Scaling Frustration Index and Balanced State Discovery for Real Signed Graphs

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

***Abstract*—Structural balance modeling for signed graph networks presents how to model the sources of conflicts. The state-of-the-art focuses on computing the frustration index of a signed graph, a critical step toward solving problems in social and sensor networks and scientific modeling. The proposed approaches do not scale to large signed networks of tens of millions of vertices and edges. In this paper, we propose two efficient algorithms, a tree-based *graphBpp* and a gradient descent-based *graphL*. We show that both algorithms outperform state-of-art in terms of efficiency and effectiveness for discovering the balanced state for *any* size of the network. We introduce the first comparison for large graphs for the exact, tree-based, and gradient descent-based methods. The speedup of the methods is around *300+ times faster* than the state-of-the-art for large signed graphs. We find that the exact method excels at optimally finding the frustration for small graphs only. *graphBpp* scales this approximation to large signed graphs at the cost of accuracy. *graphL* produces a state with a lower frustration at the cost of selecting a proper variable initialization and hyperparameter tuning.**

***Index Terms*—frustration index, balanced states, signed graphs, scaling, in-memory**

I. Introduction

UNSTRUCTURED data requires a rich graph representation. The signed networks can model complex relationships with negative and positive edges and lack of an edge. Social dynamics and stability concerning friendship and enmity in more depth [1], [2] as well as brain behavior [3] were modeled using signed network analysis. The challenge right now is the size of the signed graph benchmarks [4], [5] and the complexity of the existing methods [6]: the proof of concepts for narrow-band tasks in finance [7], polypharmacy [8], bioinformatics [9], and sensor data analysis [10], [11] are simply too small to be deployed for modern networks and datasets and make assumptions that are not applicable in real signed networks [6], [12]. A salient metric in signed graphs is the frustration index, and finding it is NP-hard [7]. The frustration index quantifies the degree to which a signed graph deviates from a state of balance. When each cycle in the signed graph contains an even number of negative edges, the signed graph is balanced. The frustration index is also the count of edge sign alterations required to ensure that no cycle contains an odd number of negative edges. This interpretation provides a practical way to understand and apply the concept of the frustration index in signed network analysis.

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

Manuscript received April 19, 2024; revised June 28, 2024.

In this paper, we focus on scaling the computation of the frustration index and the associated balanced state for large signed networks. We propose a novel and efficient tree-based method, *graphBpp*, and a loss optimization method, *graphL*. We demonstrate the proof-of-concept on large (millions of vertices and edges) signed graphs derived from the actual data.

Balance theory represents a theory of changes in attitudes [13]: people's attitudes evolve in networks so that friends of a friend will likely become friends, and so will enemies of an enemy [13]. Heider established the foundation for social balance theory [14], and Harary established the mathematical foundation for signed graphs and introduced the k-way balance [15], [16].

The balanced-theory-based algorithms helped solve the tasks of predicting edge sentiment, recommending content and products, and identifying unusual trends [17]–[20]. The frustration index is one measure of network property in many scientific disciplines, that is, in chemistry [21], biology [22], brain studies [23], physical chemistry [24] and control [25]. Finding the maximum cut of the graph in a particular case of all opposing edges is equivalent to the calculation of the frustration index [26]. The authors showed that the process is NP-hard [26]. State-of-the-art methods address the computation of the frustration index for signed graphs with up to 100,000 vertices [27], and the approach does not scale to modern large signed networks with tens of millions of vertices and edges. Signed networks can have multiple nearest-balanced states, and *graphB* algorithm [28] implements the first approach to scale Algorithm 1. Nearest Balanced states S are a subset of all possible balanced states of a signed graph in which *graphB* produces these states by a minimal number of edge sign changes using a tree-sampling method. In other words, the algorithm always produces this subset of balanced states by avoiding the tedious calculations of finding all balanced states, some of which are only present by passing through another balanced state [28]. We designate $S(i)$ to indicate the i^{th} nearest balanced state produced by *graphB* in the i^{th} iteration. In the for loop in line 1, the algorithm loops over k sampled spanning trees instead of all trees(Algorithm 2 line 1). Next, the *graphB+* algorithm scaled the computation of fundamental cycles for the spanning tree T in Algorithm 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 $|V|$ denotes the number of vertices and $|E|$ the number of edges in Σ, there are precise $|E| - (|V| - 1)$ fundamental cycles, one for each edge that

0000–0000/00$00.00 © 2024 IEEE

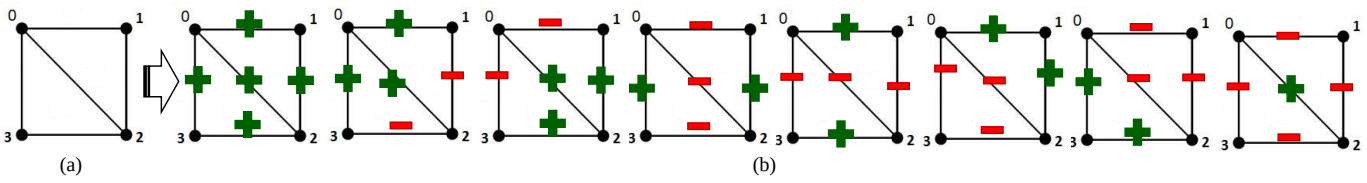

Fig. 1. (a) Unsigned graph G with 4 vertices and 5 edges; (b) Eight possible edge sign combinations for G (signed graphs Σ).

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 [29] efficiently discovers fundamental cycles for the spanning tree T and computes vertex and edge labels with linear time complexity. The algorithm requires a linear amount of memory, and 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 labels here are specific values assigned to the vertices and edges after sampling a spanning tree as a preliminary step prior to finding and traversing the fundamental cycles and balancing them. These labels aid in finding these cycles and ensure efficient traversal. The algorithm speeds up the balancing to more than 14 million identified, traversed, and balanced fundamental cycles. Next, the algorithm traverses or visits each cycle in a specific order and takes note of which edge signs switched within each cycle to obtain an even number of negative edges along that cycle. We define the memory-bound frustration cloud as a container that contains a collection of nearest balanced states for a signed graph, restricted in size based on the computer's random access memory. Figure 1 shows eight examples of balanced states for an unsigned graph with four vertices and five edges. We can observe that every cycle in each of the states contains an even number of negative edges. The contributions are:

- We extend the frustration cloud from a set in [28] to a *(key, value)* tuple collection $\mathcal{F}_{Sigma} = \mathcal{B}{:}(\mathcal{C}, \mathcal{S})$. We store the nearest balanced states with their associated frequency and edge switches as a tuple in the memory-bound frustrated cloud.
- We propose *graphBpp*, a robust improvement over previous tree balancing algorithms [30] for finding the nearest balanced frustration state and index for any real-world signed network of any size or density with time complexity O($|T_k|*|E|*log(|V|*d)$), 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, and $|T_k|$ is the number of spanning-trees generated. We analyze the algorithmic effectiveness in Section IV-A and demonstrate its scalability and efficiency over the state-of-the-art exact method in the literature in Section VII.
- We propose *graphL*, a gradient descent algorithm that produces a more optimal balanced state with a lower index than *graphBpp* in linear time. We analyze the algorithmic effectiveness in Section V-A and demonstrate its scalability and efficiency over the state-of-the-art exact method in the literature in Section VIII.
- We propose the tree-sampling method and heuristics and confirm it with the benchmark comparison of seven spanning tree-sampling methods, frustration index computation, and timing for networks with tens of millions of nodes. To the best of our knowledge, this marks the first instance where exact, tree-based, and gradient descent-based methods are evaluated and directly compared in terms of their effectiveness in estimating the frustration index.

