Scaling Frustration Index and Corresponding Balanced State Discovery for Real Signed Graphs

Muhieddine Shebaro and Jelena Tešić

Oct 30 2023

Abstract

Structural balance modeling for signed graph networks presents how to model the sources of conflicts. The state-of-the-art has focused on computing the frustration index of a signed graph as a critical step toward solving problems in social and sensor networks and for scientific modeling. However, the proposed approaches do not scale to modern large sparse signed networks. Also, they do not address that there is more than one way in some networks to reach a consensus with the minimum number of edge-sign switches needed. We propose an efficient balanced state discovery algorithm and a network frustration computation that will discover the nearest balanced state for the *any* size of the graph network and compute the frustration of the network. The speedup of the proposed method is around 300 times faster than the state-of-the-art for signed graphs with hundreds of thousands of edges. The technique successfully scales to find the balanced states and frustration of the networks with millions of nodes and edges in real time where state-of-the-art fails.

frustration index, balanced states, signed graphs, scaling, in-memory

1 Introduction

Unstructured data requires a rich graph representation; the signed networks model complex relationships through the interdependence between entities complementary to instance features. When opposing edges are included in a network, we can study social dynamics and stability concerning friendship and enmity in more depth [5, 30], or expand to new application domains, such as brain behavior [39]. However, signed graph benchmarks are currently too small and too similar in topology to actual data, hindering progress in signed graph analysis [29, 31]. Singed graph extensions have been demonstrated for narrow-band tasks in finance [7], polypharmacy [33], bioinformatics [32] and sensor data analysis [12, 34]. To date, signed graph benchmark evaluations of the algorithms are small in size, do not have the same topology as signed graphs derived from accurate data, and the algorithms make assumptions that are not applicable in real signed networks [14, 43].

The work focuses on scaling the computing of the frustration index and finding the associated balanced state for large signed networks. Balance theory represents a theory of

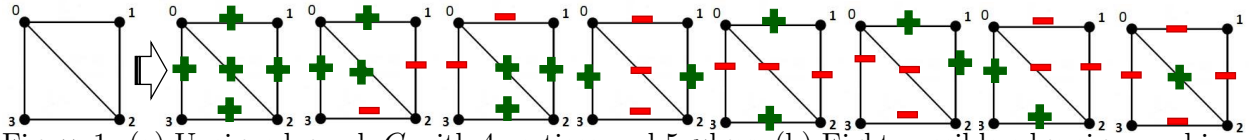


Figure 1: (a) Unsigned graph G with 4 vertices and 5 edges; (b) Eight possible edge sign combinations for G that are balanced.

changes in attitudes [1]: people's attitudes evolve in networks so that friends of a friend will likely become friends, and so will enemies of an enemy [1]. Heider established the foundation for social balance theory [24], and Harary established the mathematical foundation for signed graphs and introduced the k-way balance[11, 22]. Balance theory concepts have been used to predict edge sentiment, to recommend content and products, or to identify unusual trends [3, 15, 19, 28]. A network is considered balanced only if every fundamental cycle contains an even number of opposing edges. One measure of the network is its distance from its balanced state. A signed network's *frustration index* is the smallest number of edge-sign switches needed to reach a balanced state. The frustration index is one measure of network property in many scientific disciplines, that is, in chemistry [42], biology [27], brain studies [37], physical chemistry [46] and control [18]. The calculation of the frustration index can also be reduced to the maximum cut of the graph in a particular case of all opposing edges, which proved to be NP-hard [26]. State-of-the-art methods address the computation of the frustration index for signed graphs with up to 100,000 vertices [9], and the approach does not scale to modern large-scale sparse networks. The signed networks can have multiple nearest balanced states, e.g., multiple paths to a balanced state with the corresponding minimal or near-minimal number of edge switches to reach more than one closest balanced state [36]. The contributions are:

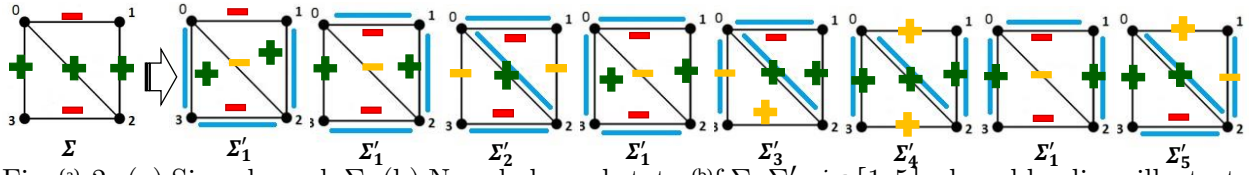
(1) Scaling discovery of the *frustration index* for any real-world signed network of any size or density based on efficiently finding a fundamental cycle basis.

(2) Extending the frustration cloud from a set in [36] to a *(key,value)* tuple collection $\mathcal{F}_\Sigma = \mathcal{B}{:}(\mathcal{C}, \mathcal{S})/$ The nearest balanced states with their associated frequency and edge switches will be stored in the memory-bound frustrated cloud as a tuple.

(3) Benchmark comparison of seven spanning tree sampling methods including a breakthrough study of frustration index computation and timing for networks with tens of millions of nodes.

This paper introduces an algorithm to discover the nearest balanced state for a signed graph of *any* size under real-time processing constraints. The outcome is the edges that need to change signs for the network to achieve a balanced state. The number of edges in the output set captures the frustration of the network. The connection between the fundamental cycle basis and the frustration index is that each unbalanced cycle will be balanced by switching the sign of the edges. The counter is increased by one for each switch. The more unbalanced cycles a signed network has, the greater the frustration index of this network. *graphB++* initially obtains a spanning tree and performs these sign switches on *non-tree* edges.

The paper is organized as follows: in Section 2, we summarize related work in the field; in Section 3, we present preliminary findings; in Section 5, we introduce a novel algorithm for computing the frustration index and its scalable version; in Section 6, we analyze different sampling approaches of the spanning tree and propose two new methods; in Section 8, we conduct experiments to evaluate the efficiency of the proposed algorithm on ten different



(a) (b)

Figure 2: (a) Signed graph Σ; (b) Near-balanced states of Σ, $\Sigma'_i : i \in [1, 5]$ where blue lines illustrate the spanning tree and yellow signs note the edge sign change in Algorithm 1. If a fundamental cycle contains an odd number of negative edges, sign switching occurs on *non-tree* edges (non-blue edges) to balance the signed network.

signed graph benchmarks and apply and measure the findings on signed graphs derived from the 26 Amazon ratings and reviews datasets [23], and in Section 8.5 we summarize our findings.

2 Related Work and Motivation

In this section, we *motivate* and discuss the applications and *significance* of computing the frustration index in various fields. There is currently no open-source code for calculating the frustration index except for the Binary Linear Programming (BLP) model baseline [7] so we use BLP as a baseline for comparison.

