Signed Graph Balancing in Linear Time

Avery VanAusdal
Texas State University
San Marcos, Texas, USA
arv107@txstate.edu

Lucas Rusnak
Texas State University
San Marcos, Texas, USA
lucas.rusnak@txstate.edu

Martin Burtscher
Texas State University
San Marcos, Texas, USA
burtscher@txstate.edu

Abstract

Signed graphs are often used to represent sentiments between entities, and balancing them can provide valuable insight, for example, in social network analysis. This paper presents ECL-SGB, a fast work-efficient algorithm and CUDA implementation for balancing signed graphs without needing to traverse each cycle individually. ECL-SGB's time complexity is linear in the size of the graph, providing better scalability than prior approaches. ECL-SGB outperforms the previously leading approach on every one of our 20 inputs. On average, it is 11.9x and 13.6x faster on an RTX 4090 and an A100 GPU, respectively. On the largest tested input with over 17 million vertices and 50 million edges, ECL-SGB balances over 754 million cycles per second on the RTX 4090.

CCS Concepts

• **Computing methodologies → Massively parallel algorithms**.

Keywords

Fundamental cycles, signed-graph balancing, parallelization, GPU computing

ACM Reference Format:
Avery VanAusdal, Lucas Rusnak, and Martin Burtscher. 2026. Signed Graph Balancing in Linear Time. In *Proceedings of The 18th Workshop on General Purpose Processing Using GPU (GPGPU '26).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Social network analysis has become an important workload since a large amount of human interaction is now occurring online. People express their opinions via social networks and online surveys. Some social network analyses focus on quantifying influence [2] and improving recommendation systems [9]. Majority voting is typically used for making decisions in these tasks [15], but this approach is prone to bias and does not consider *network-wide* consensus states. Yet, consensus in small-scale social networks has been well-researched in the field of psychology [1, 5–7].

Signed social networks are graphs whose edges model attitudes between vertices (people). Each edge can describe an agreeable or antagonistic relationship [6]. Two antagonistic edges cancel each other out, so if every cycle in a signed graph contains an even number of antagonistic edges, the graph is *balanced*, that is, in a global consensus state.

To improve the equity of consensus-based social network analysis, Rusnak and Tesic proposed the concept of the frustration cloud [14], which considers all nearest balanced consensus states. However, their approach, called graphB, does not scale to inputs with more than a few thousand vertices. To improve upon this, Alabandi et al. [3] introduced graphB+, a parallel GPU algorithm for computing nearest consensus states of large real-world social networks.

This paper focuses on reducing and simplifying the graph operations needed for discovering and balancing the fundamental cycles with a new approach. To this end, we present ECL-SGB, a fast and efficient parallel algorithm able to process large signed graphs. Our CUDA implementation can handle real-world inputs with millions of edges on a single GPU significantly faster than graphB+. This paper makes the following contributions.

- It introduces the ECL-SGB algorithm for balancing each fundamental cycle in a simple, constant-time operation without needing to traverse any cycle individually.
- It describes an efficient parallelization approach of the ECL-SGB algorithm for CUDA.
- It presents a performance evaluation that demonstrates the improved speed and efficiency of our GPU implementation.

We will open source our CUDA implementation of ECL-SGB.

The rest of the paper is organized as follows. Section 2 provides background information on balancing signed graphs. Section 3 explains the ECL-SGB algorithm and its parallelization. Section 4 summarizes related work. Section 5 describes the experimental methodology of the performance evaluation. Section 6 presents and analyzes the results. Section 7 concludes the paper with a summary.

2 Background

This section briefly covers how signed graphs are balanced and why it is important. More detail can be found in the graphB+ paper [3].

As mentioned in Section 1, edges in a signed graph describe agreeable (+1) or antagonistic (-1) relationships between vertices [6]. The sign of a path in such a graph is the product of the signs of its constituent edges. Since two negative edges cancel each other out, a positive cycle contains an even number of negative edges, while a negative cycle contains an odd number of negative edges. A signed graph is balanced if every cycle is positive. The vertices of a balanced signed graph can be bipartitioned into two sets of vertices that internally agree with each other but disagree with vertices in the other set. This is known as the Harary bipartition. All of the negative edges occur between the sets [5]. Fig. 1(a) shows a signed graph Σ with 4 vertices and 5 edges, and Fig. 1(b) shows all balanced states of Σ, where the negative edges comprise the Harary cuts. These Harary bipartitions are needed for deriving vertex and edge

