ABCD: Algorithm for Balanced Component Discovery in Signed Networks

Muhieddine Shebaro and Jelena Tešić

Oct 30 2023

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 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. However, it is still bounded by computational capability. The proposed approach builds on the graph characteristics and a scalable fundamental cycle discovery method to minimize the number of vertices discarded. We evaluate the proposed approach against state-of-the-art and demonstrate over two times higher graph size regarding the number of vertices selected of the discovered subset on an extensive signed network with millions of vertices and edges over the state-of-art in the same time frame.

maximum balanced subgraph, signed networks, balanced states, balance theory

1 Problem Specification

$$\Sigma' \subseteq \Sigma \wedge Fr(\Sigma') = 0 \wedge \underset{N' \leq N}{\arg\max}\, \Sigma' \implies \Sigma' \qquad (1)$$

In this paper, we consider the solution of the discovery of the largest balanced component $\Sigma', |Sigma'| = N'$ in *any* size signed graph $\Sigma, |Sigma| = N$ in Eq. 1. This is a well-known NP-hard problem [23] and existing solutions do not scale to real-world graphs [22]. We consider Σ to be a structure-free signed graph derived from real-life networks with millions of vertices and vertices e.g. [9]. Signed graph balancing is defined in Section 3.2. We propose a solution based on the scalable graph cycle basis computation of the underlying unsigned graph G of Σ. we use the edge sign switching technique using a fundamental cycle basis discovery method to *search* for the maximum balanced subgraph. The proposed approach finds the largest balanced subgraph Σ' of *any* Σ is $O(K*(N*M))$ where N is the number of vertices, M is the number of edges in Σ and algorithm is considering only the top K balanced states with the lowest frustration index. We use the state-of-art method proposed in [16] for baseline comparisons in Section 5.

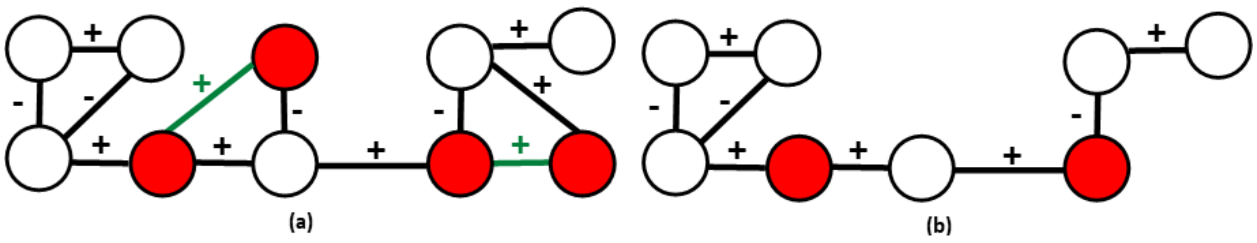

Figure 1: (a): The unbalanced signed network. Green edges are the candidate edges causing imbalance, and red vertices are the candidate vertices. (b): The maximum balanced signed sub-graph obtained after deleting one candidate node along each edge.

2 Design Considerations

Related Work Signed networks allow for negative weights, representing antagonistic relationships or conflicting opinions [22]. The task of the largest balanced sub-graph discovery has applications in portfolio system's economic risk management [8], computational and operational research [6], community analysis and structure [15], computational biology to model balanced interactions between genes and proteins [14] and social network analysis [5]. 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. Still, they are essential for understanding how the system behaves. Moreover, by locating the maximum balanced sub-graphs, we can simplify the system into sub-systems with balanced interactions and eliminate inconsistencies regarding unbalanced cycles. [7] proposed the GGMZ algorithm that computes graphs minimum spanning tree, selects a subset of vertices, and all the edges crossing that subset are inverted to create positive edges. The resulting set of vertices disconnected by negative edges is returned as the final output of the algorithm, and the complexity of the algorithm is $O(N^3)$. [17] show that any signed graph contains a balanced sub-graph with at least 0.5M + 0.25(N- 1) edges. [6] propose GRASP algorithm that randomly selects a subset of vertices and then greedily adds vertices that maximize the number of edges connecting them to the current subset while keeping the size of the subset balanced. [2] propose the EIGEN algorithm that relies on the dominant eigenvector of the adjacency matrix to partition the graph into two disjoint sets. The algorithm then recursively this partitioning process on each of the two sets until the desired level of balance is achieved. [19] uses 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. [16] proposes the *trimming iteratively to maximize balance* TIMBAL algorithm that uses a two-stage method approach where the first stage removes vertices. The second one restores them as long as it does not cause imbalance. TIMBAL proposes a novel bound for perturbations of the graph Laplacian pre-processing and sub-sampling techniques to scale the processing for large graphs. The vertices deleted from at least one of these sub-graphs are then deleted from the original graph. TIMBAL performs well on the artificially created graphs, and its performance is used as a benchmark for real signed graphs in this paper for the first time. TIMBAL relies on the costly eigenvalue computation ($O(N^2)$). [3] showed that TIMBAL performance decreases due to spectral pollution in eigenvalue computation. Recently, [11] 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. [1] has shown that GPU code takes less than 15 minutes to find 1000 fundamental cycle bases for 10M vertices 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 vertices and edges.

This paper proposes an algorithm for balanced consensus 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 [1] and utilizes the graph's node density distribution and near-optimal balanced states to minimize the number of vertices removed from the balanced sub-graph. The paper is organized as follows: in Section 1, we formally describe the objective and the problem related to finding the maximum balanced subgraph in the signed network; in Section 3, we present related definitions and corollaries that lead to the proposed solution. In Section 4 we introduce the novel ABCD algorithm and the implementation details; in Section 5, we present proof of concept; and in Section 6 we summarize our findings.

