challenging, the existing approximation algorithms have made considerable progress in solving it efficiently for many signed networks, and propose a novel, scalable algorithm for balance component discovery (ABCD). We capture the information on the unbalanced fundamental cycles and the Harary bipartition labeling for the top unique total cycle bases with the lowest number of unstable cycles. A balanced state with the lowest frustration index for a specific signed network does not necessarily yield a maximum balanced subgraph. A balanced state with a high frustration index skyrockets the number of vertices discarded due to the increase in the number of candidate vertices and edges to be processed. We introduce a novel set of conditions (neighborhood degree, bi-cut) to remove the vertices from the graph. The output of the ABCD algorithm is guaranteed to be balanced. ABCD eliminates the unbalanced cycle bases by removing the edges. Thus, the cycle turns into an open path. The resulting subgraph has the largest size regarding the number of vertices; it is balanced as it has no unbalanced cycles, and it is a subgraph as the algorithm removes the *vertices*. Overall, ABCD's unique contribution lies in taking advantage of the top-K states with the lowest frustration and carefully selecting vertices for removals along candidate edges. This enables it to obtain substantially larger balanced subgraphs than TIMBAL which relies on eigenvalue

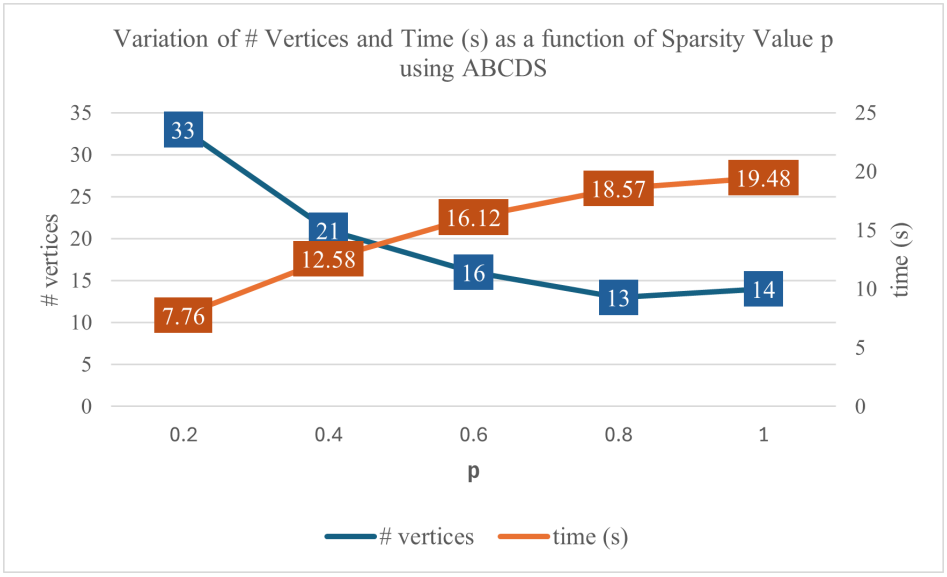

FIGURE 7. **Variation of largest balanced subgraph size and execution time as a function of sparsity value using the ABCDS variant with $I = 5000$ and $K = 4000$.**

computations and subsampling the signed graph.

Future work includes the OpenMP and GPU code accelerations as the GPU implementation of the baseline takes less than 15 minutes to find 1000 fundamental cycle bases for 10M vertices and 22M edges [16]. Since the runtime is roughly proportional to the input size, the ABCD parallel implementation can balance ten times larger inputs in a few seconds per sample, making it tractable to analyze graphs with 100s of millions of vertices and edges.

REFERENCES

[1] Y. Wu, D. Meng, and Z.-G. Wu, "Disagreement and antagonism in signed networks: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 7, pp. 1166–1187, 2022.

[2] R. P. Abelson and M. J. Rosenberg, "Symbolic psycho-logic: A model of attitudinal cognition," *Behavioral Science*, vol. 3, no. 1, pp. 1–13, 1958.

[3] F. Heider, "Attitudes and cognitive organization," *Journal of Psychology*, vol. 21, pp. 107–112, 1946.

[4] D. Cartwright and F. Harary, "Structural balance: A generalization of Heider's theory," *Psychological Review*, vol. 63, pp. 277–293, 1956.

[5] F. Harary and D. Cartwright, "On the coloring of signed graphs," *Elemente der Mathematik*, vol. 23, pp. 85–89, 1968.