II. Definitions and Corollaries

A. Fundamental Cycle Basis

Definition 2.1: **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.**Simple Cycle** is a route that begins and concludes at an identical vertex, and it doesn't pass through any other vertex more than one time. **Cycle Basis** is a set of simple cycles that forms a basis of the cycle space.

Definition 2.2: 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 exists between vertices x and y, which does not include m. A **Fundamental Cycle** is a cycle that combines a path in the tree T and an edge m from the graph G. The cycles, denoted as c_i, are considered fundamental if they include precisely one edge that is not part of the tree. They are a collection of cycles capable of generating all possible cycles in a graph through a linear combination of its members, which is determined based on a spanning tree. For instance, the cycles 0-1-2 and 0-3-2 in Figure 1 are fundamental cycles and they can generate a larger cycle 0-1-2-3, that is not a fundamental cycle.

Corollary 2.1: A fundamental cycle basis can be derived 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 precisely $M - N + 1$ fundamental cycles for each connected component.

B. Balanced Graphs and Frustration

Definition 2.3: **Signed graph** $\Sigma = (G, \sigma, V, E)$ 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^-. **Fully Signed Graph** is a signed graph with vertex signs (assigned +1 or -1) [31]. **Sign** of a sub-graph is the *product* of the edges signs. **Balanced Signed graph** is a signed graph where every cycle is positive. **Frustration** of a signed graph (Fr) is defined as the number of candidate edges whose sign needs to be switched for the graph to reach

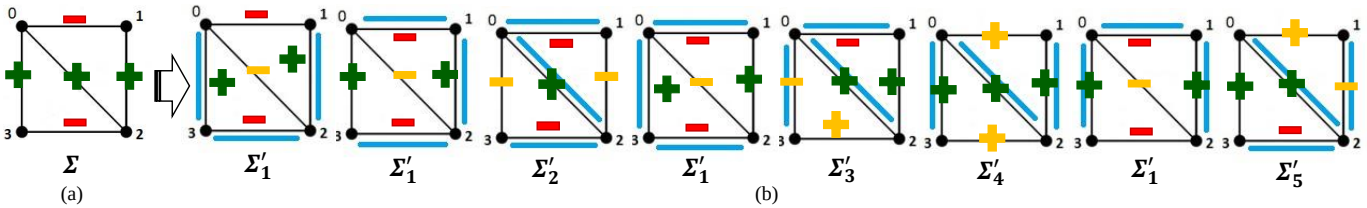

Fig. 2. (a) Signed graph Σ (b) Near-balanced states of Σ, $\Sigma'_i : i \in [1, 5]$ where blue lines illustrate the spanning tree and yellow signs note the edge sign change in Algorithm 1. If a fundamental cycle contains an odd number of negative edges, sign switching occurs on *non-tree* edges (non-blue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

a balanced state. **Frustration Cloud** contains a collection of nearest balanced states for a particular signed graph.

Definition 2.4: A balanced state is **optimal** if and only if it requires a minimum number of edge sign switches in the original graph to reach a balanced state.

Theorem 2.1 ([15]): 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 Σ', 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*.

In this paper, Section III summarizes related work and state-of-the-art in the field. We introduce a *balancing* algorithmic improvements for approximating the frustration index as the *graphBpp* algorithm or *graphB++* in Section IV. We introduce the gradient descent-based heuristic for finding the frustration index and the novel loss function in the *graphL* algorithm in Section V. We assess the effectiveness and efficiency of *graphBpp* in Section VII, and of *graphL* in Section VIII, and compare and contrast them with state-of-art using real-world signed graph benchmarks [5], [32]. In Section IX, we summarize our findings.

III. Related Work

Frustration index computation has various applications in bioinformatics, engineering, and science, and the only existing open-source code for calculating the frustration index is the Binary Linear Programming (BLP) [7].

Frustration Applications: In chemistry, the stability of fullerenes is related to the frustration index [21]. The frustration index measures how an incoherent system responds to perturbations in large-scale signed biological networks [22]. 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 [25]. In these networks, the strength is determined by measuring the social commitment of agents, particularly when there are disorder or adversarial actions. Winning the disorder is the process of overcoming the chaos caused by frustration in the network. The frustration arises from the mix of collaborative and antagonistic interactions, leading to an unbalanced signed graph. To overcome this disorder and make significant decisions, agents need a high level of social commitment. Moreover, 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 [23]. Physical chemists predict the protein-protein interaction using the frustration index of the protein signed network [24]. Saberi et al. investigated the pattern for the formation of frustrating connections in different brain regions during multiple life stages [33].

Computing the Frustration Index: Researchers have focused on calculating the exact frustration index. Calculating the frustration index is an NP-hard problem equivalent to calculating the ground state of the spin glass model on unstructured graphs [34]. The frustration index for small fullerene graphs can be calculated in polynomial time [35], and the finding was used to estimate the genetic algorithm of the frustration index in [21]. Bansal et al. introduced the correlation clustering problem, which is a problem in computing the minimum number of frustrated edges for several subsets [36]. 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]. Recent improvements in the algorithm include binary programming models and the use of multiple powerful mathematical solvers by Gurobi [37], and the algorithm can handle up to $|E| = 100,000$ edges and compute the frustration index of the network in 10 hours [27]. The 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. The use of a parallel genetic algorithm for solving large integer programming models [38] does not scale as the authors postulate that as these models grow, the efficiency decreases greatly, making it impossible to have any output, and we demonstrate this in experiments.

Gradient Descent in Signed Networks: Tang et al. [39] propose a statistically principled latent space approach for modeling signed networks and accommodating the well-known balance theory. They build a balanced inner-product model that has three different kinds of latent variables to optimize: vertex degree heterogeneity α vector of size n, z vector of size n, which encodes for the latent position, and the latent polar variable vector q of size n in which it encodes the placement of the vertices in one of the two Harary subsets. They model the distribution of signs through their product, which satisfies

the balance. An edge between two vertices will likely have a positive sign when their latent variables q_i and q_j have the same sign and a negative sign otherwise. Finally, they propose a loss function to minimize and find the optimal polar variable values using the *projected* gradient descent. They present the error rates for these estimates using simulation studies.

In our work, we introduce one balancing-based approach and one frustration-based approach. The methodology is to first scale the approximation of the frustration index by introducing a more efficient tree-based approach in Section IV and to introduce a gradient descent-based heuristic for further optimizing the frustration index computation in Section V.

IV. The graphBpp, Graph Balancing Methodology

In this section, we propose the improved graph balancing algorithm, the *graphBpp* algorithm. The *graphBpp* extends the fundamental cycle algorithm *graphB+* we have proposed earlier in [29] to *approximate* the frustration index Fr for a signed graph Σ as Fr_{Sigma} using a particular tree-sampling technique. Note that we focus on the tree sampling tree selection for *graphBpp* for minimizing the index in subsection VII-2).

$$Fr_{\Sigma} = min_i(\mathcal{S}(I)) \tag{1}$$

The objective function for approximating frustration index is outlined in Eq. 1, and S is a container that stores the number of edge sign switches for a given i^{th} nearest balanced state.