3 Definitions and Corollaries

3.1 Fundamental Cycle Basis

Definition 3.1. *Graph Σ' is a **subgraph** of a graph Σ if **all** edges and vertices of Σ' are contained in Σ.*

Definition 3.2. ***Path** is a sequence of distinct edges m that connect a sequence of distinct vertices n 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. **Cycle Basis** is a set of simple cycles that forms a basis of the cycle space.*

Definition 3.3. *For the underlying graph G, let T be the spanning tree of G, and let an edge m 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 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. For the graph G with N vertices and M edges, there are exactly $M - N + 1$ fundamental cycles.*

3.2 Balanced Graphs and Frustration

Definition 3.4. ***Signed graph** $\Sigma = (G, \sigma)$ consists of underlying unsigned graph G and an edge signing function $\sigma : m \rightarrow \{+1, -1\}$. 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.* ***Frustration*** *of a signed graph is defined as the number of candidate edges whose sign needs to be switched for the graph to reach the balanced state.*

Theorem 3.2 ([4])**.** *If a signed subgraph Σ' is balanced, the following are equivalent:*

1. *Σ' is balanced. (All circles are positive.)*
2. *For every vertex pair $(n_i, n_j) in \Sigma'$, all (n_i, n_j)-paths have the same sign.*
3. *$Fr(\Sigma') = 0$.*
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.

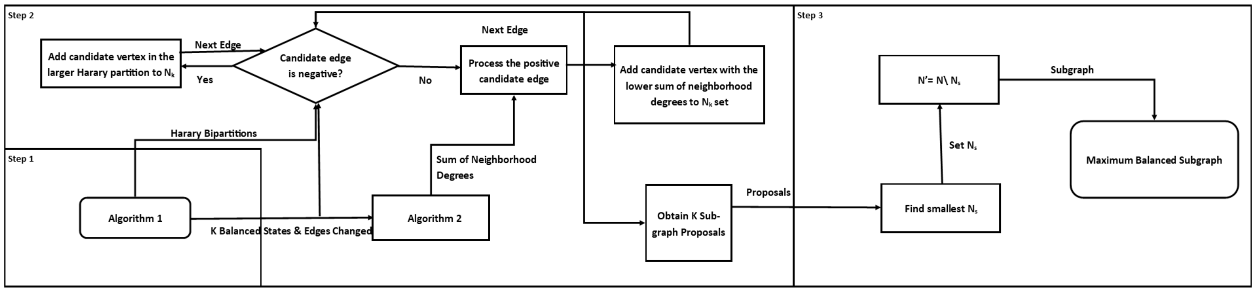
Step 2
Add candidate vertex in the larger Harary partition to N_k
Next Edge
Yes
Candidate edge is negative?
No
Next Edge
Process the positive candidate edge
Add candidate vertex with the lower sum of neighborhood degrees to N_k set
Harary Bipartitions
Step 1
Sum of Neighborhood Degrees
Algorithm 1
K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

Figure 2: The algorithmic illustration of the Algorithm for Balanced Component Discovery (ABCD).

4 Methodology

Balancing a signed network via edge sign switching identifies a set of candidate edges to have their sign switched. Removal of such edges yields the removal of the fundamental cycles that are unbalanced and produces a unique partitioning of input graphs to balanced subgraphs. Optimal balancing states require minimum edge signed switched and thus we consider only the candidates to reach such states. Note that optimal balanced states do not necessarily have the same amount of candidate edges that need to be switched. [18], and their frustrations are different. First, we propose to obtain the variety of optimal balanced state candidates by sampling the graph multiple times and using the fundamental cycle basis discovery method as a base [1, 18]. Next, we propose to process the edges that cause imbalance by deleting one of the vertices along each candidate edge. The proposed approach removes *one* of the vertices along these candidate edges to minimize the number of vertices simultaneously lost and obtain the largest possible balanced sub-graph. The criteria for choosing the vertices to purge is that if the candidate edge is positive, we delete the one that "carries" fewer vertices (degree/sum of neighborhood degree). If the edge is negative, we exploit the concept of Harary bipartition [4] and remove the vertex in the smaller partition because it would be connected to a smaller number of vertices. Thus, the loss of vertices in the process is minimized to reach a balanced sub-graph and maximize the vertex cardinality of that sub-graph simultaneously.

In this paper, we introduce the **Algorithm for the Balanced Component Discovery** (ABCD) as a scalable solution for the discovery of the largest balanced subgraph in Alg. 1. [18] have proven that the spanning tree-based approach can discover fundamental cycles and balance the graph. The approach produces different optimal balanced states of Σ, as defined in 4.1. In [1], we have proposed an efficient data structure and algorithm to discover fundamental cycles if given the spanning tree T. We demonstrated that the discovery and analysis of the fundamental cycles can be computed with linear time complexity and only requires a linear amount of memory. The algorithm outline in 1 consists of three steps, as illustrated in Figure 2.

Algorithm 1 ABCD Phase 1

```
Data: Signed graph $\Sigma$, $I$,$K$
Result: $\mathcal{M}_\Sigma = M_k, \mathcal{H}_\Sigma = H_k, k \in [1, K]$
1  Generate set $\mathcal{T}_i$ of $I$ spanning trees of $\Sigma$
2  Counter to keep only the top optimal balanced states $i = 0; m_K = m$;
3  for $i = 0; i++; i < I$ do
4      for edges $m$, $m \in \Sigma \setminus T_i$ do
5          if fundamental cycle $T \cup m$ is negative then
6              Add edge $m$ to $M_i$
7      if $|M_i| < m_K$ then
8          $\mathcal{M}_\Sigma \cup = M_k$
9      if $|\mathcal{M}_\Sigma| > K$ then
10         Remove the largest set and update $m_K$
11 for $i = 0; i++; i < K$ do
12     Create zero vector $H_k$ of dimension $n$
13     for edge $m \in M_i$ do
14         Switch edge sign in $\Sigma_k$: $m^- \rightarrow m^+; m^+ \rightarrow m^-$
15     Cut all the negative edges to create Harary bi-partitions $A$ and $B$ so that $|A| > |B|$
16     for $n$ in $N$ do
17         if $n \in A, H_k(n) = 1$
```

Definition 4.1. *The balanced states are* ***optimal*** *if and only if it requires a minimum number of edge sign switches in the original graph to reach a balanced state.*

ABCD phase 1 This phase 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 optimal 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.