GPGPU '26, Pittsburgh, PA

ACM ISBN XXX-X-XXXX-XXXX-X/XXXX/XX
https://doi.org/XXXXXXX.XXXXXXX

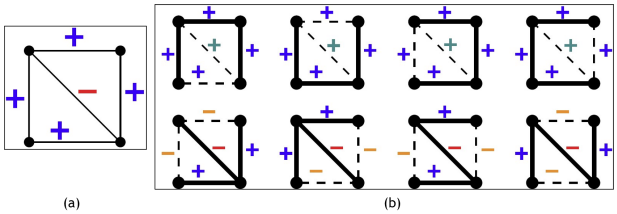

Figure 1: (a) signed input graph Σ; (b) all spanning trees and the corresponding nearest structurally balanced states [3]

attributes that are important for global consensus analysis [14]. For example, the *status* of a vertex is the probability that the vertex will be in the majority, i.e., the larger set, over all nearest balanced states of Σ [14]. Fig. 2(a) shows the Harary bipartitions of our example graph Σ, and Fig. 2(b) shows the resulting status values for each vertex. For instance, in Fig. 2(a), the top left vertex belongs to the majority set 6 out of 8 times, represented by its status of 6/8 = 0.75 as shown in Fig. 2(b). Rusnak and Tesic found 1000 trees to be a good sample size for computing the attributes of larger graphs [14].

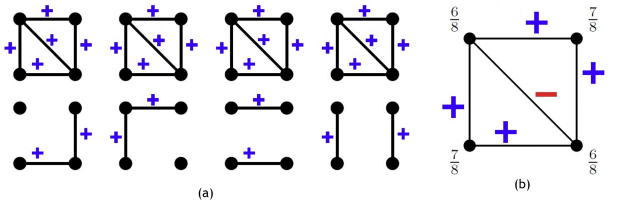

Figure 2: (a) Harary bipartitions of the balanced states; (b) corresponding *status* of the vertices of Σ [3]

2.1 Tree-based Signed Graph Balancing

Alg. 1 computes a nearest balanced state for the signed input graph Σ and a given spanning tree T, as proven elsewhere [3, 14].

Algorithm 1 Tree-based Signed Graph Balancing [3]

Require: Signed graph $\Sigma = (G, \sigma)$
Require: Spanning tree T of Σ
  for all edges e, $e \in \Sigma \setminus T$ **do**
    if fundamental cycle $T \cup e$ is negative **then**
      switch edge sign: $e^- \rightarrow e^+; e^+ \rightarrow e^-$
    end if
  end for
Ensure: Balanced graph Σ_T

Given any spanning tree T of a connected signed graph Σ, each edge e of Σ that does not belong to T forms a fundamental cycle when combined with the simple path in T that connects the endpoints of e. That is, there is exactly one fundamental cycle for each edge that does not belong to T. Each of these fundamental cycles is unique because it contains an edge e that is not present in any other fundamental cycle.

2.2 Complexity Analysis of graphB+

Given a provided spanning tree T of a signed graph with n vertices and m edges, graphB+ relabels the vertex IDs based on a pre-order traversal of the tree, then labels the tree edges with the range of IDs reachable by each edge [3]. Next, graphB+ uses these labels to traverse and balance the fundamental cycles induced by adding the non-tree edges to the tree. The relabeling is implemented in parallel as two $O(n)$ steps, which are a bottom-up and then a top-down pass over the tree levels. All vertices at a given level (i.e., tree depth) are processed in parallel. Next, graphB+ traverses the induced cycles by using the range information to follow the appropriate edges from vertex to vertex. It counts the number of negative edges along the cycle and sets the sign of the non-tree edge to make that total number even. Traversing all cycles in this manner requires $O(m \times k \times t)$ work, where k is the average cycle length and t is the average tree-degree of the vertices on a cycle. For random BFS trees, the expected tree depth is $O(log(n))$, providing an upper bound for k. The CUDA implementation of graphB+ parallelizes across the vertices and the edges.

To perform the Harary bipartitioning on each resulting nearest balanced state, graphB+ first computes the connected components (CCs) of the graph while ignoring negative edges, which requires $O(n + m)$ work. Each CC is collapsed into a single vertex. It then performs a BFS on the resulting graph to count the number of hops (negative edges) between the root and each component. Counting the number of vertices that are an odd number of hops from the root (in $O(n)$ time) gives the size of the Harary bipartitions, allowing further graph attributes to be calculated [14].