Algorithm 1 Tree-Based Signed Graph Balancing

1: Input signed graph Σ and spanning tree T of Σ
2: **for** Edges e, $e \in \Sigma \setminus T$ **do**
3: **if** fundamental cycle $T \cup e$ is negative **then**
4: Flip edge sign for edge e: $e^- \rightarrow e^+; e^+ \rightarrow e^-$
5: **end if**
6: **end for**
7: Return balanced graph Σ'_T

The *graphB+* is an efficient algorithm alternative for computing the fundamental cycles [29]. The *graphBpp* algorithm builds on the *graphB* [28] and *graphB+* [29] as it combines the efficiency of *graphB+* with the functionality of *graphB*. The *graphBpp* integrates different tree-sampling approaches, as outlined in Algorithm 2 for the frustration index computation. Next, the *graphBpp* 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 Algorithm 3. The *graphBpp* finds the approximate frustration index and the nearest balanced state associated with the index for *any* large signed graph.

We extend the definition of frustration cloud $\mathcal{F}_{\Sigma}$ 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 (Algorithm 2) Σ'_T in relation to $\mathcal{B}$. We represent the balanced state Σ'_T as a string B to make the process more efficient. The balanced state Σ'_T represents the three edge 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. The number of edge sign switches for each iteration of the *graphBpp* algorithm is counted by comparing Σ with the produced balanced state Σ'_i in Alg 1, and the value is stored in $\mathcal{S}(i))$ for the i^{th} iteration. Thus, by choosing the nearest balanced state with the lowest number of edge sign switches, we can *approximate* the frustration index with the lowest value available from the tree sampling. Next, we introduce the update to the *frustration cloud* [28] to be memory-bound, and we define the new frustration cloud $\mathcal{F}_{Sigma}$ in Eq. 2.

Algorithm 2 Tree-Based Graph Balancing and Frustration Index (Non-scalable Version for Small and Medium-Sized Graphs)

1: Input signed graph Σ and spanning trees sampling method M
2: Generate set $\mathcal{T}_{M^k}$ of k spanning trees of Σ using M
3: Empty $\mathcal{F}_{\Sigma}$ (frustration cloud of nearest balanced states)
4: **for** spanning trees T, $T \in \mathcal{T}_k$ where $\mathcal{T}_k$ is a set of k spanning trees of Σ **do**
5: Find nearest balanced state Σ'_i using Algorithm 1
6: s = edge signs difference count from Σ to Σ'_i
7: Transform Σ'_i balanced state to string B
8: **if** $B \notin \mathcal{B}$ **then**
9: Add key B to $\mathcal{B}$
10: S(B) = s
11: C(B) = 1
12: **else**
13: C(B)++
14: **end if**
15: **end for**
16: Return frustration index Fr(Σ) = $min_i(\mathcal{S})$ and frustration cloud $\mathcal{F}_{\Sigma}$ = $\mathcal{B}$:($\mathcal{C}, \mathcal{S}$)

$$\mathcal{F}_{\Sigma} = (\mathcal{B}(i), \mathcal{C}(i), \mathcal{S}(i)), i \leq \mathcal{F}_{max} \tag{2}$$

In equation 2, the $\mathcal{B}(i)$ is a container for storing the i^{th} balanced state, $\mathcal{C}(i)$ is a container for saving the number of i^{th} balanced state produced, and $\mathcal{S}(i)$ is the number of the edge switches to achieve the i^{th} balanced state from Σ. The $\mathcal{F}_{max}$ represents the number of balanced states where a memory limit is reached during the frustration cloud creation. Figure 2 all the nearest balanced states produced by *graphBpp*.

For *graphBpp* 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, as outlined in Algorithm 2 line 5. 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'_i,(1,number of switched edge signs) pair to the collection. If the state was previously unseen, we add the new balanced state to the hashmap as a string key as illustrated in Algorithm 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 Algorithm 2. The minimum number of edge switches in all balanced states approximates the frustration index, and we approximate the frustration index as Fr(Σ) = min($\mathcal{S}$). The size of the frustration cloud grows with the number of iterations as the probability of the previously unseen nearest balanced state grows. The size of the frustration cloud can become an issue for graphs with millions of vertices and vertices as the frustration cloud is too big for the main memory. The underlying data structure for the implementation in C++ is $std :: map < std :: string, std :: pair < int, int >>$. The key, as discussed above, is of type string that represents a balanced state, and the pair stores two integers, one for the number of switches and the other for the frequency of the corresponding stable state. The string keys are indeed slower than using integer key types in the map data structure in regards to comparing keys. However, in our case, storing the balanced states using strings or integers should have a similar comparison performance, and the integer key approach would be much more complicated. First, if we use $std :: map < int, std :: pair < int, int >>$, then it would be difficult and more complex to represent the signed network using one integer key, whereas the string can intuitively concatenate all the edges with their signs to represent that graph. There is no direct way of representing this collection of edges forming the signed graphs using solely the integer key. If we were to use an integer as a key for the map, we would still have to loop through the edges in that key for the comparisons, and the cost would be $O(|E|)$ equivalent to using the string key.

The *graphBpp* adapts Algorithm 2 as outlined in Algorithm 3 to scale the computation. First, we compute the number of balanced states that we can keep in the memory as $\mathcal{F}_{max}$ (Algorithm 3, line 2). Next, we keep only the $\mathcal{F}_{max}$ best-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 map 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, assuming that some vital system processes are running in the background.

Algorithm 3 Tree-Based Graph Balancing and Frustration Index (Scalable Version for Large-sized Graphs)

1: Input Σ signed graph and M tree sampling method
2: Generate set $\mathcal{T}_k$ of k spanning trees of Σ
3: Determine $\mathcal{F}_{max}$, it is the maximum number of nearest balanced states the cloud can store before it reaches the memory limit, memory consumption of $\mathcal{F}_{Sigma} < CAP$ where $\mathcal{F}_{Sigma}$ is the frustration cloud of nearest balanced states
4: Matrix $\mathcal{B}$, count vector $\mathcal{C}$, and edge switch count vector $\mathcal{S}$, $i = 0$, frInd=0
5: **for** T spanning tree, $T \in \mathcal{T}_k$ where $\mathcal{T}_k$ is a set of k spanning trees of Σ **do**
6: Find nearest balanced state Σ'_i using Algorithm 1
7: frInd = number of edge sign switches (Algorithm 1, line 3)
8: **if** $\mathcal{S}$ is empty **then**
9: $\mathcal{S}(i)$ = frInd
10: $\mathcal{B}(i)$ = Σ'_i
11: $\mathcal{C}(i) = 1$
12: **end if**
13: **if** frInd $< max_i(\mathcal{S}(i))$ **then**
14: **if** $\Sigma'_T \notin \mathcal{B}$ **then**
15: **if** $i < \mathcal{F}_{max}$ **then**
16: i++
17: **else**
18: $i \leftarrow \arg\min_i S(i)$
19: **end if**
20: $\mathcal{S}(i)$ = frInd
21: $\mathcal{B}(i)$ = Σ'_i
22: $\mathcal{C}(i) = 1$
23: **else**
24: $\mathcal{C}_i$++
25: **end if**
26: **end if**
27: **end for**
28: Return frustration index $Fr_\Sigma = min_i(\mathcal{S}(i))$ and frustration cloud $\mathcal{F}_\Sigma = (B(i), C(i), S(i))_{i=1}^{F_{max}}$

A. The graphBpp Complexity Analysis

Concerning the time complexity of *graphBpp*, it 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 [29]. GraphB+ with (BFS) implementation 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 and approximating the frustration index, which builds upon graphB+, adds O($|E|$). O($|E| * log(|V| * d)$) is still the dominant term for one iteration (generating one spanning tree and nearest balanced state). When generating $|T_k|$ spanning trees (iterations), the complexity then becomes O($|T_k| * |E| * log(|V| * d)$). On the other hand, for adapting *graphBpp* to utilize other tree-sampling techniques, the reimplemented vertex relabeling step

takes O($|V| + |E|$) because DFS is used to perform the pre-order traversal on the random spanning tree generated. The edge relabeling has been implemented, resulting in a complexity of 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. The efficient fundamental cycle balancing method [29] has a complexity of O($|E| * log(|V| * d)$). The adapted version of graphB+ for index computation has a complexity of O($|E|$). Hence, the total time complexity for the adapted version of *graphBpp* is O($|V| * |E| * \alpha$) unless the complexity of the selected custom sampler is high enough to exceed this complexity. For $|T_k|$ iterations of the algorithm, the complexity is O($|T_k| * |V| * |E| * \alpha$). Subsection IV-B outlines the time complexity for each tree-sampling technique.

