Scaling Frustration Index and Balanced State Discovery for Real Signed Graphs

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

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 balanced state for any number of edge-sign switches needed. We propose an efficient, approximate technique that can scale to find the balanced states and the frustration index of networks with millions of nodes and edges in real time where state-of-the-art fails.

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

I. 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 [1], [2], or expand to new application domains, such as brain behavior [3]. However, signed graph benchmarks are currently too small and too similar in topology to actual data, hindering progress in signed graph analysis [4], [5]. Signed graph extensions have been demonstrated for narrow-band tasks in finance [6], polypharmacy [7], bioinformatics [8], and sensor data analysis [9], [10]. 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 [11], [12]. This work focuses on scaling the approximation of the frustration index and the associated balanced state for large signed networks. We extend graphB+ [13] to approximate the frustration index \( F_{\Sigma} \) 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 VII-C):

\[
F_{\Sigma} = \min_{i} (S(i))
\]  

(1)

where \( S \) is a container that stores the number of edge sign switches for a given \( i \)th 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. It is assigned to \( S(i) \). We aim to find a balanced state with minimum 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:

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

(2)

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

Balance theory represents a theory of changes in attitudes [14]: people’s attitudes evolve in networks so that friends of a friend will likely become friends, and so will enemies of an enemy [14]. Heider established the foundation for social balance theory [15], and Harary established the mathematical foundation for signed graphs and introduced the k-way balance [16], [17]. Balance theory concepts have been used to predict edge sentiment, to recommend content and products, or to identify unusual trends [18]–[21]. The frustration index is one measure of network property in many scientific disciplines, that is, in chemistry [22], biology [23], brain studies [24], physical chemistry [25] and control [26]. 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 [27]. State-of-the-art methods address the computation of the frustration index for signed graphs with up to 100,000 vertices [28], and the approach does not scale to modern large-scale sparse networks. 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 [29]. The graphB algorithm was introduced in [29] as a scaled adjustment of the Algorithm 1. 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
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. 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 \( T \) between vertices \( x \) and \( y \) does not include \( m \). The fundamental cycle is any cycle that is built using path in \( T \) plus edge \( m \) in graph \( G \).

Corollary 2.1: A fundamental cycle basis may be formed from a spanning tree or spanning forest of the given graph by selecting the cycles formed by combining a path in the tree and a single edge outside the tree. For the graph \( G \) with \( N \) vertices and \( M \) edges, there are precisely \( M - N + 1 \) fundamental cycles.

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 \( \sigma : m \to \{+1, -1\} \). The edge \( m \) can be positive \( m^+ \) or negative \( m^- \). Sign of a sub-graph is product of the edges signs. Balanced Signed graph is a signed graph where every cycle is positive. Frustration of a signed graph \( \text{Fr} \) is defined as the number of candidate edges whose sign needs to be switched for the graph to reach the balanced state. Frustration Cloud contains a collection of nearest balanced states for a particular signed graph.

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

Theorem 2.1 ([16]): If a signed subgraph \( \Sigma' \) is balanced, the following are equivalent:

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

The paper is organized as follows: in Section III, we summarize related work in the field; in Section IV, we introduce a novel algorithm for approximating the frustration index and its scalable version; in Section V, we analyze different sampling approaches of the spanning tree and propose two new methods; in Section VI, we analyze the time complexity of our algorithm; in Section VII, we conduct experiments to evaluate the efficiency of the proposed algorithm on ten different signed graph benchmarks and apply and measure the findings on signed graphs derived from the 26 Amazon ratings.
Researchers have focused Computing the Frustration Index: multiple life stages [31]. Saberi et al. investigated the pattern for the formation using the frustration index of the protein signed network chemistry, the protein-protein interaction can be predicted a person’s cognitive performance and the frustration of the networks [26]. The frustration in neuroplasticity assesses the a decision and win the disorder in adversarial multi-agent been determining the strength of agent commitment to make the system responds to perturbations in large-scale signed bio-

Frustration Applications: In chemistry, the stability of fullerenes is related to the frustration index [22]. The frustration index has been used to measure how an incoherent system responds to perturbations in large-scale signed biological networks [23]. 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 [26]. 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 [24]. In physical chemistry, the protein-protein interaction can be predicted using the frustration index of the protein signed network [25]. Saberi et al. investigated the pattern for the formation of frustrating connections in different brain regions during multiple life stages [31].

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 [32]. The frustration index for small fullerenes graphs can be calculated in polynomial time [33], and the finding has been extended to estimate the genetic algorithm of the frustration index in [22]. Bansal et al. introduced the correlation clustering problem, which is a problem in computing the minimum number of frustrated edges for several subsets [34]. Aref et al. provided an exact algorithm to calculate the partial balance and frustration index with $\Omega(2^b)|E|^2$ complexity where $b$ is a fixed parameter, and $|E|$ is the number of edges [6]. Recent improvements in the algorithm include binary programming models and the use of multiple powerful mathematical solvers by Gurobi [35], and the algorithm can handle up to $|E| = 100,000$ edges and compute the frustration index of the network in 10 hours [28]. These approaches are still too complex for millions of vertices in Amazon’s bi-partite ratings and reviews network. 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 [36] because the authors postulate that as these models grow, the efficiency decreases greatly, making it impossible to have any output.