3 The ECL-SGB Algorithm

ECL-SGB is a new streamlined graph-balancing algorithm that requires significantly less work than prior algorithms such as graphB+. It uses a novel approach for efficiently identifying and balancing all fundamental cycles of a graph without needing to traverse the individual cycles. Alg. 2 outlines how ECL-SGB works on the signed graph $\Sigma = (G, \sigma)$ using a provided spanning tree T of Σ. The tree can be generated with any spanning tree algorithm. ECL-SGB's memory footprint is linear in the size of the graph.

Algorithm 2 ECL-SGB Algorithm

Require: $\Sigma = (G, \sigma)$
Require: Spanning tree $T \in \Sigma$
1: $pathsign[root] \leftarrow positive$
2: **for** level l of T from top to bottom **do** ▷ Path sign labeling
3: **for all** vertices v_{src}, $v_{src} \in level[l]$ **do**
4: **for** vertices v_{dst}, $v_{dst} \in child[v_{src}]$ **do**
5: $e = edge\ v_{src} \rightarrow v_{dst}$
6: $pathsign[v_{dst}] \leftarrow pathsign[v_{src}] \times sign[e]$
7: **end for**
8: **end for**
9: **end for**
10: **for all** edges e, $e \in \Sigma \setminus T$, $e = (v_{src}, v_{dst})$ **do** ▷ Cycle balancing
11: $sign[e] \leftarrow pathsign[v_{src}] \times pathsign[v_{dst}]$
12: **end for**
Ensure: Balanced graph Σ_T

Given a tree of a signed graph as input, the ECL-SGB algorithm performs the following two steps to compute the nearest balanced state:

1) It labels each vertex v with the sign of the path from the root vertex to v. The sign of the root starts as positive. As ECL-SGB traverses the paths from the root, each reached vertex receives the product of the previous vertex's path sign and the connecting edge's sign. In this way, the signs of the paths from the root are calculated level-by-level until every vertex is reached.

2) It balances the fundamental cycles created by each each non-tree edge $e = (v_{src}, v_{dst})$ by setting the sign of e to the product of the path signs of v_{src} and v_{dst}. The resulting edge sign balances the fundamental cycle $T \cup e$ (cf. Section 2.1). Balancing all fundamental cycles guarantees that all other cycles are balanced [3]. Once all non-tree edges are processed, i.e., all fundamental cycles are balanced, the entire graph is balanced.

The path sign labels also identify which Harary bipartition each vertex belongs to. Vertices with the same path sign belong to the same set. Determining the majority set is as simple as counting the number of vertices with positive (or negative) path signs.

3.1 Complexity Analysis and Comparison

Assume Σ to be a connected graph with n vertices and m edges. Since Σ is connected, $m \geq n - 1$ and any spanning tree T of Σ has $n - 1$ edges. Like graphB+, the space complexity of ECL-SGB is $O(n + m)$ because it uses a constant amount of storage for each vertex and each edge. Step 1 of the ECL-SGB algorithm computes the path signs by performing a top-down traversal of T, which requires $O(n)$ work as it processes each vertex and each tree edge. Step 2 balances the signs of all non-tree edges using a constant time operation per edge, requiring $O(m)$ work. Counting the number of vertices in each set to determine the majority also takes $O(n)$ work, but this can be amortized by counting as the path signs are set in Step 1 (Line 6 of Alg. 2). In contrast, graphB+ is dominated by the $O(m \times log(n) \times t)$ work of traversing and balancing all cycles, where t is the average tree-degree of the vertices on a cycle [3]. graphB+ also requires additional work to identify the sets of the Harary bipartition (see Section 2.2), while ECL-SGB just needs to refer to the calculated path signs. Since the overall work complexity of ECL-SGB is $O(m)$, linear in the size of the graph, it represents a significant improvement.

3.2 GPU Implementation & Parallelization

This section discusses how our ECL-SGB CUDA code is implemented and parallelized. It also includes further comparisons with the graphB+ algorithm.

Since the vertex path signs and edge signs can only be −1 or +1, our implementation stores signs as Boolean values. Instead of calculating the product of integer signs as shown in Alg. 2, our code simply compares the Boolean values for equality; if the signs are the same, the resulting sign is positive. By storing these values as Booleans instead of integers, the data structures require less space and enjoy greater locality.

