Maximum Balanced (k, ϵ)-Bitruss Detection in Signed Bipartite Graph

Kai Hiu CHUNG
Hong Kong University of Science and Technology
qzhongaf@cse.ust.hk

Alexander ZHOU
Hong Kong University of Science and Technology
atzhou@cse.ust.hk

Yue WANG
Shenzhen Institute of Computing Sciences
yuewang@sics.ac.cn

Lei CHEN
Hong Kong University of Science and Technology
leichen@cse.ust.hk

ABSTRACT

Signed bipartite graphs represent relationships between two sets of entities, including both positive and negative interactions, allowing for a more comprehensive modeling of real-world networks. In this work, we focus on the detection of cohesive subgraphs in signed bipartite graphs by leveraging the concept of balanced butterflies. A balanced butterfly is a cycle of length 4 that is considered stable if it contains an even number of negative edges. We propose a novel model called the balanced (k, ϵ)-bitruss, which provides a concise representation of cohesive signed bipartite subgraphs while enabling control over density (k) and balance (ϵ). We prove that finding the largest balanced (k, ϵ)-bitruss is NP-hard and cannot be efficiently approximated to a significant extent. Furthermore, we extend the unsigned butterfly counting framework to efficiently compute both balanced and unbalanced butterflies. Based on this technique, we develop two greedy heuristic algorithms: one that prioritizes followers and another that focuses on balanced support ratios. Experimental results demonstrate that the greedy approach based on balanced support ratios outperforms the follower-based approach in terms of both efficiency and effectiveness.

PVLDB Reference Format:
Kai Hiu CHUNG, Alexander ZHOU, Yue WANG, and Lei CHEN. Maximum Balanced (k, ϵ)-Bitruss Detection in Signed Bipartite Graph. PVLDB, 17(3): 332 - 344, 2023.
doi:10.14778/3632093.3632099

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/qixiaoz/BalancedBitruss.

1 INTRODUCTION

Bipartite graphs are widely used in various areas, including social network analysis and recommendation systems. They are particularly useful in modeling positive relationships between two distinct sets of entities, such as consumers and products [44], authors and papers [5], users and items in e-commerce websites [9]. However, many real-world networks involve not only positive interactions but also negative ones, which can represent conflicts, competition, or distrust between entities [13]. For example, consumers may not only "like" some products, but also "dislike" some products. To capture such polarized relationships, signed bipartite graphs have been introduced as a natural extension of unsigned bipartite graphs [13]. Balance theory, a concept in social psychology that proposes people strive for consistency and balance in their attitudes and beliefs towards others [18], is a well-established approach to studying signed graphs.

In unipartite graphs, balance theory suggests that a triangle is balanced if it contains an even number of negative edges, and unbalanced if it contains an odd number of negative edges [6], reflecting the intuition that the "friends of friends are friends" and "the enemies of enemies are friends". Despite many works applying balance theory on unipartite graphs [12][43][3][37], balance analysis on bipartite graphs has not yet been extensively explored. The first comprehensive analysis and validation of balance theory using the smallest cycle in signed bipartite networks, signed butterfly, was conducted in a previous work [13]. A butterfly is a cycle of length 4, and it is balanced (i.e. stable) if it contains an even number of negative edges, and unbalanced otherwise. By examining the isomorphism classes of balanced butterflies, we can gain insights into the underlying patterns of balanced relationships in signed bipartite graphs. Figure 1 shows the isomorphism classes of balanced butterflies.

To interpret the meaning of a balanced butterfly formed by nodes u_0, u_1, v_0, v_1 in a signed bipartite graph, we can conceptualize u_0, u_1 as consumers and v_0, v_1 as products. The sign of edges reflects the consumers' attitudes towards the products. A balanced butterfly involving these nodes signifies that consumers u_0, u_1 possess either identical (in class (A), (C), (E)) or opposite (in class (B), (D)) preferences concerning products v_0, v_1. In cases of identical preferences, u_0, u_1 may share a liking or disliking for both products, or both consumers might independently dislike both products, corresponding to isomorphism class (A), (C), (E) in Figure 1, respectively. Conversely, opposite preferences could manifest as u_0 liking v_0 and v_1 while u_1 dislikes both, or u_0 liking v_0 but disliking v_1 while u_1 exhibits the opposite preference, corresponding to isomorphism class (B), (D) in Figure 1, respectively. Notably, in either case, these relationships are stable according to the balanced theory, with consumers in classes (A), (C), and (E) viewing each other as friends due to their identical preferences, while those in classes (B) and (D) considering each other as enemies owing to their divergent preferences. On the other hand, an unbalanced butterfly (in class

Proceedings of the VLDB Endowment, Vol. 17, No. 3 ISSN 2150-8097.
doi:10.14778/3632093.3632099

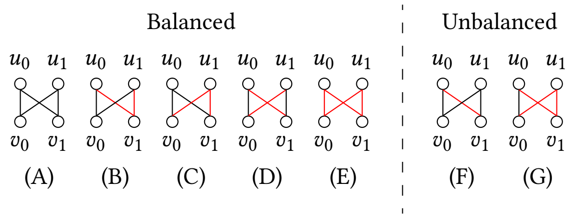


Figure 1: Isomorphism Classes of Signed Butterflies. In this paper, the positive edges are colored in black and the negative edges are colored in red.

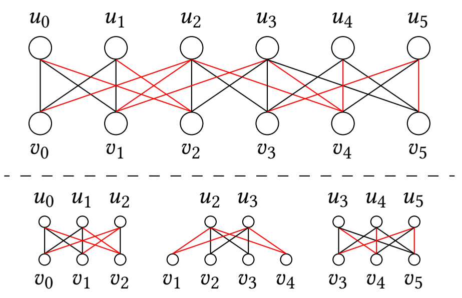


Figure 2: Example of a balanced bitruss and the maximal balanced signed bicliques it contains.

(F), (G)) formed by these nodes does not imply that consumers u_0, u_1 share friendship or enmity preferences regarding the products. To determine whether u_0 and u_1 are friends or enemies, at least one edge needs to switch its sign, illustrating the inherent instability of unbalanced butterflies according to the balance theory.

Detecting cohesive subgraphs in signed bipartite graphs is a crucial task that helps uncover patterns of positive and negative relationships, providing valuable insights into the underlying social dynamics [16]. One of the popular models for cohesive subgraphs is the fully-connected subgraph, known as biclique [27][17]. While previous studies about biclique mining mainly focus on unsigned bipartite graphs, Sun et. al. [35] pay initial attention to the bicliques in signed bipartite graphs. To consider the stability of cohesive structure regarding the sign information, they employ balance theory and investigate the balanced signed biclique. The balanced bicliques in a signed bipartite graph are defined as the bicliques free from unbalance butterflies. This work enumerates all maximal balanced signed bicliques over a certain size threshold. However, bicliques are the most stringent model of cohesive structures on bipartite graphs [26]. In practice, bicliques in real bipartite graphs are often numerous, small, scattered, and likely to overlap, and only a small proportion of them are useful for practical applications after being found by an algorithm [1]. Furthermore, the balanced signed biclique may not be able to capture more comprehensive signed relationships between entities, as it only considers bicliques and does not allow for edges that are not part of a biclique. The balanced signed biclique does not take into account the density or proportion of unbalanced butterflies. Therefore, bicliques may not always be the best model for dense subgraphs.

