ABCD: Algorithm for Balanced Component Discovery in Signed Networks

Muhieddine Shebaro, *Student Member, IEEE*, Lucas Rusnak, and Jelena Tešić, *Senior Member, IEEE*

Abstract—The most significant 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 nodes 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 nodes 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 sub-graph, frustration index, balanced states, and signed graphs.

1 Introduction

Signed networks allow for unsigned and negative weights in the graph-based representation. Negative weights represent antagonistic relationships and model the conflicting opinions between the nodes well [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 its core [6], [7], [8], [9]. The task of the most extensive balanced sub-graph 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 nodes that are part of the maximum balanced sub-graph Σ' of Σ may not necessarily have a high degree of centrality between them. Still, they are essential for understanding how the system behaves. *TODO: I do not understand this sentence – Mo*

By locating the largest balanced sub-graphs, we can simplify the system into sub-systems with balanced interactions and eliminate inconsistencies regarding unbalanced cycles. Finding the largest balanced sub-graph 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, and we show that it discovers larger signed sub-graphs faster than TIMBAL. Section 2 summarizes the related work and state-of-the-art approaches. The approach builds on the scalable discovery of fundamental cycles in [24] and utilizes the graph's node density distribution and near-balanced states to minimize the number of nodes removed from the balanced sub-graph. Section 3 explains the notations and the definitions and theorems behind the singed graph balancing, as well as the algorithm for the scalable graph cycle-basis computation of the underlying unsigned graph G of Σ. In Section 4, we introduce 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 focuses on comparing the proposed method to the state-of-art method proposed in [17]. TIMBAL method achieved the highest node cardinality (number of nodes in the largest balanced sub-graph) across all signed graphs, among other baselines in the literature [17]. We evaluate our algorithm on the Konect, and Amazon signed graphs in Section 6.1 and Section 6.2, respectively. Moreover, we perform parameter sensitivity analysis on our proposed algorithm in Section 6.4. In Section 7, we summarize our findings.

The problem definition of finding the largest balanced component $\Sigma^G, |\Sigma^G| = g$ in *any* size signed graph $\Sigma, |\Sigma| = n$ is in Eq. 1.

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

Note that $Fr(\Sigma^G)$ represents the Frustration of balanced sub-graph Σ^G. The goal 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. For definitions and corollaries, see Section 3. The paper contributions are:

• A novel algorithm for balanced component discovery (ABCD) in signed networks. The task at hand is the discovery of the near-balanced subgraph *without* changing any edge signs (not to be mixed with the task of balancing the

M. Shebaro and J. Tešić are with the Department of Computer Science, and L. Rusnak is with the Department of Mathematics at Texas State University, San Marcos, TX, USA 78666.
E-mail: m.shebaro, lucas.Rusnak, jtesic@txstate.edu

Manuscript received Dec 21, 2023; revised July 22, 2024.

graph [18] where the authors shift the edge signs to find a balanced state of the graph).

• ABCD is based on the balance theory and efficient implementation to *find* the largest balanced sub-graph in signed graphs. ABCD considers the top K near-balanced states with the lowest frustration index in the process, and we provide theoretical proof.

• ABCD can discover a balanced subgraph in large signed networks with millions of nodes and edges, and the size of the discovered subset is double that of the state-of-the-art in a similar time frame.

2 Related Work

Finding the near-balanced sub-graph in a signed graph is known to be an NP-hard problem. Gülpinar et al. proposed the GGMZ algorithm, which begins by computing the input graph's minimum spanning tree. Next, a subset of nodes is selected, and all the edges crossing that subset are inverted to create positive edges. We run this step for the entire graph, and the result is the set of nodes disconnected by negative edges. The system's overall complexity is $O(|N|^3)$ if N is a set of nodes [19]. Poljak and Daniel Turzík show that any signed graph that has |N| nodes and |M| edges contains a balanced sub-graph with at least 0.5|M| + 0.25(|N|- 1) edges [20]. Crowston et al. propose a discovery of a balanced sub-graph of size 0.5|M| + 0.25(|N|- 1+k), k is a parameter, of the complete system's overall complexity is $O(|N|^3)$ if N is a set of nodes [19]. They achieved data reduction by finding small separators and a novel gadget construction scheme. The fixed-parameter algorithm is based on iterative compression with a very effective heuristic speedup. *TODO: what is so effective about it? – Mo*

Figueiredo et al. first introduced a polyhedral-based branch-and-cut algorithm to find the largest sub-graph [21], followed by preprocessing routines and initial heuristic improvements in [11]. The proposed GRASP algorithm randomly selects a subset of nodes. It then greedily adds nodes that maximize the number of edges connecting them to the current subset while keeping the size of the subset balanced [11]. The EIGEN algorithm [22] 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 node 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 [18]. 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 nodes and the second one restores them as long as it does not cause imbalance [17]. 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 ($O(|N|^2)$), and its performance decreases due to the spectral pollution in eigenvalue computation [23]. TIMBAL proposes a novel bound for perturbations of the graph Laplacian preprocessing techniques to scale the processing for large graphs. The algorithm randomly samples sub-graphs and runs TIMBAL. The nodes deleted from at least one of these sub-graphs are deleted from the original graph. 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 [17]. Note that the task for this paper - finding the largest balanced subgraph – differs from the balancing graph task proposed by [18]: the authors modify the graph after seeing an initial maximum balanced sub-graph using TIMBAL to maximize balance using edge deletions. In addition, the authors employ a budget to delete edges in the candidate edge set. Our algorithm does not have a limit on the number of vertices and edges to be deleted to find the largest balanced sub-graph.

TABLE 1
Summary of notations and their meanings.