Algorithm 2 ABCD Phase 2

```
Data: $H_k$, $M_k$
Result: $N_k$ which is the entire set of vertices of the $k$th graph to keep when reconstructing
        the original graph
18 Counter to keep only the top optimal balanced states $i = 0; m_K = m$;
19 Initialize empty set $N_k = \varnothing$ for all values of k
20 for $i = 0; i++; i < K$ do
21     for edges $m$, $m \in M_k$ do
22         if $m+$ then
23             Append the vertex $x$ along $m+$ that has a lower sum of neighborhood degrees to
               set $N_k$
24         else
25             Append the vertex $x$ along $m-$ where $H_k(x) = 1$ to set $N_k$
```

ABCD phase 2 This phase 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 vertices removed from the graph increases the cardinality of the largest balanced sub-graph. *Harary bipartition* separates the vertices of the balanced graph into two sets such that the vertices of both sets internally agree with each other but disagree with the vertices of the other set[4]. As a labeling vector, the H_k set is created 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 M_k list of edges that should have a different sign for the entire graph to be balanced, we initialize an empty set of vertices N_k. For every edge m in M_k, Alg. 2 repeats the following steps:
2.1. If the edge m is positive, it will be negative in the balanced state. If either vertices is already in N_k, move on to the next edge. Else, add the edge-defining node n to N_k so that $H_k(n) = 0$. If they are both 0 or both 1, move this edge to the end of the M_k set and revisit. The remaining node is in the largest Harary partition for a fully balanced graph so that it will be connected to more vertices than the node ending up in the smaller Harary set after partitioning.
2.2. If the edge m is negative and marked for switching to positive: if either vertex is already

in N_k, nothing needs to be done; move on to the next edge. We add the node with the lower neighborhood degree to N_k.

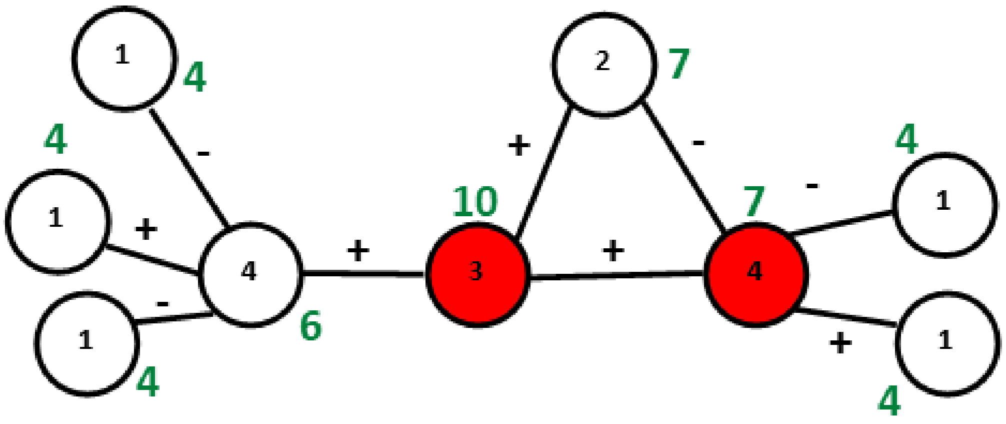

Figure 3: Degree (black, in node) vs. Sum of Neighborhood Degrees (green, next to the node) computation. The sum of neighborhood degrees labels the red vertices connected by a positive link that will be deleted.

Fig. 3 illustrates how we compute the neighborhood degree for an exemplar signed graph. Two red vertices in the image indicate the balancing algorithm 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. Thus, 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 the one connected to another edge in M_k set of edges marked to form an unbalanced fundamental cycle. Note that the *neighborhood* degree is computed for all vertices in the signed graph once and re-used for computation.

ABCD phase 3 This phase finds the index of the smallest sized N_k set among all $N_k, k \in [1, K]$ sets. Let it be index $s : |N_s| \leq |N_k|, k \in [1, K]$. The resulting maximized balanced sub-graph proposal N_s' is finally obtained by removing specific vertices as $N' = N \setminus N_s$. The remaining subset is balanced as all fundamental cycles in the graph are balanced.

Illustrative Example of the ABCD algorithm The full ABCD algorithmic flow is illustrated in Figure 2. Figure 4 illustrates the step-by-step ABCD method on the sampled signed graph. In the top row, the signed graph with 7 vertices and 10 edges is introduced. Balancing occurs in ABCD phase 1 (Alg.1), and green text on the edges with the changed signs. Green numbers beside vertices are the sum of neighborhood degrees. Orange edges are the edges of the unbalanced fundamental cycles. Green edges are the candidate edges. Red vertices are the candidate for deletion in ABCD phase 2 (Alg.2). Final candidate selection and comparison is in ABCD phase 3, and we found that by removing node $v2$, the remaining subgraph is balanced. The red dotted oval over the graph in Step 3 signifies that the final output N_s' of ABCD has a maximal cardinality.