Frustration Applications: In chemistry, the stability of fullerenes is related to the frustration index [42]. The frustration index has been used to measure how an incoherent system responds to perturbations in large-scale signed biological networks [27]. The frustration of the network has been determining the strength of agent commitment to make a decision and win the disorder in adversarial multi-agent networks [18]. The frustration in neuroplasticity assesses the development of brain networks, as studies have shown that a person's cognitive performance and the frustration of the brain network have a negative correlation [37]. In physical chemistry, the protein-protein interaction can be predicted using the frustration index of the protein-signed network [46]. Saberi et al. investigated the pattern for the formation of frustrating connections in different brain regions during multiple life stages[38].

Computing the Frustration Index: Researchers have focused on calculating the exact frustration index, not finding the balanced state of the network. Calculating the frustration index is an NP-hard problem equivalent to calculating the ground state of the spin glass model on unstructured graphs [41]. The frustration index for small fullerene graphs can be calculated in polynomial time[16], and the finding has been extended to estimate the genetic algorithm of the frustration index in [42]. Bansal et al. introduced the correlation clustering problem, which is a problem in computing the minimum number of frustrated edges for several subsets [10]. Aref et al. provided an exact algorithm to calculate the partial balance and frustration index with $O((2^b)|E|^2)$ complexity where b is a fixed parameter, and $|E|$ is the number of edges [7]. Researchers have focused on approximating the frustration computation. Aref et. al introduced the approximate algorithm of complexity $O(\sqrt{log(|V|)})$, $|V|$ is the number of vertices that theoretically converged to the exact frustration index when tested on networks up to 2,000 edges [8]. Recent improvements in the algorithm include binary programming models and the use of multiple powerful mathematical solvers by Guribo [21], and the algorithm can handle up to $|E| = 100,000$ edges and compute the frustration index of the network in 10 hours [9]. For millions of vertices in Amazon's bi-partite ratings and reviews network, these approaches are still too complex. This is because integer and

binary programming models are known to be slow, computationally expensive, and have a huge search space for large problems. They might output an approximate solution to the problem, but that doesn't necessarily mean they are better than a heuristic approach that is much faster and scalable. For example, we can see the use of a parallel genetic algorithm for solving large integer programming models [17] because the authors postulate that as these models grow, the efficiency for solving decreases greatly where it becomes impossible to have any output.

3 Preliminary Data

Graph $G = (V, E)$ is a set of vertices V connected by the set of edges E. The number of vertices in the graph G is $|V|$, and the number of edges is $|E|$. *Path* is a sequence of distinct edges that connect a sequence of distinct vertices. *Cycle* is a path that begins and ends at the same vertex. *Connected graph* is a graph in which a path joins two vertices. *Subgraph* is a graph whose all edges and vertices are contained in a larger graph, for example, path and cycle. *Signed graph* Σ is a tuple of a graph $G = (V, E)$ and an edge signing function $\sigma : E \to \{+1, -1\}$. The edge can be positive + or negative −, $e \in [e^+, e^-]$, E^+, and E^- are sets of positive and negative edges of G. *Sign* of a subgraph is *product* of the edges signs. *Balanced signed graph* is a signed graph where every cycle is positive. *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 [11]. *Near-balanced* graph Σ' is a balanced graph that requires a minimum number of edge sign switches to produce a balanced graph from the signed graph Σ. *Frustration cloud* $\mathcal{F}_\Sigma$ is a set of all signed near-balanced graphs of Σ: $\mathcal{F}_\Sigma = \{\Sigma'_i | i \in [1, N] \wedge Fr(\Sigma'_i) = 0\}$. The unsigned graph G with four vertices and five edges is illustrated in Figure 1(a), and all possible combinations of edge signs that result in balanced signed graphs are illustrated in Figure 1(b). The frustration cloud $\mathcal{F}_\Sigma$ for a signed graph Σ is formed using the tree balance algorithm outlined in Alg. 1 and explained in detail in [36]. Only five of these balanced graphs are nearest balanced states for a specifically signed graph Σ illustrated in Fig. 2(a), as illustrated by the frustration cloud $\mathcal{F}_\Sigma = \{\Sigma'_i | i \in [1, 5]\}$ in Fig. 2(b).

Algorithm 1 Tree-Based Signed Graph Balancing

Data: Signed graph Σ
Data: Spanning tree T of Σ
Result: Balanced graph Σ'_T
1 **forall** *edges* e, $e \in \Sigma \setminus T$ **do**
2 **if** *fundamental cycle* $T \cup e$ *is negative* **then**
3 switch edge sign in Σ'_T: $e^- \to e^+; e^+ \to e^-$
4 **end**

Balance only affects the signs of the edges as it negates some of them to reach a consensus state (line 3 in Alg.1). The minimum number of edge sign switches is the frustration cloud. To discover all the states of a signed graph, Alg. 1 should be run with *all* spanning trees T of Σ. The cloud $\mathcal{F}_\Sigma$ encompasses all the individual closest balanced states that result from this algorithm. The exact number for any graph G can be calculated in polynomial time as the determinant of a matrix derived from the chart, using Kirchhoff's matrix-tree

theorem [44]. For a small social network such as the Highland Tribes [35] with 16 vertices and 29 positive and 29 opposing edges, the number of spanning trees is $402,506,278,163$ [36]. The *graphB* algorithm was introduced in [36] as a scaled adjustment of the Alg. 1 **for** loop in line 1: instead of all trees, the algorithm loops over k spanning trees discovered using the method M (Alg. 2 line 1). Next, the *graphB+* algorithm scaled the computation of fundamental cycles for the spanning tree T in Alg. 1. If T is a spanning tree of Σ and e is an edge of Σ that does not belong to T, then *fundamental Cycle* C_e, defined by e, is the cycle consisting of e together with the straightforward Path in T connecting the endpoints of e. If $|V|, v \in V$ denotes the number of edges and $|E|, e \in E$ the number of vertices in Σ, there are precise $|E| - (|V| - 1)$ fundamental cycles, one for each edge that does not belong to T. Each C_e is linearly independent of the remaining cycles because it includes an edge e not present in any other fundamental process. The *graphB+* algorithm was introduced in [2] as an efficient way to balance the signed graph given the spanning tree T. The main benefits of *graphB+* are that labels can be computed with linear time complexity, only require a linear amount of memory, and that the running time for balancing a cycle is linear in the length of the cycle times the vertex degrees but *independent* of the size of the graph. The algorithm speeds up the balancing to more than 14 million fundamental cycles identified, traversed, and balanced. This paper proposes the *graphB++* algorithm to scale the frustration cloud computation (with an extension to its definition) and approximate the frustration index for large real signed graphs.

4 Definition of the Objective

We extend *graphB+* to approximate the frustration index Fr_Σ for a given signed network using a particular tree-sampling technique (we will see later which technique is ideal for *graphB++* for minimizing this value in Section 8.3):

$Fr_\Sigma = min_i(\mathcal{S}(i))$;

The S is a container that stores the number of edge sign switches for a given ith nearest balanced state. For every state in each iteration, the number of edge sign switches is counted by comparing the originally signed network with the produced balanced state and is assigned to $\mathcal{S}(i)$). We aim to find a balanced state with a minimum number of switches and return it to the frustration index using the extended version *graphB++*. We also construct the memory-bound frustration cloud $\mathcal{F}_\Sigma$ and extend its definition to the following:

$\mathcal{F}_\Sigma = (\mathcal{B}(i), \mathcal{C}(i), \mathcal{S}(i)), i \leq \mathcal{F}_{max}$

The $\mathcal{B}(i)$ is a container for storing the ith balanced state, $\mathcal{C}(i)$ is a container for saving the number of ith balanced state produced, and $\mathcal{S}(i)$ is a container that keeps the number of edge sign switches for a given ith nearest balanced state. $\mathcal{F}_{max}$ represents the number of balanced states where a memory limit is reached during the frustration cloud creation.