Notation	Meaning
Σ	signed network
G	underlying graph of a signed network
T	spanning tree
n	# of nodes
N	set of nodes *TODO: set of all nodes? – Mo*
v	a node
V	set of nodes *TODO: set of all nodes, see above ... difference between N and V?? – Mo*
m	# of edges
M	set of edges
e	an edge
e^+	positive edge
e^-	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, $\lvert U\rvert \geq \lvert W\rvert$
I	# of spanning trees/iterations
$\mathcal{F}_\Sigma$	the frustration cloud set of Σ
$\lvert\mathcal{F}_\Sigma\rvert$	the size of the frustration cloud.
K	# of nearest balanced states w lowest frustration index, $K \leq \lvert\mathcal{F}_\Sigma\rvert$
m_k	minimal # of edge signs changed, see Alg. 1)
$\mathcal{F}_\Sigma$	frustration cloud of Σ as defined in [25]
$\mathcal{E}_\Sigma$	container of Σ for storing a set of edges in each element that switched signs during balancing
$\mathcal{M}_\Sigma$	K size array of the number of edge sign switches of top K nearest balanced states.
$\mathcal{H}_\Sigma$	binary array of size n to separate the nodes into Harary bipartitions where $\mathcal{H}_\Sigma[v] = 1$ if v is in set U, and 0 if v is in set W
$\mathcal{V}$	set of nodes § in each of the top-K nearest balanced states to keep
$\mathcal{V}_s$	smallest set of nodes to purge to recover largest balanced sub-graph
$\mathcal{V}'_s$	set of nodes obtained after purging the set of nodes $\mathcal{V}_s$

3 ABCD Support in Theory

In this section, we define the fundamental cycle basis and relevant signed graph network terms and outline the theorems and corollaries that provide theoretical bounds for the proposed ABCD approach.

Definition 3.1. **Signed graph** $\Sigma = (G, \sigma)$ consists of underlying unsigned graph G and an edge signing function

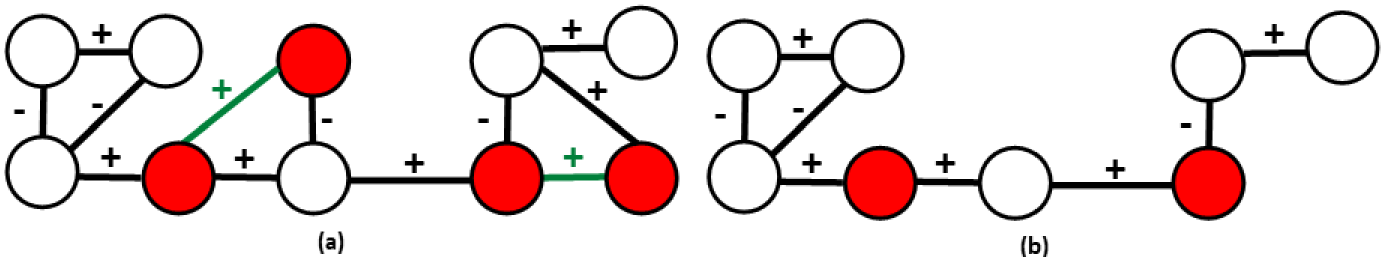

Fig. 1. (a): The unbalanced signed network. Green edges are the candidate edges causing imbalance, and red nodes are the candidate nodes. (b): The maximum balanced signed sub-graph obtained after deleting one candidate node along each edge.

$\sigma : m \rightarrow \{+1, -1\}$. The edge m can be positive m^+ or negative m^-. **Sign** of a sub-graph is *product* of the edges signs. **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.

TODO: fix the subgraph vs. the balanced graph notation – Mo

Definition 3.2. Graph Σ^G is a **sub-graph** of a graph Σ if **all** edges and nodes of Σ^G are contained in Σ.

Definition 3.3. **Path** is a sequence of distinct edges m that connect a sequence of distinct nodes n in a graph. **Connected graph** has a path that joins any two nodes. **Cycle** is a path that begins and ends at the same node. **Cycle Basis** is a set of simple cycles that forms a basis of the cycle space.

Definition 3.4. For the underlying graph G, let T be the spanning tree of G, and let an edge m be an edge in G between nodes x and y that is *NOT* in the spanning tree T. Since the spanning tree spans all nodes, a unique path in T between nodes x and y does not include m. **The fundamental cycle** is any cycle that is built using path in T plus edge m in graph G.

Corollary 3.1. A fundamental cycle basis may be formed from a spanning tree or spanning forest of the given graph by selecting the cycles formed by combining a path in the tree and a single edge outside the tree. or the graph G with n nodes and m edges, there are precisely $m - n + 1$ fundamental cycles.

Definition 3.5. The balanced states are **near-balanced** if and only if it requires a minimum number of edge sign switches in the original graph to reach a balanced state. We label the near-balanced 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 [25]

TODO: I want to move from Σ' to Σ_i for all near-balanced states. – Mo

Theorem 3.1. If a signed sub-graph Σ' is balanced, the following are equivalent [4]:

1) Σ' is balanced. (All circles are positive.)
2) For every node pair (n_i, n_j) in Σ', all (n_i, n_j)-paths have the same sign.
3) $Fr(\Sigma') = 0$.
4) There exists a bipartition of the node set into sets U and W such that an edge is negative if, and only if, it has one node in U and one in W. The bipartition (U,W) is called the *Harary-bipartition*.

We note the sets so that U always contains a more significant or equal number of nodes than W

In this study, we aim to identify the largest balanced sub-graph within a signed network. A balanced sub-graph has an even number of opposing edges in each fundamental cycle. The size of a sub-graph, which we aim to maximize, is quantified by its node cardinality or the number of nodes it contains. We summarize the notations used in this paper and their meanings in Table 1.

If the graph is balanced, the frustration index is 0, and the largest subgraph is the graph itself. Frustration is a measure of how many edges need to be changed in a signed graph Σ to achieve *any* near balancing state Σ_i, $i \in [1, |\mathcal{F}_\Sigma|]$. The nearest balanced states to Σ do not necessarily have the same amount of candidate edges that need to be switched [25]. Different near-balanced states result in different Harary bisects. The approach in [26] produces different near-balanced states of Σ, as defined in Def. 3.5. For $\forall i \in [1, |\mathcal{F}_\Sigma|]$, there are two steps, and now we know the exact edges that need to be switched and nodes connected to them to achieve a balanced state Σ_i. *Missing: Lucas proposes to connect clustering here as the Harary cut corresponds to the first spectral split. Why is that optimal in terms of finding the largest balanced subgraph?*

Step 1: remove Fr_i **edges** from Σ whose signs need to be switched to achieve Σ_i, $\forall i$. We break the negative fundamental cycles, and the resulting graph is a spanning subgraph of Σ with m nodes, $n - Fr_i$ edges, and $n-m-Fr_i+1$ positive cycles. Note that not a single original positive cycle was removed in this step.