Algorithm 4 Hybridized RDFS-BFS Sampling

1: Input signed graph Σ and a root vertex n get uniformly distributed random number 0 or 1, z
2: **if** z is 0 **then**
3: Run BFS algorithm [40]
4: **else**
5: Run RDFS algorithm
6: **end if**
7: Return spanning tree T of Σ

B. Sampling Spanning Trees

To maximize the chances of discovering the optimal nearest balanced state in Algorithm 2, we propose to utilize randomization and hybridization of the standard tree sampling. The **Depth-First Search (DFS)** algorithm [41] is a graph traversal method characterized by its time complexity of O($|V| + |E|$). The traversal commences at a root vertex and continues to explore as deeply as possible along each pathway (branch) before backtracking. This process repeats until a vertex is reached where all adjacent vertices have already been visited. The **Breadth first search (BFS)** algorithm [41] with time complexity O($|V|+|E|$) is a graph traversal approach in which the algorithm first passes through all vertices on the same level before moving on to the next level. A graph traversal technique is a strategy employed to visit all the vertices of a graph. A level is a group of vertices that are equidistant from the root vertex. 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 vertex establishes a link to the first unvisited vertex based on the network's randomized order of the adjacency list. **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 vertices adjacent to a specific vertex 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 [42]. For the popular **Kruskal's algorithm** [43] 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. The method finds the minimum spanning tree of a connected and weighted graph. Randomizing the weights of **Prim's algorithm** [44] [43] 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. Algorithm 4 outlines the steps of the RDFS-BFS sampler.

V. The graphL, Loss Function Methodology

We use gradient descent to approximate the frustration index and balance the signed graph in linear time. We adopt the equation from Du et al. [31] that calculates the imbalance of a fully signed network. The definition of structural balance in these networks is different. According to the theory of homophily, a fully signed network is balanced if every edge is positive and the corresponding vertices have the same sign. If there is a negative edge, the vertices should have different signs. I suppose the fully signed network is balanced based on the homophily theory. In that case, the underlying signed network (ignoring vertex signs) is also balanced because the fully signed network is a generalization of the signed network [31]. The equation for computing imbalance in the fully signed network is outlined in Eq. 3

$$L = \sum_{\forall (i,j) \in \Sigma} \frac{1 - e_{ij}\theta_i\theta_j}{2} \tag{3}$$

where e_{ij} is the sign of the edge connecting vertex i to vertex j. θ_i and θ_j are the vertex signs (1 or -1) for vertices i and j respectively. The equation is a d differentiable loss function that we will attempt to minimize by treating the θ signs of the vertices as latent variables. Initially, the θ variables are relaxed to continuous random variables in the range between -1 to 1. We denote these continuous variables as Γ, which is essentially a vector with a size equal to the number of vertices in the signed graphs. The equation for optimization is then:

$$L = \sum_{\forall (i,j) \in \Sigma} \frac{1 - e_{ij}\Gamma_i\Gamma_j}{2} \tag{4}$$

The loss function used is outlined in Eq. 4. Next, for each gradient update iteration or round, the *graphL* algorithm computes the loss using Eq. 3. Note that algorithm in lines 5-6 sets θ_i and θ_j to -1 if Γ_i and Γ_j are negative respectively and to 1 if Γ_i and Γ_j are positive respectively. The algorithm computes the gradients with respect to each latent variable Γ_i in Γ vector in Eq. 4, and the gradients are:

$$\Gamma_i : \frac{\partial L}{\partial \Gamma_i} = -\frac{1}{2} \sum \Gamma_j e_{ij} \tag{5}$$

Algorithm 5 Gradient Descent-Based Graph Balancing and Frustration Index

1: Input signed Network Σ, learning rate α, number of gradient updates λ
2: x=0
3: Initialize random float vector Γ of size equal to the number of nodes.
4: **while** $x < \lambda$ **do**
5: Compute loss function (also frustration index) using $L = \sum \frac{1-e_{ij}\theta_i\theta_j}{2}$ where θ_i and θ_j is 1 if Γ_i and Γ_j is greater than 0 respectively, otherwise -1
6: Induce relaxation and allow continuous values for θ vector by substituting it with Γ: $L = \sum \frac{1-e_{ij}\Gamma_i\Gamma_j}{2}$
7: Compute the gradient with respect to each Γ_i: $\frac{\partial L}{\partial v_i} = -\frac{1}{2}\sum \Gamma_j e_{ij}$ where Γ_j is the neighbor of Γ_i
8: Update Γ: $v \leftarrow \Gamma - \alpha\frac{\partial L}{\partial \Gamma}$
9: x=x+1
10: **end while**
11: Initialize frustration=0
12: Initialize set $visit = \phi$
13: Assign $\Sigma' = \Sigma$
14: **while** all edges have not been visited **do**
15: Fetch unvisited edge e_{ij} between vertices i and j
16: **if** $\Gamma_i >= 0$ **then**
17: θ_i=1
18: **end if**
19: **if** $\Gamma_j >= 0$ **then**
20: θ_j=1
21: **end if**
22: **if** $\Gamma_i < 0$ **then**
23: θ_i=-1
24: **end if**
25: **if** $\Gamma_j < 0$ **then**
26: θ_j=-1
27: **end if**
28: frustration+=$\frac{1-e_{ij}\theta_i\theta_j}{2}$
29: **if** $\frac{1-e_{ij}\theta_i\theta_j}{2}$=1 **then**
30: Flip the sign of e_{ij} in Σ'
31: **end if**
32: Add e_{ij} to $visit$
33: **end while**
34: Return Σ' and frustration

Γ_j is the neighbor of Γ_i, and we update the values of the elements of Γ using Γ: $\Gamma \leftarrow \Gamma - \alpha\frac{\partial L}{\partial \Gamma}$. In this way, we are *directly* minimizing the loss that represents the level of imbalance in the signed network. We repeat the whole process until we reach a predefined number of gradient updates λ. These latent variables in Γ should converge to be either above 0 or below 0. Finally, we loop over every edge in the network and discretize the values of Γ_i and Γ_j along e_{ij} to be integers 1 or -1 to be assigned back in θ vector. If Γ_i and Γ_j along edge (i, j) have values above 0, we set n_i and n_j to 1. Otherwise, we set them to -1. We use Eq. 3 to approximate the frustration and increase the frustration counter by $\frac{1-e_{ij}\theta_i\theta_j}{2}$ for each edge. In addition, for each edge, if it is causing an imbalance, then we flip its sign and finally return the balanced state. Algorithm 5 summarizes the steps of the complete algorithm.