5 Scalable Computation

We propose *graphB++*, an efficient algorithm that computes balanced states and frustration index. The *graphB++* algorithm builds on the *graphB* and *graphB+* baseline. First, the

algorithm integrates different tree-sampling approaches, as outlined in Alg. 2) for frustration index computation. Next, the *graphB++* algorithm scales the calculation of the frustration index and associated optimal balanced state by iteratively keeping in memory only the subset of nearest balanced states with the smallest number of edge negations, as outlined in Alg. 3. The result of *graphB++* is that the frustration index and the closest balanced associated state can be computed in minutes for the *any* large signed graph.

Algorithm 2 Graph Balancing and Frustration Index

Data: Signed graph Σ and spanning trees sampling method M
Result: frustration index Fr(Σ)=$min_i(\mathcal{S})$ and frustration cloud $\mathcal{F}_\Sigma = \mathcal{B}{:}(\mathcal{C},\mathcal{S})$.
5 Generate set $\mathcal{T}_{M^k}$ of k spanning trees of Σ using M Empty $\mathcal{F}_\Sigma$ **foreach** *spanning trees* T, $T \in \mathcal{T}_k$ **do**
6 Find nearest balanced state Σ'_T using Alg. 1 s = edge signs difference count from Σ to Σ'_T Transform Σ'_T balanced state to string B **if** $B \notin \mathcal{B}$ **then**
7 add key B to $\mathcal{B}$ S(B) = s C(B) = 1
8 **else**
9 C(B)++
10 **end**

We extend the definition of frustration cloud $\mathcal{F}_\Sigma$ (Sect. 3) from a set to a *(key,value)* tuple collection $\mathcal{F}_\Sigma = \mathcal{B}{:}(\mathcal{C},\mathcal{S})$. The key is the unique balanced state $\mathcal{B}(i)$, and the value is the count of balanced states occurring in iteration $\mathcal{C}(i)$, and the edge count switches to the balanced state $\mathcal{S}(i)$. In each balancing iteration, we examine the resulting balance state (Alg. 2). Σ'_T in relation to $\mathcal{B}$. To make the process more efficient, we represent the balanced state Σ'_T as a string B. The balanced state Σ'_T is represented as the 3 edges vectors (src,tgt,sign). If an edge i is defined by two vertices (u, v) and a sign s, the algorithm balances the graph and stores the edges as src(i)=u, tgt(i)=v, sign(i)=s.

For *graphB++* implementation, we propose an efficient transform (O(—E—)) of the balanced state output Σ' to the string hash key B for comparison with other balanced states (Alg. 2 line 5). First, the triple edge vector (src(i),tgt(i),sign(i)) is inserted into a set of tuple data structures to organize the edges and prepare for string conversion automatically. Then, it is transformed to a string format "src(i)-¿tgt(i): sign(i)," and then all edge strings are concatenated in order, separated by the delimiter "—" and stored as the B key in $\mathcal{B}$. If B is in $\mathcal{B}$, we increase the corresponding $C(B)$ value count, where B is the existing balanced state Σ'_T. If Σ'_T is not in $\mathcal{B}$, we add (Σ'_T,(1,number of switched edge signs) pair to the collection. If the state was previously unseen, we add the new balanced state inefficient matrix format to the hashmap as a string key as illustrated in Alg. 2. Then, we add 1 to the end of the count stack $\mathcal{C}$ and add the number of edge switches in the graph for this balanced state to the frustration cloud frequency stack $\mathcal{S}$. These two values (count stack and frequency stack) are stored as a pair, and the value of the hashmap of the balanced state string as a key is that pair. If the balanced state exists in $\mathcal{B}$, we increase the count at the same string key in $\mathcal{C}$ only (the first element of the pair is modified), as illustrated in the Algorithm 2. Now, Alg. 2 line 5 takes $O(E)$ for comparison. And the minimum number of edge switches in all balanced states defines the frustration index, and we compute the frustration index as Fr(Σ)=min($\mathcal{S}$). As the number of iterations increases, more elements will be added to $\mathcal{B}$, and the space complexity is linear. This can become an issue for graphs with millions of nodes and vertices as the frustration cloud is too big for the main memory.

Algorithm 3 Scalable Graph Balancing and Frustration Index

Data: Σ signed graph; M tree sampling method

Result: frustration index $Fr_\Sigma = min_i(\mathcal{S}(i))$ and frustration cloud $\mathcal{F}_\Sigma = (\mathcal{B}(i), \mathcal{C}(i), \mathcal{S}(i))$, $i \leq \mathcal{F}_{max}$

11 Generate set $\mathcal{T}_k$ of k spanning trees of Σ Determine $\mathcal{F}_{max}$, $|\mathcal{F}_\Sigma| < CAP$ Empty matrix $\mathcal{B}$, empty count vector $\mathcal{C}$, and empty edge switch count vector $\mathcal{S}$, $i = 0$, frInd=0. **foreach** T *spanning tree,* $T \in \mathcal{T}_k$ **do**

12 Find nearest balanced state Σ'_T using Alg. 1 frInd = # Σ'_T switch edge signs (Alg. 1, line 3) **if** $frInd < max_i(\mathcal{S}(i))$ **then**

13 **if** $\Sigma_T \notin \mathcal{B}$ **then**

14 **if** $i < \mathcal{F}_{max}$ **then**

15 i++

16 **else**

17 i takes the index of current $max_i(\mathcal{S}(i))$

18 $\mathcal{S}(i)$ = frInd $\mathcal{B}(i) = \Sigma_T$ $\mathcal{C}(i) = 1$

19 **else**

20 $\mathcal{C}_i$ ++

21 **end**

Thus, for large graphs, we adapt Alg. 2 as outlined in Alg. 3 to scale the computation. First, we compute the number of balanced states that we can keep in the memory as $\mathcal{F}_{max}$ (Alg. 3, line 2). Next, we keep only the $\mathcal{F}_{max}$ closest balanced states in $\mathcal{B}$, their count through all k iterations in $\mathcal{C}$, and the number of switched edges for each of them in $\mathcal{S}(i)$. Note that there can be different balanced states of Σ with the same number of switched edges. Finally, we compute the frustration index as a minimum of $\mathcal{S}$). The proposed approach scales well with the size of the signed graph, as we limit the number of the nearest balanced states that we keep in the memory based on their frustration index and capacity of the frustration cloud hashmap in-memory storage, as illustrated in Algorithm 3. The comparison of balanced states now has up to k iterations times $\mathcal{F}_{max}$ closest balanced states. We determine $\mathcal{F}_{max}$ so that the size of the frustration cloud in memory is smaller than CAP. In experiments, we define CAP as 75% of the total RAM size under the assumption that some vital system processes are running in the background. The algorithmic complexity remains $O(|E|log(|V|)d_a)$ where d_a is the average degree of a vertex, $|V|$ is the number of vertices, and $|E|$ is the number of edges [2].