Step 2: remove nodes For all near balancing states to Σ, Σ_i, $i \in [1, ...I]$, create Harary bi-set U_i, W_i where $\Sigma_i = U_i + W_i$ and $|U_i| > |W_i|$ [4], one per near balanced state, as in [26]. *Missing: add illustration here on different Harary sets*. For each edge removed in step 1, remove the connecting node as follows: (i) remove all nodes that fall into W_i (assigned smaller by default); (ii) for the remaining nodes in the U_i, delete the node that connects fewer nodes (degree/sum of neighborhood degree) in the original graph Σ. Note that this procedure minimizes the number of positive cycles broken from the original graph Σ. *Missing: how? why? how do we prove that we have ensured that we deleted the smallest number of nodes to reach a balanced subgraph? Can we identify the largest*

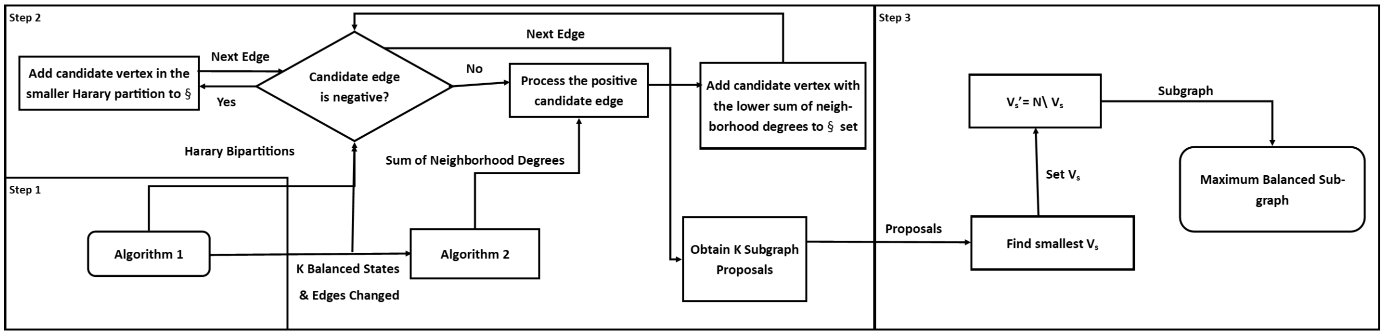
Step 2
Add candidate vertex in the smaller Harary partition to §
Next Edge
Yes
Candidate edge is negative?
No
Process the positive candidate edge
Next Edge
Add candidate vertex with the lower sum of neighborhood degrees to § set
Harary Bipartitions
Sum of Neighborhood Degrees
Step 1
Algorithm 1
K Balanced States & Edges Changed
Algorithm 2
Obtain K Subgraph Proposals
Step 3
V_s'= N\ V_s
Subgraph
Set V_s
Maximum Balanced Subgraph
Proposals
Find smallest V_s

Fig. 2. Algorithm for Balanced Component Discovery (ABCD): graph processing flow and illustration of Algorithm 1, Algorithm 2, and phase 3 of the algorithm.

balanced subgraph by either picking the largest subgraph from Step 1 (when only edges are deleted in Sigma or from Step 2 (($max_i(||U_i| - |W_i||)$)

4 Methodology

In this section, we introduce the **Algorithm for the Balanced Component Discovery** (ABCD), a scalable algorithm for the discovery of the largest balanced sub-graph. The ABCD algorithm consists of three steps: *TODO: list the steps here – Mo*, as illustrated in Figure 2. *TODO: next paragraph should describe both the process and the Figure 2 – Mo* Next, we explain the three phases of the algorithm in Algorithm 1 *TODO: make sure the flow is correct here – Mo*.

ABCD phase 1 creates a candidate list of fundamental cycle bases with minimal unbalanced cycles. This algorithm is the backbone of our proposed algorithm in Alg. 1. I is the number of iterations we run the algorithm and the upper bound on how many near-balanced states we discover in the process. The steps are:
1.1. Discover the fundamental cycle bases for each of the I spanning trees (Alg. 1).
1.2. For each of the cycles in the basis, count the number of cycles that contain the odd number of negative edges (Alg. 1).
1.3. Keep only the $K, K << I$ fundamental cycle basis out of I accessed that have the smallest number of fundamental cycles with an odd number of negative edges (imbalanced Cycle). This translates into lowest cardinality $m_k, m_k < m - n + 1$ in Alg. 1.

ABCD phase 2 employs a smart edge deletion approach for all K discovered balanced states as outlined in Alg. 2. The illustrative example is outlined in Figure 3. Minimizing the number of nodes removed from the graph increases the cardinality of the largest balanced sub-graph. *Harary bipartition* separates the nodes of the balanced graph into two sets such that the nodes of both sets internally agree with each other but disagree with the nodes of the other set [4]. The $\mathcal{H}_\Sigma$ set is created as a labeling vector in Alg. 1. We repeat the following steps for all K identified balanced states for a signed graph Σ, and the heuristic on how we remove the unbalanced fundamental cycles out of possible $m - n + 1$ cycles for the balanced state $k, k \in [1, K]$.
For the edges that should have a different sign for the entire graph to be balanced $\mathcal{E}_\Sigma$, we initialize an empty set §, Alg. 2