Our approach is different than the work by Tang et al. [39] in several ways. First, we estimate the different types of latent variables, and we only use a random float vector Γ of size equal to the number of vertices to determine the optimal membership of each vertex in the Harary subsets. Second, we are not modeling any signed networks. Our focus is to flip the sign of the edges after estimating the latent variables of each vertex and approximate the frustration index. Tang et al. did not explicitly and directly modify the edge signs of the graph. Third, we propose the usage of a loss function for computing the frustration index directly and for computing the gradients. Fourth, we use the vanilla gradient descent approach instead of the projected gradient descent to minimize our loss function. We simply threshold the latent variables after optimization (ex, if an element in the latent vector is 25 after the gradient-descent step, which is above 0, we assign that element to be 1). Fifth, our loss function is adopted from Du et al. [31] in which they propose to measure imbalance for fully signed networks as presented in Eq. 3. The loss function used by Tang et al. is $L = \sum_{i<j}^{n} |A_{ij}|\frac{1+A_{ij}}{2}\eta_{ij} + |A_{ij}|\log(1 - \ Sigma(\eta_{ij})))$ where $\eta_{ij} = v_i v_j$, A is the adjacency matrix, σ is the sigmoid function, and n is the number of nodes.

A. The graphL Complexity Analysis

For every gradient update and computation, it is sufficient to loop over every edge in the signed graph in order to update the elements in the Γ vector. The λ is the number of gradient updates, and the total time complexity becomes O($\lambda * |E|$). We utilize the compressed sparse row (CSR) format to model the signed graph instead of the adjacency matrix because CSR scales better memory-wise. The construction of the adjacency matrix has a space complexity of O(V^2), which isn't computationally feasible when dealing with large signed graphs. Computing the amount of imbalance also occurs in constant time in each gradient update iteration as in Algorithm 5 in line 5, and it does not affect the total time complexity.

TABLE I
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**		
[4]	$\|V\|$	$\|E\|$	cycles	% positive
test10 [29]	10	13	4	53.85
highland [45]	16	58	43	50
sampson18 [46]	18	112	95	54.4
rainFall [12]	306	93,636	93,331	68.78
S&P1500 [12]	1,193	711,028	709,836	75.13
wikiElec [4]	7,539	112,058	104,520	73.33
wikiRfa [4]	7,634	175,787	168,154	77.91
epinions [4]	119,130	704,267	585,138	83.23
slashdot [4]	82,140	500,481	418,342	77.03

VI. SETUP, IMPLEMENTATION, AND DATA

Signed Graph Benchmarks used in the experiments are **SNAP** [4], **Konect** [5], and **Amazon** ratings [32]. Table I and Table II summarizes **SNAP** and **Konect** signed graph benchmarks.

The **Amazon** ratings and reviews data [32] provides rating information between 0 (low) and 5 (high) of the Amazon users on different products. We have transformed the graphs into 18 signed bipartite graphs. The raw Amazon data was originally in .json form and and maintained the rating, the item I.D., and the user I.D. Here, the user I.D.s and the item I.D.s are the nodes, and the edges between them are constructed based on the rating value. If the rating is 5 nd 4, it implies a positive edge; the rating is 3 and 2, it gives no edge, and if the ratings is 0 and 1, it gives a negative edge. Table VI summarizes the large signed graphs stemming from the process.

TABLE II
KONECT LARGEST CONNECTED COMPONENT (LCC) GRAPH ATTRIBUTES [5] (EXCEPT TWITTERREF, WHICH IS NOT A KONECT GRAPH). $|E|$ IS THE NUMBER OF EDGES, AND $|V|$ IS THE NUMBER OF VERTICES IN THE LCC OF THE GRAPH. THE *% posit ve* LABEL MARKS THE PERCENTAGE OF POSITIVE EDGES IN THE LCC.

	Largest Connected Component Graph			
Konect [5]	vertices $\lvert V\rvert$	edges $\lvert E\rvert$	cycles $\lvert E\rvert - \lvert V\rvert + 1$	% 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

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. The GPU is Nvidia G Force and has 8GB of 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 relies on the published Binary Linear Programming (BLP) code [47]. The binary linear m del runs on a Jupyter notebook in Python [47] and is based on a Gurobi mathematical solver and has several parameters like the termination parameter where we can set a time limit on how long the optimization process should last in Gurobi [37]. The binary terms in the objective function depend on the single AND constraints and two standard XOR constraints per edge, respectively.

Two replacements in the ABS model's objective function linearized two absolute value terms [27]. The code [47] was run with the following modifications: (1) -1 for the method parameter that indicates that the method for optimization is automatic, and the setting will typically choose the non-deterministic concurrent method in the Gurobi's documentation [37] for this linear programming problem; (2) the lazy parameter is set to 1 with enabled speedup; (3) thread parameter is set to *multiprocessing.cpu_count(*, and (4) the time limit for the model run is set to up to 30 hours.

Note that the value of the lazy attribute influences how aggressively the model is constrained. A value of 1 allows the constraint to cut off a feasible solution. The code provided [47] 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 *graphBpp* and to (2) detect and eliminate duplicates, inconsistencies, self-loops, and invalid signs in the input graph. The **graphBpp** implementation extends the open-source implementation [29] to include and test proposed tree sampling strategies while keeping the original speedup optimization for finding fundamental cycles intact. The **graphBpp** algorithm works with different tree-sampling strategies. It is achieved by using C++ and involves minimizing the number of loops used, incorporating OpenMP directives for parallel processing, and promptly freeing up memory resources when they are no longer needed. In the implementation of Algorithm 3, the code checks the total RAM size of the Linux system during runtime and the amount of memory currently being used by the frustration cloud. These two values are compared in each iteration to decide how many balanced states can be stored. The code for graphBpp is available on GitHub, and the data it uses is publicly accessible. The references for these resources are [48] for the code and [4], [5], [32] for the data. On the other hand, for the gradient-based heuristic, we set λ to 1000 and the learning rate α to 0.001.

VII. THE GRAPHBPP PROOF OF CONCEPT

We compare the proposed method to BLP baseline [7] on SNAP [4], Konect [5], and Amazon [32] open-source benchmarks. For the Amazon sign d graphs, we ran the scalable version of *graphBpp* intended for large graphs, which is Algorithm 3. For other smaller graphs, we ran Algorithm 2. Running Algorithm efalg-index for the Amazon signed graphs would not work because the 1000 balanced states for these graphs would not fit in memory and would crash the program. Thus, we cannot use both versions of the algorithm on all signed graphs. The five experiments are summarized in subsections as follows:

- Subsection VII-1 shows which tree-sampling technique for *graphBpp* is ideal for minimizing the approximation of the frustration index.
- Subsection VII-2 demonstrates the effect of increasing the number of iterations on the convergence of the frustration index to the minimum in *graphBpp* for a given signed network.
- Subsection VII-3 shows how the proposed *graphBpp* algorithm compares to Binary Linear Programming model (BLP) [47] for computing the amount of frustration on real-world signed networks in terms of performance and time.
- Subsection VII-4 answers how to overcome the memory restrictions of extracting and saving balanced states with their associated frequencies and frustration in the frustration cloud for *graphBpp*.