6 Sampling Spanning Trees

In this section, we propose to utilize the randomization and hybridization of the standard tree sampling approaches to maximize the chances of discovering the optimal nearest balanced state in Alg. 2. The following techniques have been dominantly used in maze generation. **Depth first search (DFS)** algorithm [13] with time complexity O($|V| + |E|$) begins the traverse at the root node and proceeds through the nodes as far as possible until it reaches the node with all the nearby nodes visited. **Breadth first search (BFS)** algorithm [13]

with time complexity O($|V| + |E|$) is a graph traversal approach in which the algorithm first passes through all nodes on the same level before moving on to the next level.

We propose to use the randomized algorithms as follows: in each iteration, we shuffle and randomize a node's neighborhood using a uniformly distributed random seed number before applying a static algorithm. The idea is that a node establishes a link to the first unvisited node based on the randomized order of the adjacency list in the network. **Randomized Depth First Search (RDFS)** algorithm transforms DFS into a non-deterministic algorithm by eliminating the static ordering of the adjacency lists. The time complexity of the DFS is known to be O($|V| + |E|$), where $|V|$ is the number of vertices and $|E|$ is the number of edges in the signed network. The algorithm also runs in linear time O(n), where n is the number of nodes adjacent to a specific node in the network, so the total time complexity is O($|V| + |E|$). **Aldous-Broder algorithm** with complexity O($|V|$) produces a random uniform spanning tree by performing a random walk on a finite graph with any initial vertex and stops after all vertices have been visited [25]. For the popular **Kruskal's algorithm** [45] that has a time complexity O($|E|log|V|$) or O($|E|log|E|$), we intend to generate random spanning trees by assigning random weights to every edge in each iteration before running the algorithm. It is used to find the minimum spanning tree of a connected and weighted graph. Randomizing the weights of **Prim's algorithm** [20] [45] with complexity O($|V|^2$) can also generate random spanning trees. We propose a new algorithm, the **RDFS-BFS** sampler, to minimize the frustration index and maximize the number of unique stable states to increase algorithmic chances of finding the optimal state among all the nearest balanced states.

7 Complexity Analysis

graphB++ that is based on the original implementation of *graphB+* (BFS) has a complexity of O($|E| * log(|V| * d)$) time, where —E— is the number of edges, —V— is the number of vertices, and d is the average spanning-tree degree of the vertices on each cycle. The code for scaling the processing and saving of balanced states in the memory-bound frustration cloud as well as approximating the frustration index which builds upon graphB+ adds O(—E—). O($|E| * log(|V| * d)$) is still the dominant term. On the other hand, for *graphB++* that is adapted for other tree-sampling techniques, the reimplemented vertex relabeling step takes O($|V| + |E|$) instead because DFS is used to perform the pre-order traversal on a random spanning tree generated by a custom sampler of certain complexity. For the edge relabeling, specifically for assigning the beg and end ranges on each edge to any spanning tree, it has been reimplemented which compromised the efficiency resulting in a complexity of approximately O($|V||E| * \alpha$) where α is the average depth from a certain vertex of an edge to the deepest relabeled vertex where the assignment of the end range of the edge takes place. Both implementations have been connected to the efficient fundamental cycle balancing method [2] with complexity O($|E| * log(|V| * d)$). Likewise, the same code was employed for index computation which takes O($|E|$). Hence, the total time complexity for the adapted version of *graphB++* is O($|V||E| * \alpha$) unless the complexity of the selected custom sampler is high enough to exceed this complexity. Section 6 outlines the time complexity for each tree-sampling technique.

8 Experiments

• Section 8.2 shows which tree-sampling technique for *graphB++* is ideal for minimizing the approximation of the frustration index.
• Section 8.3 demonstrates the effect of increasing the number of iterations on the convergence of the frustration index to the minimum in *graphB++* for a given signed network.
• Section 8.4 shows how the proposed *graphB++* algorithm compares to Binary Linear Programming model (BLP) [6] for computing the amount of frustration on real-world signed networks in terms of performance and time.
• Section 8.5 answers how to overcome the memory restrictions of extracting and saving balanced states with their associated frequencies and frustration in the frustration cloud for *graphB++*.

Table 1: SNAP Signed graph Largest connected component (LCC) attributes. $|V|$ is the number of vertices and $|E|$ is the number of edges in the largest connected component LCC; The label *% positive* is the number of positive edges divided by e;

SNAP	**vertices**	**edges**		
[31]	$\|V\|$	$\|E\|$	cycles	% positive
test10 [2]	10	13	4	53.85
highland [35]	16	58	43	50
sampson18 [40]	18	112	95	54.4
rainFall [14]	306	93,636	93,331	68.78
S&P1500 [14]	1,193	711,028	709,836	75.13
wikiElec [31]	7,539	112,058	104,520	73.33
wikiRfa [31]	7,634	175,787	168,154	77.91
epinions [31]	119,130	704,267	585138	83.23
slashdot [31]	82,140	500,481	418342	77.03

Table 2: Konect Largest Connected Component (LCC) graph attributes [29]. $|E|$ is the number of vertices, and $|V|$ is the number of edges in the LCC of the graph. The *% positive* label marks the percentage of positive edges in the LCC.

	Largest Connected Component Graph			
Konect	vertices	edges	cycles	%
[29]	$\|V\|$	$\|E\|$	$\|E\| - \|V\| + 1$	positive
Sampson	18	126	145	51.32
ProLeague	16	120	105	49.79
DutchCollege	32	422	391	31.51
Congress	219	521	303	80.44
BitcoinAlpha	3,775	14,120	10,346	93.64
BitcoinOTC	5,875	21,489	15,615	89.98
Chess	7,115	55,779	48,665	32.53
TwitterReferendum	10,864	251,396	240,533	94.91
SlashdotZoo	79,116	467,731	388,616	76.092
Epinions	119,130	704,267	585,138	85.29
WikiElec	7,066	100,667	93,602	78.77
WikiConflict	113,123	2,025,910	1,912,788	43.31
WikiPolitics	137,740	715,334	577,595	87.88

8.1 Setup, Implementation, and Data

Signed Graph Benchmarks used in the experiments are **SNAP** [31], **Konect** [29], and **Amazon** ratings [23]. Both **SNAP** and **Konect** signed graph characteristics are described in Table 1 and 2 respectively. Concerning **Amazon**, ratings and reviews data [23] provides rating information between 0 (low) and 5 (high) of the Amazon users on different products. We have transformed the graphs to 18 signed bipartite graphs, where ratings 5 and 4 imply a positive edge, rating 3 and 2 give no edge, and ratings 0 and 1 give a negative edge. The graph characteristics are described in Table 6 and result in tens of millions of vertices and edges in a signed graph.

Setup The operating system used for all experimental evaluations 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.