Algorithm 1 ABCD Phase 1

```
1: Fetch signed graph $\Sigma$, number of iterations $I$, and integer $K$ that determines the near-balanced states with the lowest frustration index to keep
2: Generate set of $I$ spanning trees of $\Sigma$
3: Counter to keep only the near-balanced states with the lowest frustration index $i = 0; m_K = m;$
4: for i = 0; i++; i < I do
5:   for edges $e$, $e \in \Sigma \setminus T_i$ do
6:     if fundamental cycle $T \cup e$ is negative then
7:       add edge $e$ to $M_i$
8:     end if
9:   end for
10:  if $|M_i| < m_K$ then
11:    $\mathcal{M}_\Sigma \cup = m_k$
12:  end if
13:  if $|\mathcal{M}_\Sigma| > K$ then
14:    Remove the largest set and update $m_K$
15:  end if
16: end for
17: Create $\mathcal{H}_\Sigma$ vector to store vector in each element of size $K$
18: Create $\mathcal{E}_\Sigma$ vector to store container in each element of size $K$
19: for i = 0; i++; i < K do
20:   Create zero vector $H_i$ of dimension $n$
21:   Create empty container $D_i$ of dimension $m$
22:   for edge $e \in M_i$ do
23:     switch edge sign in $\Sigma_i$: $e^- \to e^+; e^+ \to e^-$
24:   end for
25:   Cut all the negative edges to create Harary bipartitions $U$ and $W$ so that $|U| > |W|$
26:   for $v$ in $N_i$ do
27:     if $v \in U$,H$_i(v) = 1$
28:   end for
29:   Push $H_i$ to $\mathcal{H}_\Sigma$
30:   Push the edges that switched signs to $D_i$
31:   Push $D_i$ to $\mathcal{E}_\Sigma$
32: end for
33: return $\mathcal{H}_\Sigma$ and $\mathcal{E}_\Sigma$
```

repeats the following steps:

2.1. If the edge is positive, it will be negative in the

Algorithm 2 ABCD Phase 2

```
1: Harary bipartition array for each node $\mathcal{H}_\Sigma$, edges that switched signs for each top-$K$ stable states with the lowest frustration $\mathcal{E}_\Sigma$
2: Initialize empty set $\mathcal{V} = \emptyset$
3: for i = 0; i++; i < K do
4:   Initialize empty set $\S = \emptyset$
5:   for edges $e$, $e \in \mathcal{E}_\Sigma[i]$ do
6:     if any of the nodes along $e \in \S$ then
7:       Skip iteration
8:     end if
9:     if $e$ is positive then
10:      Append the node $v$ along $e$ that has a lower sum of neighborhood degrees to set $\S$
11:    else
12:      Append the node $v$ along $e$ where $\mathcal{H}_\Sigma[i][v] = 0$ to set $\S$
13:    end if
14:  end for
15:  Push $\S$ to $\mathcal{V}$
16: end for
17: return $\mathcal{V}$ which is the entire set of nodes in each of the top-$K$ graphs with the lowest index to keep when reconstructing the original graph
```

balanced state. If either node is already in $\S$, move on to the next edge. Else, add the edge-defining node v to $\S$ so that $\mathcal{H}_\Sigma[i][v] = 0$ in iteration i. The remaining node is in the largest Harary partition for a fully balanced graph so that it will be connected to more nodes than the node ending up in the smaller Harary set after partitioning.

2.2. If the edge is negative and marked for switching to positive, if either node is already in $\S$, nothing needs to be done; move on to the next edge. We add the node with the lower neighborhood degree to $\S$. The computation of the neighborhood degree is demonstrated in Alg 3. We observe that iterating 3 times where in each iteration, we set degree[] to be equal to neighborhood_degree[] before computing an empty neighborhood_degree[] using the new degree[] to improve the results generally.

Algorithm 3 Computation of the sum of neighborhood degree

```
1: Input signed graph $\Sigma$
2: Declare and initialize array neighborhood_degree = []
3: for v = 0; v++; v < n do
4:   Declare and initialize integer sum = 0
5:   for every neighbor $nei$ connected to v via an edge do
6:     sum += degree[$nei$]
7:   end for
8:   neighborhood_degree[v]=sum
9: end for
10: return array neighborhood_degree
```

Fig. 3 illustrates how we compute the neighborhood degree for an exemplar signed graph. Two red nodes 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 node is 3, and the right node is 4. The neighborhood degree of the left node is 10 (neighbors of a neighbor), and the neighborhood degree of the right node is 7. We chose the node on the right to delete and the node on the left to keep. If both have the same sum of neighborhood degrees, choose a random node along that edge to discard. Note that the *neighborhood* degree is computed for all nodes in the signed graph once and re-used for computation.

ABCD phase 3 finds the index of the smallest sized $\S$ in $\mathcal{V}$. Let it be index $s : |\mathcal{V}_s| \leq |\mathcal{V}|$. The resulting maximized balanced sub-graph proposal $\mathcal{V}'_s$ is finally obtained by removing specific nodes as $\mathcal{V}'_s = N \setminus \mathcal{V}_s$. The remaining subset is balanced as all fundamental cycles in the graph are balanced.

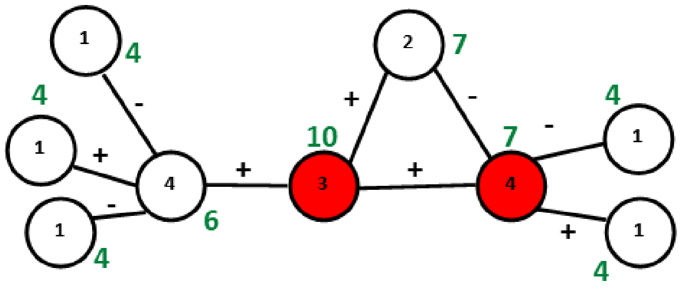

Fig. 3. Degree (black, in node) vs. Sum of Neighborhood Degrees (green, next to the node) computation. The sum of neighborhood degrees labels the red nodes connected by a positive link that will be deleted.

Illustrative Example of the ABCD algorithm Figure 1 illustrates an example execution of the proposed ABCD algorithm. First, we identify the candidate imbalance edges in the graph. Second, the candidates for the deletion are red nodes connected to unbalanced edges that belong to the most minor Harary cut.

Figure 4 illustrates the step-by-step ABCD method on the sampled signed graph. The signed graph with seven nodes and ten edges is introduced in the top row. Balancing occurs in ABCD phase 1 (Alg.1), and green text on the edges indicates the changed signs. Green numbers beside nodes are the sum of neighborhood degrees. Orange edges are the edges of the unbalanced fundamental cycles. Green edges are the candidate edges. Red nodes are the candidates for deletion in ABCD phase 2 (Alg.2). The final candidate selection and comparison is in ABCD phase 3, and we found that the remaining sub-graph is balanced by removing node $v2$. The red dotted oval over the graph in Step 3 signifies that the final output $\mathcal{V}'_s$ of ABCD has a maximal cardinality.