1) Selecting the Spanning Tree Method: We compare the timing and frustration computation of *graphBpp* implementation of Algorithm 2 of **SEVEN** different tree sampling methods described in subsection IV-B 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 *graphBpp* runs are non-deterministic, and we run the methods multiple times. The frustration committed 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. The results are summarized in Table IV in terms of the approximated frustration index and Figure 3 (top) in terms of the approximated frustration index as a percentage of the total number of edges in the graph. Table V summarizes the resulting frustration index per method. 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 3 (top). RDFS/Kruskal/Aldous Broder's 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 3 (bottom) as BLP takes 30 hours for larger datasets (far right navy bar in Fig. 3 (bottom)). RDFS-BFS hybrid app each is competitive with BFS in terms of frustration index (green and blue bars in Figure 3 (top)) with the small timing overhead for large graphs (Fig. 3 (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 III. 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 (AB) did not terminate for DutchCollege because, in uncommon scenarios, AB would get stuck looping when performing a random walk after all the current vertex's neighbors have already been visited.

2) Iteration Timing: Here, we evaluate the efficiency of the proposed algorithm by comparing *graphBpp* frustration and timing if the number of iterations increases. Figure 4 shows the change in performance for the two best tree sampling methods when the number of iterations grows. As discussed in subsection IV-B, more iterations will not impact BFS sampling in smaller graphs. The frustration shows a slight improvement for the larger graphs for both methods when the number of iterations increases in Figure 4.

3) graphBpp vs. SOTA: In this experiment, we compare the baseline BLP [27] with graphBpp implementation with breadth-first search (BFS) spanning tree sampling in 1000 iterations in terms of the frustration index and the time it takes to approximate the frustration index for 13 benchmark graphs in Table V. The space complexity of BLP is $O(|V|^2)$, where $|V|$ is the number of vertices on the graph. Aref et al. state that

TABLE III
AVERAGE FRUSTRATION INDEX COMPUTATION TIME PER ITERATION USING SPANNING TREE SAMPLING METHODS FOR 1000 ITERATIONS FOR KONECT DATA IN TABLE II (EXCEPT TWITTERRE,F WHICH IS NOT A KONECT GRAPH). **BFS** SAMPLING METHOD IS THE FASTEST. THE ALGORITHM IS HIGHLY PARALLELIZABLE.

Konect [5]	**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

TABLE IV
SNAP FRUSTRATION FOR 1000 ITERATIONS OF GRAPHBPP WITH SEVEN DIFFERENT TREE SAMPLERS INTRODUCED IN SUBSECTION IV-B, 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

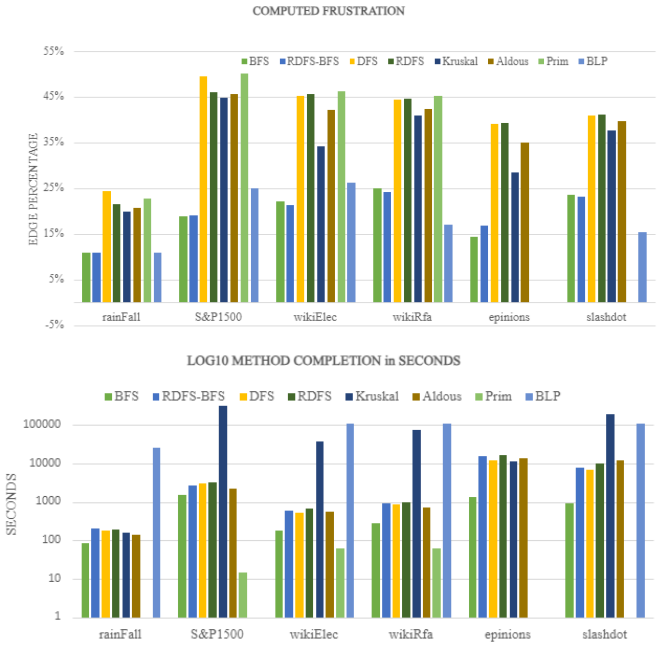

Fig. 3. Frustration index (top) and timing (bottom) comparison computed using Binary Linear Programming (BLP) [27] and graphBpp 1000 iterations for different tree sampling methods over different real large signed graphs except for Prim (1 iteration). BLP never finished computing the frustration index for epinions and sp1500 within the 30 hours allocated.

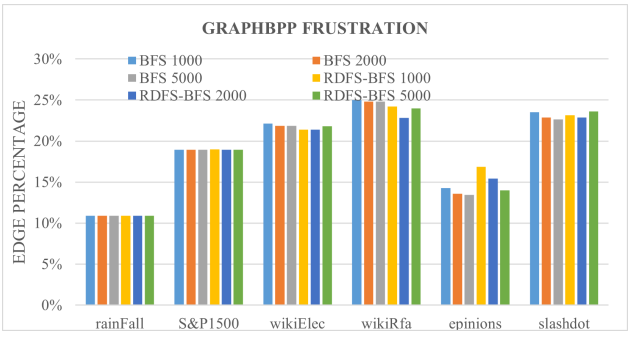

Fig. 4. *graphBpp* frustration for six benchmark datasets, two spanning three sampling approaches (BFS and RDFS-BFS), and three different iteration counts.

TABLE V
SNAP SIGNED GRAPH BASELINE PERFORMANCE AS A FUNCTION OF THE NUMBER OF FUNDAMENTAL CYCLES $|E| - |V| + 1$ FOR THE BASELINE BINARY LINEAR PROGRAMMING (BLP) [27] AND PROPOSED *graphBpp* ALGORITHM WITH BFS TREE SAMPLING FOR FRUSTRATION INDEX COMPUTATION. THE BLP BASELINE NEVER COMPLETES FOR OPINIONS AND NEVER CONVERGES BEYOND THE GUROBI HEURISTIC FOR S&P1500 WIKIELEC AND WIKIRFA DATASETS (*).

SNAP [4]	Cycles $\|E\| - \|V\| + 1$	**BLP** index	time	**graphBpp** index	time
test10 [29]	4	**2**	0.053s	**2**	0.08s
highland [45]	43	**7**	0.037s	**7**	0.13s
sampson18 [46]	95	**39**	0.08s	**39**	0.27s
rainFall [12]	93,331	**10,150**	7.26hrs	10,271	83.4s
S&P1500 [12]	709,836	176,965*	N/A	**134,515**	1478s
wikiElec [4]	104,520	29,360*	30hrs	**24,827**	184s
wikiRfa [4]	168,154	29,971*	30 hrs	43,971	281s
epinions [4]	585,138	N/A	N/A	**100,450**	1360s
slashdot [4]	418,342	**77,306***	30 hrs	117,587	937s

the signed graphs with up to $100,000$ edges will be solved in 10 hours [27]. Since our computer can store all 1000 nearest balanced states for each of these 13 signed graphs in memory, we ran the *graphBpp* implementation of Algorithm 2 (non-scalable version of the algorithm for small and medium-sized graphs). This algorithm does not handle the case when the memory is complete and it is slightly faster than Algorithm 3 where it employs nearest balanced state replacement after it reaches the memory limit. All external processes are closed to prevent interference with time measurements. The measurement for both methods includes the time it takes to input the file, process it, and output the results.