[6] V. Amelkin and A. K. Singh, "Fighting opinion control in social networks via link recommendation," in *Proc. 25th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining (KDD)*, pp. 677–685, 2019.

[7] T. Derr, Z. Wang, J. Dacon, and J. Tang, "Link and interaction polarity predictions in signed networks," *Social Network Analysis and Mining*, vol. 10, no. 1, pp. 1–14, 2020.

[8] K. Garimella, T. Smith, R. Weiss, and R. West, "Political polarization in online news consumption," in *Proc. Int. AAAI Conf. Web and Social Media*, vol. 15, pp. 152–162, 2021.

[9] R. Interian, R. G. Marzo, I. Mendoza, and C. C. Ribeiro, "Network polarization, filter bubbles, and echo chambers: An annotated review of measures, models, and case studies," *arXiv preprint*, arXiv:2207.13799, 2022.

[10] F. Harary, M.-H. Lim, and D. C. Wunsch, "Signed graphs for portfolio analysis in risk management," *IMA Journal of Management Mathematics*, vol. 13, no. 3, pp. 201–210, 2002.

[11] R. Figueiredo and Y. Frota, "The maximum balanced subgraph of a signed graph: Applications and solution approaches," *European Journal of Operational Research*, vol. 236, no. 2, pp. 473–487, 2014.

[12] K. T. Macon, P. J. Mucha, and M. A. Porter, "Community structure in the United Nations General Assembly," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 1, pp. 343–361, 2012.

[13] C. Liu, Y. Dai, K. Yu, and Z. Zhang, "Enhancing cancer driver gene prediction by protein–protein interaction network," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 19, no. 4, pp. 2231–2240, 2022.

[14] C. Chen, Y. Wu, R. Sun, and X. Wang, "Maximum signed θ-clique identification in large signed graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 2, pp. 1791–1802, 2023.

[15] T. Zaslavsky, "A mathematical bibliography of signed and gain graphs and allied areas," *Electron. J. Combin.*, 2012.

[16] G. Alabandi, J. Tešić, L. Rusnak, and M. Burtscher, "Discovering and balancing fundamental cycles in large signed graphs," in *Proc. Int. Conf. High Performance Comput., Netw., Storage and Anal. (SC)*, New York, NY, USA, 2021, Association for Computing Machinery.

[17] B. Ordozgoiti, A. Matakos, and A. Gionis, "Finding large balanced subgraphs in signed networks," in *Proc. Web Conf. (WWW)*, pp. 1378–1388, New York, NY, USA, 2020, Association for Computing Machinery.

[18] L. Rusnak and J. Tešić, "Characterizing attitudinal network graphs through frustration cloud," *Data Mining and Knowledge Discovery*, vol. 6, 2021.

[19] G. Facchetti, G. Iacono, and C. Altafini, "Computing global structural balance in large-scale signed social networks," *Proc. Natl. Acad. Sci. USA*, vol. 108, no. 52, pp. 20953–20958, 2011.

[20] F. Berger, P. Gritzmann, and S. de Vries, "Minimum cycle bases for network graphs," *Algorithmica*, vol. 40, no. 1, pp. 51–62, 2004.

[21] N. Gülpinar, G. Gutin, G. Mitra, and A. Zverovitch, "Extracting pure network submatrices in linear programs using signed graphs," *Discrete Appl. Math.*, vol. 137, no. 3, pp. 359–372, 2004.

[22] S. Poljak and D. Turzík, "A polynomial-time heuristic for certain subgraph optimization problems with guaranteed worst-case bound," *Discrete Math.*, vol. 58, no. 1, pp. 99–104, 1986.

[23] R. Crowston, G. Gutin, M. Jones, and G. Muciaccia, "Maximum balanced subgraph problem parameterized above the lower bound," *Theor. Comput. Sci.*, vol. 513, pp. 53–64, 2013.

[24] R. M. Figueiredo, M. Labbé, and C. C. de Souza, "An exact approach to the problem of extracting an embedded network matrix," *Comput. Oper. Res.*, vol. 38, no. 11, pp. 1483–1492, 2011.

[25] F. Bonchi, E. Galimberti, A. Gionis, B. Ordozgoiti, and G. Ruffo, "Discovering polarized communities in signed networks," in *Proc. 12th ACM Int. Conf. Web Search and Data Mining (WSDM)*, pp. 195–203, 2019.

