ABCD: ALGORITHM FOR BALANCED COMPONENT DISCOVERY IN SIGNED NETWORKS

Muhieddine Shebaro and Jelena Tešić

January 31, 2025

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. In this paper, we propose a scalable state-of-the-art approach for the maximum balanced sub-graph detection in the network of *any* size. The proposed approach finds the largest balanced sub-graph by considering only the top K balanced states with the lowest frustration index. We show that the ABCD method selects a 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.

balanced sub-graph, frustration index, balanced states, and signed graphs

1 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 model the conflicting opinions between the vertices well [Wu et al., 2022]. Balance theory represents a theory of changes in attitudes [Abelson and Rosenberg, 1958]: people's attitudes evolve in networks so that friends of a friend will likely become friends, and so will enemies of an enemy [Abelson and Rosenberg, 1958]. Heider established the foundation for social balance theory [Heider, 1946], and Harary established the mathematical foundation for signed graphs and introduced the k-way balance [Cartwright and Harary, 1956, Harary and Cartwright, 1968]. The solutions to the tasks to predict edge sentiment, to recommend content and products, or to identify unusual trends have had balanced theory at its core [Derr et al., 2020, Garimella et al., 2021, Interian et al., 2022, Amelkin and Singh, 2019]. The task of the **most extensive balanced sub-graph** discovery has applications in portfolio system's economic risk management [Harary et al., 2002], computational and operational research [Figueiredo and Frota, 2014], community analysis and structure [Macon et al., 2012], computational biology to model balanced interactions between genes and proteins [Liu et al., 2022] and social network analysis [Chen et al., 2023]. The vertices that are part of the maximum balanced sub-graph Σ' of Σ may not necessarily have a high degree of centrality between them. By locating the largest balanced sub-graphs, we can simplify the system into sub-systems with balanced interactions and eliminate inconsistencies regarding unbalanced cycles. The largest balanced sub-graph is the largest possible subset of the signed network that satisfies the balance theory (every cycle has an even number of negative edges). This sub-graph doesn't have to be a **clique** where a clique is a subset of nodes in the graph where every node has an edge over every other node in that subset.

Finding the largest balanced sub-graph is a well-known NP-hard problem [Zaslavsky, 2012], and existing solutions do not scale to real-world graphs [Wu et al., 2022]. This paper proposes an algorithm for balanced component discovery (ABCD) in signed graphs, and we show that it discovers *larger* signed sub-graphs faster than TIMBAL. The approach builds on the scalable discovery of fundamental cycles in [Alabandi et al., 2021] and utilizes the graph's vertex density distribution and stable states to minimize the number of vertices removed from the balanced sub-graph. Section 2 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 4 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 sub-graph. In Section 5, we analyze the complexity of our algorithm. Section 6 compares the proposed method to the state-of-art method proposed in [Ordozgoiti et al., 2020]. The TIMBAL method achieved the highest vertex cardinality (number of vertices in the largest balanced sub-graph) across all signed graphs, among other baselines in the literature [Ordozgoiti et al., 2020]. We evaluate our algorithm on the Konect and Amazon signed graphs in Subsection 6.1 and Subsection 6.2, respectively. Next, we perform an ablation study on our proposed algorithm in Subsection 6.3. In Section 7, 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.

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

where $Fr(\Sigma^G)$ is the frustration of balanced sub-graph Σ^G defined in [Rusnak and Tešić, 2021]. 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 sub-graph 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.

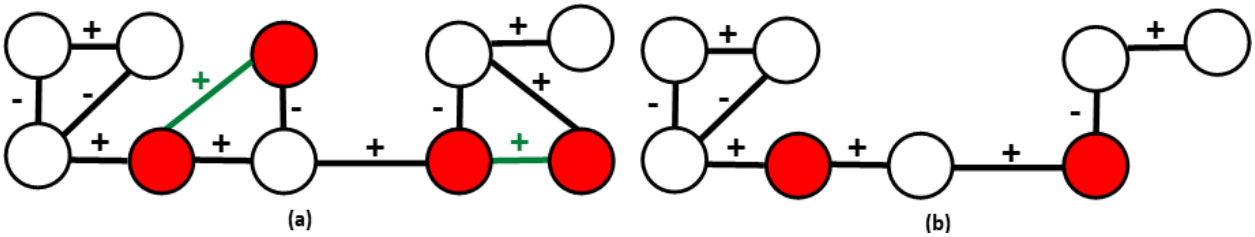

Figure 1: (a): The unbalanced signed network. Green edges are the candidate edges causing imbalance, and red vertices are the candidate vertices. (b): The maximum balanced signed sub-graph obtained after deleting one candidate vertex along each edge.

2 Definitions & Corollaries

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.