The last four columns of Table V summarize our findings on the SNAP benchmark. BLP and graphBpp computation for small graphs was fast and yielded equal indices such as highland and sampson18. Both methods retrieve correct frustration indices for the three datasets. The baseline code f ils for two graphs with over 700,000 edges, as described in Table II. The BLP code fails or the sparse opinions (over 700,000 vertices) and produces no results (adjacency matrix cannot fit in memory and crashes the jupyter notebook), where *graphBpp* 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 and crashes without outputting the time. The source of the failure is that the Python kernel stops working. On the other hand, *graphBpp* finds 1000 near-balanced states of the network and associated frustration 134,515 in 24.6 minutes (Table II). The most extensive signed graph in which we were able to run the linear binary solver was the fully connected rainFall network [12] due to $|E| < 100,000$ [27] algorithm limitation. The linear binary solver finished with a frustration index of 10,150 and a time of 7.26 hours while the *graphBpp* 's computed frustration index was 10,217 in 83.58s (0.02 hours), more than 300 times faster. The linear binary solver fails to complete the computations for wikiElec within 30 hours: the frustration index is 29,360, where *graphBpp* calculated the frustration 24,827 and provides an associated balanced state in 185s (3min+). Note that the BLP c de produces a heuristics frustration estimate in 30 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 30 hours is much lower than the discovered balanced states. On the other hand, able II outlines that *graphBpp* 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 *graphBpp* outputs the corresponding balanced state in the same time frame, while the BLP code does not. The *graphBpp* 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, *graphBpp* frustration is exact (4), close to actual (rainFall), or better than the heuristic (wikiElec, S&P1500, and epinions). The proposed *graphBpp* balanced state discovery is equal to or superior to the 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.

4) Scaling Balanced State Discovery: We implement Algorithm 3 and set the CAP to 75% of the total RAM size. We apply it to Amazon data in Table VI. The BLP model only worked and converged for the smallest 3 Amazon signed networks, the Core5 reviews in Table VI. All Amazon ratings and 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. The serialized process takes about an extra hour for each 1 million edges, and the processing time, for a fixed CAP, is way less than BLP across the board with an increasing number of edges and vertices, see Figure 5. The BLP algorithm c converges within 30 hours for smaller signed graphs and finds the optimal frustration index. *graphBpp* 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 VI. 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. In

TABLE VI
AMAZON RATINGS AND REVIEWS GRAPH CHARACTERISTICS [32]: THE NUMBER OF VERTICES, EDGES, AND CYCLES REFLECT THE NUMBER IN THE LARGEST CONNECTED COMPONENT OF EACH DATASET. (*) INDICATES THAT GUROBI NEVER CONVERGED.

Amazon Ratings	$\|V\|$	$\|E\|$	**BLP** index	**BLP** time	**graphBpp** index	**graphBpp** 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 Reviews	$\|V\|$	$\|E\|$	**BLP** index	**BLP** time	**graphBpp** index	**graphBpp** time
Digital Music	9,109	64,706	10,482*	30 hrs	19,926	101s
Instant Video	6,815	37,126	6,001*	30hrs	10,833	101s
Musical Instr	2,329	10,261	1,162	136.15s	2,311	17.3s

conclusion, we have demonstrated that *graphBpp* can scale and find the nearest balanced state for *any* size of a signed graph.

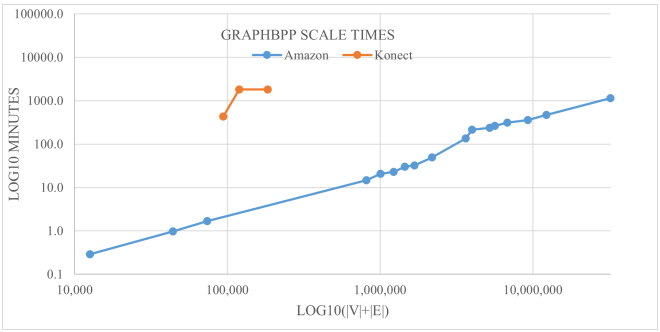

Fig. 5. *graphBpp* scales better with increasing graph size $|V| + |E|$ for a fixed number of iterations (1000) for the benchmark Konect Table II) and Amazon Table VI signed graphs.

VIII. THE GRAPHL PROOF OF CONCEPT

We pit *graphL* against *graphBpp* for approximating the frustration index and in order to obtain better stable states. We run both heuristics on the Konect signed graphs. We use Breadth-First Search as the tree-sampling technique for *graphBpp* using Algorithm 3 since we proved it yields minimal edge sign switches previously. Since we might obtain different results for each run for the gradient descent-based method due to the random initialization of the Γ vector, we run the heuristic five times and choose the minimal edge sign switches produced out of them. Table VII summarizes the results.

First, we can observe that the gradient descent-based method is much more efficient across the board because the heuristic runs in linear time, and it does not have to extract and save multiple nearest balanced states in memory because only *graphBpp* is capable of forming the frustration cloud. Second, the gradient descent-based approach generally produces more optimal balanced states for every signed graph than that of *graphBpp except* Sampson, Congress, and TwitterRef. for the same number of iterations/gradient updates. However, the former comes with a downside, which is tuning the learning rate hyperparameter and finding the proper initialization. Hence, *graphBpp* is advantageous in the sense that it does not need any hyperparameter tuning, and it is a non-trainable algorithm. Unlike *graphBpp*, *graphL* produces *only* one balanced state and cannot generate multiple nearest balanced states, which are essential for computing the consensus features proposed in [28]. These features are used in clustering and signed network analysis [49].

TABLE VII
COMPARISON OF THE FRUSTRATION INDEX APPROXIMATION AND EXECUTION TIME USING GRAPHBPP AND GRAPHL WITH 1000 ITERATIONS FOR BOTH ON THE KONECT DATA IN TABLE II (EXCEPT TWITTERRE,F WHICH IS NOT A KONECT GRAPH).

Konect [5] Graphs	**graphL index**	**graphL time**	**graphBpp index**	**graphBpp time**
Sampson	37	0.030s	**35**	0.304s
ProLeague	**13**	0.003s	**13**	0.27s
DutchCollege	**2**	0.010s	**2**	0.80s
Congress	38	0.008s	**21**	1.021s
BitcoinAlpha	**900**	0.18s	1,105	24.38s
BitcoinOTC	**1,426**	0.27s	1,827	40.55s
Chess	**14,684**	0.70s	20,991	85.35s
TwitterRef.	19,500	2.72s	**16,183**	456.80s
SlashdotZoo	**80,787**	6.31s	109,930	837.76s
Epinions	**57,874**	9.66s	100,646	1,367.74s
WikiElec	**15,389**	1.11s	22,289	168.59s
WikiConflict	**167,003**	25.29s	252,400	6503.24s
WikiPolitics	**58,438**	9.32s	86,833	1582.30s

IX. CONCLUSION

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. 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 [28]. In this paper, we extend our findings 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 *graphBpp* 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 [29]. And 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. In addition, we propose the use of gradient descent as a way to

approximate the frustration index in linear time. We compared both *graphBpp* and *graphL*, and we deduce that the latter is efficient and generally yields more optimal balanced states as tested on Konect signed graphs.

REFERENCES

[1] T. Antal, P. L. Krapivsky, and S. Redner, "Social balance on networks: The dynamics of friendship and enmity," *Physica D: Nonlinear Phenomena*, vol. 224, no. 1, pp. 130–136, 2006. [Online]. Available: http://arxiv.org/abs/physics/0605183

[2] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. ACM, 2010, pp. 641–650.

[3] M. Saberi, R. Khosrowabadi, A. Khatibi, B. Mišić, and G. Jafari, "Topological impact of negative links on the stability of resting-state brain network," *Scientific Reports*, vol. 11, no. 1, p. 2176, 2021. [Online]. Available: http://www.nature.com/articles/s41598-021-81767-7

[4] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, June 2014.