Our implementation uses the same deterministic parallel BFS tree generation code that graphB+ uses. Like graphB+, ECL-SGB parallelizes the processing of a single tree at a time. Calculating the path signs (Step 1) is performed by a top-down pass over the tree levels as outlined in Alg. 2. All vertices at a given level (i.e., tree depth) are processed in parallel, meaning the for-loop on Line 3 of Alg. 2 is parallel. Each vertex is assigned a set of 32 adjacent threads (i.e., a warp) to process its set of neighbors in parallel (Line 4), which minimizes any load imbalance caused by non-uniform vertex degrees. No synchronization is needed within a level since each child can only have one parent in the tree and thus will only be read and written to by one thread each. Alabandi et al. also found level-by-level parallelization to be efficient for graphB+ due to the low expected diameter of signed social networks [3].

Processing the cycles (Step 2) using the path signs is embarrassingly parallel. It is parallelized over the edges while only processing the non-tree edges (Line 10 of Alg 2). Furthermore, the non-tree edges are only processed in one direction to minimize wasted work. No synchronization is needed for this step since the path signs are only read and the edge metrics are only updated by a single thread per edge.

The following are implementation aspects that ECL-SGB and graphB+ share [3]. We employ the widely-used compressed sparse row (CSR) format to represent the graph in memory. We keep only a single copy of the graph in memory by marking edges' tree membership in the adjacency lists using 1-bit flags. The BFS tree generator used by these codes creates a queue of vertices that is partitioned by tree level. Both codes re-use this queue to efficiently traverse the tree level-by-level when needed.

4 Related Work

Harary and Kabell [8] proposed a serial algorithm for *testing* whether a signed graph is balanced by propagating path signs from an arbitrary root vertex and comparing the path signs of the endpoints of non-tree edges. Loukakis [11] improved the efficiency of their approach using BFS traversal. We derived a similar algorithm that *balances* the cycles formed by non-tree edges to discover the nearest balanced states of many trees and compute statistical attributes of vertices and edges. Moreover, our approach is parallel and targets GPUs.

Alabandi et al. [3] introduced graphB+, the first parallel algorithm for balancing signed graphs. graphB+ uses a new vertex and edge labeling method to efficiently traverse and balance all fundamental cycles in parallel. Our work performs the same task as graphB+, so we use it as the baseline for our performance study. Shebaro and Tesic published extensions of graphB+ for efficiently storing stable states [16] and computing the frustration index [17] using a variety of tree generators. Their work uses the same core algorithm and parallelization strategy as graphB+.

5 Experimental Methodology

We evaluate the performance of our CUDA implementation of ECL-SGB and compare it to the public graphB+ CUDA code [4]. The reported execution times include building the trees and the Harary bipartitioning but exclude reading input graphs, building internal representations, and allocating data structures. We ensure that graphB+ and ECL-SGB generate the same 1000 trees per input for a fair comparison. To validate the results, we confirm that ECL-SGB produces the exact same result values as graphB+.

We used two GPUs from different generations for these experiments. One is an RTX 4090 GPU with 16,384 processing elements and 24 GB of global memory, which is based on the Ada Lovelace architecture. The clock frequency is 2235 MHz for the processing elements and 1313 MHz for the GDDR6X memory. The other GPU is an A100 with 6912 processing elements and 40 GB of global memory, which is based on the Ampere architecture. The clock frequency is 765 MHz for the processing elements and 1215 MHz for the HBM2e memory. We used nvcc version 12.6 with the "-O3 -arch=sm_89" flags to compile for the 4090, and nvcc version 12.0 with the "-O3 -arch=sm_80" flags for the A100.

The 20 graphs from Table 1 served as inputs. They are similar to the inputs that Alabandi et al. [3] used to evaluate graphB+ and were obtained from the Stanford Network Analysis Platform (SNAP) [10] and from Amazon. Note that we use the significantly larger inputs from the 2018 version of the Amazon dataset [12, 13]. The table describes the characteristics of the largest connected component and the overall graph. Note that we only process the largest connected component with graphB+ and ECL-SGB, which encompasses nearly the entirety of each input. To ensure a representative sample size [14], both codes compute the nearest balanced states of 1000 BFS trees for each input.