### 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) [6].

**Frustration Applications:**
- In chemistry, the stability of fullerenes is related to the frustration index [22].
- The frustration index has been used to measure how an incoherent system responds to perturbations in large-scale signed biological networks [23].
- 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 [26].
- 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 [24].
- In physical chemistry, the protein-protein interaction can be predicted using the frustration index of the protein signed network [25].

### IV. Methodology

**Algorithm 1: Tree-Based Signed Graph Balancing**

```plaintext```
input signed graph $\Sigma$ and spanning tree $T$ of $\Sigma$
for edges $e, e \in \Sigma \setminus T$ do
  if fundamental cycle $T \cup e$ is negative then
    flip edge sign for edge $e$: $e^- \rightarrow e^+; e^+ \rightarrow e^-$
  end if
end for
return balanced graph $\Sigma'_T$.
```

GraphB+ is introduced as an efficient algorithm alternative for computing the nearest balanced states and frustration index approximation. The graphB+ algorithm builds on the graphB and graphB+ baseline. First, the algorithm integrates different tree-sampling approaches, as outlined in Algorithm 2 for the 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 Algorithm 3. The result of graphB++ is that the frustration index and the closest balanced associated state can be computed in minutes for any large signed graph.

We extend the definition of frustration cloud $\mathcal{F}_\Sigma$ from a set to a (key,value) tuple collection $\mathcal{F}_\Sigma = \mathcal{B}(C, S)$. The key is the unique balanced state $B(i)$, and the value is the count of balanced states occurring in iteration $C(i)$, and the edge count switches to the balanced state $S(i)$. In each balancing iteration, we examine the resulting balance state (Algorithm 2). $\Sigma'_T$ in relation to $\mathcal{B}$. We represent the balanced state $\Sigma'_T$ as a string $\mathcal{B}$ to make the process more efficient. The balanced state $\Sigma'_T$ 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$(i) = s$. 
Algorithm 2 Graph Balancing and Frustration Index

input signed graph $\Sigma$ and spanning trees sampling method $M$
generate set $T_{Mk}$ of $k$ spanning trees of $\Sigma$ using $M$ empty $F_{\Sigma}$
for spanning trees $T$, $T \in T_{k}$ do
  find nearest balanced state $\Sigma_{T}$ using Algorithm 1
  $s = $ edge signs difference count from $\Sigma$ to $\Sigma_{T}$
  Transform $\Sigma_{T}$ balanced state to string $B$
  if $B \notin B$ then
    add key $B$ to $B$
    $S(B) = s$
    $C(B) = 1$
  else
    $C(B)++$
  end if
end for
return frustration index $Fr(\Sigma) = min_{i}(S)$ and frustration cloud $F_{\Sigma} = B:(C,S)$

For graphB++ implementation, we propose an efficient transform ($O(\frac{1}{2}|V|)$) of the balanced state output $\Sigma_{T}$ to the string hash key $B$ for comparison with other balanced states (Algorithm 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 ""). 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 "").

V. SAMPLING SPANNING TREES

To maximize the chances of discovering the optimal nearest balanced state in Algorithm 2, we propose to utilize the randomization and hybridization of the standard tree sampling. The Depth first search (DFS) algorithm [37] 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. The Breadth first search (BFS) algorithm [37] 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 network’s randomized order of the adjacency list.
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 [38]. For the popular **Kruskal’s algorithm** [39] 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** [40] [39] 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. The algorithm of the RDFS-BFS sampler is written in Algorithm 4.

**Algorithm 4 Hybridized RDFS-BFS Sampling**

signed graph $\Sigma$ and a root node $n$ get uniformly distributed random number 0 or 1, $z$
\[\text{if } z \text{ is 0 then}
\text{run BFS algorithm [41]}
\text{else}
\text{run RDFS algorithm}
\text{end if}
\text{return spanning tree } T \text{ of } \Sigma
\]

VI. COMPLEXITY ANALYSIS

GraphB+ that is based on the original implementation of graphB+ (BFS) has a complexity of $O(|E| \cdot \log(|V| \cdot d))$ 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 $O(|E|)$. $O(|E| \cdot \log(|V| \cdot d))$ 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 $O(|V||E| \cdot \alpha)$ where $\alpha$ 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 [13] with complexity $O(|E| \cdot \log(|V| \cdot d))$. 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 $O(|V||E| \cdot \alpha)$ unless the complexity of the selected custom sampler is high enough to exceed this complexity. Section V outlines the time complexity for each tree-sampling technique.