Definition 2.1 ***Signed graph*** $\Sigma = (G, \sigma)$ *consists of underlying unsigned graph* G *and an edge signing function* $\sigma : e \to \{+1, -1\}$ *[Leskovec et al., 2010]. The edge* $|E|$ *can be positive* e^+ *or negative* e^-. ***Sign*** *of a sub-graph is* product *of the edges 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* ***sub-graph*** *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 the balanced state [Facchetti et al., 2011].*

Definition 2.4 ***Cycle Basis*** *is a set of simple cycles that forms a basis of the cycle space [Berger et al., 2004]. A* ***Bridge node*** *is a vertex whose removal increases the number of connected components within the network.*

Definition 2.5 *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*.* ***The fundamental cycle*** *is any cycle that is built using path in* T *plus edge* e *in graph* G*.*

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 unsigned graph* G *with* $|V|$ *vertices and* $|E|$ *edges has precisely* $|V| - |E| + 1$ *fundamental cycles.*

Table 1: Summary of notations and their meanings.

Notation	Meaning
Σ/ G	signed network/ its underlying graph
Σ^G	sub-graph
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, $\|U\| \geq \|W\|$
$\mathcal{F}_\Sigma$	the frustration cloud set of Σ
$\|\mathcal{F}_\Sigma\|$	the size of the frustration cloud.
K	# of nearest balanced states w lowest frustration index, $K \leq \|\mathcal{F}_\Sigma\|$
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 sub-graphs after $V/\mathcal{V}_i$

Definition 2.6 *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 [Rusnak and Tešić, 2021]*

Theorem 2.2 *If a signed sub-graph Σ^G is balanced, the following are equivalent [Cartwright and Harary, 1956]:*

1. *Σ^G is balanced. (All circles 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.*

3 Related Work

Finding the stable sub-graph 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 [Gülpinar et al., 2004]. Poljak and Daniel Turzík show that any signed graph that has $|V|$ vertices and $|E|$ edges contains a balanced sub-graph with at least $0.5|E| + 0.25(|V| - 1)$ edges [Poljak and Turzík, 1986]. Crowston et al. [Crowston et al., 2013] propose a discovery of a balanced sub-graph 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 sub-graph [Figueiredo et al., 2011]. Then, they proposed GRASP, an improved algorithm version with the pre-processing and heuristic methods [Figueiredo and Frota, 2014]. The GRASP algorithm randomly selects a subset of vertices. Greedily adds vertices that maximize the number of edges connecting them to the current subset while keeping the size of the subset balanced [Figueiredo and Frota, 2014]. The EIGEN algorithm [Bonchi et al., 2019] 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 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 sub-graph is obtained [Sharma et al., 2021]. 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 [Ordozgoiti et al., 2020]. Both algorithms rely on signed spectral theory. The approaches do not scale to the large signed graphs as they rely on the costly eigenvalue computation ($O(|E|^2)$), and its performance decreases due to the spectral pollution in eigenvalue computation [Boulton, 2016]. TIMBAL proposes a novel bound for perturbations of the graph Laplacian pre-processing 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 sub-graphs of a specific size and recovering them [Ordozgoiti et al., 2020].

Note that the Maximizing balance via edge deletions (MBED) task is *different* than the task of discovering the maximal balanced sub-graph. 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 [Sharma et al., 2021]. Discovering the maximal balanced sub-graph *does not require* community and budget specifications.

Preserving connectivity when deleting vertices is critical to maximizing the chances of obtaining a large, balanced sub-graph. Kleinberg et al. [Kleinberg et al., 2008] 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 [Kleinberg et al., 2008].

Deng et al. [Deng et al., 2007] 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 [Lalou et al., 2018]. 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.