6 Results

Tables 2 and 3 list the average runtimes in milliseconds for graphB+ and ECL-SGB to create and balance one random BFS tree for each tested input alongside the corresponding speedup ratios of ECL-SGB over graphB+ on both GPUs. The tables' columns list the runtimes of different parts of each code, such as generating the spanning tree. Figs. 3 and 4 show the corresponding overall throughputs in millions of fundamental cycles balanced per second to visualize the performance. Note that the x-axis is logarithmic and that higher throughputs are better.

The results show that ECL-SGB is substantially faster and more scalable than graphB+. The overall speedups exhibit a moderate to strong positive correlation with the average tree-degree of the vertices on the cycles. This is because traversing cycles in graphB+ linearly scales with this metric (Section 2.2), whereas ECL-SGB processes cycles in constant time per vertex (Section 3.1). This highlights the scalability advantage of ECL-SGB, which enables the processing of larger and denser graphs. The overall geometric mean speedup of ECL-SGB over graphB+ is 11.89x and 13.63x on the 4090 and A100, respectively.

Tables 2 and 3 break down the overall runtime into the major steps of the computation. The Generate Tree step covers the parallel generation of a random spanning BFS tree. Process Cycles is the main computational step and includes the discovery and balancing of all fundamental cycles formed by the spanning tree. For graphB+, this step also includes relabeling the vertex and edges since it is an optimization for faster cycle traversal. ECL-SGB is much faster in this step due to its significantly lower work requirements as detailed in Section 3.1. Lastly, the Harary Cut step is where the codes determine set membership and identify the majority set. While graphB+ accomplishes this in $O(n + m)$ time (Section 2.2), ECL-SGB achieves the same by merely counting the number of vertices with positive and negative paths from the root, which only

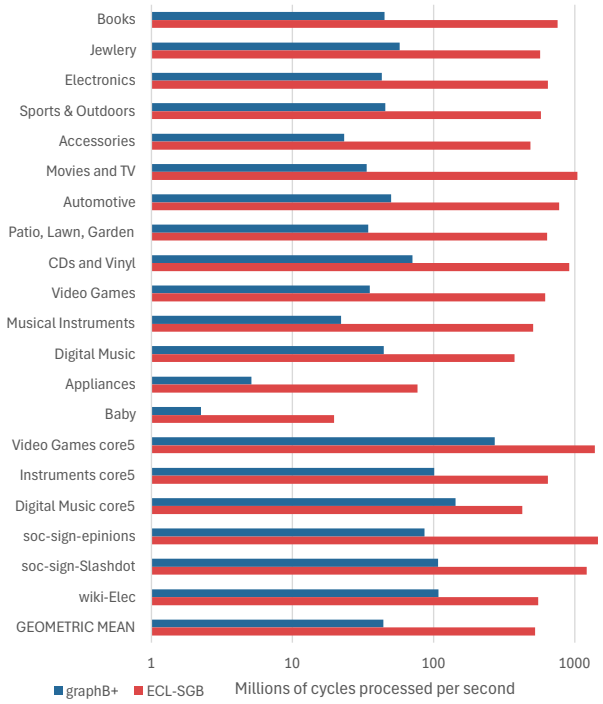

Figure 3: Throughput in millions of fundamental cycles balanced per second of graphB+ and ECL-SGB on the RTX 4090

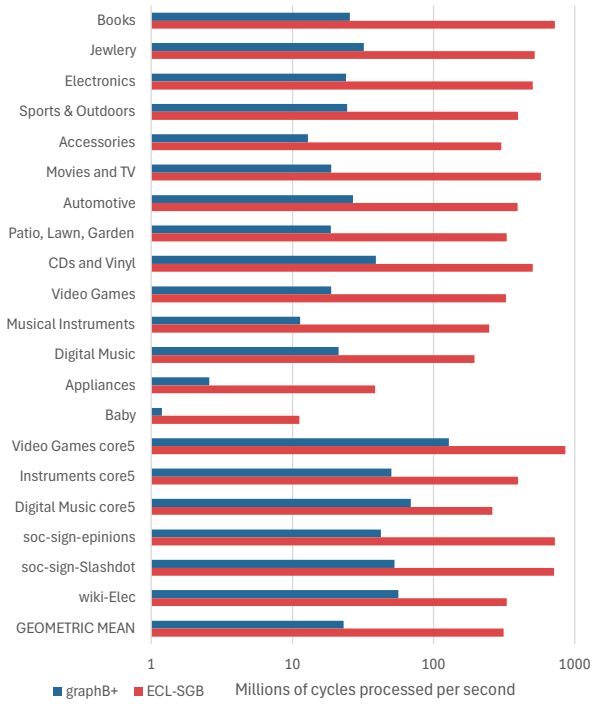

Figure 4: Throughput in millions of fundamental cycles balanced per second of graphB+ and ECL-SGB on the A100