Bitruss is a cohesive subgraph defined based on butterflies, with a parameter k that controls the density of the community it represents [47]. The larger k, the denser the community represented by a k-bitruss. Compared to other cohesive models such as quasi-bicliques [33] and (α, β)-cores [25], bitruss is a more natural and succinct model as the balance of a signed bipartite graph depends on the butterflies within it [13]. Additionally, a single maximal bitruss can cover the cohesive parts of the entire bipartite graph, making it a flexible model for bipartite graphs of different densities. To apply bitruss to signed bipartite graphs, we limit the proportion of unbalanced butterflies that each edge can be contained in within a bitruss. This is reasonable as the balance of an edge depends on the butterflies it is contained, and limiting the proportion of unbalanced butterflies ensures that the every edge in the bitruss remains balanced. Inspired by this, we propose a new model, balanced (k, ϵ)-bitruss. In a signed bipartite graph, a subgraph S is a balanced (k, ϵ)-bitruss if (1) each edge in the S is contained in at least k butterflies, and (2) for each edge, among all butterflies containing the edge, the proportion of unbalanced butterflies is at most ϵ.

Example 1. *Consider a social network where users* $(u_0, \ldots, u_5)$ *are connected to movies* $(v_0, \ldots, v_5)$ *they have watched and rated. The edges in the network are signed based on the sentiment of the user's rating. A positive edge indicates a positive sentiment, while a negative edge indicates a negative sentiment. Suppose we want to identify a group of users and movies that have strong and stable sentiment towards each other. In this scenario, we could use the balanced* (k, ϵ)*-bitruss to identify cohesive subgraphs with a specific density and proportion of unbalanced butterflies. Indeed, all butterflies in the graph are balanced, so this graph represents a cohesive and balanced community, forming a* $(3, 0)$*-bitruss. However, if we use the balanced signed biclique enumeration algorithm to identify maximal* $(2, 3)$*-balanced signed bicliques in this graph, we will find three balanced bicliques as depicted in the lower part of Figure 2. Each biclique only reveals a fragment of the community, resulting in three distinct fragment of a community. It is not desirable to have multiple fragmented communities instead of a single giant* $(3, 0)$*-bitruss that represents the entire cohesive and balanced community. Moreover, the balanced signed biclique would not be able to capture the density or proportion of unbalanced butterflies in the subgraph.*

Unlike the original k-bitruss on an unsigned bipartite graph, there may exist multiple maximal balanced (k, ϵ)-bitrusses in a signed bipartite graph. As we will explain later, it is computationally impractical to enumerate all maximal balanced (k, ϵ)-bitrusses in a signed bipartite graph. Therefore, to capture the largest possible communities or groups of entities that exhibit a certain degree of balance in their relationships, we aim to find the balanced (k, ϵ)-bitruss with the largest size, given k and ϵ. By maximizing the size of the balanced (k, ϵ)-bitrusses, our goal is to capture the most comprehensive balanced cohesive patterns in the signed bipartite graph. The balanced (k, ϵ)-bitruss can be used in various applications. A few examples are given below.

Antagonistic group detection: In a customer-product signed network, the balanced butterflies can be a useful reference for identifying friendly and hostile users. By analyzing the sign information, we can detect groups of users who tend to have similar or opposite preference to each other [15], which can help with targeted marketing or customer service interventions. In the example shown in Figure 2, the subgraph induced by $\{u_0, u_1, u_5, v_0, v_1, v_4\}$ and the subgraph induced by $\{u_2, u_3, u_4, v_2, v_3, v_5\}$ can be regarded as antagonistic groups each other, because the inner-group edges are mostly positive and the inter-group edges are mostly negative.

Stable community detection: By identifying cohesive subgraphs that satisfy the balanced (k, ϵ)-bitruss condition, we can detect closely related communities where each edge is guaranteed a certain degree of balance [28]. This can help in understanding social dynamics in various contexts, such as online social networks [4] or political organizations [14].

Recommendations: Since a balanced (k, ϵ)-bitruss in a signed bipartite graph is sufficiently balanced, existing techniques [13] can be adopted to predict unconnected vertex pairs that are likely to have positive or negative interactions, or suggest switching the sign of existing edges to improve overall network balance. This can be useful in recommendation systems [22], such as for e-commerce or social media platforms, where personalized recommendations can improve user engagement and satisfaction.

1.1 Challenges

To our best knowledge, we are the first to propose the community detection problem on signed bipartite graphs while considering the customizable density (k) and degree of balance (ϵ). Our investigation reveals that the problem of finding the largest balanced (k, ϵ)-bitruss is NP-hard, and furthermore, it cannot be efficiently approximated in any non-trivial proportion. Unlike unsigned bipartite graphs where the k-bitruss is unique [47], we show that there may exist multiple balanced (k, ϵ)-bitrusses in signed bipartite graphs, making the application of previous techniques on unsigned bitruss unsuitable for this problem. While a straightforward approach could be to first calculate the unsigned k-bitruss utilizing existing algorithms and subsequently eliminate the least number of edges required to create a maximum balanced (k, ϵ)-bitruss, the computation expense of this method is prohibitive even on small graphs due to the extensive number of edge combinations that require removal. For example, there are $\binom{1000}{100} > 10^{139}$ combinations to remove 100 edges from a bitruss of 1,000 edges.

1.2 Contributions

To handle such hardness of the problem, we first propose a technique (`count`) that extends the unsigned butterfly counting framework [39] to efficently compute both balanced and unbalanced butterflies. We then utilize this technique to modify the Bloom-Edge (BE) index [40] to adapt to signed bipartite graphs, improving the performance of updating information about signed butterflies after the removal of an edge. Moreover, we introduce a pruning strategy based on the bitruss that counts balanced butterflies only, which filters the unpromising search space. We then propose two greedy heuristic algorithms, `GreedyF` and `GreedyS`. The `GreedyF` is based on the followers of an edge, and the `GreedyS` based on the ratio of unbalanced butterfly support to the total butterfly support of an edge. In our experiments, we compare the performance of the two greedy heuristic algorithms to that of the exact algorithm (`Exact`) on a couple of small datasets. Overall, the heuristic algorithms are significantly more efficient than the exact one, without compromising too much on the result size.

Our major contributions can be summarized as following:

- We formally define balanced (k, ϵ)-bitruss and the the maximum balanced (k, ϵ)-bitruss search problem.
- We prove the NP-hardness of the problem, and the NP-hardness of an approximation solution to this problem, with any nontrivial approximation ratio.
- We develop novel strategies to speed up the counting of balanced and unbalanced butterflies, and their updates after edge removals. Equipped with these strategies, we propose two greedy heuristic algorithms.
- We conduct experiments on real-world datasets to evaluate our techniques and algorithms.