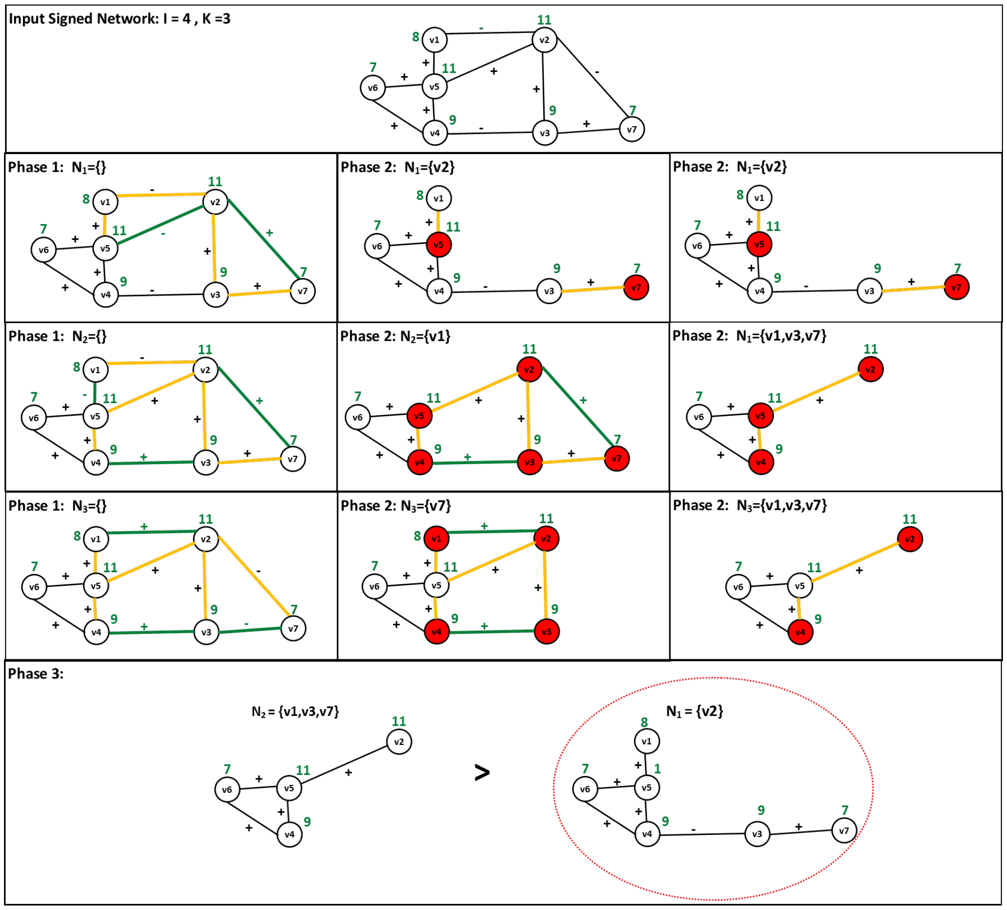

Figure 4: The ABCD algorithm applied to a sample signed graph with 7 vertices and 10 edges. Balancing occurs in Phase 1 with green text on the edges with the changed signs. Green numbers beside nodes are the sum of neighborhood degrees. Orange edges are the edges of the unbalanced fundamental cycles. Green edges are the candidate edges. Red nodes are the candidate nodes for deletion. The red dotted oval over the graph in Phase 3 signifies that the final output N'_s of ABCD has a maximal cardinality.

5 Proof Of Concept

Observations The following have helped refine and shape our method as well as establish an underpinning on which our algorithm stands: First, we observed that a balanced state with the lowest frustration index for a specific signed network does not necessarily yield a maximum balanced sub-graph using the set of conditions we laid out for processing positive and negative candidate edges (relying on Harary bipartitions for positive edges and the sum of neighborhood degrees for negative edges for the decision-making for vertex deletion). However, 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. Hence, we only capture the information on the unbalanced fundamental cycles and the Harary bipartition labeling for the top K unique fundamental cycle bases with the lowest cardinality. Second, among the tree-sampling techniques such as depth-first search, randomized depth-first search, and breadth-first search, we observed that the latter is the most suitable for this kind of task because it relatively reduces the amount of identified candidate edges compared to others. Third, the largest connected component (LCC) produced at the end is *guaranteed* to be balanced. This algorithm eliminates the unbalanced cycle bases by leaving a "hole," which prevents any consensus conflict. Therefore, this component is then called the maximum (it has the largest size in terms of the number of vertices) balanced (it has no unbalanced cycles) sub-graph (the number of nodes in this component is less than or equal to that of the entire signed network input) where edge sign switching is absent.

Implementation is in C++. The algorithm identifies the largest connected component (LCC) and applies the ABCD algorithm to LCC. All real-world benchmark graphs have one large connected component. The implementation treats edges without signs as positive edges in the fundamental cycle. If the graph has more equal connected components, the implementation accommodates that scenario. ABCD phase 1 (Alg. 1) implementation builds on [1, 18]. [20] have recently shown that the breadth-first search sampling of the spanning trees captures the diversity of the optimal balanced states, and 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, N_s' is constructed by the algorithm re-reading the original graph and skipping the entries with vertices in N_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 $300K,000 < N$ vertices. **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 $300K,000 < N$ vertices.

Baseline for the proof of concept is TIMBAL [16]. TIMBAL approach has been shown to reach the highest cardinality of the sub-graphs in various datasets and is a de-facto state of the art [16]. The input parameters of TIMBAL [16] are set as follows for all subsample_flag=False, samples=4 based on the paper implementation. The parameter max_removals=1 is set for small graphs (under 1000 vertices) and to max_removals=100 for the rest of the signed networks. We set avg_size=20 for datasets of several vertices less than 80,000, and subsample_flag=True, samples = 1000, avg_size = 200 max_removals=100 for datasets with the number of vertices greater than 80,000. TIMBAL is a non-deterministic algorithm, and we run it 5 and 10 times for Konect data to get the maximum 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 subgraph 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. It has one socket, two threads per core, and eight cores per socket. The architecture is X86_x64. GPU is Nvidia GeForce having 8GB memory. 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.

Table 1: Konect sets plus TwitterReferendum and PPI signed graphs attributes: $|N|$ is the total number of vertices; $|M|$ 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 pre-processing techniques such as removing duplicate and inconsistent edges are applied. Konect (except TwitterReferendum and PPI, which are not from the Konect site) 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.

Konect	**Largest Connected Component Graph**			TIMBAL subgraph				ABCD	
				5 runs		10 runs		5000	
Dataset	# vert	# edges	**# cycles**	# vert	time (s)	# vert	time (s)	# vert	time (s)
Highland	16	58	43	13	0.1	13	0.2	13	0.9
CrisisInCloister	18	126	145	8	0.25	8	0.5	8	1.28
ProLeague	16	120	105	10	0.1	10	0.2	10	1.40
DutchCollege	32	422	391	29	0.1	29	0.2	**30**	14
Congress	219	521	303	207	0.85	207	1.9	207	7.79
PPI	3,058	11,860	8,803	900	78.45	900	156.9	**2,072**	97.59
BitcoinAlpha	3,775	14,120	10,346	3,014	5.57	3,081	11.4	**3,146**	55.68
BitcoinOTC	5,875	21,489	15,615	4,250	10.15	4,349	20.3	**4,910**	92.29
Chess	7,115	55,779	48,665	2,230	15.25	2,320	30.5	**2,551**	174.67
TwitterReferendum	10,864	251,396	240,533	9,021	27,6	9,110	55.2	**9,438**	851.94
SlashdotZoo	79,116	467,731	388,616	39,905	259.4	40,123	518.8	**43,544**	1870.20
Epinions	119,130	704,267	585,138	73,433	774.9	74,106	1549.8	**74,843**	3272.87
WikiElec	7,066	100,667	93,602	**3,758**	18.95	**3,856**	37.9	3,506	3033.08
WikiConflict	113,123	2,025,910	1,912,788	**56,768**	539.25	56,768	1078.5	54,476	8332.22
WikiPolitics	137,740	715,334	577,595	67,009	602.65	**69,050**	1205.3	63,584	3478.08

5.1 Konect Benchmark Evaluation

The first benchmark consists of 14 signed graphs from the Konect repository [12] used in [16] TIMBAL benchmark evaluations. Konect signed graphs and their characteristics are described in the supplemental PDF document 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 of 13 konect datasets. TIMBAL performs better on the three Konect Wiki data sets. TIMBAL is faster than ABCD on smaller networks. For the largest graph in the collection, Epinions, ABCD takes double the time to recover the largest balanced sub-graph. The comparison graphs are outlined in Figure 5. 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 larger subset. For ABCD_Fast, we can observe that the performance is consistently better than TIMBAL for the same runtime for the majority of the graphs.