5 Complexity Analysis

First, computing the degrees of each node and reading the graph take O($n + m$). Computing the sum of the neighborhood degrees of each node uses the same procedure as calculating the degrees, but it's repeated three times. Hence, O($3 * (n + m)$) is added. In each iteration, balancing the signed network for all nodes takes $O(m * log(n)d_a)$ where d_a is the average degree of a node as analyzed by Alabandi et al. [24]. Obtaining the Harary bipartition takes O($n*m$) because we use Belman-Ford algorithm to compute distances on the detected connected components. The time

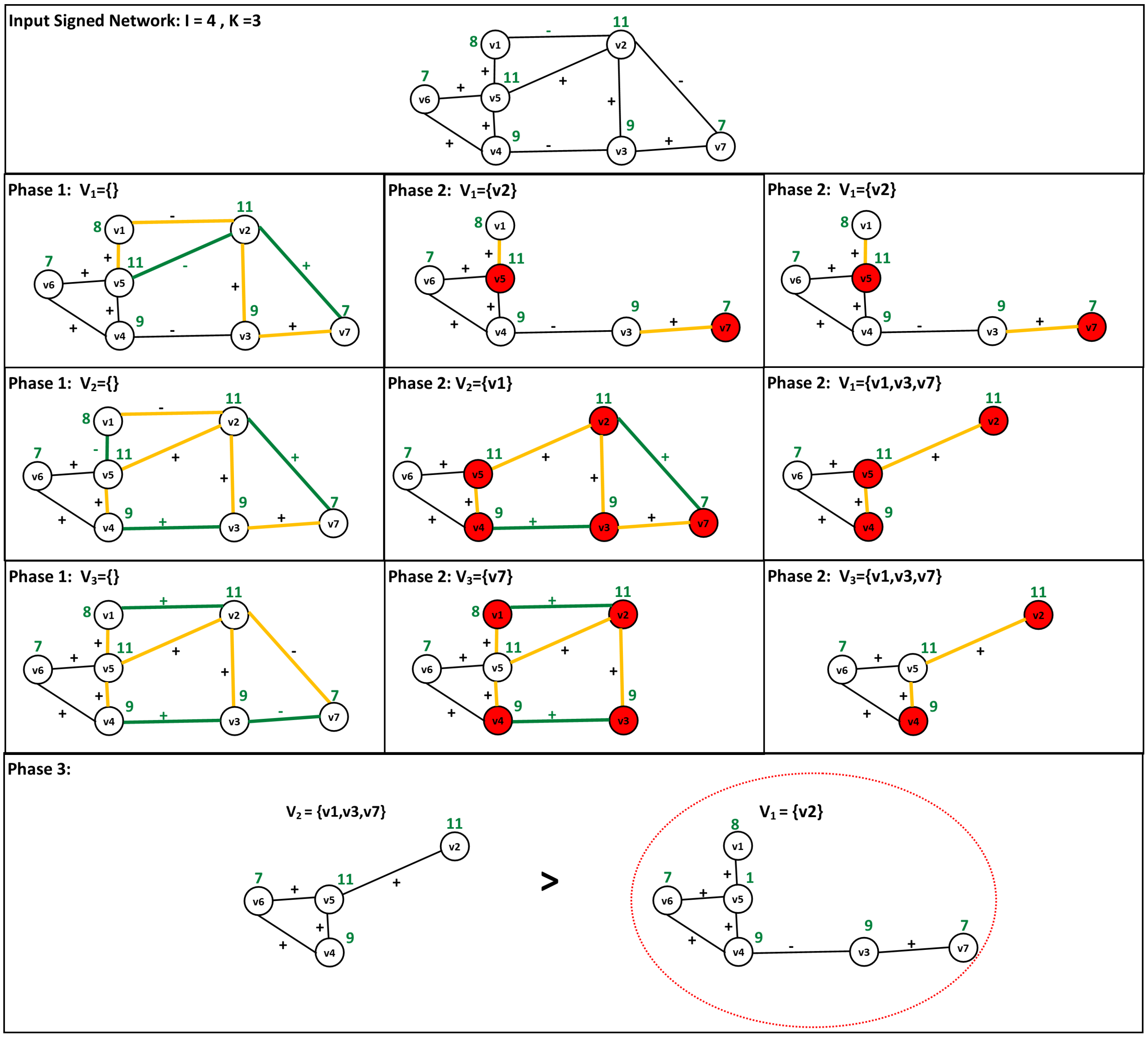

Fig. 4. The ABCD algorithm applied to a signed graph with seven nodes and ten edges. Balancing occurs in ABCD phase 1 (Alg.1), and green text on the edges indicates the changed signs. Green numbers beside nodes are the sum of neighborhood degrees. Orange edges are the edges of the unbalanced fundamental cycles. Green edges are the candidate edges. Red nodes are the candidates for deletion in ABCD phase 2 (Alg.2).

complexity to count the number of edge sign switches that occurred after is O(m). In case the number of balanced states retrieved is less than K, the time complexity for storing the state and Harary bipartition takes O(m). 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(m) in total as well.

Phase 2 takes O($K * (mlog(n) + (n + m))$) because we have to loop over every top-K balanced state. For each state, we loop over candidate edges, insert nodes to be kept in a set that takes $log(n)$ (assuming the set data structure in C++ is a red-black tree), and reread O($n + m$) to reread each top-K balanced state without the candidate nodes and detect the largest connected component using union-find. Hence, the total time complexity of the algorithm is O($K * ((m * log(n)d_a + n * m + K * (mlog(n) + (n + m))))$).

6 Proof Of Concept

All real-world benchmark graphs have one large connected component. The **implementation** of the algorithms is in C++. The algorithm identifies the largest connected component (LCC). It applies the ABCD algorithm to LCC. The implementation treats edges without signs as positive edges in the fundamental Cycle. If the graph has two or more connected components with the same size, the implementation accommodates that scenario. ABCD phase 1 (Alg. 1) implementation builds on [24]. [26] have recently shown that the breadth-first search sampling of the spanning trees captures the diversity of the frustration cloud [25]. We adopt the breath-first search method for sampling spanning trees in the Algorithm 1. ABCD phase 2 is implemented as listed in Algorithm 2. For the ABCD phase 3, $\mathcal{V}_s'$ is constructed by

Rereading the original graph and skipping the entries with nodes in $\mathcal{V}_s$. **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 $300,000 < n$ nodes. **ABCD_Fast** is a faster version of **ABCD** 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 $300,000 < n$ nodes. For this faster version, we can also study the effect of decreasing the number of iterations on the overall speed and performance. We experiment with this version solely on the Konect dataset as in Figure 5.