2 RELATED WORK

Truss and bitruss: Cohesive subgraphs have been extensively studied in social network analysis. Cohen [11] introduced the concept of a k-truss, where every edge belongs to at least $k-2$ triangles, offering efficient computation and capturing social cohesion. Che et al. [7] improved truss decomposition efficiency in massive networks. Zhao et al. [45] extended the truss model to signed networks with the signed k-truss, focusing on friend and foe relationships. Wu et al. [42] introduced the signed (k, r)-truss for signed networks. Zou [47] proposed the bitruss model for bipartite graphs, and Wang et al. [40] developed the BE-Index for efficient bitruss decomposition. Leveraging the BE-Index, we designed the signed BE-Index for balanced and unbalanced support in balanced (k, ϵ)-bitruss. While these models target cohesive subgraphs in various network settings, the balanced (k, ϵ)-bitruss uniquely considers the proportion of unbalanced butterflies, enhancing its ability to capture community cohesiveness and balance in signed bipartite graphs. This extension of the truss and bitruss concepts to signed contexts enhances its relevance in real-world applications.

Butterfly counting: Butterfly is the simplest non-trivial motif in signed bipartite graphs and serves as a building block for community structure [24][29], measuring graph cohesion [2], and computing clustering coefficients [19]. Several studies have focused on developing efficient algorithms for counting butterflies in bipartite networks. A deterministic solution for butterfly counting on regular bipartite networks has been established [38], and subsequent improvements have led to the current best solution, which utilizes a vertex priority approach [30][39]. For the the baseline signed butterfly counting algorithm in this work, we use a trivial extension of the vertex priority approach. Additionally, parallel [32] and cache-aware [41] solutions are explored to improve the efficiency of butterfly counting. Furthermore, butterfly counting has expanded to include a number of problem variations, such as counting in uncertain bipartite graphs [46], counting in bipartite graph streaming [31], and counting approximation by sampling [30], making it an active and diverse research area.

Other cohesive subgraph models in signed networks: Cohesive subgraph mining in signed networks has seen increased attention recently, with various models proposed. Li et al. [23] introduced the (α, k)-clique model, while Chen et al. [8] explored the maximum signed θ-clique. Both models employ efficient branch and bound enumeration algorithms. However, they do not consider balance theory. The balanced clique model [10][36] addresses this gap, focusing on enumerating maximal balanced cliques. Kim et al. [20] introduced the (p, n)-core model, ensuring specific positive and negative neighbor counts within cores. Sun et al. [34] proposed the stable k-core model for community detection in signed networks. Sun et al. [35] tackled the enumeration of maximal balanced signed bicliques in signed bipartite graphs. However, none of these models consider the proportion of unbalanced relationships, a key feature of our balanced (k, ϵ)-bitruss model. This additional parameter offers greater flexibility in characterizing community structures in signed bipartite graphs, capturing cohesion and balance under varying degrees of unbalanced relationships. It's worth noting that all these cohesive subgraph models in signed graphs, including ours, are NP-hard to find.

3 PRELIMINARIES

In this section, we formally describe our notations and definitions of our problem, which are summarized in Table 1.

Definition 1 (Signed bipartite graph). *A signed bipartite graph is an undirected bipartite graph $G = (U, V, E = E^+ \cup E^-)$, where U and V are two disjoint sets of nodes, and $E \subset U \times V$ is the set of edges with two partitions, E^+ and E^-.*

For an edge $e \in E^+$, we call e a positive edge and write sign(e) = '+'. For an edge $e \in E^-$, we call e a negative edge and write sign(e) = '-'. In this paper, we color positive edges in black and negative edges in red.

Definition 2 (Neighbor). *Let $G = (U, V, E = E^+ \cup E^-)$ be a signed bipartite graph. For a vertex $u \in U$, we call $N_G(u) := \{v \in V : (u, v) \in E\}$ the neighbors of u. Moreover, $N_G^+(u) := \{v \in V : (u, v) \in E^+\}$ is called the positive neighbors of u, and $N_G^-(u) := \{v \in V : (u, v) \in E^-\}$ is called the negative neighbors of u. Similar definitions are settled for a vertex $v \in V$.*

We use $\deg_G(u)$, called the *degree* of u, to indicate $|N_G(u)|$.

Table 1: Table of Notations

Notation	Definition
$G = (U, V, E = E^+ \cup E^-)$	Signed bipartite graph
$e = (u, v)$	Edge form by $u \in U$ and $v \in V$
sign(e)	Sign of e
$N_G^{(\pm)}(u)$	(Signed) neighbors of node u in G
$\deg_G(u)$	Degree of node u in G
$⧖ = (e_0, e_1, e_2, e_3)$ $= [u_0, u_1, v_0, v_1]$	Butterfly
B	Bloom
$\text{Sup}_G(e)$	Supports of edge e in G
(u, v, w)	Wedge
$<_p$	Vertex priority relation
twin(B, e)	Twin edge of e in bloom B

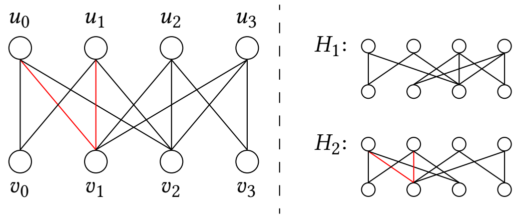


Figure 3: Example of a signed bipartite graph with two maximal balanced (k, ϵ)-bitrusses.

Definition 3 (Biclique). *Let $G = (U, V, E)$ be a bipartite graph and $S = (U_S, V_S, E_S)$ a subgraph of G. Then S is called a biclique if $E_S = U_S \times V_S$.*

We may omit the edge set E_S when we indicate a biclique. To specify the size of a biclique, we may say S is a $(|U_S|, |V_S|)$-biclique. We use the name k-bloom, or bloom, to indicate either a $(2, k)$-biclique or a $(k, 2)$-biclique [40].

Definition 4 (Signed Butterfly [13]). *In a signed bipartite graph $G = (U, V, E)$, a signed butterfly is a $(2, 2)$-biclique $(\{u_0, u_1\}, \{v_0, v_1\})$. The signed butterfly is called balanced if is has even number of negative edges, and unbalanced otherwise.*

We write $⧖ = ((u_0, v_0), (u_0, v_1), (u_1, v_0), (u_1, v_1))$ or $⧖ = [u_0, u_1, v_0, v_1]$ for simplicity. From previous work [13] we know that signed butterfly has 7 isomorphism classes, where 5 of them are balanced and 2 of them are unbalanced, as shown in Figure 1.

Definition 5 (Balanced / Unbalanced Supports). *Given a signed bipartite graph $G = (U, V, E)$ and an edge $e \in E$, the support of e in G, denoted as $\text{Sup}_G(e)$, is the number of butterflies in G containing e. Specifically, the balanced support $\text{Sup}_G^+(e)$ is the number of balanced butterflies containing e, and the unbalanced support $\text{Sup}_G^-(e)$ is the number of unbalanced butterflies containing e.*

Definition 6 (Balanced (k, ϵ)-bitruss). *For a signed bipartite graph G, a positive integer k, and a real number $\epsilon \in [0, 1]$, a balanced (k, ϵ)-bitruss is a subgraph H where,*

1. *H is a k-bitruss, i.e., for each edge e in H, $\text{Sup}_H(e) \geq k$;*
2. *for each edge e in H, $\text{Sup}_H^-(e) / \text{Sup}_H(e) \leq \epsilon$;*

Note that we do not require a balanced (k, ϵ)-bitruss to be connected.