VII. EXPERIMENTS

For the proof of concept, we compare the proposed method to BLP baseline [6] on SNAP [4], Konect [5], and Amazon [30] open-source benchmarks. The five experiments are summarized in subsections as follows:

- Section VII-B shows which tree-sampling technique for graphB++ is ideal for minimizing the approximation of the frustration index.
- Section VII-C 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 VII-D shows how the proposed graphB++ algorithm compares to Binary Linear Programming model (BLP) [42] for computing the amount of frustration on real-world signed networks in terms of performance and time.
- Section VII-E 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++.

| 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 [4] | vertices $|V|$ | $|E|$ | cycles | % positive |
| test10 [13] | 10 | 13 | 4 | 55.85 |
| highland [43] | 16 | 58 | 43 | 50 |
| sampson18 [44] | 18 | 112 | 95 | 54.4 |
| rainFall [11] | 306 | 93,636 | 93,331 | 68.78 |
| wikiElec [4] | 7,539 | 112,058 | 104,520 | 73.33 |
| wikiRfa [4] | 7,634 | 175,787 | 168,154 | 77.91 |
| slashdot [4] | 82,140 | 500,481 | 418,342 | 77.03 |

A. Setup, Implementation, and Data

**Signed Graph Benchmarks** used in the experiments are SNAP [4], Konect [5], and Amazon ratings [30]. Both SNAP and Konect signed graph characteristics are described in Table I and II respectively. Concerning Amazon, ratings and reviews data [30] 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, where ratings 5 and 4 imply a positive edge, ratings 3 and 2 give no edge, and ratings 0 and 1 give a negative edge. The graph characteristics are described in Table VI and result in tens of millions of vertices and edges in a signed graph.
### TABLE II

**Konect Largest Connected Component (LCC) graph attributes**

| Konect       | \(|V|\) | \(|E|\) | \(|E| - |V| + 1\) | \(|\%\) positive\) |
|--------------|-------|-------|----------------|-----------------|
| Sampson      | 18    | 126   | 145            | 51.32           |
| ProLeague    | 16    | 120   | 105            | 49.79           |
| DutchColledge | 322   | 322   | 291            | 31.57           |
| 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 | 93.91 |
| SlashdotZoo  | 79,116| 467,731| 388,616 | 76.09 |
| Opinions     | 119,130 | 704,267 | 585,138 | 85.29 |
| WikiElec     | 7,066 | 100,667| 93,602         | 78.77 |
| WikiConflict | 111,123| 2,025,910| 1,912,788 | 43.31 |
| WikiPolitics | 137,740| 715,334| 577,595         | 87.85 |

**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_64. The GPU is Nvidia GeForce 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 is based on the published Binary Linear Programming (BLP) code [42]. The binary linear model runs on a Jupyter notebook in Python [42] and is based on a Gurobi mathematical solver and covers several parameters [35]. 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 [28]. The code [42] 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 up to 40 hours. The code provided [42] 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 [13] to include and test proposed tree sampling strategies while keeping the original speed-up optimization for finding fundamental cycles intact. Implementing tree sampling strategies and adapting 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 Algorithm 3. This is done by retrieving the total RAM size from the Linux system during runtime and the current resident set size that 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 [45], and the data is publicly available [4], [5], [30].

**B. Selecting the Spanning Tree Method**

In this experiment, we compare the timing and frustration computation of graphB++ implementation of Algorithm 2 of SEVEN different tree sampling methods described in Section V 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 are 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 V summarizes the resulting frustration index per method. The results are summarized in Table III in terms of the approximated frustration index and Figure 4 in terms of the approximated 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. 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 5 as BLP takes 40 hours for larger datasets (far right navy bar in Fig. 5). RDFS-BFS hybrid approach is competitive with BFS in terms of frustration index (green and blue bars in Figure 4) with the small timing overhead for large graphs (Fig. 5): 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 IV. 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 DutchColledge because, in uncommon scenarios, this sampler would get stuck looping when performing a random walk after all the current vertex’s neighbors have already been visited.

**C. 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 V, 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 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.

D. graphB++ vs. SOTA

In this experiment, we compare the baseline BLP [28] with graphB++ 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. The researchers also stated that signed graphs with up to 100,000 edges could be solved in 10 hours [28]. Since the frustration cloud computation can fit in the memory, we ran the graphB++ implementation of Algorithm 2. 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 results are outlined in the last four columns of Table V. 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 optimization,
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 II. 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 II). The most extensive signed graph in which we were able to run the linear binary solver was the fully connected rainFall network [11] due to $|E| < 100,000$ [28] 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 (0.2 hours), more than 300 times faster.

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 epinions) 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 the discovered balanced states. On the other hand, Table II 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). 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.
E. Scaling Balanced State Discovery

In this experiment, 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 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. The BLP algorithm converges within 40 hours for smaller signed graphs 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 VI. The serialized process 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 6. 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 (Table VI) 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.

VIII. 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. 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 [29]. 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 [13], and we plan to do this in the future. This work reports 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


---

**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 Electrical 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.