Baseline for the proof of concept is TIMBAL [17]. The TIMBAL approach has reached the highest cardinality of the sub-graphs 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 nodes) and to max_removals=100 for the rest of the signed networks. e set avg_size=20 for datasets of several nodes less than 80,000, and subsample_flag=True, samples = 1000, avg_size = 200 max_removals=100 for datasets with the number of nodes 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 node cardinality.

Setup ABCD is run on the same graphs as TIMBAL, and the results are compared side-by-side for 14 Konect and 17 Amazon datasets in terms of runtime in seconds and the size of the produced sub-graph. We verify the balanced state of the discovered sub-graph for both methods. The operating system used for the experiments is Linux Ubuntu 20.04.3, running on the 11th Gen Intel(R) Core(TM) i9-11900K @ 3.50GHz with 16 physical cores. t 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.

6.1 ABCD for the Konect Benchmark

Konect signed graphs are from [27], 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. Nodes 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 who mostly did not know each other before starting university. Each student was asked to rate the other at seven different time points. A node represents a student, and an edge between two students shows that the left rated the right. The edge weights show how good their friendship is in the eye of the left node. The weight ranges from -1 for the risk of conflict to +3 for best friend. *Congress* is a signed network where nodes are politicians speaking in the United States Congress, and a directed edge denotes that a speaker mentions another speaker. In the *Chess* network, each node 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 on which Bitcoins are traded. *BitcoinOTC* is a user-user trust/distrust network from the Bitcoin OTC platform on which Bitcoins are traded.

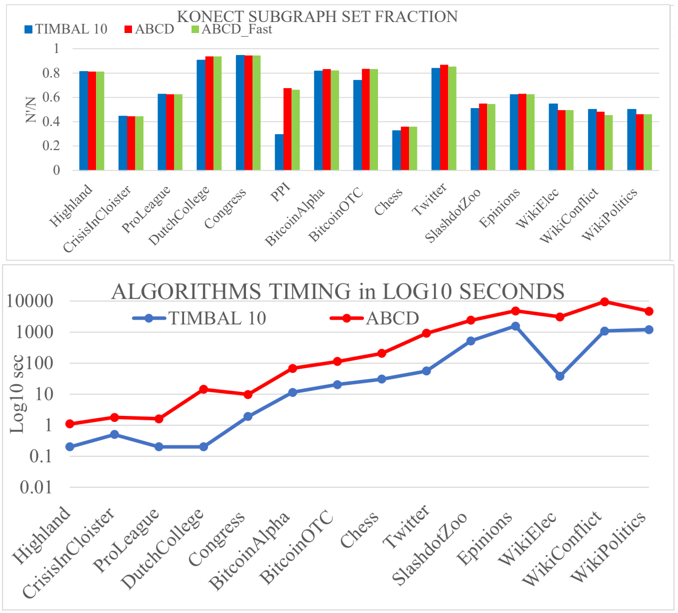

Fig. 5. ABCD and TIMBAL performance comparison for Konect benchmark in terms of subset graph fractions (top) and algorithmic timing (bottom).

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 [28]. *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. Nodes 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 given a positive or negative value. *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 [29].

TABLE 2
Konect sets plus TwitterReferendum and PPI signed graphs attributes: n is the total number of nodes; m is the total number of edges in the graph; The number of edges of the entire input signed network is the reported numbers from the Konect site. The number of edges of LCC is computed locally in our machine with preprocessing as removing duplicate and inconsistent edges is applied. Konect plus TwitterReferendum and PPI datasets graph performance comparisons: The largest sub-graph regarding the number of nodes discovered for TIMBAL 5 runs, TIMBAL 10 runs, and ABCD 5000 iterations approach in a given runtime in seconds.

	Entire Input		**Largest Connected**			TIMBAL 5 runs		TIMBAL 10 runs		ABCD	
Konect	**Signed Network**		**Component Graph**			sub-graph	Run	sub-graph	Run	sub-graph	Run
Dataset	# nodes	# edges	# nodes	# edges	**# cycles**	# nodes	time (s)	# nodes	time (s)	# nodes	time (s)
Highland	16	58	16	58	43	13	0.1	13	0.2	13	0.9
CrisisInCloister	18	126	18	126	145	8	0.25	8	0.5	8	1.28
ProLeague	16	120	16	120	105	10	0.1	10	0.2	10	1.40
DutchCollege	32	3,062	32	422	391	29	0.1	29	0.2	**30**	14
Congress	219	764	219	521	303	207	0.85	207	1.9	207	7.79
PPI	3,058	11,860	3,058	11,860	8,803	900	78.45	900	156.9	**2,072**	97.59
BitcoinAlpha	3,783	24,186	3,775	14,120	10,346	3,014	5.57	3,081	11.4	**3,146**	55.68
BitcoinOTC	5,881	35,592	5,875	21,489	15,615	4,250	10.15	4,349	20.3	**4,910**	92.29
Chess	7,301	65,053	7,115	55,779	48,665	2,230	15.25	2,320	30.5	**2,551**	174.67
TwitterReferendum	10,884	251,406	10,864	251,396	240,533	9,021	27,6	9,110	55.2	**9,438**	851.94
SlashdotZoo	79,120	515,397	79,116	467,731	388,616	39,905	259.4	40,123	518.8	**43,544**	1870.20
Epinions	131,828	841,372	119,130	704,267	585,138	73,433	774.9	74,106	1549.8	**74,843**	3272.87
WikiElec	7,118	103,675	7,066	100,667	93,602	**3,758**	18.95	**3,856**	37.9	3,506	3033.08
WikiConflict	118,100	2,917,785	113,123	2,025,910	1,912,788	**56,768**	539.25	56,768	1078.5	54,476	8332.22
WikiPolitics	138,592	740,397	137,740	715,334	577,595	67,009	602.65	**69,050**	1205.3	63,584	3478.08

The first benchmark consists of 14 signed graphs from the Konect repository [27] used in [17] TIMBAL benchmark evaluations. The supplemental PDF document describes Konect signed graphs and their characteristics in great detail. ABCD and TIMBAL performance are outlined in Figure 5. ABCD matches TIMBAL performance in the smallest three networks. ABCD algorithm finds a more significant subset for 11 Konect datasets. TIMBAL performs better on the three Konect Wiki data sets. 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 nodes obtained after 5 and 10 runs for TIMBAL, and only for one dataset did the repeated runs discover a more significant subset. For ABCD_Fast, we can observe that the performance is consistently better than TIMBAL for the exact runtime for the majority of the graphs.