Lemma 1. *The maximal balanced (k, ϵ)-bitruss in a signed bipartite graph G is not unique. In other words, there may exists multiple maximal balanced (k, ϵ)-bitrusses in G.*

Proof. Consider the signed bipartite graph shown in Figure 3, suppose $k = 1$ and $\epsilon = 0$. Then there are two balanced (k, ϵ)-bitruss, namely, $H_1 = G(\{(u_0, v_0), (u_0, v_2), (u_1, v_0), (u_1, v_2), (u_2, v_1), (u_2, v_2), (u_2, v_3), (u_3, v_1), (u_3, v_2), (u_3, v_3)\})$ and $H_2 = G(\{(u_0, v_0), (u_0, v_1), (u_0, v_2), (u_1, v_0), (u_1, v_1), (u_1, v_2), (u_2, v_1), (u_2, v_3), (u_3, v_1), (u_3, v_3)\})$. □

3.1 Problem Definition

Given a signed bipartite graph $G = (U, V, E)$, a positive integer k, and a real number $\epsilon \in [0, 1]$, we aim to find the balanced (k, ϵ)-bitruss $H = (U_H, V_H, E_H)$ with the largest edge size $|E_H|$. The largest edge size is preferred over the largest vertex size because the edge size reflects both the capacity and density of communities.

4 PROBLEM ANALYSIS

Theorem 1. *Given a signed bipartite graph G, the problem of computing the maximum balanced (k, ϵ)-Bitruss is NP-hard.*

Theorem 2. *Given a signed bipartite graph G, it is NP-hard to approximate the maximum balanced (k, ϵ)-bitruss within a factor of $|E|^{1-\delta}$, for any $\delta > 0$.*

See proofs in our supplementary materials[1].

5 SOLUTIONS

In this section, we first propose techniques for signed butterfly counting and support maintenance, as well as a pruning method based on the supports. Since the problem is computationally challenging, we also propose two heuristic strategies in addition to an exact solution.

5.1 Signed Butterfly Counting

To determine a maximum balanced (k, ϵ)-bitruss, we need to know the balanced support and unbalanced support of every edge. Therefore, the first step to solve the maximum balanced (k, ϵ)-bitruss problem is to count the balanced butterflies and the unbalanced butterflies in the given signed bipartite graph.

5.1.1 Baseline. Our baseline algorithm of signed butterfly counting is an extension from an existing unsigned butterfly counting algorithm (BFC-VP)[39], which relies on the concept of wedges and vertex priority.

Definition 7 (Wedge). *In a bipartite graph $G = (U, V, E)$, a wedge redis a path of length 2 and we denote a wedge as (u, v, w), where u and w are the endpoint vertices, and v is the middle vertex.*

Definition 8 (Vertex Priority [39]). *Let $G = (U, V, E)$ be a bipartite graph and $id : U \cup V \rightarrow \{1, \dots, |U|+|V|\}$ a bijective function called the* **index** *of vertices. Then the vertex priority $<_p$ is the strict total order on vertices ($U \cup V$) satisfying the following:*

$$u <_p v \Leftrightarrow \begin{cases} id(u) < id(v) & \textit{if } \deg_G(u) = \deg_G(v) \\ \deg_G(u) < \deg_G(v) & \textit{otherwise.} \end{cases}$$

According to the literature [39], the vertex priority is effective to avoid repetitive counting of butterflies and reduce the time cost of butterfly counting algorithm. To import the BFC-VP algorithm to our problem, we need to distinguish the balanced butterflies and the unbalanced butterflies in the given signed bipartite graph.

The details of the baseline algorithm for signed butterfly counting is shown in Algorithm 1, where Line 1 through Line 8 is the framework of BFC-VP [39]. We sort the neighbors of each node in the order of vertex priority $<_p$ (Line 1). For every node u (Line 3), we create a Hash-map H of vertex lists taking the vertices with the same side with u as its keys, and for each two-hop neighbor w of u with $w <_p u$, we store each node v in $H(u, w)$ if $v <_p u$ and the wedge (u, v, w) exists (Line 4-7). Guaranteed by the properties of the vertex priority [39], every wedge will be iterated exactly once (note

Algorithm 1 Signed-BFC-Base

```
Input:  $G = (U, V, E)$: Signed bipartite graph
Output: $c^+$: Balanced butterfly count
        $c^-$: Unbalanced butterfly count
1:  Sort $N_G(u)$ of each $u \in U \cup V$ by vertex priority $<_p$
2:  $c^+ \leftarrow 0, c^- \leftarrow 0$
3:  for each $u \in U \cup V$ do
4:      Create $H(u, w)$ for each node $w$ at same side as $u$
5:      for each $v \in N_G(u) : v <_p u$ do
6:          for each $w \in N_G(v) : w <_p u$ do
7:              $H(u, w)$.append($v$)
8:      for each node $w : H(u, w) > 1$ do
9:          for each pair of distinct $v_0, v_1 \in H(u, w)$ do
10:             if $[u, w, v_0, v_1]$ is balanced then
11:                 $c^+ += 1$
12:             else $c^- += 1$
```

that (u, v, w) and (w, v, u) are identical). We observe the butterfly $[u, w, v_0, v_1]$ exists if there are wedges (u, v_0, w) and (u, v_1, w) for each pair of distinct vertices v_0 and v_1. Clearly there are $\binom{|H(u,w)|}{2}$ butterflies for each nontrivial $H(u, w)$, but we have to figure out which of them are balanced and which of them are unbalanced (Line 8-12).

Theorem 3. *The time-complexity of the Signed-BFC-Base algorithm is $O\left(\sum_{(u,v)\in E} \min\left\{\deg_G(u)^2, \deg_G(v)^2\right\}\right)$.*

Proof. We know [39] that the time-complexity of the BFC-VP algorithm is $O\left(\sum_{(u,v)\in E} \min\left\{\deg_G(u), \deg_G(v)\right\}\right)$, which is dominated by the wedge iteration (Algorithm 1, Line 3-8). According to the vertex priority, for each pair of vertices u, w with $w <_p u$, there are at most $\deg_G(w)$ wedges in which u and w are the endpoints. In addition to the BFC-VP, since the Signed-BFC-Base algorithm iterates every combination of two wedges (u, v_0, w) and (u, v_1, w), while there are at most $\binom{\deg_G(w)}{2} \in O(\deg_G(w)^2)$ such combinations, then the time-complexity of Signed-BFC-Base turns out to be $O\left(\sum_{(u,v)\in E} \min\left\{\deg_G(u)^2, \deg_G(v)^2\right\}\right)$. □

Corollary 1. *The space-complexity of the Signed-BFC-Base algorithm is $O\left(\sum_{(u,v)\in E} \min\left\{\deg_G(u), \deg_G(v)\right\}\right)$.* □

5.1.2 Improvements. Indeed, it is possible to calculate the number of balanced (and unbalanced) butterflies induced by u, w and $H(u, w)$ in the Algorithm 1 without enumeration. To do so, we need to categorize the wedges in a signed bipartite graph into two types.