Implementation The baseline implementation is based on the published Binary Linear Programming (BLP) code [6]. The binary linear model runs on a Jupyter notebook in Python [6] and is based on a Guribo mathematical solver and covers several parameters [21]. The dependencies of the binary terms in the objective function of the AND and XOR models are considered using AND constraints and two standard XOR constraints per edge, respectively. Two replacements in the ABS model's objective function linearized two absolute value terms [9]. The code [6] was run with the following modifications: (1) the method parameter is -1, automatic; (2) the lazy parameter is 1 with enabled speedup; (3) *multiprocessing.cpu_count()* is added as the thread parameter, and (4) the time limit for the model run is set to up to 40 hours. The code provided [6] generates random graphs based on the specified number of nodes, edges, and probability of negative edges. Our improvements to the code allow for the code to (1) accept the same input format as *graphB++*; and to (2) detect and eliminate duplicates, inconsistencies, self-loops, and invalid signs in the input graph. The **graphB++** implementation extends the open-source implementation [2] to include and test proposed tree sampling strategies while keeping the original speed-up optimization for finding fundamental cycles intact. The implementation of tree sampling strategies and the adaptation of **graphB++** to work with them are highly optimized in C++. This is achieved by employing the least number of loops possible, incorporating OpenMP directives for parallelization, and freeing dynamically allocated objects when unused. The *graphB++* also implements frustration cloud computation based on the memory size in Alg. 3. This is done by retrieving the total RAM size from the Linux system during runtime and the current resident set size in which the current execution is consuming. Then, these two values are compared in each iteration to determine the limit of the number of balanced states to save. The code is released on the anonymous gitHub [4], and the data is publicly available [23, 29, 31].

8.2 Selecting the Spanning Tree Method

In this experiment, we compare the timing and frustration computation of *graphB++* implementation of Alg. 2 of **SEVEN** different tree sampling methods described in Section 6

and look for the most effective and efficient sampling method for the frustration index computation. The seven methods are: Prim, Kruskal, Breadth First Search (BFS), Depth First Search (DFS), Randomized DFS (RDFS), Hybrid RDFS-BFS and Aldous-Broder sampling. Note that *graphB++* runs are non-deterministic, and we run the methods multiple times. The frustration computed and completion time is always the same for smaller graphs and within 0.1% for larger graphs. We compare the findings to the BLP baseline implementation for SNAP datasets. Table 1 summarizes the resulting frustration index per method. The results are summarized in Table 3 in terms of the actual frustration index value and in Figure 4 (top) in terms of the computed frustration index as a percentage of the total number of edges in the graph. BFS-spanning trees produce balanced states of minimum edge switches, and DFS-spanning trees make trees with maximum edge switches, as evident from the frustration computed in Figure 4 (top). RDFS/Kruskal/Aldous-Broder frustration scores are slightly better due to the randomization step. BFS discovers the optimal trees for the frustration computation, but they are repetitive.

The timing is reported on the log 10 scale in seconds in Figure 4 (bottom) as BLP takes 40 hours for larger datasets (far right navy bar in Fig. 4 (bottom)). RDFS-BFS hybrid approach is competitive with BFS in terms of frustration index (green and blue bars in Figure 4 (top)) with the small timing overhead for large graphs (Fig. 4 (bottom)): BFS produces 117,587 frustrations while BFS-RDFS produces 115,932 frustrations in 1000 iterations for the slashdot dataset. The Prim approach is too slow for large datasets, and the baseline BLP takes too long, or it does not complete. We also tabulated the timing per iteration for each tree-sampling technique in Table 4. Since the time complexity of Prim is $O(V^2)$, the number of iterations is set to 1, and it was very inefficient and slow for large graphs such as WikiConflict. Moreover, Aldous-Broder did not terminate for DutchCollege because, in uncommon scenarios, this sampler would get stuck looping when performing a random walk. After all, the current vertex's neighbors are already visited.

Table 3: SNAP frustration for 1000 iterations of graphB++ with SEVEN different tree samplers introduced in Section 6, and the baseline. AB stands for Aldous-Broder. BFS and Hybrid consistently perform the best.

	rainFall	**S&P 1500**	**wikiElec**	**wikiRfa**	**epinions**	**slashdot**
BFS	**10,217**	**134,515**	24,827	43,971	**100,450**	117,587
RDFS	20,047	326,957	51,197	78,215	276,584	205,236
DFS	22,879	351,135	50,617	78,151	275,211	205,089
Hybrid	**10,217**	134,863	**23,970**	**42,573**	118,588	**115,932**
Kruskal	18,576	318,733	38,255	71,990	200,264	188,495
AB	19,403	323,657	47,252	74,438	246941	198785
Prim	21,312	355,819	51,732	79,482	N/A	N/A
BLP	10,150	176,965	29,257	26,778	N/A	77,283

8.3 Iteration Timing

Here, we evaluate the efficiency of the proposed algorithm by comparing *graphB++* frustration and timing if the number of iterations increases. Figure 3 shows the change in performance for the two best tree sampling methods when the number of iterations grows. As discussed in Section 6, more iterations will not impact BFS sampling in smaller graphs.

Table 4: Average Frustration Index computation time per iteration using spanning tree sampling methods for 1000 iterations for Konect data in Table 2. **BFS** sampling method is the fastest. The algorithm is highly parallelizable.

Konect [29]	**Computation Time for Spanning Tree Method**					
Sampling	**BFS**	**RDFS**	**DFS**	**Hybrid**	**Kruskal**	**AB**
Sampson	**0.0003s**	0.00084s	0.00053s	0.00106s	0.00055s	0.00057s
ProLeague	**0.00027s**	0.0008s	0.0035s	0.00088s	0.0005s	0.0005s
DutchCollege	**0.0008s**	0.00216s	0.00179s	0.00288s	0.00158s	N/A
Congress	**0.00102s**	0.00505s	0.00301s	0.00629s	0.00317s	0.00340s
BitcoinAlpha	**0.024s**	0.126s	0.103s	0.143s	0.098s	0.102s
BitcoinOTC	**0.041s**	0.265s	0.229s	0.268s	0.214s	0.231s
Chess	0.085s	0.388s	0.283s	0.375s	0.250s	0.282s
TwitterRef.	0.457s	2.826s	2.277s	2.566s	2.111s	2.155s
SlashdotZoo	0.838s	19.131s	13.803s	15.102s	9.849s	11.880s
Epinions	1.368s	20.794s	12.795s	16.715s	11.902s	13.373s
WikiElec	0.16859s	0.63977s	0.487s	0.559s	0.462s	0.502s
WikiConflict	6.503s	94.102s	65.282s	77.045s	40.720s	41.293s
WikiPolitics	1.582s	21.585s	17.374s	19.613s	15.925s	17.010s

The frustration shows a slight improvement for the larger graphs for both methods when the number of iterations increases in Figure 3. Note that the *graphB++* running time linearly increases with the number of iterations. The frustration improvements are insignificant on the sampled signed graph to justify the increased iteration count.