[26] K. Sharma, I. A. Gillani, S. Medya, S. Ranu, and A. Bagchi, "Balance maximization in signed networks via edge deletions," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD)*, pp. 752–760, New York, NY, USA, 2021, Association for Computing Machinery.

[27] L. Boulton, "Spectral pollution and eigenvalue bounds," *Appl. Numer. Math.*, vol. 99, pp. 1–23, 2016.
[28] Y. Shang, "On the structural balance dynamics under perceived sentiment," *Bull. Iranian Math. Soc.*, vol. 46, no. 3, pp. 717–724, 2020.
[29] J. Kleinberg, M. Sandler, and A. Slivkins, "Network failure detection and graph connectivity," *SIAM J. Comput.*, vol. 38, no. 4, pp. 1330–1346, 2008.
[30] K. Deng, H. Zhao, and D. Li, "Effect of node deleting on network structure," *Physica A*, vol. 379, no. 2, pp. 714–726, 2007.
[31] M. Lalou, M. A. Tahraoui, and H. Kheddouci, "The critical node detection problem in networks: A survey," *Comput. Sci. Rev.*, vol. 28, pp. 92–117, 2018.
[32] Y. Shang, "Upper bounds on the order of nearly regular induced subgraphs in random graphs," *An. Univ. Craiova Ser. Mat. Inform.*, vol. 38, 2011.
[33] Y. Shang, "Poisson approximation of induced subgraph counts in an inhomogeneous random intersection graph model," *Bull. Korean Math. Soc.*, vol. 56, no. 5, pp. 1199–1210, 2019.
[34] R. Tarjan, "A note on finding the bridges of a graph," *Inf. Process. Lett.*, vol. 2, no. 6, pp. 160–161, 1974.
[35] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *Proc. 25th Int. Conf. World Wide Web (WWW)*, pp. 507–517, 2016.
[36] J. Kunegis, "KONECT – The Koblenz Network Collection," in *Proc. 22nd Int. Conf. World Wide Web (WWW)*, pp. 1343–1350, 2013.
[37] B. Anna D. and C. Aaron, "Scale-free networks are rare," *Nat. Commun.*, vol. 10, no. 1, pp. 1–10, 2019.
[38] M. Tomasso, L. Rusnak, and J. Tešić, "Advances in scaling community discovery methods for signed graph networks," *J. Complex Netw.*, vol. 10, no. 3, 2022.
[39] M. Lai, V. Patti, G. Ruffo, and P. Rosso, "Stance evolution and Twitter interactions in an Italian political debate," in *Natural Language Processing and Information Systems*, M. Silberztein, F. Atigui, E. Kornyshova, E. Métais, and F. Meziane, Eds. Cham: Springer, 2018, pp. 15–27.
[40] Y. He, G. Reinert, S. Wang, and M. Cucuringu, "SSSNET: Semi-supervised signed network clustering," in *Proc. SIAM Int. Conf. Data Min. (SDM)*, pp. 244–252, 2022.
[41] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Signed networks in social media," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. (CHI)*, pp. 1361–1370, New York, NY, USA, 2010.
[42] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.
[43] Alan Turing Institute, "SigNet: A signature verification tool," GitHub repository, 2025. [Online]. Available: https://github.com/alan-turing-institute/SigNet

MUHIEDDINE SHEBARO, PH.D. is currently an independent researcher. He received his B.S. degree in Computer Science from Beirut Arab University, Lebanon, in 2021, and the Ph.D. degree in Computer Science from Texas State University, USA, in 2025. His research interests include network science, machine learning, and data science.

JELENA TEŠIĆ, PH.D. is an Associate Professor at Texas State University. Before that, she was a research scientist at Mayachitra (CA) and IBM Watson Research Center (NY). She received her Ph.D. (2004) and M.Sc. (1999) in Electrical and Computer Engineering from the University of California Santa Barbara, CA, USA, and Dipl. Ing. (1998) in Electrical Engineering from the University of Belgrade, Serbia. Dr. Tešić served as Area Chair for ACM Multimedia 2019-present and IEEE ICIP and ICME conferences; she has served as Guest Editor for IEEE Multimedia Magazine for the September 2008 issue and as a reviewer for numerous IEEE and ACM Journals. She has authored over 40 peer-reviewed scientific papers and holds six US patents. Her research focuses on advancing the analytic application of EO remote sensing, namely object localization and identification at scale.