Definition 9 (S-Wedge And D-Wedge [35]). *Let (u, v, w) be a wedge in a signed bipartite graph. Then (u, v, w) is called an s-wedge if* $\text{sign}((u, v)) = \text{sign}((v, w))$*, or is called a d-wedge if* $\text{sign}((u, v)) \neq \text{sign}((v, w))$*, as shown in Figure 4.*

Lemma 2. *Let $b = [u_0, u_1, v_0, v_1]$ be a butterfly in a signed bipartite graph. Then b is balanced if (u_0, v_0, u_1) and (u_0, v_1, u_1) are both s-wedges or both d-wedges; b is unbalanced if there are one s-wedge and one d-wedge from (u_0, v_0, u_1) and (u_0, v_1, u_1).*

[1] https://github.com/qixiaoz/BalancedBitruss

PROOF. Observe that an s-wedge has either 0 or 2 negative edges, and a d-wedge has 1 negative edge. Therefore, butterfly b has even number of negative edges, i.e. balanced, if and only if there are even number of d-wedges from (u_0, v_0, u_1) and (u_0, v_1, u_1). □

Utilizing Lemma 2, we present our improvement of signed butterfly counting in Algorithm 2.

Algorithm 2 Signed-BFC

```
Input: $G = (U, V, E)$: Signed bipartite graph
Output: $c^+$: Balanced butterfly count
        $c^-$: Unbalanced butterfly count
1: Sort $N_G(u)$ of each $u \in U \cup V$ by vertex priority $<_p$
2: $c^+ \leftarrow 0, c^- \leftarrow 0$
3: for each $u \in U \cup V$ do
4:     Create $H_s(u, w)$ and $H_d(u, w)$ for each node $w$ at same side as $u$
5:     for each $v \in N_G(u) : v <_p u$ do
6:         for each $w \in N_G(v) : w <_p u$ do
7:             if $(u, v, w)$ is an s-wedge then
8:                 $H_s(u, w)$.append($v$)
9:             else $H_d(u, w)$.append($v$)
10:    for each node $w : H_s(u, w) + H_d(u, w) > 1$ do
11:        $c^+ += \binom{|H_s(u,w)|}{2} + \binom{|H_d(u,w)|}{2}$
12:        $c^- += |H_s(u, w)||H_d(u, w)|$
```

The framework of Algorithm 2 is similar to that of Algorithm 1, except that for each vertex u, we create two different Hash-maps, namely $H_s(u, w)$ and $H_d(u, w)$, to store s-wedges and d-wedges respectively (Line 4). Specifically, for each wedge (u, v, w) where $v, w <_p u$, we put v into $H_s(u, w)$ if (u, v, w) is an s-wedge, and into $H_d(u, w)$ if (u, v, w) is a d-wedge (Line 5-9). According to Lemma 2, each pair of s-wedges or pair of d-wedges with endpoints at u and w forms a balanced butterfly, while there are $\binom{|H_s(u,w)|}{2} + \binom{|H_d(u,w)|}{2}$ such pairs (Line 11). On the other hand, each pair of one s-wedge and one d-wedge endpoints at u and w forms an unbalanced butterfly, while there are $|H_s(u, w)||H_d(u, w)|$ such pairs (Line 12).

THEOREM 4. *The time-complexity of the Signed-BFC algorithm is* $O\left(\sum_{(u,v)\in E} \min\left\{\deg_G(u), \deg_G(v)\right\}\right)$. *Also, The space-complexity of Signed-BFC is* $O\left(\sum_{(u,v)\in E} \min\left\{\deg_G(u), \deg_G(v)\right\}\right)$.

PROOF. Immediate from Theorem 3 and Corollary 1. □

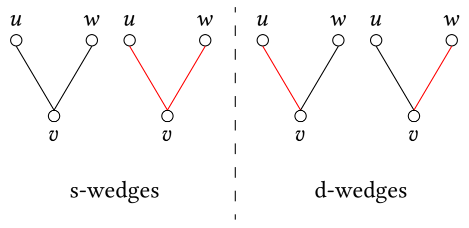


Figure 4: Isomorphism classes of s-wedges and d-wedges.

5.2 Balanced / Unbalanced Supports Maintenance

With our improved balanced and unbalanced butterfly counting algorithm, obtaining the balanced and unbalanced supports of a given edge becomes straightforward, enabling us to efficiently prune edges that do not meet the constraints of a maximal balanced (k, ϵ)-bitruss. Nevertheless, when an edge is removed, updates are required for the edges that participate in at least one butterfly that contains the removed edge. Hence, an efficient management system for tracking the balanced and unbalanced supports of each edge is essential to optimize the overall performance of the algorithm. For the sake of convenience, we adopted the definition of maximal priority-obeyed bloom [40] to indicate the hash maps $H_s(u, w)$ and $H_d(u, w)$ in the Algorithm 2.

DEFINITION 10 (MAXIMAL PRIORITY-OBEYED BLOOM [40]). *In a bipartite graph, a maximal priority-obeyed bloom is a k-bloom* $B = (U_S, V_S)$ *such that*

(1) *If a node* $u \in U_S$ *(or* $v \in V_S$ *respectively) has the greatest vertex priority among all nodes in the bloom, then* $|U_S| = 2$ *(or* $|V_S| = 2$ *respectively) and* $|U_S|$ *(or* $|V_S$ *respectively) is called the dominant node set of B, denoted as* dom(B).
(2) *No other bloom containing B satisfies the condition above.*

Clearly for each pair of two-hop neighbors u and w mentioned in Algorithm 2, $\{u, w\}$ and $H_s(u, w) \cup H_d(u, w)$ form a maximal priority-obeyed bloom. In addition to butterfly counting, the maximal priority-obeyed blooms are also useful to calculate the balanced and unbalanced supports. A maximal priority-obeyed bloom can be viewed as a set of s-wedges and d-wedges. In each such wedge, we say one edge is the twin edge [40] of the other.

DEFINITION 11 (TWIN EDGE [40]). *Given a maximal priority-obeyed k-bloom* $B = (U_S, V_S)$ *and an edge e in the bloom, the twin edge of e in the bloom B, denoted as* twin(B, e)*, is the edge in the bloom that forms a wedge with e where the endpoints are the dominant node set of B.*

The wedge formed by e and twin(B, e) is called a priority-obeyed wedge in B, and there are k priority-obeyed wedges in B.

EXAMPLE 2. *In the signed bipartite graph shown in Figure 3,* $(\{u_0, u_1, u_2, u_3\}, \{v_1, v_2\})$ *is a maximal primary-obeyed bloom, in which the priority-obeyed wedges are* (v_1, u_0, v_2), (v_1, u_1, v_2), (v_1, u_2, v_2), *and* (v_1, u_3, v_2).

LEMMA 3. *Let B be a maximal priority-obeyed bloom in a signed bipartite graph, and e an edge in B. Suppose* n_s *is the number of priority-obeyed s-wedges in B, and* n_d *is the number of priority-obeyed d-wedges in B. Then*

$$\mathrm{Sup}^+_B(e) = \begin{cases} n_s - 1 & \text{if } \mathrm{sign}(e) = \mathrm{sign}(\mathrm{twin}(B, e)) \\ n_d - 1 & \text{if } \mathrm{sign}(e) \neq \mathrm{sign}(\mathrm{twin}(B, e)) \end{cases}$$

$$\mathrm{Sup}^-_B(e) = \begin{cases} n_d & \text{if } \mathrm{sign}(e) = \mathrm{sign}(\mathrm{twin}(B, e)) \\ n_s & \text{if } \mathrm{sign}(e) \neq \mathrm{sign}(\mathrm{twin}(B, e)). \end{cases}$$

PROOF. If sign(e) = sign(twin(B, e)), then the priority-obeyed constructed by e and twin(B, e) is an s-wedge, and there are $n_s - 1$ other priority-obeyed s-wedges in B. According to Lemma 2, each of the $n_s - 1$ s-wedges forms a balanced butterfly with e and $\text{twin}(B, e)$), and each of the n_d priority-obeyed d-wedges in B forms an unbalanced butterfly with e and $\text{twin}(B, e)$).