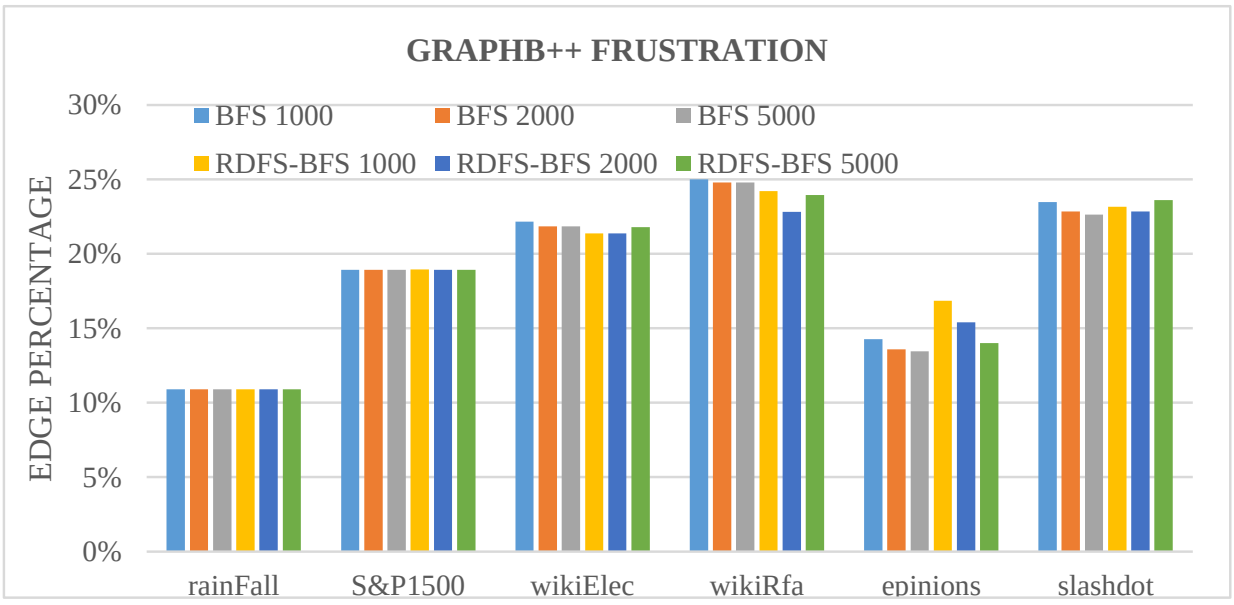


Figure 3: *graphB++* frustration for six benchmark datasets, two spanning three sampling approaches (BFS and RDFS-BFS), and three different iteration counts.

8.4 graphB++ vs. SOTA

In this experiment, we compare the baseline BLP [9] with graphB++ implementation with breadth-first search (BFS) spanning tree sampling in 1000 iterations in terms of the frustra-

Table 5: SNAP Signed graph baseline performance as a function of the number of fundamental cycles $|E|-|V|+1$ for the baseline Binary Linear Programming (BLP) [9] and proposed *graphB++* algorithm with BFS tree sampling for frustration index computation. The BLP baseline was never completed for epinions and never converged beyond Guribo heuristic for S&P1500 and wikiElec and wikiRfa datasets (*).

SNAP	Cycles	**BLP**		**graphB++**	
[31]	$\|E\|-\|V\|+1$	index	time	index	time
test10 [2]	4	2	0.06s	2	0.08s
highland [35]	43	7	0.035s	7	0.13s
sampson18 [40]	95	43	0.06s	39	0.27s
rainFall [14]	93,331	10,150	7.5hrs	10,271	83.4s
S&P1500 [14]	709,836	176,965*	14.15*hrs	134,515	1478s
wikiElec [31]	104,520	29,257*	40hrs	24,827	184s
wikiRfa [31]	168,154	26,778*	40 hrs	43,971	281s
epinions [31]	585,138	N/A	N/A	100,450	1360s
slashdot [31]	418,342	77,283*	40 hrs	117,587	937s

tion index and the time it takes to compute the frustration index for 10 benchmark graphs in Table 2. The space complexity of BLP is $O(|V|^2)$, where $|V|$ is the number of vertices on the graph. The researchers also stated that real signed graphs with up to $100,000$ edges could be solved in 10 hours [9]. Since the frustration cloud computation can fit in the memory, we ran the *graphB++* implementation of Algorithm 2 with the breadth-first search (BFS) spanning tree. All external processes are closed to prevent interference with time measurements. The measurement for both methods include the time it takes to input the file, process it, and output the results.

The results are outlined in the last four columns of Table 2. BLP and graphB++ computation for small graphs was fast and close. Both methods retrieve correct frustration indices for the three datasets. For the sampson18 dataset, the BLP heuristic gets stuck in the local minima and results in 43 instead of 39 for the frustration index. During the optimization process, the BLP model's bound value increases from 9 to 40.55088 while the incumbent value is at 43. After, the best bound also increases to reach an integer solution of 43 resulting in a gap 0%, whereas *graphB++* approximates the frustration index to be 39. The baseline code fails for two graphs with over 700,000 edges, as described in Table 2. The BLP code fails for the sparse epinions (over 700,000 vertices) and produces no results, where *graphB++* finds 1000 nearest balanced states of the graph, the most optimal one with frustration 100,450 in under 23 minutes. BLP code on the fully connected S&P1500 signed graph produces a heuristic frustration estimate of 176,965 after 14.15 hours of processing and fails during the model-solving phase. On the other hand, *graphB++* finds 1000 near-balanced states of the network and associated frustration 134,515 in 24.6 minutes (Table 2). The most extensive signed graph in which we were able to run the linear binary solver was the fully connected rainFall network [14] due to $|E| < 100,000$ [9] algorithm limitation. The linear binary solver finished with a frustration index of 10,150 and a time of 26,054.25s (7.5 hours) while the *graphB++*'s computed frustration index as 10,217 in 83.58s (0.02 hours), more than 300 times faster.

Figure 4: Frustration index computed (with timing comparison) using Binary Linear Programming (BLP) [9] and graphB++ 1000 iterations for different tree sampling methods over different real large signed graphs except for Prim (1 iteration). BLP never finished computing the frustration index for epinions and sp1500 within the 40 hours allocated.

The linear binary solver fails to complete the computations for wikiElec within 40 hours: a proximity value of the frustration index was estimated to be between 21,299 and 29,257,

where *graphB++* calculated the frustration 24,827 and provides an associated balanced state in 185s (3min+). Note that the BLP code produces a heuristics frustration estimate in 40 hours for wikiRfa and slashdot (it fails for epinion) and a gap with *no associated balanced state* as a guide on balancing the graph. For wikiRfa and slashdot, the heuristic upper bound computed in 40 hours is much lower than discovered balanced states. On the other hand, Table 2 outlines that *graphB++* finds 1000 unique near-balanced states for wikiRfa and slashdot in 5min and 15min, respectively, and offers frustration as a measure of the nearest balanced state it discovered in the process.

In summary, the *graphB++* outputs the corresponding balanced state in the same time frame, while the BLP code does not. The *graphB++* computes the nearest balanced states in minutes for large graphs compared to hours for BLP if the computation does not fail. For eight out of ten graphs tested, *graphB++* frustration is exact (4), close to actual (rainFall), or better than the heuristic (wikiElec, S&P1500, and epinions). Next, we investigate how the tree sampling methods influence the discovered state frustration. We have shown that the proposed *graphB++* balanced state discovery is equal to or superior to state of the art for small networks and efficient for more extensive networks, and scales for large networks both in terms of processing time and producing outcomes where BLP either fails or makes heuristics without the associated balanced state.