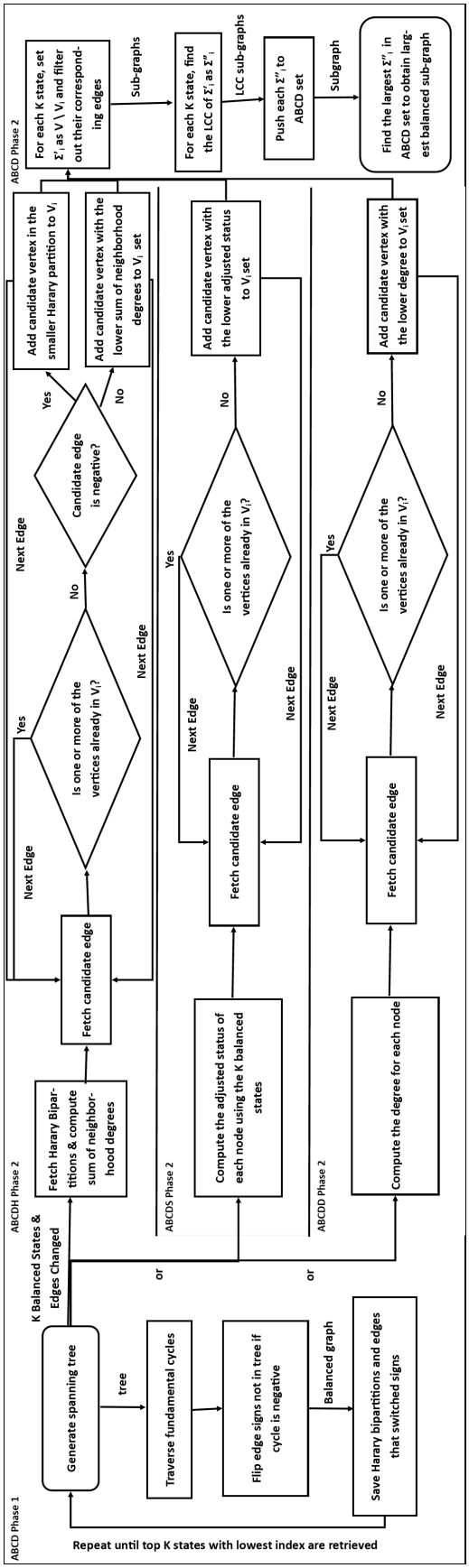

Figure 2: The ABCD pipeline.

4 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 [Alabandi et al., 2021, Rusnak and Tešić, 2021]. Section 4.1 outlines the ABCD Phase 1 approach of collecting multiple candidate edge sets for deletion. Section 4.2 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 Candidate Edge Identification" and Phase 2 as "Candidate Vertex Purging and Largest Sub-graph Retrieval." Figure 1 demonstrates an example execution of our algorithm where the balancing algorithm identifies the candidate edges causing imbalance (green) [Rusnak and Tešić, 2021]. One of the red vertices along these candidate edges has been removed based on handling criteria.

4.1 ABCD Phase 1

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 $\mathcal{F}_\Sigma$, $\mathcal{E}_\Sigma$, and $\mathcal{H}_\Sigma$
Initialize $c = 0$
for $i = 0$; i++; $i <$ I **do**
    Create empty set M_i
    for edges e, $e \in \Sigma \setminus T_i$ **do**
        if fundamental cycle $T_i \cup e$ is negative **then**
            Add edge e to M_i
        end if
    end for
    if $|\mathcal{F}_\Sigma| < K$ **then**
        $\mathcal{F}_\Sigma[c] = |M_i|$
        $\mathcal{E}_\Sigma[c] = M_i$
        Fetch H_i by executing Algorithm 4 with inputs Σ and M_i
        $\mathcal{H}_\Sigma[c]$=H_i
        $c = c + 1$
    else
        Get index l such that $\mathcal{F}_\Sigma[l]$ is the largest
        if $\mathcal{F}_\Sigma[l] < |M_i|$ **then**
            Delete $\mathcal{F}_\Sigma[l]$, $\mathcal{E}_\Sigma[c]$, and $\mathcal{H}_\Sigma[c]$
            $\mathcal{F}_\Sigma[l] = |M_i|$
            $\mathcal{E}_\Sigma[l] = M_i$
            Fetch H_i by executing Algorithm 4 with inputs Σ and M_i
            $\mathcal{H}_\Sigma[l]$=H_i
        end if
    end if
end for
Return $\mathcal{F}_\Sigma$, $\mathcal{E}_\Sigma$, and $\mathcal{H}_\Sigma$

The 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 the 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).

4.2 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 sub-graph. 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 sub-graph. 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 over 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 [Tarjan, 1974]. 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.

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 $\mathcal{V}_\} = \emptyset$
  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.

Here, we propose THREE efficient approximations of connectivity that rely on the scale-free characteristic of the prominent real graphs [Kunegis, 2013, He and McAuley, 2016]. 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, 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 [Anna D. and Aaron, 2019] and recently analyzed real signed graphs exhibit scale-free behavior [Tomasso et al., 2022].

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 or equal to $\mathcal{F}_\Sigma[i]$ for the i^{th} nearest balanced state saved as an output of the Algorithm 1. Let's 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 $|E| - |V| + 1$. The worst-case scenario for finding the largest balanced sub-graph would be for the graph that requires disconnecting each of the $|E| - |V| + 1$ cycles, and none of the edges removed along these cycles share 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 Phase 2 steps for each stable state. First, we initialize an empty set $\mathcal{V}_i$ to save the vertices from being deleted along the candidate edges (edges that flip signs). If either vertex is already in $\mathcal{V}_i$, move on to the next edge. 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 4.2.1, app = 1; (2) ABCDH: Subsection 4.2.2, app = 2; (3) ABCDS: Subsection 4.2.3, app = 3. We repeat this step K times, $\forall i$, $i \in [1, K]$. The resulting outcome of the Algorithm 2 is the set V_i of vertices to be erased for each of the K stable states where V_i is going to be used to obtain the graph Σ_i'' as the largest connected component of $(V \setminus \mathcal{V}_i, E \setminus \mathcal{E}_\Sigma[i])$. Finally, each K Σ_i'' will be stored in $\mathcal{ABCD}$, and the largest $Sigma_i'$ will be returned as the outcome for this task. Figure 2 summarizes the entire ABCD pipeline with Phase 2 containing the three handling criteria for purging vertices split by horizontal lines.