[5] J. Kunegis, "KONECT – The Koblenz Network Collection," in *Proceedings of the 22nd International Conference on World Wide Web*, ser. WWW '13. ACM, 2013, pp. 1343–1350. [Online]. Available: http://dl.acm.org/citation.cfm?id=2488173

[6] M. Tomasso, L. Rusnak, and J. Tešić, "Advances in scaling community discovery methods for signed graph networks," *Journal of Complex Networks*, vol. 10, no. 3, 06 2022, cnac013. [Online]. Available: https://doi.org/10.1093/comnet/cnac013

[7] S. Aref, A. J. Mason, and M. C. Wilson, "An exact method for computing the frustration index in signed networks using binary programming," *CoRR*, vol. abs/1611.09030, 2016. [Online]. Available: http://arxiv.org/abs/1611.09030

[8] T. Liu, J. Cui, H. Zhuang, and H. Wang, "Modeling polypharmacy effects with heterogeneous signed graph convolutional networks," *Applied Intelligence*, vol. 51, pp. 8316–8333, 2021.

[9] R. Li, X. Yuan, M. Radfar, P. Marendy, W. Ni, T. J. O'Brien, and P. M. Casillas-Espinosa, "Graph signal processing, graph neural network and graph learning on biological data: A systematic review," *IEEE Reviews in Biomedical Engineering*, pp. 1–1, 2021.

[10] S. Casas, C. Gulino, R. Liao, and R. Urtasun, "Spagnn: Spatially-aware graph neural networks for relational behavior forecasting from sensor data," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9491–9497.

[11] T. Liu, A. Sheng, G. Qi, and Y. Li, "Admissible bipartite consensus in networks of singular agents over signed graphs," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 8, pp. 2880–2884, 2021.

[12] M. Cucuringu, A. V. Singh, D. Sulem, and H. Tyagi, "Regularized spectral methods for clustering signed networks," *Journal of Machine Learning Research*, vol. 22, no. 264, pp. 1–79, 2021.

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

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

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

[16] F. Harary and D. Cartwright, "On the coloring of signed graphs." *Elemente der Mathematik*, vol. 23, pp. 85–89, 1968. [Online]. Available: http://eudml.org/doc/140892

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

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

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

[20] V. Amelkin and A. 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*, 2019, pp. 677–685.

[21] Z. Seif and M. Ahmadi, "Computing frustration index using genetic algorithm," *Communications in Mathematical and in Computer Chemistry*, vol. 71, pp. 437–443, 2014.

[22] G. Iacono and C. Altafini, "Average frustration and phase transition in large-scale biological networks: a statistical physics approach," *IFAC Proceedings Volumes*, vol. 43, no. 14, pp. 320–325, 2010, 8th IFAC Symposium on Nonlinear Control Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S147466701536986X

[23] M. Saberi, R. Khosrowabadi, A. Khatibi, B. Misic, and G. Jafari, "Requirement to change of functional brain network across the lifespan." *PLoS ONE*, vol. 16, no. 11, pp. 1 – 19, 2021. [Online]. Available: https://libproxy.txstate.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=fsr&AN=153656396&site=eds-live&scope=site

[24] X. Zhou, H. Song, and J. Li, "Residue frustration based prediction of protein-protein interactions using machine learning." *Journal of physical chemistry*, vol. 126, no. 8, pp. 1719–727, 2022. [Online]. Available: https://libproxy.txstate.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=agr&AN=IND607691906&site=eds-live&scope=site

[25] A. Fontan and C. 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*, pp. 109 – 114, 2018. [Online]. Available: https://libproxy.txstate.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.8619615&site=eds-live&scope=site

[26] F. Hüffner, N. Betzler, and R. Niedermeier, "Separator-based data reduction for signed graph balancing," *Journal of combinatorial optimization*, vol. 20, no. 4, pp. 335–360, 2010.

[27] S. Aref and Z. P. Neal, "Identifying hidden coalitions in the us house of representatives by optimally partitioning signed networks based on generalized balance," *Scientific reports*, vol. 11, no. 1, pp. 1–9, 2021.

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

[29] G. Alabandi, J. Tešić, L. Rusnak, and M. Burtscher, "Discovering and balancing fundamental cycles in large signed graphs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476153

[30] *Muhieddine Shebaro* and **Jelena Tešić**, "Scaling frustration index and corresponding balanced state discovery for real signed graphs," in ***Submitted to a Conference***, 2023.

[31] H. Du, X. He, and M. W. Feldman, "Structural balance in fully signed networks," *Complexity*, vol. 21, no. S1, pp. 497–511, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cplx.21764

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

[33] M. Saberi, R. Khosrowabadi, A. Khatibi, B. Misic, and G. Jafari, "Pattern of frustration formation in the functional brain network." *NETWORK NEUROSCIENCE*, vol. 6, no. 4, pp. 1334 – 1356, 2022. [Online]. Available: https://libproxy.txstate.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edswsc&AN=000889309400010&site=eds-live&scope=site

[34] M. T. Schaub, N. O'Clery, Y. N. Billeh, J.-C. Delvenne, R. Lambiotte, and M. Barahona, "Graph partitions and cluster synchronization in networks of oscillators," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 26, no. 9, p. 094821, 2016.

[35] T. Došlić and D. Vukičević, "Computing the bipartite edge frustration of fullerene graphs," *Discrete Applied Mathematics*, vol. 155, no. 10, pp. 1294–1301, 2007.

[36] N. Bansal, A. Blum, and S. Chawla, "Correlation clustering." *Machine Learning*, vol. 56, no. 1-3, pp. 89 – 113, 2004. [Online]. Available: https://libproxy.txstate.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edssjs&AN=edssjs.975B661D&site=eds-live&scope=site

[37] GUROBI, "Gurobi optimization," https://www.gurobi.com/.

[38] M. K. Fallah, M. Mirhosseini, M. Fazlali, and M. 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)*, 2020, pp. 390–394.

[39] W. Tang and J. Zhu, "Population-level balance in signed networks," 2023.

[40] M. Burtscher, "graphB+: Balancing algorithm for large graphs," https://userweb.cs.txstate.edu/~burtscher/research/graphB/, 2021.
[41] T. H. Cormen, *Introduction to algorithms.* MIT Press, 2009. [Online]. Available: https://libproxy.txstate.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=cat00022a&AN=txi.b5115051&site=eds-live&scope=site
[42] Y. Hu, R. Lyons, and P. Tang, "A reverse aldous-broder algorithm." *ANNALES DE L INSTITUT HENRI POINCARE-PROBABILITES ET STATISTIQUES*, vol. 57, no. 2, pp. 890 – 900, 2021. [Online]. Available: https://libproxy.txstate.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edswsc&AN=000677592900014&site=eds-live&scope=site
[43] Y. Wu, D. Meng, and Z.-G. Wu, "geeksforgeeks," https://www.geeksforgeeks.org/, accessed: 2023-06-01.
[44] S. I. Gass and M. C. Fu, Eds., *Prim's Algorithm*. Boston, MA: Springer US, 2013, pp. 1160–1160. [Online]. Available: https://doi.org/10.1007/978-1-4419-1153-7_200635
[45] K. Read, "Cultures of the central highlands, new guinea," *Southwestern Journal of Anthropology*, vol. 10, no. 1, pp. 1–43, 1954.
[46] S. Sampson, "A novitiate in a period of change: An experimental and case study of relationships," Ph.D. thesis, Cornell University, 1968.
[47] S. Aref, "frustration-index-xor," https://github.com/saref/frustration-index-XOR, 2021.
[48] 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.
[49] M. Tomasso, L. Rusnak, and J. Tešić, "Cluster boosting and data discovery in social networks," in *Proceedings of the 37th ACM/SIGAPP Symposium On Applied Computing (SAC)*, 2022.

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

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