5.2 Amazon Benchmark Evaluation

Amazon benchmark consists of 17 signed graphs derived from the Amazon rating and review files [9]. 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

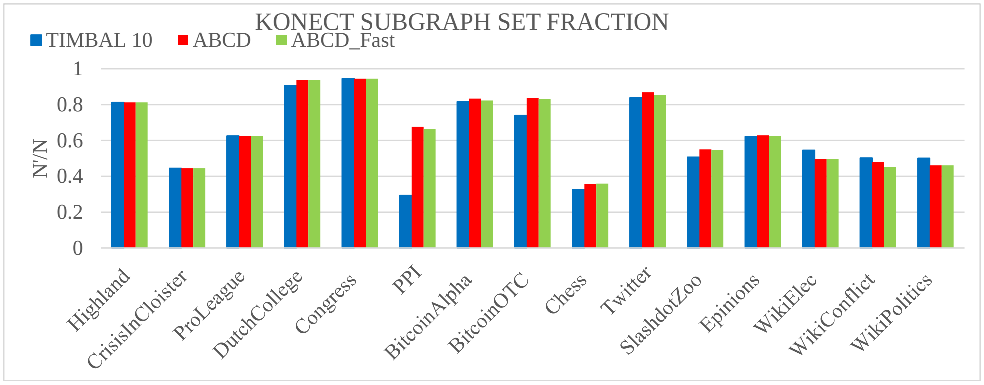

Figure 5: ABCD and TIMBAL performance comparison for Konect benchmark in terms of subset graph fractions

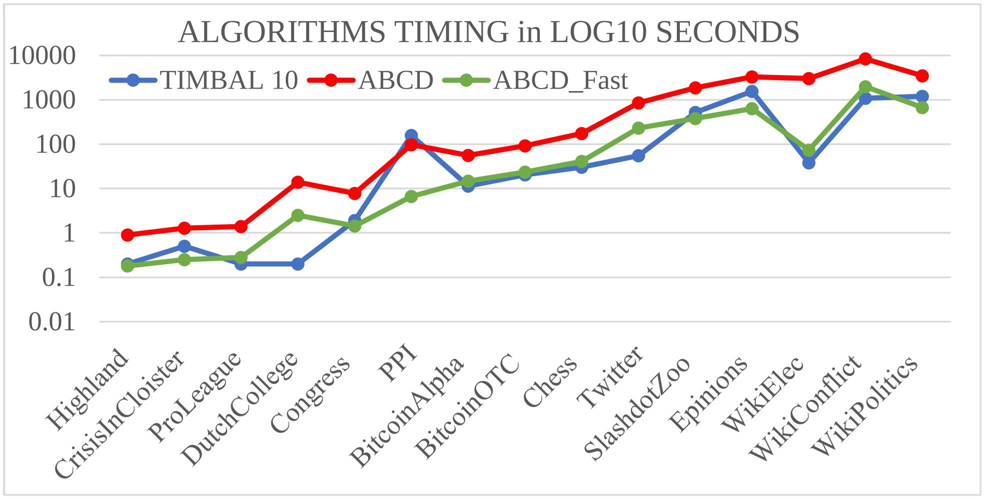

Figure 6: ABCD and TIMBAL performance comparison for Konect benchmark in terms of algorithmic timing

as follows $(5, 4) \to m^+$, $3 \to m$ (no sign), and $(2, 1) \to m^-$ [9]. The characteristics of the largest connected component are outlined in Table 2. Figure 7 illustrates the sub-graph size TIMBAL recovers (blue box), the sub-graph ABCD algorithm recovers (red box). Amazon data is large. For millions of vertices, 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, ABCD algorithm has a superior runtime and performance regarding the graph size it discovers, as illustrated in Figure 7 and Figure 9. TIMBAL's performance degrades with the graph size, and the discovered sub-graphs are much smaller than what ABCD finds, as illustrated in Figure 7.

Table 2: Amazon Benchmark LCC [9], and ratings and reviews ABCD and TIMBAL results. The time in seconds for ABCD includes 5000 iterations. The time in seconds for TIMBAL is the total time. For graphs over a million vertices, we had only 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.

	Largest Connected			Methods			
Amazon	**Component Graph**			**TIMBAL**		**ABCD**	
Ratings	**# vertices**	**# edges**	**# cycles**	# vertices	time s	# vertices	time s
Books	9,973,735	22,268,630	12,294,896	N/A	N/A	**7,085,285**	116897
Electronics	4,523,296	7,734,582	3,211,287	N/A	N/A	**3,104,399**	37677
Clothing	3,796,967	5,484,633	1,687,667	530,363	47046.34	**2,769,431**	21468
Movies/TV	2,236,744	4,573,784	2,337,041	891,106	11379.43	**1,579,760**	17146
CDs	1,959,693	3,684,143	1,724,451	612,700	11529.94	**1,452,496**	13011
Outdoors	2,147,848	3,075,419	927,572	683,846	12717.01	**1,640,544**	11295
Android App	1,373,018	2,631,009	1,257,992	437,740	5052.82	**977,536**	12254
Toys and Games	1,489,764	2,142,593	652,830	565,301	6251	**1,150,782**	7617.5
Automotive	950,831	1,239,450	288,620	140,711	12989	**744,474**	4157
Garden	735,815	939,679	203,865	122,844	5204	**522,340**	3200
Baby	559,040	892,231	333,192	229,545	3591	**397,940**	2986
Digital Music	525,522	702,584	177,063	351,124	3223	**451,320**	2203
Instant Video	433,702	572,834	139,133	121,694	4280	**360,665**	2176
Music Instr.	355,507	457,140	101,634	97,486	1785	**285,233**	1464.8
Reviews	**# vertices**	**# edges**	**# cycles**	# vertices	time s	# vertices	time s
Music 5	9,109	64,706	55,598 5	4,193	30.3	**5,143**	200.4
Video 5	6,815	37,126	30,312	3,419	23.7	**3,934**	128.3
Music Instr 5	2,329	10,261	7,933 5	3,419	23.7	**3,934**	128.3