On the other hand, if $\text{sign}(e) \neq \text{sign}(\text{twin}(B, e))$, then the priority-obeyed constructed by e and $\text{twin}(B, e)$ is a d-wedge, and there are $n_d - 1$ other priority-obeyed d-wedges in B. According to Lemma 2, each of the $n_d - 1$ d-wedges forms a balanced butterfly with e and $\text{twin}(B, e)$, and each of the n_s priority-obeyed s-wedges in B forms an unbalanced butterfly with e and $\text{twin}(B, e)$. □

According to Wang et al. [40], every butterfly in a bipartite graph is included in a single maximal priority-obeyed bloom. As a result, we can use Lemma 3 to determine the balanced support and unbalanced support of an edge e in a signed bipartite graph G, which can be computed as follows: $\text{Sup}^+_G(e) = \sum_{e \in B_i} \text{Sup}^+_{B_i}(e)$ and $\text{Sup}^-_G(e) = \sum e \in B_i \, \text{Sup}^-_{B_i}(e)$.

When an edge is removed from a signed bipartite graph, we must update the balanced and unbalanced supports for any edges that share at least one butterfly with the removed edge. To accelerate this process, we propose Signed Bloom-Edge Index (or SBE index), which is inspired from the BE-Index proposed in the literature [40].

Definition 12 (Signed Bloom-Edge Index). *Given a signed bipartite graph $G = (U, V, E)$, the Signed Bloom-Edge Index of G is a 3-tuple of functions $I = (H_B, H_s, H_d)$, where*

(1) *H_B maps every edge e to the set $\{\text{dom}(B) : e \in B\}$, where B is a priority-obeyed bloom;*
(2) *H_s maps every pair of two-hop neighbors u, w to the set $\{v : (u, v, w)$ is a priority-obeyed s-wedge$\}$; and*
(3) *H_d maps every pair of two-hop neighbors u, w to the set $\{v : (u, v, w)$ is a priority-obeyed d-wedge$\}$.*

All three functions can be implemented by hash-maps. Indeed H_s and H_d are already constructed in Algorithm 2. We extend it to complete the SBE index and calculate balanced supports and unbalanced supports, as described in Algorithm 3. For simplicity, we use $e_{u,v}$ to denote the edge induced by u and v.

Line 4-9 of Algorithm 3 follows the framework of the construction of the BE-Index [40], but in addition, we need to distinguish s-wedges and d-wedges during iteration and handle them respectively. Also, note that Line 12-17 of Algorithm 3 is an application of Lemma 3.

Theorem 5. *The time-complexity of the SBE-Index-Construction is $O\left(\sum_{(u,v) \in E} \min\{\deg_G(u), \deg_G(v)\}\right)$. Also, The space-complexity of SBE-Index-Construction is $O\left(\sum_{(u,v) \in E} \min\{\deg_G(u), \deg_G(v)\}\right)$.*

Proof. In addition to the framework of Algorithm 2, the balanced / unbalanced support updates in Line 12-21 is a priority-obeyed wedge iteration in different pattern, which takes time complexity of $O\left(\sum_{(u,v) \in E} \min\{\deg_G(u), \deg_G(v)\}\right)$, as discussed in the proof of Theorem 3. The space complexity is immediate from Corollary 1. □

After the construction of the SBE index, we can efficiently update the supports caused by edge removal. Algorithm 4 elaborate the process to remove an edge, in which we use $B_{u,w}$ to denote the maximal priority-obeyed bloom with $\text{dom}\, B_{u,w} = \{u, w\}$.

Algorithm 3 SBE-Index-Construction

```
Input:  $G = (U, V, E)$: Signed bipartite graph
Output: $c^+$ : Balanced butterfly count
        $c^-$ : Unbalanced butterfly count
        $\text{Sup}^+_G$ : balanced support of every edge
        $\text{Sup}^-_G$ : unbalanced support of every edge
        $I = (H_B, H_s, H_d)$: SBE Index
 1: Line 1-2 of Algorithm 2
 2: for each $e \in E$ do
 3:     $\text{Sup}^+_G(e) \leftarrow 0, \text{Sup}^-_G(e) \leftarrow 0$
 4: for each $u \in U \cup V$ do
 5:     for each $v \in N_G(u) : v <_p u$ do
 6:         for each $w \in N_G(v) : w <_p u$ do
 7:             $H_B(e_{u,v})$.append($(u, w)$)
 8:             $H_B(e_{w,v})$.append($(u, w)$)
 9:             Line 7-9 of Algorithm 2
10:     for each node $w : H_s(u, w) + H_d(u, w) > 1$ do
11:         Line 11-12 of Algorithm 2
12:         for each node $v \in H_s(u, w)$ do
13:             $\text{Sup}^+_G(e_{u,v}), \text{Sup}^+_G(e_{w,v}) += |H_s(u, w)| - 1$
14:             $\text{Sup}^-_G(e_{u,v}), \text{Sup}^-_G(e_{w,v}) += |H_d(u, w)|$
15:         for each node $v \in H_d(u, w)$ do
16:             $\text{Sup}^+_G(e_{u,v}), \text{Sup}^+_G(e_{w,v}) += |H_d(u, w)| - 1$
17:             $\text{Sup}^-_G(e_{u,v}), \text{Sup}^-_G(e_{w,v}) += |H_s(u, w)|$
```

Algorithm 4 Remove-Edge

```
Input:  $G = (U, V, E)$: Signed bipartite graph
        $e$ : the edge to be removed
 1: for each $(u, w) \in H_B(e)$ do
 2:     if sign($e$) = sign(twin($B_{u,w}, e$)) then
 3:         for each $v \in H_s(u, w)$ do
 4:             if $e = e_{u,v} \vee e = e_{w,v}$ then
 5:                 Remove $v$ from $H_s(u, w)$
 6:             else $\text{Sup}^+_G(e_{u,v}), \text{Sup}^+_G(e_{w,v}) -= 1$
 7:         for each $v \in H_d(u, w)$ do
 8:             $\text{Sup}^-_G(e_{u,v}), \text{Sup}^-_G(e_{w,v}) -= 1$
 9:         $\text{Sup}^+_G(\text{twin}(B_{u,w}, e)) -= |H_s(u, w)|$
10:         $\text{Sup}^-_G(\text{twin}(B_{u,w}, e)) -= |H_d(u, w)|$
11:     else
12:         for each $v \in H_d(u, w)$ do
13:             if $e = e_{u,v} \vee e = e_{w,v}$ then
14:                 Remove $v$ from $H_d(u, w)$
15:             else $\text{Sup}^+_G(e_{u,v}), \text{Sup}^+_G(e_{w,v}) -= 1$
16:         for each $v \in H_s(u, w)$ do
17:             $\text{Sup}^-_G(e_{u,v}), \text{Sup}^-_G(e_{w,v}) -= 1$
18:         $\text{Sup}^+_G(\text{twin}(B_{u,w}, e)) -= |H_d(u, w)|$
19:         $\text{Sup}^-_G(\text{twin}(B_{u,w}, e)) -= |H_s(u, w)|$
20:     Remove $(u, w)$ from $H_B(\text{twin}(B_{u,w}, e))$
21: Remove $e$
```