6.2 ABCD for the Amazon Benchmark

Amazon benchmark consists of 17 signed graphs derived from the Amazon rating and review files [16]. 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 $(5,4) \to e^{+}$, $3 \to m$ (no sign), and $(2,1) \to e^{-}$ [16]. The characteristics of the most significant connected component are outlined in Table 3. Figure 6 (Top) and Table 4 illustrate the sub-graph size TIMBAL recovers (blue box) and the sub-graph ABCD algorithm recovers (red box). Amazon data is extensive. For millions of nodes, the ABCD algorithm performs much better than TIMBAL. One iteration of TIMBAL (blue line) takes as long as the entire ABCD algorithm (red line) for larger graphs. We detail the timing and the experiment in the supplemental PDF tables. In this experiment, the ABCD algorithm has a superior runtime and performance regarding the graph size it discovers, as illustrated in Figure 6 (Bottom). 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 (Top).

TABLE 3
Amazon ratings and reviews [16] (a subset of Amazon ratings constructed from core5 reviews) [16] mapped to signed graphs. The number of nodes, edges, and cycles reflects the number in the largest connected component of each dataset.

Amazon	**Input graph**	**Largest Connected Component**		
Ratings	**# ratings**	**# nodes**	**# 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**	**# nodes**	**# 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

6.3 ABCD vs. TIMBAL Runtime

The experiment compares the runtime for TIMBAL and ABCD as a function of the number of edges m and nodes n in the graph. We use all 31 signed graphs for this evaluation. Table 2 and Table 3 summarize the Konect and Amazon signed graph characteristics, respectively. Figure 7 illustrates a single TIMBAL run time and ABCD run time as a function of the graph size for all Konect and Amazon graphs. ABCD's performance in the most extensive sub-graph discovery is superior to TIMBAL. TIMBAL performance significantly degrades in terms of balanced graph recovery for all graphs over 350,000 nodes.

TABLE 4
Amazon ratings and reviews graph results. The time in seconds for ABCD includes 5000 iterations. The time in seconds for TIMBAL is the total time. For graphs of over a million nodes, we only had data on one run, as it takes over 100 minutes per run for TIMBAL. N/A indicates that a method does not terminate within two days.

Amazon	**TIMBAL**		**ABCD**	
Ratings	# nodes	time s	# nodes	time s
Book	N/A	N/A	**7,085,285**	116897
Electronics	N/A	N/A	**3,104,399**	37677
Jewelry	530,363	47046.34	**2,769,431**	21468
TV	891,106	11379.43	**1,579,760**	17146
Vinyl	612,700	11529.94	**1,452,496**	13011
Outdoors	683,846	12717.01	**1,640,544**	11295
AndrApp	437,740	5052.82	**977,536**	12254
Games	565,301	6251	**1,150,782**	7617.5
Automoto	140,711	12989	**744,474**	4157
Garden	122,844	5204	**522,340**	3200
Baby	229,545	3591	**397,940**	2986
Music	351,124	3223	**451,320**	2203
Video	121,694	4280	**360,665**	2176
Instruments	97,486	1785	**285,233**	1464.8
Reviews	# nodes	time s	# nodes	time s
Core Music 5	4,193	30.3	**5,143**	200.4
Core Video 5	3,419	23.7	**3,934**	128.3
Core Instrum 5	**1,725**	19.1	1,559	36.9

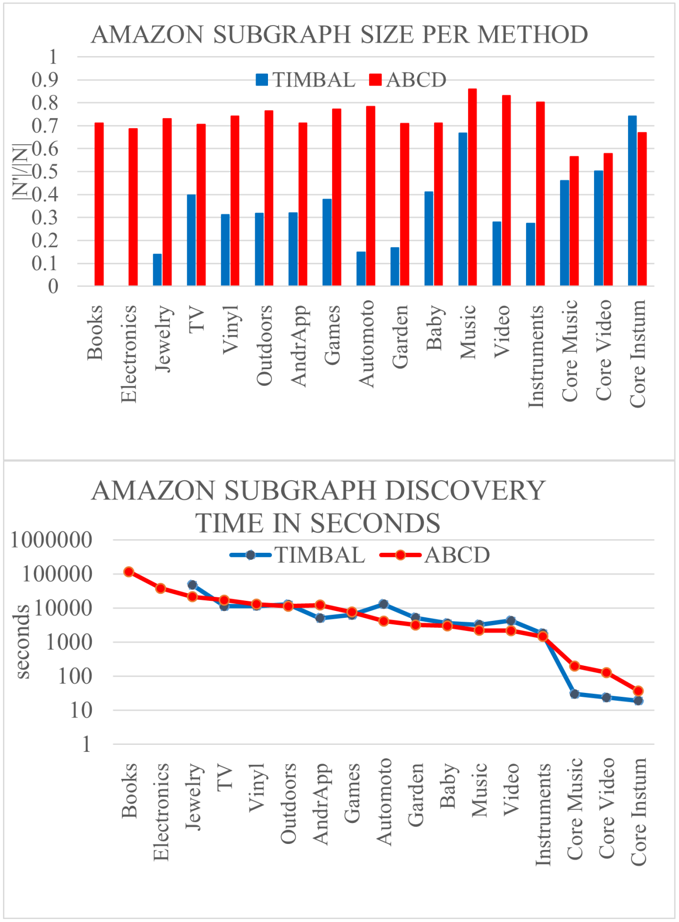

Fig. 6. ABCD and TIMBAL performance (top) and running time (bottom) comparison for Amazon data.