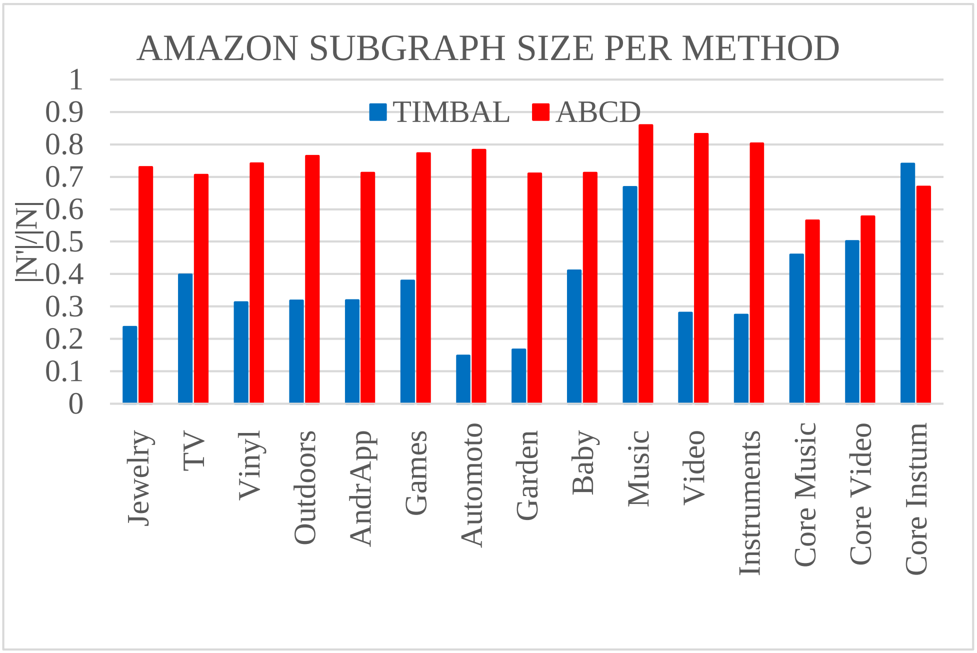

Figure 7: ABCD and TIMBAL performance comparison for Amazon data.

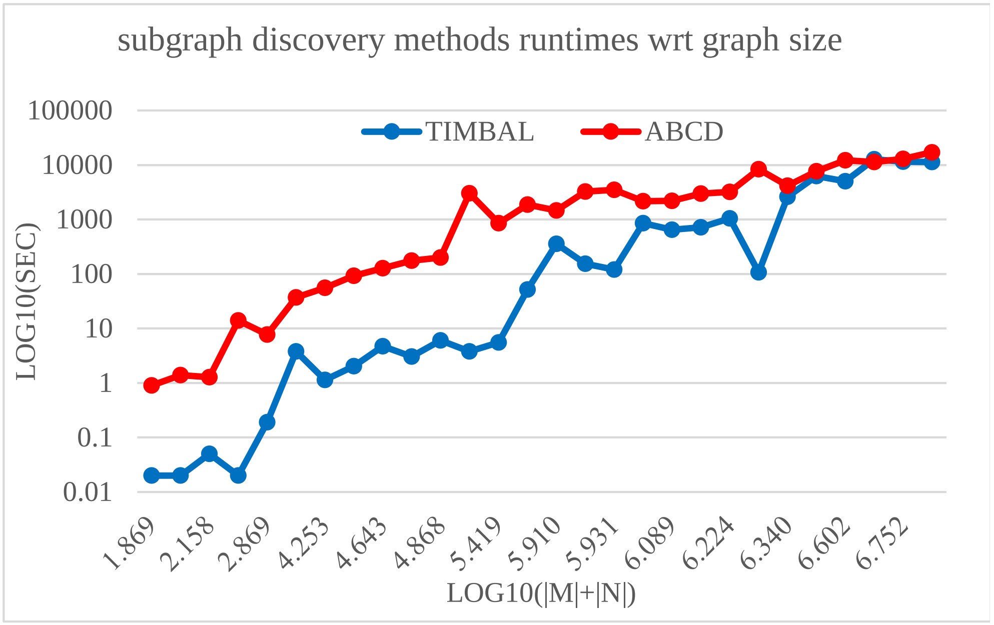

Figure 8: ABCD and TIMBAL algorithm are linear with the graph size M + N for 28 signed graphs tested. TIMBAL failed to complete for the largest two graphs.