In the next three subsections, we will describe the three approaches in this phase that are efficient approximations for connectivity modeling. Figure 4 displays three versions of 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 phase's end, one candidate vertex is marked for deletion and removed in the figure; the red vertices are candidate vertices that remain. The largest balanced sub-graph has the most vertices among the three produced sub-graphs.

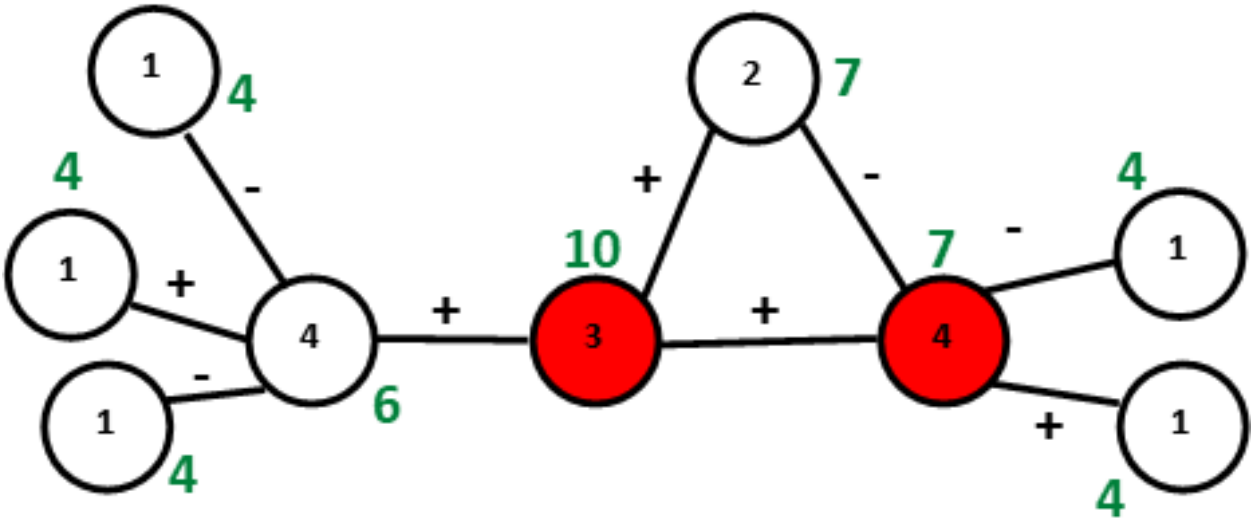

Figure 3: Degree (black, in node) vs. Sum of Neighborhood Degrees (green, next to the node) computation. The sum of neighborhood degrees labels the red vertices connected by a positive link that are candidates for deletion based on the ABCDD criteria.