8.5 Scaling Balanced State Discovery

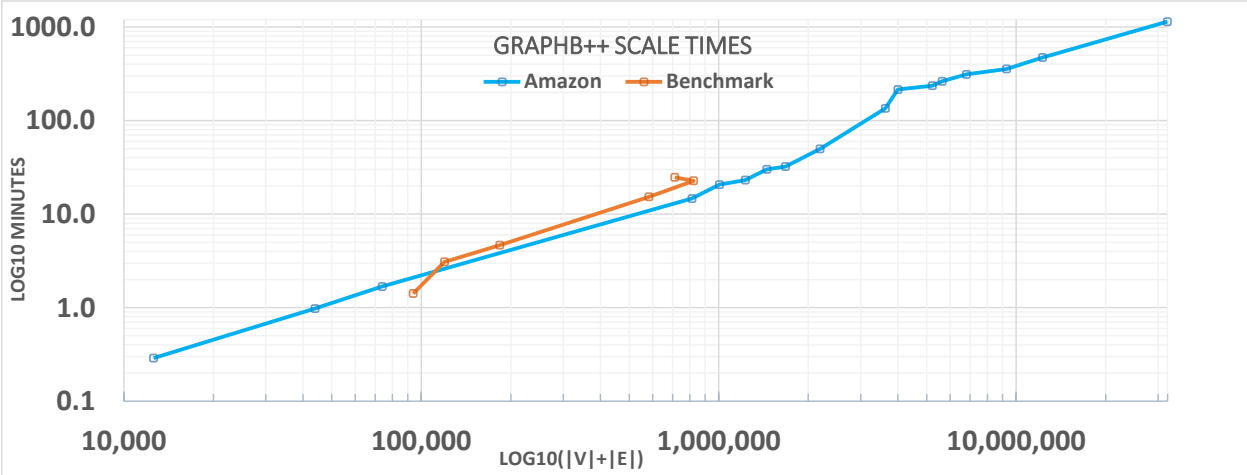


Figure 5: *graphB++* time complexity is linear with graph size —V—+—E— for a fixed number of iterations (1000) for the benchmark (Tab. 2) and Amazon (Tab 6) signed graphs.

In this experiment, we implement Alg. 3 and set the CAP to 75% of the total RAM size. We apply it to Amazon data in Table 6. The BLP model only worked and converged for the smallest 3 Amazon signed networks, the Core5 reviews in Tab. 6. All Amazon rating signed graphs have several vertices —V— higher than 300,000, and BLP outputs a memory error before initializing the model. The algorithm attempts to construct an adjacency matrix that does not fit into memory for any graph with more than 100,000 vertices. For smaller signed graphs BLP algorithm converges within 40 hours and finds the optimal frustration index. *graphB++* recovers the balanced state and associated frustration index for small graphs and in minutes for under 2 million edges; see the last column of Table 6. The serialized process

Table 6: Amazon ratings and reviews graph characteristics [23]: The number of vertices, edges, and cycles reflect the number in the largest connected component of each dataset.

Amazon	—V—	—E—	**BLP**		**graphB++ scale**	
Ratings			index	time	index	time
Books	9,973,735	22,268,630	N/A	N/A	3,146,316	19hrs
Electronics	4,523,296	7,734,582	N/A	N/A	1,025,401	7.8 hrs
Jewelry	3,796,967	5,484,633	N/A	N/A	613,129	6hrs
TV	2,236,744	4,573,784	N/A	N/A	636,568	5.2hrs
Vinyl	1,959,693	3,684,143	N/A	N/A	412,859	4.4hrs
Outdoors	2,147,848	3,075,419	N/A	N/A	264,497	4hrs
Android App	1,373,018	2,631,009	N/A	N/A	386,947	3.6hrs
Games	1,489,764	2,142,593	N/A	N/A	173,063	2.2hrs
Automotive	950,831	1,239,450	N/A	N/A	85,859	50min
Garden	735,815	939,679	N/A	N/A	70,690	32.2min
Baby	559,040	892,231	N/A	N/A	106,092	30.1min
Digital Music	525,522	702,584	N/A	N/A	34,019	23min
Instant Video	433,702	572,834	N/A	N/A	32,001	20.7min
Musical Inst.	355,507	457,140	N/A	N/A	24,959	14.7min
Amazon	—V—	—E—	**BLP**		**graphB++**	
Reviews			index	time	index	time
Digital Music	9,109	64,706	10,311	38.5 hrs	19,926	101s
Instant Video	6,815	37,126	6,001	40hrs	10,833	101s
Musical Instr	2,329	10,261	1,162	136.4s	2,311	17.3s

takes about an extra hour for each 1 million edges, and the processing, for a fixed CAP, is linear in time with the number of edges and vertices, see Figure 5. The most extensive graph we have processed is Amazon books with close to 10 million vertices and over 22 million edges, and it took 19 hours to find the nearest balanced state with frustration 3,146,316. The *graphB++* processing times of the 17 Amazon graphs (Tab. 6) exhibit the same linear trend with the size of the graph. In conclusion, we have demonstrated that graphB++ can scale and find the nearest balanced state for *any* size of a signed graph.

9 Summary and Future Work

There is more than one way to achieve balance in the network. The frustration index characterizes the optimal nearest balanced state where the minimum edge switches are required to achieve balance in the network. It has been shown that the tree-spanning approach to graph balancing produces the nearest balanced states, e.g., there can be no other balanced state nearest balanced state derived from [36]. In this paper, we extend this finding and propose a novel algorithm for discovering the nearest balanced states for any graph size in a fraction of the time. Our approach converges to the global optimum for the small graphs that the state-of-the-art binary linear programming (BLP) model computes. BLP does not work for graphs larger than 100,000 vertices while *graphB++* seamlessly scales with the graph size to discover one or more nearest balanced states for the network. The state might not be optimal for a minimal number of edge switches, but it is close to optimal, and the algorithm produces a list of edges to switch to achieve the balanced state. We have shown that the iterations of the underlying algorithm can be parallelized [2], and we plan to do this in the future. In this work, we report the result on one computer for 1000, 2000 or 5000 iterations. The timing of one iteration will help us scale the process even further as we will spawn the jobs in parallel for large signed graphs.

References

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

[2] Ghadeer Alabandi, Jelena Tešić, Lucas Rusnak, and Martin Burtscher. Discovering and balancing fundamental cycles in large signed graphs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[3] Victor Amelkin and Ambuj K Singh. Fighting opinion control in social networks via link recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 677–685, 2019.

[4] Anonimoys. Codebase for scaling frustration index and corresponding balanced state discovery for real signed graphs paper. `https://anonymous.4open.science/r/graphBplusplus-547A/README.md`, 2023.

[5] T. Antal, P. L. Krapivsky, and S. Redner. Social balance on networks: The dynamics of friendship and enmity. *Physica D: Nonlinear Phenomena*, 224(1):130–136, 2006.

[6] Samin Aref. frustration-index-xor. `https://github.com/saref/frustration-index-XOR`, 2021.