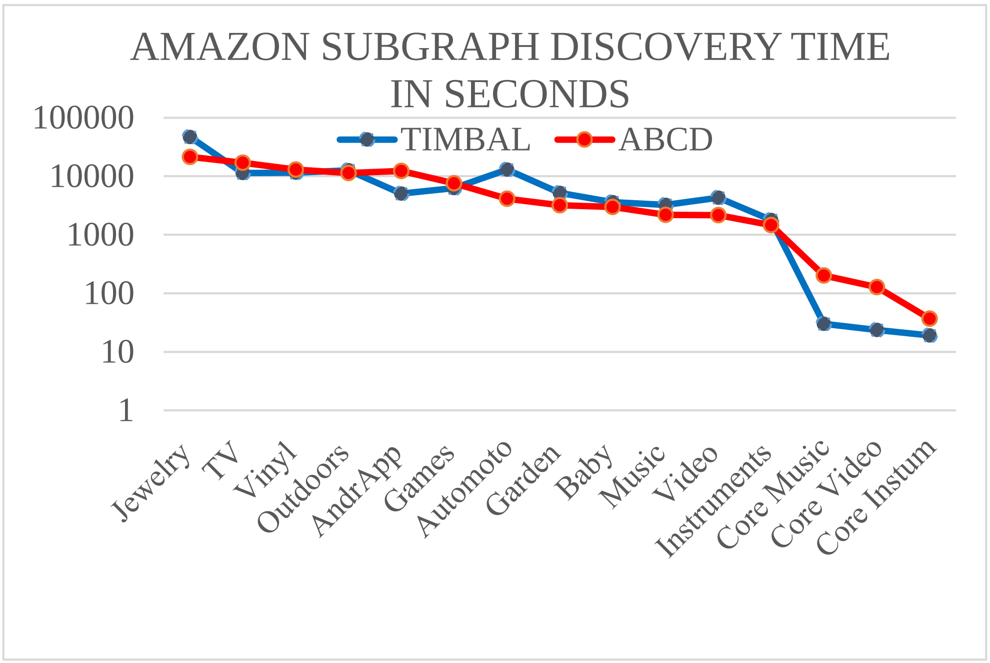

Figure 9: ABCD and TIMBAL running time comparison for Amazon data.

5.3 Graph Size vs. Runtime Experiment

evaluation considers runtime for TIMBAL and total runtime for ABCD as a function of number of edges M and number of vertices N in the graph. We use all 31 signed graphs for this evaluation. Details on the graph characteristics are listed in Table 3 and Table 2.

Figure 8 illustrates a single TIMBAL run time and ABCD run time as a function of the graph size for all Konect and Amazon graphs. Both algorithms have approximately run times linear with the size of the graph. 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 vertices.

6 Conclusion and Future work

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. In this paper, we 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 fundamental cycle bases with the lowest number of unbalanced 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, bicut) to remove the vertices from the graph. The output of the ABCD algorithm is guaranteed to be balanced. ABCD eliminates the unbalanced cycle bases by removing the edges. Thus, the cycle turns into an open path. The resulting subgraph has the the largest size in terms of 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 subgraphs, over two times larger than the state of art while keeping the processing time linear with the size of the graph.

7 Appendix

7.1 Konect Graphs and ABCD Performance

Konect Signed Graphs Konect signed graphs and their characteristics are described in Table 3. *Highland* is the signed social network of tribes of the GahukuGama alliance structure of the Eastern Central Highlands of New Guinea, from Kenneth Read [12]. *CrisisInCloister* is a directed network that contains ratings between monks related to a crisis in a cloister (or monastery) in New England (USA) which led to the departure of several of the monks [12]. *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 positive if the home team wins, negative if the away team wins, and zero for a draw [12]. *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

Table 3: Konect sets plus TwitterReferendum and PPI signed graphs attributes: N is the total number of vertices; 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 pre-processing techniques such as removing duplicate and inconsistent edges are applied.

Konect	**Entire Input Signed Network**		**Largest Connected Component Graph**		
Dataset	# vertices	# edges	# vertices	# edges	**# cycles**
Highland	16	58	16	58	43
CrisisInCloister	18	126	18	126	145
ProLeague	16	120	16	120	105
DutchCollege	32	3,062	32	422	391
Congress	219	764	219	521	303
PPI	3,058	11,860	3,058	11,860	8,803
BitcoinAlpha	3,783	24,186	3,775	14,120	10,346
BitcoinOTC	5,881	35,592	5,875	21,489	15,615
Chess	7,301	65,053	7,115	55,779	48,665
TwitterReferendum	10,884	251,406	10,864	251,396	240,533
SlashdotZoo	79,120	515,397	79,116	467,731	388,616
Epinions	131,828	841,372	119,130	704,267	585,138
WikiElec	7,118	103,675	7,066	100,667	93,602
WikiConflict	118,100	2,917,785	113,123	2,025,910	1,912,788
WikiPolitics	138,592	740,397	137,740	715,334	577,595

from -1 for the risk of conflict to +3 for best friend [12]. *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 [12]. 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 [12]. *BitcoinAlpha* is a user-user trust/distrust network from the Bitcoin Alpha platform on which Bitcoins are traded [12].*BitcoinOTC* is a user-user trust/distrust network, from the Bitcoin OTC platform, on which Bitcoins are traded [12]. *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 [13]. *WikiElec* is the network of users from the English Wikipedia that voted for and against each other in admin elections [12]. *Slashdot* is the reply network of the technology website Slashdot. vertices are users, and edges are replies [12]. The edges of *WikiConflict* represent positive and negative conflicts between users of the English Wikipedia [12]. *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 [12]. *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 [12].[10]. The characteristics of the signed graphs are listed in Table 3.

7.2 Amazon Graphs and ABCD Performance

Amazon benchmark consists of 17 signed graphs derived from the Amazon rating and review files [9]. The dataset contains product reviews and metadata from Amazon, spanning May 1996 - July 2014. We use the Amazon rating dataset as (user, item, rating, timestamp) tuples for these experiments. We also use a smaller version of Digital Music (64,706 reviews), Musical Instruments (10,261 reviews), and Amazon Instant Video (37,126 reviews) ratings, where there are five or more reviews provided per product. Rating score is mapped into an edge between the user and the product as follows $(5,4) \to m^{+}$, $3 \to m$ (no sign), and $(2,1) \to m^{-}$ [9]. The summary of the datasets is outlined in Table 4.

Table 4: Amazon ratings and reviews (a subset of Amazon ratings constructed from core5 reviews) [9] 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
Clothing, Shoes, and Jewelry	5,748,920	3,796,967	5,484,633	1,687,667
Movies and TV	4,607,047	2,236,744	4,573,784	2,337,041
CDs and Vinyl	3,749,004	1,959,693	3,684,143	1,724,451
Sports and Outdoors	3,268,695	2,147,848	3,075,419	927,572
Android App	2,638,172	1,373,018	2,631,009	1,257,992
Toys and Games	2,252,771	1,489,764	2,142,593	652,830
Automotive	1,373,768	950,831	1,239,450	288,620
Patio, Lawn, and Garden	993,490	735,815	939,679	203,865
Baby	915,446	559,040	892,231	333,192
Digital Music	836,006	525,522	702,584	177,063
Instant Video	583,993	433,702	572,834	139,133
Musical Instruments	500,176	355,507	457,140	101,634
Reviews	**# reviews**	**# vertices**	**# edges**	**# cycles**
Digital Music core5	64,706	9,109	64,706	55,598
Instant Video core5	37,126	6,815	37,126	30,312
Musical Instruments core5	10,621	2,329	10,261	7,933

References

[1] Ghadeer Alabandi, Jelena Tešić, Lucas Rusnak, and Martin Burtscher. 2021. Discovering and Balancing Fundamental Cycles in Large Signed Graphs. In *Proc. of the Intl. Conf. for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. ACM, NY, USA, Article 68, 17 pages.