In Algorithm 4, we iterate every maximal priority-obeyed bloom $B_{u,w}$ containing e using the SBE index (Line 1). According to Lemma 2, if e and its twin edge in $B_{u,w}$ form an s-wedge (Line 2), then each edge from other s-wedges in $B_{u,w}$ lose one balanced support (Line 6) and each edge from d-wedges in $B_{u,w}$ should lose one unbalanced support (Line 8); if e and its twin edge in $B_{u,w}$ form a d-wedge (Line 11), then each edge from other d-wedges in $B_{u,w}$ lose one balanced support (Line 15) and each edge from d-wedges in $B_{u,w}$ should lose one unbalanced support (Line 17). Since e is removed from $B_{u,w}$, so is its twin (Line 20). Moreover, twin$(B_{u,w}, e)$ loses all butterfly supports from $B_{u,w}$, so its balanced support and unbalanced support decrement as indicated in Lemma 3 (Line 9-10, 18-19).

Theorem 6. *The time-complexity of the RemoveEdge algorithm is* $O(\mathrm{Sup}_G(e))$.

Proof. Algorithm 4 iterates every butterfly containing e exactly once, and operations of constant time are performed in each iteration. Thus, the overall time-complexity of the RemoveEdge algorithm is $O(\mathrm{Sup}_G(e))$. □

5.3 Pruning Unpromising Edges

With the supports update mechanism due to edge removal, we can now propose a pruning strategy that removes those edges that are certainly not part of a balanced (k, ϵ)-bitruss, for given k and ϵ.

Definition 13 (Pruned (k, ϵ)-bitruss). *Given a signed bipartite graph $G = (U, V, E)$, let $P(G)$ be a subgraph of G such that*

(1) *for every edge e in G_P we have $\mathrm{Sup}_{P(G)}(e) \geq k$ and $\mathrm{Sup}^+_{P(G)}(e) \geq k(1-\epsilon)$;*
(2) *no other subgraph containing G_P satisfies the condition above.*

Then $P(G)$ is called the pruned (k, ϵ)-bitruss of G.

Lemma 4. *The pruned (k, ϵ)-bitruss of G is unique.*

Proof. Suppose for contradiction that there are two distinct subgraphs $P_1(G), P_2(G)$ of G satisfying both conditions. Consider H', the union graph of $P_1(G)$ and $P_2(G)$. For an edge e in H', e is in either $P_1(G)$ or $P_2(G)$. If e is in $P_1(G)$, then $\mathrm{Sup}_{H'}(e) \geq \mathrm{Sup}_{P_1(G)}(e) \geq k$ and $\mathrm{Sup}^+_{H'}(e) \geq \mathrm{Sup}^+_{P_1(G)}(e) \geq k(1-\epsilon)$; if e is in $P_2(G)$, then $\mathrm{Sup}_{H'}(e) \geq \mathrm{Sup}_{P_2(G)}(e) \geq k$ and $\mathrm{Sup}^+_{H'}(e) \geq \mathrm{Sup}^+_{P_2(G)}(e) \geq k(1-\epsilon)$. Thus, H' is a nontrivial supergraph of H_1 that satisfies the first condition, which contradicts to the second condition of H_1. □

We can obtain $P(G)$ by iteratively removing the edges e from G where $\mathrm{Sup}_G(e) < k$ and $\mathrm{Sup}^+_G(e) < k(1-\epsilon)$ by Algorithm 4, until no such edges remain. Indeed, the purpose to have H is that H has the same maximum balanced (k, ϵ)-bitruss as G.

Lemma 5. *Given a signed bipartite graph $G = (U, V, E)$, and G_P be its pruned (k, ϵ)-bitruss, then a balanced (k, ϵ)-bitruss is a subgraph of G_P.*

Proof. Let I be a balanced (k, ϵ)-bitruss of G, and e an edge of I. By definition we have $\mathrm{Sup}_I(e) \geq k$ and $\mathrm{Sup}^-_I(e)/\mathrm{Sup}_I(e) \leq \epsilon$. Namely,

$$\frac{\mathrm{Sup}^-_I(e)}{\mathrm{Sup}^+_I(e)+\mathrm{Sup}^-_I(e)} = 1 - \frac{\mathrm{Sup}^+_I(e)}{\mathrm{Sup}^+_I(e)+\mathrm{Sup}^-_I(e)} \leq \epsilon$$
$$\Rightarrow \frac{\mathrm{Sup}^+_I(e)}{\mathrm{Sup}^+_I(e)+\mathrm{Sup}^-_I(e)} \geq 1-\epsilon$$
$$\Rightarrow \frac{\mathrm{Sup}^+_I(e)}{\mathrm{Sup}^+_I(e)+\mathrm{Sup}^-_I(e)} \cdot \mathrm{Sup}_I(e) = \mathrm{Sup}^+_I(e) \geq k(1-\epsilon).$$

Hence I is a subgraph of G_P. □

By removing edges from G that do not meet the conditions of $P(G)$, we can construct a smaller graph that is computationally efficient and the maximum balanced (k, ϵ)-bitruss still remains unchanged.

5.4 Exact Approach

We first propose an exact algorithm for our problem. Based on the greedy framework presented in Algorithm 5 and partition by butterfly-connected components, the exact algorithm explores the search space of all possible subgraphs of the given signed bipartite graph by branch-and-bound, and prunes subgraphs that are guaranteed not to be the maximum balanced (k, ϵ)-bitruss. Although the exact algorithm provides the optimal solution to our problem, its running time is exponential in the worst case.

5.5 Greedy Heuristic Approaches

To address the computational complexity of the maximum balanced (k, ϵ)-bitruss problem, we propose a framework of greedy heuristic approach shown in Algorithm 5. The algorithm begins by constructing the signed BE-index of the signed bipartite graph G (Line 1). Next, we prune the unpromising edges as discussed in Lemma 13 and Lemma 5 (Line 2-3). We use the notation E_C to represent the set of edges that do not satisfy the condition of a balanced (k, ϵ)-bitruss (Line 4). The algorithm then iteratively selects the best edge to remove (Line 6-7), updates E_C and E based on Lemma 13 and Lemma 5 again (Line 8-9), and repeats this process until $E_C = \varnothing$ (Line 5), meaning that all remaining edges satisfy the balanced (k, ϵ)-bitruss condition.