Table 1: Information about the signed input graphs. The number of vertices, edges, and cycles reflect the number in the largest connected component of each dataset

	Largest Connected Component					Entire graph
Amazon Ratings	**# vertices**	**# edges**	**# cycles**	**max degree**	**average degree**	**# ratings**
Books	17,856,079	50,796,519	32,940,441	58,147	2.84	51,311,621
Clothing, Shoes, and Jewlery	14,659,532	31,383,157	16,723,626	12,845	2.14	32,292,099
Electronics	10,404,617	20,444,865	10,040,249	28,530	1.96	20,994,353
Sports and Outdoors	7,253,621	12,365,977	5,112,357	9,990	1.70	12,980,837
Cell Phones and Accessories	6,515,991	9,868,709	3,352,719	13,543	1.51	10,063,255
Movies and TV	3,964,100	8,479,831	4,515,732	24,542	2.14	8,765,568
Automotive	4,457,387	7,617,840	3,160,454	9,585	1.71	7,990,166
Patio, Lawn and Garden	3,238,068	4,971,840	1,733,773	10,263	1.54	5,236,058
CDs and Vinyl	2,269,286	4,392,234	2,122,949	7,208	1.94	4,543,369
Video Games	1,584,495	2,471,591	887,097	6,462	1.56	2,565,349
Musical Instruments	948,992	1,428,303	479,312	4,704	1.51	1,512,530
Digital Music	936,780	1,297,556	360,777	3,411	1.39	1,584,082
Appliances	498,099	557,338	59,240	6,505	1.12	602,777
Baby	127,832	137,526	9,695	2,823	1.08	164,849
Amazon Reviews	**# vertices**	**# edges**	**# cycles**	**max degree**	**average degree**	**# reviews**
Video Games core5	72,631	473,427	400,797	782	6.52	497,577
Musical Instruments core5	38,150	219,156	181,007	1,756	5.74	231,392
Digital Music core5	28,348	145,278	116,931	572	5.12	169,781
SNAP Signed Networks	**# vertices**	**# edges**	**# cycles**	**max degree**	**average degree**	**# signed**
soc-sign-epinions	119,130	704,267	585,138	3,558	5.91	841,372
soc-sign-Slashdot090221	82,140	500,481	418,342	2,548	6.09	549,202
wiki-Elec	7,136	106,807	99,672	1,115	14.97	114,040

takes $O(n)$ time because the path signs have already been computed in the Process Cycles step. Since both implementations use the same code to generate random trees, this portion of the runtime is similar, as shown in the "Generate Tree" columns. Whereas generating a tree on the 4090 only accounts for an average of 5.51% of graphB+'s overall runtime, it accounts for 53.24% of ECL-SGB's overall runtime as the runtime is so much shorter. If the tree generation time is ignored, the geometric mean speedup of ECL-SGB over graphB+ is 24.3x and 29.3x on the 4090 and A100, respectively.

7 Summary and Conclusions

This paper describes an efficient parallel algorithm called ECL-SGB for balancing large signed graphs. It runs in $O(m)$ time and requires $O(n + m)$ storage, where m is the number of edges and n the number of vertices in the input graph. ECL-SGB accomplishes this with a new path-sign labeling technique that allows cycles to be balanced in a simple constant-time operation without requiring the traversal of any cycle. This marks a significant work efficiency improvement over the state-of-the-art graphB+ algorithm, which runs in $O(m \times log(n) \times t)$ time, where t is the average tree-degree of the vertices on each cycle [3].

We parallelized ECL-SGB using CUDA and found it to be significantly faster in practice than the public graphB+ CUDA implementation. Based on the geometric mean, ECL-SGB is 11.89x and 13.63x faster than graphB+ on an RTX 4090 and an A100 GPU, respectively. Moreover, it is at least twice as fast on every tested input graph. ECL-SGB discovers and balances up to 1.46 billion cycles per second on the tested inputs using an RTX 4090.

Acknowledgments

This work has been supported by the NSF under Award #1955367 and by an equipment donation from NVIDIA Corporation.