4.2.1 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

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
```

scale-free graphs. Our goal is exactly the opposite. The approach eliminates the vertex with a smaller (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. Figure 3 demonstrates how the sum of a neighborhood 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 sub-graph 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.

4.2.2 ABCDH: Algorithm for Balanced Component Discovery Degree Harary

The stable states to Σ, defined in Def. 2.6, do not have the same sets of candidate edges for balancing. The balancing can be achieved in multiple ways, as explained in detail in [Rusnak and Tešić, 2021]. The proposed balancing algorithm identifies the stable states and the exact edges to remove to balance the remaining sub-graph. 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 with 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 [Rusnak and Tešić, 2021]. Thus, for the deletion criteria in the ABCDH, there is less chance for the edge to end up in one of the bipartitions. The majority case is now if the original edge was positive, and now it is negative; one vertices 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 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 re-used for computation.

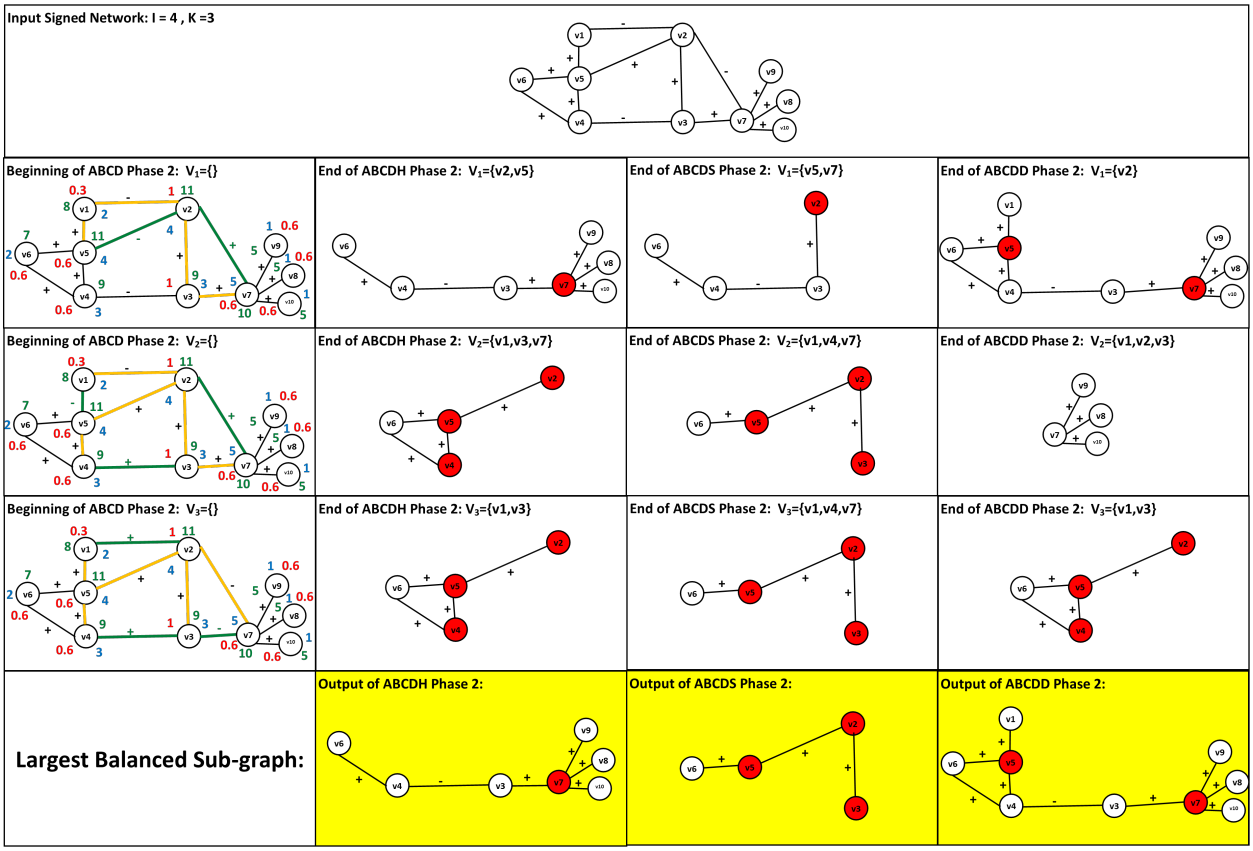

Figure 4: The ABCD algorithm applied to a sample signed graph with ten vertices and thirteen edges. For connectivity approximations, we compute the Harary bipartition in ABCD Phase 1 (Algorithm 1) and compute the sum of neighborhood (green), degree (blue), and adjusted status (red) at the beginning of 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 sub-graph defined by black and red vertices.

4.2.3 ABCDS: Algorithm for Balanced Component Discovery Status

Algorithm 5 Adjusted Status Computation

Input Σ, $\mathcal{H}_\Sigma$, and K
Initialize zero array $\mathcal{O}$ of size $|V|$
for $i = 0$; i++; $i < K$ **do**
    for every vertex of index $x \in \Sigma$ **do**
        $\mathcal{O}_\Sigma[x]$ += $\mathcal{H}_\Sigma[i][x]$
    end for
end for
for every vertex of index $x \in \Sigma$ **do**
    $\mathcal{O}_\Sigma[x]$ /= K
end for
Return $\mathcal{O}$

The Algorithm for Balanced Component Discovery Degree Status (ABCDS) approximates the connectivity by placing the vertex with the smaller adjusted status measure of the two vertices along an edge that switched sign in the candidate deletion set $\mathcal{V}_\backslash$. The ABCDS uses the adjusted status measure from [Rusnak and Tešić, 2021] 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 state's Harary bipartition is that it is more robust than using a single Harary bipartition as shown in [Rusnak and Tešić, 2021], 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.

5 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 [Alabandi et al., 2021]. 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 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 take O($|E|$), assuming Harary bipartitions are not computed. So the total beginning-to-end complexity of ABCDD is O($|E| * log(|V|) * d_a$).

ABCDH: Obtaining the Harary bipartition takes O($|V| * |E|$) because we use the Belman-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 take 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($|E| * log(|V|) * d_a + |V| * |E|$).

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 pre-computed 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($|E| * log(|V|) * d_a + |V| * |E|$).

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

6 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 [Alabandi et al., 2021] and has recently shown that the breadth-first search sampling of the spanning trees captures the diversity of the frustration cloud [Rusnak and Tešić, 2021]. We adopt the breath-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.

Table 3: Comparison of TIMBAL and ABCD I = 5000 runs on Konect plus TwitterReferendum and PPI signed graphs. The numbers between the parenthesis in the ABCDH column are the sub-graph # 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	**sub-graph # vertices**	**Run t (s)**	**sub-graph # vertices**	**Run t (s)**	**sub-graph # vertices**	**Run t (s)**	**sub-graph # vertices**	**Run t (s)**	**sub-graph # 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