6.4 Parameter Sensitivity Analysis

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 heuristic 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 nodes discarded due to the increase in the number of candidate nodes and edges to be processed. We introduce a novel set of conditions (neighborhood degree, bi-cut) to remove the nodes 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 most significant size regarding the number of nodes; it is balanced as it has no unbalanced cycles, and it is a subgraph as the algorithm removes the *nodes*. ABCD recovers significantly balanced sub-graphs, which are over two times larger than state-of-the-art.

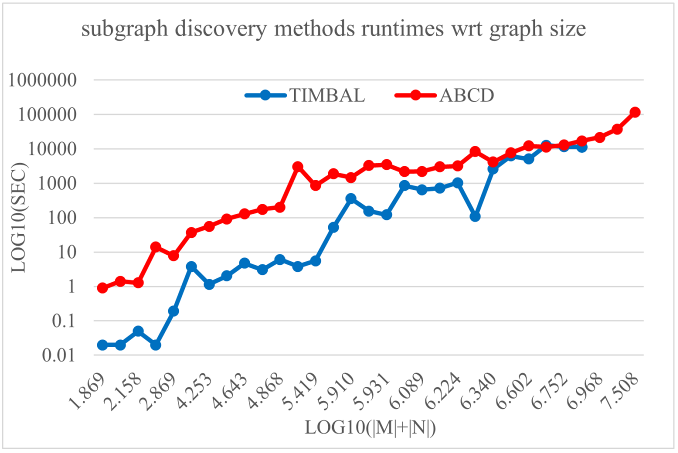

Fig. 7. ABCD and TIMBAL algorithm runtime for 28 signed graphs tested. TIMBAL failed to complete the largest two graphs.

Recently, [30] proposed faster O(m) heuristic and efficient implementation for balancing a graph and for typically obtaining a lower number of flips to reach consensus. Their paper suggests that edges, regardless of whether they belong to the spanning tree, can be chosen to be switched for balance. We plan to investigate this next, as multiple unbalanced fundamental cycles can share an edge in the spanning edge. Changing the sign of a tree edge might cause processing instabilities. Future work also includes integrating the OpenMP and GPU code accelerations. [24] has shown that GPU code takes less than 15 minutes to find 1000 fundamental cycle bases for 10M nodes and 22M edges. 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 nodes 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," *J. Psychology*, vol. 21, pp. 107–112, 1946.

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

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

[6] 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.

[7] K. Garimella, T. Smith, R. Weiss, and R. West, "Political polarization in online news consumption," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 15, 2021, pp. 152–162.

[8] 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.

[9] V. Amelkin and A. K. Singh, "Fight"g opinion control in social networks via link recommendation," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2019, pp. 677–685.

[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 minimum 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 Transactions on Computational Biology and Bioinformatics*, 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 Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 1791–1802, 2023.

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

[16] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *Proceedings of the 25th International Conference on World Wide Web*, WWW ACM, 2016, pp. 507–517.

[17] B. Ordozgoiti, A. Matakos, and A. Gionis, "Findi" large balanced subgraphs in signed networks," in *Proceedings of The Web Conference 2020* (WWW)., pp. 1378–1388, ACM, 2020.

[18] K. Sharma, I. A. Gillani, S. Medya, S. Ranu, and A. Bagchi, *Balance Maximization in Signed Networks via Edge Deletions, Proceedings of the International Conference on World Wide Web*, WWW ACM.

[19] N. Gülpinar, G. Gutin, G. Mitra, and A. Zverovitch, "Extracting pure network submatrices in linear programs using signed graphs," *Discrete Applied Mathematics*, vol. 137, no. 3, 2004.

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

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

[22] F. Bonchi, E. Galimberti, A. Gionis, B. Ordozgoiti, and G. Ruffo, "Discovering Polarized communities in signed networks," 2019.

[23] L. Boulton, "Spectral pollution and eigenvalue bounds," *Applied Numerical Mathematics*, vol. 99, pp. 1–23, 2016.

[24] G. Alabandi, J. Tešić, L. Rusnak, and M. Burtscher, "Discovering and balancing fundamental cycles in large signed graphs," in *International Conference for High-Performance Computing, Networking, Storage and Analysis*, SC '21. ACM, 2021.

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

[26] *Muhieddine Shebaro* and **Jelena Tešić**, "Identifying stable states of large signed graphs," in *Companion Proceedings of the ACM Web Conference 2023 (WWW Companion)*, 2023.

[27] J. Kunegis, "KONECT – The Koblenz Network Collection," in *Proceedings of the 22nd International Conference on World Wide Web*, WWW ACM, 2013, pp. 1343–1350.

[28] M. Lai, V. Patti, G. Ruffo, and P. Rosso, "Stance evolution and Twitter interactions in an Italian political debate," *Natural Language Processing and Information Systems*, Springer, 2018, pp. 15–27.

[29] Y. He, G. Reinert, S. Wang, and M. Cucuringu, "SSSNET:the semi-supervised signed network clustering," in *SIAM International Conference on Data Mining (SDM)*, 2021, pp. 244–252.

[30] S. Kundu and A. A. Nanavati, "A more"powerful heuristic for balancing an unbalanced graph," in *Complex Networks and Their Applications XI*, Springer, 2023, pp. 31–42.

Muhieddine Shebaro is a Computer Science Ph.D. student at Texas State University. He received his B.S.c degree in Computer Science from Beirut Arab University, Lebanon, in 2021 and defended his Ph.D. dissertation proposal in January 2024. His research interests include network science, machine learning, and data science.

Lucas Rusnak is an Associate Professor at the Department of Mathematics, Texas State University. He has a PhD in Mathematics from Binghamton University. Dr. Rusnak introduced oriented hypergraphs and hypergraphic balance theory, through which he has generalized the Matrix-tree theorem, Kirchhoff's Laws, and graph grammars to oriented hypergraphs. His research interests include anything that challenges the boundaries of classical graph theory, matrix theory, and matroid theory.

Jelena Tešić, Ph.D. is an Associate Professor at the Department of Computer Science, Texas State University. She received her Ph.D. (2004) from the University of California Santa Barbara, CA, USA. Dr. Tešić has authored over 50 peer-reviewed scientific papers and holds six US patents. Her research focuses on unstructured data representation and analysis at scale, machine learning, and network science. She is a senior IEEE member.