Table 2: Average execution times (in milliseconds) for graphB+ and ECL-SGB to create and balance one BFS tree for each input on the 4090 GPU. The columns list the times of different parts of each algorithm. The last column is the overall speedup of ECL-SGB over graphB+

4090 GPU	graphB+ (ms)				ECL-SGB (ms)				
Input Graph	**Generate Tree**	**Process Cycles**	**Harary Cut**	**Overall Total**	**Generate Trees**	**Process Cycles**	**Harary Cut**	**Overall Total**	**Overall Speedup**
Books	21.469	646.492	64.936	732.948	21.391	20.173	1.991	43.650	16.79
Clothing, Shoes, and Jewlery	15.006	243.741	31.121	289.930	14.902	13.285	1.234	29.467	9.84
Electronics	7.728	210.204	15.914	233.900	7.637	7.096	0.789	15.560	15.03
Sports and Outdoors	4.586	100.998	6.736	112.385	4.586	3.750	0.496	8.862	12.68
Cell Phones and Accessories	3.619	134.299	5.980	143.952	3.466	3.013	0.400	6.909	20.84
Movies and TV	1.870	127.658	4.904	134.485	1.900	2.055	0.333	4.323	31.11
Automotive	1.773	58.049	3.451	63.324	1.793	1.942	0.305	4.078	15.53
Patio, Lawn and Garden	1.176	46.457	2.782	50.467	1.150	1.340	0.205	2.725	18.52
CDs and Vinyl	1.039	26.902	1.902	29.895	1.045	1.063	0.179	2.315	12.91
Video Games	0.639	23.105	1.281	25.078	0.653	0.654	0.098	1.436	17.47
Musical Instruments	0.523	20.393	0.771	21.739	0.555	0.311	0.052	0.945	23.00
Digital Music	0.587	6.849	0.630	8.118	0.604	0.278	0.049	0.961	8.45
Appliances	0.461	10.621	0.464	11.594	0.477	0.233	0.032	0.769	15.08
Baby	0.305	3.767	0.190	4.314	0.313	0.124	0.026	0.491	8.79
Video Games core5	0.161	1.038	0.224	1.476	0.162	0.069	0.029	0.288	5.12
Musical Instruments core5	0.152	1.408	0.174	1.786	0.163	0.061	0.026	0.280	6.39
Digital Music core5	0.164	0.473	0.130	0.819	0.169	0.052	0.026	0.276	2.97
soc-sign-epinions	0.246	6.166	0.302	6.766	0.240	0.102	0.030	0.400	16.91
soc-sign-Slashdot090221	0.198	3.418	0.215	3.883	0.201	0.085	0.028	0.343	11.31
wiki-Elec	0.099	0.666	0.107	0.920	0.094	0.035	0.024	0.181	5.08

Table 3: Average execution times (in milliseconds) for graphB+ and ECL-SGB to create and balance one BFS tree for each input on the A100 GPU. The columns list the times of different parts of each algorithm. The last column is the overall speedup of ECL-SGB over graphB+

A100 GPU	graphB+ (ms)				ECL-SGB (ms)				
Input Graph	**Generate Tree**	**Process Cycles**	**Harary Cut**	**Overall Total**	**Generate Trees**	**Process Cycles**	**Harary Cut**	**Overall Total**	**Overall Speedup**
Books	23.845	1189.083	79.945	1292.953	23.199	20.351	2.086	45.682	28.30
Clothing, Shoes, and Jewlery	17.633	455.461	49.905	523.081	17.425	13.579	1.216	32.268	16.21
Electronics	10.342	379.550	29.101	419.076	10.143	9.099	0.735	20.022	20.93
Sports and Outdoors	7.114	186.768	15.167	209.131	7.043	5.368	0.456	12.913	16.20
Cell Phones and Accessories	5.990	241.750	12.875	260.697	6.000	4.723	0.371	11.143	23.40
Movies and TV	4.149	228.696	7.846	240.768	4.246	3.245	0.304	7.842	30.70
Automotive	4.553	105.259	7.682	117.575	4.595	3.109	0.284	8.034	14.64
Patio, Lawn and Garden	2.813	84.976	4.954	92.824	2.840	2.172	0.197	5.255	17.66
CDs and Vinyl	2.236	48.576	3.685	54.578	2.278	1.716	0.172	4.214	12.95
Video Games	1.399	43.176	2.549	47.197	1.459	1.119	0.111	2.735	17.26
Musical Instruments	1.013	39.647	1.670	42.404	1.074	0.745	0.077	1.944	21.82
Digital Music	1.004	14.629	1.351	17.057	1.052	0.687	0.074	1.859	9.17
Appliances	0.865	20.844	1.262	23.045	0.910	0.538	0.048	1.543	14.94
Baby	0.530	7.116	0.460	8.179	0.548	0.243	0.032	0.868	9.42
Video Games core5	0.255	1.937	0.858	3.127	0.256	0.129	0.039	0.468	6.68
Musical Instruments core5	0.245	2.603	0.685	3.605	0.263	0.117	0.034	0.458	7.87
Digital Music core5	0.251	0.849	0.524	1.703	0.264	0.106	0.034	0.450	3.78
soc-sign-epinions	0.452	12.157	1.119	13.800	0.480	0.233	0.048	0.810	17.03
soc-sign-Slashdot090221	0.358	6.631	0.890	7.951	0.343	0.161	0.038	0.588	13.51
wiki-Elec	0.170	1.263	0.270	1.775	0.160	0.068	0.030	0.304	5.85