[7] Samin Aref, Andrew J. Mason, and Mark C. Wilson. An exact method for computing the frustration index in signed networks using binary programming. *CoRR*, abs/1611.09030, 2016.

[8] Samin Aref, Andrew J. Mason, and Mark C. Wilson. A modeling and computational study of the frustration index in signed networks. *Networks*, 75(1):95–110, 2020.

[9] Samin Aref and Zachary P Neal. Identifying hidden coalitions in the us house of representatives by optimally partitioning signed networks based on generalized balance. *Scientific reports*, 11(1):1–9, 2021.

[10] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89 – 113, 2004.

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

[12] Sergio Casas, Cole Gulino, Renjie Liao, and Raquel Urtasun. Spagnn: Spatially-aware graph neural networks for relational behavior forecasting from sensor data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9491–9497. IEEE, 2020.

[13] Thomas H. Cormen. *Introduction to algorithms.* MIT Press, 2009.

[14] Mihai Cucuringu, Apoorv Vikram Singh, Déborah Sulem, and Hemant Tyagi. Regularized spectral methods for clustering signed networks. *Journal of Machine Learning Research*, 22(264):1–79, 2021.

[15] Tyler Derr, Zhiwei Wang, Jamell Dacon, and Jiliang Tang. Link and interaction polarity predictions in signed networks. *Social Network Analysis and Mining*, 10(1):1–14, 2020.

[16] Tomislav Došlić and Damir Vukičević. Computing the bipartite edge frustration of fullerene graphs. *Discrete Applied Mathematics*, 155(10):1294–1301, 2007.

[17] Mohammad K Fallah, Mina Mirhosseini, Mahmood Fazlali, and Masoud Daneshtalab. Scalable parallel genetic algorithm for solving large integer linear programming models derived from behavioral synthesis. In *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 390–394, 2020.

[18] Angela Fontan and Claudio Altafini. Achieving a decision in antagonistic multi agent networks: frustration determines commitment strength. *2018 IEEE Conference on Decision and Control (CDC), Decision and Control (CDC), 2018 IEEE Conference on*, pages 109 – 114, 2018.

[19] Kiran Garimella, Tim Smith, Rebecca Weiss, and Robert West. Political polarization in online news consumption. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 15, pages 152–162, 2021.

[20] Saul I. Gass and Michael C. Fu, editors. *Prim's Algorithm*, pages 1160–1160. Springer US, Boston, MA, 2013.

[21] GUROBI. Gurobi optimization. `https://www.gurobi.com/`.

[22] Frank Harary and Dorwin Cartwright. On the coloring of signed graphs. *Elemente der Mathematik*, 23:85–89, 1968.

[23] Ruining He and Julian 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 '16, pages 507–517. ACM, 2016.

[24] Fritz Heider. Attitudes and cognitive organization. *J. Psychology*, 21:107–112, 1946.

[25] Yiping Hu, Russell Lyons, and Pengfei Tang. A reverse aldous-broder algorithm. *ANNALES DE L INSTITUT HENRI POINCARE-PROBABILITES ET STATISTIQUES*, 57(2):890 – 900, 2021.

[26] Falk Hüffner, Nadja Betzler, and Rolf Niedermeier. Separator-based data reduction for signed graph balancing. *Journal of combinatorial optimization*, 20(4):335–360, 2010.

[27] Giovanni Iacono and Claudio Altafini. Average frustration and phase transition in large-scale biological networks: a statistical physics approach. *IFAC Proceedings Volumes*, 43(14):320–325, 2010. 8th IFAC Symposium on Nonlinear Control Systems.

[28] Ruben Interian, Ruslan G Marzo, Isela Mendoza, and Celso C Ribeiro. Network polarization, filter bubbles, and echo chambers: An annotated review of measures, models, and case studies. *arXiv preprint arXiv:2207.13799*, 2022.

[29] Jérôme Kunegis. KONECT – The Koblenz Network Collection. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 1343–1350. ACM, 2013.

[30] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 641–650. ACM, 2010.

[31] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[32] Rui Li, Xin Yuan, Mohsen Radfar, Peter Marendy, Wei Ni, Terence J. O'Brien, and Pablo M. Casillas-Espinosa. Graph signal processing, graph neural network and graph learning on biological data: A systematic review. *IEEE Reviews in Biomedical Engineering*, pages 1–1, 2021.

[33] Taoran Liu, Jiancong Cui, Hui Zhuang, and Hong Wang. Modeling polypharmacy effects with heterogeneous signed graph convolutional networks. *Applied Intelligence*, 51:8316–8333, 2021.

[34] Tianya Liu, Andong Sheng, Guoqing Qi, and Yinya Li. Admissible bipartite consensus in networks of singular agents over signed graphs. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(8):2880–2884, 2021.

[35] Kenneth Read. Cultures of the central highlands, new guinea. *Southwestern Journal of Anthropology*, 10(1):1–43, 1954.

[36] Lucas Rusnak and Jelena Tešić. Characterizing attitudinal network graphs through frustration cloud. *Data Mining and Knowledge Discovery*, 6, November 2021.

[37] Majid Saberi, Reza Khosrowabadi, Ali Khatibi, Bratislav Misic, and Gholamreza Jafari. Requirement to change of functional brain network across the lifespan. *PLoS ONE*, 16(11):1 – 19, 2021.

[38] Majid Saberi, Reza Khosrowabadi, Ali Khatibi, Bratislav Misic, and Gholamreza Jafari. Pattern of frustration formation in the functional brain network. *NETWORK NEUROSCIENCE*, 6(4):1334 – 1356, 2022.

[39] Majid Saberi, Reza Khosrowabadi, Ali Khatibi, Bratislav Mišić, and Gholamreza Jafari. Topological impact of negative links on the stability of resting-state brain network. *Scientific Reports*, 11(1):2176, 2021.

[40] S. Sampson. A novitiate in a period of change: An experimental and case study of relationships. Ph.D. thesis, Cornell University, 1968.

[41] Michael T Schaub, Neave O'Clery, Yazan N Billeh, Jean-Charles Delvenne, Renaud Lambiotte, and Mauricio Barahona. Graph partitions and cluster synchronization in networks of oscillators. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 26(9):094821, 2016.

[42] Z Seif and MB Ahmadi. Computing frustration index using genetic algorithm. *Communications in Mathematical and in Computer Chemistry*, 71:437–443, 2014.

[43] Maria Tomasso, Lucas Rusnak, and Jelena Tešić. Advances in scaling community discovery methods for signed graph networks. *Journal of Complex Networks*, 10(3), 06 2022. cnac013.

[44] William T. Tutte. *Graph theory*, volume 21 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley Publishing Company Advanced Book Program, Reading, MA, 1984.

[45] Yuxin Wu, Deyuan Meng, and Zheng-Guang Wu. geeksforgeeks. `https://www.geeksforgeeks.org/`. Accessed: 2023-06-01.

[46] Xiaozhou Zhou, Haoyu Song, and Jingyuan Li. Residue frustration based prediction of protein-protein interactions using machine learning. *Journal of physical chemistry*, 126(8):1719–727, 2022.