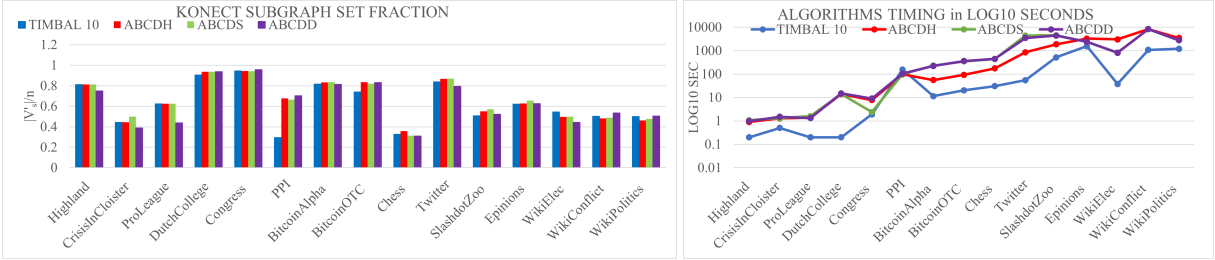

Figure 5: ABCD and TIMBAL performance comparison for Konect benchmark in terms of subset graph fractions (left) and algorithmic timing (right).

Baseline: The TIMBAL approach has reached the highest cardinality of the sub-graphs in various datasets and is a de-facto state-of-the-art [Ordozgoiti et al., 2020]. The input parameters of TIMBAL [Ordozgoiti et al., 2020] 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. e 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.

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 compare three implementations of the ABCD algorithm (ABCDH, ABCDS, ABCDD) to TIMBAL for 14 Konect (plus Twitter Ref. and PPI signed graphs) and 17 Amazon datasets in terms of runtime in seconds and the size of the produced sub-graph and verify the balanced state of the discovered sub-graph for both methods.

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.

Table 4: Amazon ratings and reviews [He and McAuley, 2016] mapped to signed graphs. The number of vertices, edges, and cycles reflect the number in the largest connected component of each dataset.

AMAZON	**Input graph**	**Largest Connected Component**		
Ratings	**# ratings**	**# 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

Table 5: Comparison between TIMBAL and ABCD on Amazon ratings and reviews [He and McAuley, 2016] 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	**TIMBAL 1 run**		**ABCDH**		**ABCDS**		**ABCDD**	
Ratings	**# vertices**	**t (hr)**	**# vertices**	**t (hr)**	**# vertices**	**time (hr)**	**# 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	2179.59	**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

6.1 TIMBAL vs. ABCD for the Konect Benchmark

Konect signed graphs are from [Kunegis, 2013], 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 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 [Lai et al., 2018].

WikiElec is the network of users from the English Wikipedia that 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 that 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 [He et al., 2021].

The first benchmark consists of 14 signed graphs from the Konect repository [Kunegis, 2013] (plus TwitterRef. and PPI) used in TIMBAL benchmark evaluations. Figure 5 and Table 3 summarizes baseline TIMBAL and proposed

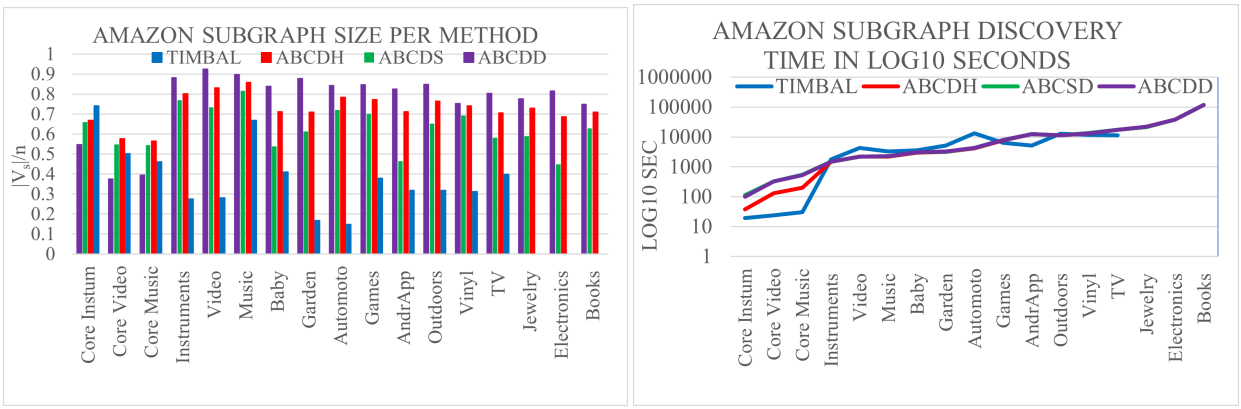

Figure 6: ABCD and TIMBAL performance and running time comparison for Amazon data.

ABCD performance. The ABCD matches 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 most extensive graph in the collection, Epinions, ABCD takes double the time to recover the largest balanced sub-graph.

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 the parenthesis in the ABCDH column. For this benchmark, ABCD algorithms outperform TIMBAL with comparable runtimes.

6.2 TIMBAL vs. ABCD for the Amazon Benchmark

Amazon benchmark consists of 17 signed graphs derived from the Amazon rating and review files [He and McAuley, 2016]. The dataset contains product reviews and metadata from Amazon, spanning May 1996 to July 2014. Rating score is mapped into an edge between the user and the product as follows $(5, 4) \to e^+$, $3 \to e$ (no sign), and $(2, 1) \to e^-$ [He and McAuley, 2016].

Table 4 summarizes Amazon data used and the characteristics of the largest connected component of the graph. Figure 6 (left) and Table 5 illustrate the sub-graph size TIMBAL recovers (blue box) and the sub-graph 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 Figure 6 (right). TIMBAL's performance degrades with the graph size, and the discovered sub-graphs are much smaller than what ABCD finds, as described in Figure 6 (left).

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