References

[1] Robert P. Abelson and Milton J. Rosenberg. 1958. Symbolic psycho-logic: A model of attitudinal cognition. *Behavioral Science* 3, 1 (1958), 1–13. doi:10.1002/bs.3830030102

[2] S. Al-Yazidi, J. Berri, M. Al-Qurishi, and M. Al-Alrubaian. 2020. Measuring Reputation and Influence in Online Social Networks: A Systematic Literature Review. *IEEE Access* 8 (2020), 105824–105851. doi:10.1109/ACCESS.2020.2999033

[3] Ghadeer Alabandi, Jelena Tešić, Lucas Rusnak, and Martin Burtscher. 2021. Discovering and balancing fundamental cycles in large signed graphs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 68, 17 pages. doi:10.1145/3458817.3476153

[4] Ghadeer Alabandi, Avery Vanausdal, and Martin Burtscher. 2024. graphB+ v0.3. https://userweb.cs.txstate.edu/~burtscher/research/graphBplus/.

[5] D. Cartwright and F. Harary. 1956. Structural balance: a generalization of Heider's theory. *Psychological Rev.* 63 (1956), 277–293. doi:10.1037/h0046049

[6] F. Harary. 1953. On the notion of balance of a signed graph. *Michigan Math. J.* 2(2) (1953), 143–146. doi:10.1307/mmj/1028989917

[7] F. Harary. 1959. On the measurement of structural balance. *Behavioral Sci.* 4 (1959), 316–323. doi:10.1002/bs.3830040405

[8] Frank Harary and Jerald A. Kabell. 1980. A simple algorithm to detect balance in signed graphs. *Mathematical Social Sciences* 1, 1 (1980), 131–136. doi:10.1016/0165-4896(80)90010-4

[9] Jure Leskovec. 2015. New Directions in Recommender Systems. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (WSDM '15)*. ACM, Shanghai, China, 3–4. doi:10.1145/2684822.2697044

[10] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[11] E. Loukakis. 2003. A Dynamic Programming Algorithm To Test A Signed Graph For Balance. *International Journal of Computer Mathematics* 80, 4 (2003), 499–507. doi:10.1080/0020716021000009246

[12] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Amazon Review Data (2018). https://nijianmo.github.io/amazon/index.html.

[13] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 188–197. doi:10.18653/v1/D19-1018

[14] Lucas Rusnak and Jelena Tešić. 2021. Characterizing Attitudinal Network Graphs through Frustration Cloud. *Data Mining and Knowledge Discovery* 35, 6 (Sept. 2021), 2498–2539. doi:10.1007/s10618-021-00795-z

[15] Farah Saab, Imad H. Elhajj, Ayman Kayssi, and Ali Chehab. 2019. Modelling Cognitive Bias in Crowdsourcing Systems. *Cognitive Systems Research* 58 (2019), 1 – 18. doi:10.1016/j.cogsys.2019.04.004

[16] Muhieddine Shebaro and Jelena Tešić. 2023. Identifying Stable States of Large Signed Graphs. In *Companion Proceedings of the ACM Web Conference 2023* (2023-04-30) *(WWW '23 Companion)*. Association for Computing Machinery, New York, NY, USA, 594–597. doi:10.1145/3543873.3587544

[17] Muhieddine Shebaro and Jelena Tešić. 2023. Scaling Frustration Index and Corresponding Balanced State Discovery for Real Signed Graphs. doi:10.48550/arXiv.2311.00869