[2] Francesco Bonchi, Edoardo Galimberti, Aristides Gionis, Bruno Ordozgoiti, and

Table 5: Amazon ratings and reviews graph characteristics [9]. The number of vertices, edges, and cycles reflect the number in the largest connected component of each dataset. The time in seconds for TIMBAL is the total time. For graphs over million vertices, we had only data on one run as it takes over 100 minutes per run for TIMBAL. The time in seconds for ABCD includes 5000 iterations. N/A indicates that a method does not terminate within two days.

Amazon			**TIMBAL**			**ABCD**	
Ratings	N	M	# vertices	time s	#runs	# vertices	time s
Books	9,973,735	22,268,630	N/A	N/A	1	**7,085,285**	116897
Electronics	4,523,296	7,734,582	N/A	N/A	1	**3,104,399**	37677
Jewelry	3,796,967	5,484,633	530,363	47046.34	1	**2,769,431**	21468
TV	2,236,744	4,573,784	891,106	11379.43	1	**1,579,760**	17146
Vinyl	1,959,693	3,684,143	612,700	11529.94	1	**1,452,496**	13011
Outdoors	2,147,848	3,075,419	683,846	12717.01	1	**1,640,544**	11295
Android App	1,373,018	2,631,009	437,740	5052.82	1	**977,536**	12254
Games	1,489,764	2,142,593	565,301	6251	1	**1,150,782**	7617.5
Automotive	950,831	1,239,450	140,711	12989	5	**744,474**	4157
Garden	735,815	939,679	122,844	5204	5	**522,340**	3200
Baby	559,040	892,231	229,545	3591	5	**397,940**	2986
Digital Music	525,522	702,584	351,124	3223	5	**451,320**	2203
Instant Video	433,702	572,834	121,694	4280	5	**360,665**	2176
Musical Instruments	355,507	457,140	97,486	1785	5	**285,233**	1464.8
Reviews	N	M	# vertices	time s	#runs	# vertices	time s
Digital Music	9,109	64,706	4,193	30.3	5	**5,143**	200.4
Instant Video	6,815	37,126	3,419	23.7	5	**3,934**	128.3
Musical Instruments	2,329	10,261	**1,725**	19.1	5	1,559	36.9

Giancarlo Ruffo. 2019. Discovering Polarized Communities in Signed Networks. arXiv:1910.02438 [cs.DS]

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

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

[5] Chen Chen, Yanping Wu, Renjie Sun, and Xiaoyang Wang. 2023. Maximum Signed θ-Clique Identification in Large Signed Graphs. *IEEE Transactions on Knowledge and Data Engineering* 35, 2 (2023), 1791–1802.

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

[7] N Gülpinar, G Gutin, G Mitra, and A Zverovitch. 2004. Extracting pure network

submatrices in linear programs using signed graphs. *Discrete Applied Mathematics* 137, 3 (2004), 359–372.

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

[9] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *Proc. of the 25th Intl. Conf. on World Wide Web (WWW '16).* ACM, 507–517.

[10] Yixuan He, Gesine Reinert, Songchao Wang, and Mihai Cucuringu. 2021. SSSNET: Semi-Supervised Signed Network Clustering. *Proc. of the 2022 SIAM Intl. Conf. on Data Mining (SDM)* 244–252 (2021). arXiv:2110.06623

[11] Sukhamay Kundu and Amit A. Nanavati. 2023. A More Powerful Heuristic for Balancing an Unbalanced Graph. In *Complex Networks and Their Applications XI*, Springer, Cham, 31–42.

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

[13] Mirko Lai, Viviana Patti, Giancarlo Ruffo, and Paolo Rosso. 2018. Stance Evolution and Twitter Interactions in an Italian Political Debate. In *Natural Language Processing and Information Systems.* Springer, Cham, 15–27.

[14] C. Liu, Y. Dai, K. Yu, and Z. Zhang. 2022. Enhancing Cancer Driver Gene Prediction by Protein-Protein Interaction Network. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 19, 4 (2022), 2231 – 2240.

[15] Kevin T. Macon, Peter J. Mucha, and Mason A. Porter. 2012. Community structure in the United Nations General Assembly. *Physica A: Statistical Mechanics and its Applications* 391, 1 (2012), 343–361.

[16] Bruno Ordozgoiti, Antonis Matakos, and Aristides Gionis. 2020. Finding Large Balanced Subgraphs in Signed Networks. In *Proc. of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20).* ACM, 1378–1388.

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

[18] Lucas Rusnak and Jelena Tešić. 2021. Characterizing Attitudinal Network Graphs through Frustration Cloud. *Data Mining and Knowledge Discovery* 6 (November 2021).

[19] Kartik Sharma, Iqra Altaf Gillani, Sourav Medya, Sayan Ranu, and Amitabha Bagchi. 2021. *Balance Maximization in Signed Networks via Edge Deletions.* ACM, NY, USA, 752–760.

[20] Muhieddine Shebaro and Jelena Tešić. 2023. Identifying Stable States of Large Signed Graphs. In *Proc. of the 32nd International Conference Companion on World Wide Web (WWW '23)*. ACM.

[21] Arunachalam Vinayagam, Jonathan Zirin, Charles Roesel, Yanhui Hu, Bahar Yilmazel, Anastasia A Samsonova, Ralph A Neumüller, Stephanie E Mohr, and Norbert Perrimon. 2014. Integrating protein-protein interaction networks with phenotypes reveals signs of interactions. *Nature Methods* 11, 1 (2014), 94 – 99.

[22] Yuxin Wu, Deyuan Meng, and Zheng-Guang Wu. 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 (2022), 1166 – 1187.

[23] Thomas Zaslavsky. 2012. A Mathematical Bibliography of Signed and Gain Graphs and Allied Areas. *Electronic Journal of Combinatorics* 9, 7 (2022), 1166 – 1187.