6.3 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 sub-graph 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 sub-graph 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 sub-graph found with a greater value of I. Thus, the vertex cardinality of the discovered sub-graph increases when the K is larger. For $K = 100$, Table 8 summarizes the improvements as the size of the sub-graph increases for comparable Table 7 performance. The execution timing also increases, and we use the size of the frustration cloud K as a balancing barometer between the size of the sub-graph found and execution time.

Next, as the number of iterations I for the larger $K = 100$ increases, the size of the discovered balanced sub-graph 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 sub-graph with the largest vertex cardinality. Therefore,

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

7 Conclusion

Finding maximum balanced sub-graphs 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 sub-graph. 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 sub-graph has the largest size regarding the number of vertices; it is balanced as it has no unbalanced cycles, and it is a sub-graph as the algorithm removes the *vertices*. ABCD recovers significantly balanced sub-graphs over two times larger than state-of-the-art.

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 [Alabandi et al., 2021]. 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

[Abelson and Rosenberg, 1958] Abelson, R. P. and Rosenberg, M. J. (1958). Symbolic psycho-logic: A model of attitudinal cognition. *Behavioral Science*, 3(1):1–13.

[Alabandi et al., 2021] Alabandi, G., Tešić, J., Rusnak, L., and Burtscher, M. (2021). Discovering and balancing fundamental cycles in large signed graphs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA. Association for Computing Machinery.

[Amelkin and Singh, 2019] Amelkin, V. and Singh, A. K. (2019). Fighting opinion control in social networks via link recommendation. In *Proceedings of the 25th ACM SIGKDD Intl. Conf. Knowledge Discovery & Data Mining*, pages 677–685.

[Anna D. and Aaron, 2019] Anna D., B. and Aaron, C. (2019). Scale-free networks are rare. *Nature Communications*, 10(1):1 – 10.

[Berger et al., 2004] Berger, F., Gritzmann, P., and de Vries, S. (2004). Minimum cycle bases for network graphs. *Algorithmica*, 40(1):51–62.

[Bonchi et al., 2019] Bonchi, F., Galimberti, E., Gionis, A., Ordozgoiti, B., and Ruffo, G. (2019). Discovering polarized communities in signed networks.

[Boulton, 2016] Boulton, L. (2016). Spectral pollution and eigenvalue bounds. *Applied Numerical Mathematics*, 99:1–23.

[Cartwright and Harary, 1956] Cartwright, D. and Harary, F. (1956). Structural balance: a generalization of Heider's theory. *Psychological Rev.*, 63:277–293.

[Chen et al., 2023] Chen, C., Wu, Y., Sun, R., and Wang, X. (2023). Maximum signed θ-clique identification in large signed graphs. *IEEE Transactions on Knowledge and Data Engineering*, 35(2):1791–1802.

[Crowston et al., 2013] Crowston, R., Gutin, G., Jones, M., and Muciaccia, G. (2013). Maximum balanced subgraph problem parameterized above lower bound. *Theoretical Computer Science*, 513:53 – 64.

[Deng et al., 2007] Deng, K., Zhao, H., and Li, D. (2007). Effect of node deleting on network structure. *Physica A: Statistical Mechanics and its Applications*, 379(2):714–726.

[Derr et al., 2020] Derr, T., Wang, Z., Dacon, J., and Tang, J. (2020). Link and interaction polarity predictions in signed networks. *Social Network Analysis and Mining*, 10(1):1–14.

[Facchetti et al., 2011] Facchetti, G., Iacono, G., and Altafini, C. (2011). Computing global structural balance in large-scale signed social networks. *Proceedings of the National Academy of Sciences*, 108(52):20953–20958.

[Figueiredo and Frota, 2014] Figueiredo, R. and Frota, Y. (2014). The maximum balanced subgraph of a signed graph: Applications and solution approaches. *European Journal of Operational Research*, 236(2):473–487.

[Figueiredo et al., 2011] Figueiredo, R. M., Labbé, M., and de Souza, C. C. (2011). An exact approach to the problem of extracting an embedded network matrix. *Computers and Operations Research*, 38(11):1483 – 1492.

[Garimella et al., 2021] Garimella, K., Smith, T., Weiss, R., and West, R. (2021). Political polarization in online news consumption. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 15, pages 152–162.

[Gülpinar et al., 2004] Gülpinar, N., Gutin, G., Mitra, G., and Zverovitch, A. (2004). Extracting pure network submatrices in linear programs using signed graphs. *Discrete Applied Mathematics*, 137(3):359–372.

[Harary and Cartwright, 1968] Harary, F. and Cartwright, D. (1968). On the coloring of signed graphs. *Elemente der Mathematik*, 23:85–89.

[Harary et al., 2002] Harary, F., Lim, M.-H., and Wunsch, D. C. (2002). Signed graphs for portfolio analysis in risk management. *IMA Journal of Management Mathematics*, 13(3):201–210.

[He and McAuley, 2016] He, R. and McAuley, J. (2016). Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th Intl. Conf. World Wide Web*, WWW '16, pages 507–517. ACM.

[He et al., 2021] He, Y., Reinert, G., Wang, S., and Cucuringu, M. (2021). SSSNET: semi-supervised signed network clustering. In *Proceedings of the 2022 SIAM Intl. Conf. Data Mining (SDM)*, pages 244–252.

[Heider, 1946] Heider, F. (1946). Attitudes and cognitive organization. *J. Psychology*, 21:107–112.

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

[Kleinberg et al., 2008] Kleinberg, J., Sandler, M., and Slivkins, A. (2008). Network failure detection and graph connectivity. *SIAM Journal on Computing*, 38(4):1330–1346.

[Kunegis, 2013] Kunegis, J. (2013). KONECT – The Koblenz Network Collection. In *Proceedings of the 22nd Intl. Conf. World Wide Web*, WWW '13, pages 1343–1350. ACM.

[Lai et al., 2018] Lai, M., Patti, V., Ruffo, G., and Rosso, P. (2018). Stance evolution and Twitter interactions in an Italian political debate. In Silberztein, M., Atigui, F., Kornyshova, E., Métais, E., and Meziane, F., editors, *Natural Language Processing and Information Systems*, pages 15–27, Cham. Springer International Publishing.

[Lalou et al., 2018] Lalou, M., Tahraoui, M. A., and Kheddouci, H. (2018). The critical node detection problem in networks: A survey. *Computer Science Review*, 28:92–117.

[Leskovec et al., 2010] Leskovec, J., Huttenlocher, D., and Kleinberg, J. (2010). Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 1361–1370, New York, NY, USA. Association for Computing Machinery.

[Liu et al., 2022] Liu, C., Dai, Y., Yu, K., and Zhang, Z. (2022). Enhancing cancer driver gene prediction by protein-protein interaction network. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, Computational Biology and Bioinformatics, IEEE/ACM Transactions on IEEE/ACM Trans. Comput. Biol. and Bioinf*, 19(4):2231 – 2240.

[Macon et al., 2012] Macon, K. T., Mucha, P. J., and Porter, M. A. (2012). Community structure in the United Nations General Assembly. *Physica A: Statistical Mechanics and its Applications*, 391(1):343–361.

[Ordozgoiti et al., 2020] Ordozgoiti, B., Matakos, A., and Gionis, A. (2020). Finding large balanced subgraphs in signed networks. In *Proceedings of The Web Conference 2020*, WWW '20, pages 1378–1388, New York, NY, USA. Association for Computing Machinery.

[Poljak and Turzík, 1986] Poljak, S. and Turzík, D. (1986). A polynomial-time heuristic for certain subgraph optimization problems with guaranteed worst-case bound. *Discrete Mathematics*, 58(1):99–104.

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

[Sharma et al., 2021] Sharma, K., Gillani, I. A., Medya, S., Ranu, S., and Bagchi, A. (2021). *Balance Maximization in Signed Networks via Edge Deletions*, pages 752–760. Association for Computing Machinery, New York, NY, USA.

[Tarjan, 1974] Tarjan, R. (1974). A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161.

[Tomasso et al., 2022] Tomasso, M., Rusnak, L., and Tešić, J. (2022). Advances in scaling community discovery methods for signed graph networks. *Journal of Complex Networks*, 10(3).

[Wu et al., 2022] Wu, Y., Meng, D., and Wu, Z.-G. (2022). Disagreement and antagonism in signed networks: A survey. *IEEE/CAA Journal of Automatica Sinica, Automatica Sinica, IEEE/CAA Journal of, IEEE/CAA J. Autom. Sinica*, 9(7):1166 – 1187.

[Zaslavsky, 2012] Zaslavsky, T. (2012). A mathematical bibliography of signed and gain graphs and allied areas. *ELECTRONIC JOURNAL OF COMBINATORICS*.