Algorithm 5 Greedy

```
Input:  $G = (U, V, E)$: Signed bipartite graph
        $k$: support constraint
        $\epsilon$: balanced supports ratio constraints
Output: The maximum balanced $(k, \epsilon)$-bitruss
 1: SBE-Index-Construction (Algorithm 3) with $G$
 2: while $\exists e \in E, \mathrm{Sup}_G(e) < k$ or $\mathrm{Sup}^+_G(e) < k(1-\epsilon)$ do
 3:     Remove-Edge (Algorithm 4) with $G, e$
 4: $E_C \leftarrow \{e \in E : \mathrm{Sup}^-_G(e)/\mathrm{Sup}_G(e) > \epsilon\}$
 5: while $E_C \neq \varnothing$ do
 6:     $e \leftarrow$ the best edge
 7:     Remove-Edge (Algorithm 4) with $G, e$
 8:     Prune $E$ by Line 2-3 of Algorithm 5
 9:     Update $E_C$
10: return $G$
```

We next present two strategies to choose the best edge to remove.

5.5.1 Greedy Heuristic By Followers. When an edge is removed, it may caused other edges to become unpromising edges as indicated in Lemma 13 and Lemma 5. Referring to the literature [34], these edges are called the *followers* of the removed edge.

Definition 14 (Followers). *Given a signed bipartite graph $G = (U, V, E)$, let $H = P(G)$ be the pruned bitruss of G. Then the followers of an edge e is defined as $F_H(e) := E_H \backslash P(H \backslash \{e\})$, where E_H is the set of edges in H, and $H \backslash \{e\}$ is the graph H with e removed.*

Hence to determine the best edge for removal in Line 6 of Algorithm 5, it is reasonable to choose the edge in E_C with the minimum number of followers. However, identifying this edge can be computationally intensive. In fact, it may take up to $O(|E| \left(\sum_{e \in E} \operatorname{Sup}_G(e)\right)$ time, as we need to simulate an edge removal for every edge, and in each removal simulation, we must remove the pruned edges in cascade. The worst-case scenario occurs when all edges are pruned.

The bottleneck to identify the edge with the least followers is that the number of followers of every edge is calculated by edge removal simulation. To overcome this bottleneck, we propose two techniques to terminate the algorithm early. Firstly, record the current minimum number of followers $f_{\min}$. If an edge is found having more than $f_{\min}$ followers, then this edge cannot have the fewer followers than $f_{\min}$ and its followers counting can stop earlier. Secondly, if the minimum number of followers for any edge e' becomes 1, as indicated by $f_{\min} = 1$, then the edge e' must have the minimum number of followers. This is because any edge must have at least one follower, which is itself. In this case, the followers counting can terminate without considering the followers of the remaining edges.

5.5.2 Greedy Heuristic By Ratio of Balanced Supports. Apart from the greedy heuristic strategy by followers, we propose an alternative approach that is greedy by balanced supports ratio. Instead of the edge with minimum number of followers, this strategy takes the edge e with maximum value of $\operatorname{Sup}_G^-(e)/\operatorname{Sup}_G(e)$ as the best edge to remove in Line 6 of Algorithm 5.

Compared to the greedy heuristic by followers, the greedy heuristic by balanced supports ratio takes a different perspective on best edge selection. The most noticeable advantage of the greedy heuristic strategy by balanced supports ratio is that is only takes up to $O(|E|)$ running time to identify the best edge to remove.

5.5.3 Complexities of Greedy Heuristics.

Theorem 7. *The time-complexity of the Greedy framework is $O\left(\sum_{(u,v) \in E} \min\{\deg_G(u), \deg_G(v)\} + \operatorname{Sup}_G((u,v)) + S\right)$, where S is the time-complexity to select the best edge to remove in each iteration.*

Proof. Theorem 3 states that the SBE-index construction takes $O\left(\sum_{(u,v) \in E} \min\{\deg_G(u), \deg_G(v)\}\right)$ time, and Theorem 4 shows that edge removal takes $O(\operatorname{Sup}_G(e))$ time, where each edge can be removed at most once. Meanwhile, note that the best edge selection can occur at most $O(|E|)$ times. Thus, the theorem holds. □

5.6 Selecting k and ϵ

When applying the maximum balanced (k, ϵ)-bitruss model to analyze signed bipartite graphs, choosing the right values for k and ϵ is critical. Users should consider the trade-off between cohesive community detection and comprehensiveness based on user goals.

5.6.1 Effect of k on Cohesion and Community Size. The parameter k dictates the minimum number of balanced butterflies an edge must participate in within the resulting subgraph. Higher k values emphasize tighter connections, resulting in smaller, more cohesive communities within the graph. Additionally, the graph's density plays a role; dense graphs may require larger k values to capture cohesive substructures [47].

5.6.2 Effect of ϵ on Tolerance for Unbalanced Structures. The parameter ϵ represents the proportion of unbalanced butterflies the subgraph can contain. Lower ϵ values (e.g., 0.3 or lower) prioritize balanced structures, yielding subgraphs with fewer unbalanced elements. Higher ϵ values allow for more tolerance of unbalanced structures, potentially including diverse or noisy patterns.

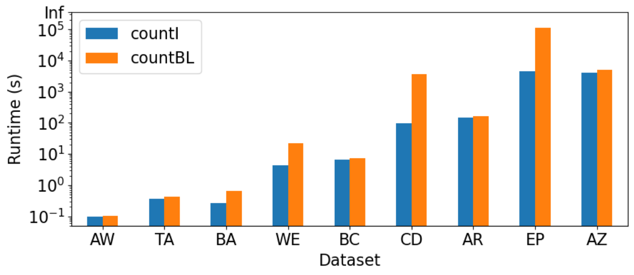


Figure 5: Performance of Counting Algorithms

6 EXPERIMENTAL STUDY

6.1 Experiment Setup

Since the balanced (k, ϵ)-bitruss is a new problem, while current bitruss decomposition approaches [47][40] does not consider signs and balanced butterflies, we evaluate the following algorithms, in the experiments:

- `countBL`: Algorithm 1, the baseline signed butterfly counting.
- `countI`: Algorithm 2, our improved signed butterfly counting.
- `Exact`: Our exact solution mentioned in Section 5.4.
- `GreedyF`: Algorithm 5, the greedy heuristic maximum balanced (k, ϵ)-bitruss searching algorithm that regards the best edge as the edge with the fewest followers, introduced in Section 5.5.1.
- `GreedyS`: Algorithm 5, regarding the best edge as the edge with highest unbalanced support ratio, introduced in Section 5.5.2.
- `Random`: Algorithm 5 but randomly selecting an edge in E_C as the best edge. For each tested k and ϵ in each dataset, we repeat this algorithm 10 times and record the average runtime and result sizes. Since E_C varies from time to time, it takes $O(|E_C|)$ time to select a random edge in E_C in our implementation.

Table 2 lists the details of the 9 real-world datasets we consider. Dataset Bonanza[2] and WikiElec[3] are native signed bipartite graphs, while others[4][5] are rating networks which can also be regarded as

[2] http://www.bonanza.com
[3] http://snap.stanford.edu
[4] http://konect.cc
[5] http://jmcauley.ucsd.edu/data/amazon/

Table 2: Statistics of Datasets. “Balanced” is short for balanced butterflies and “Unbalanced” is short for unbalanced butterflies.

Dataset	$\|U\|$	$\|V\|$	$\|E^+\|$	$\|E^-\|$	Index Size	Avg. deg	k	Balanced	Unalanced
AmazonWang (AW)	26,112	799	22,900	6,001	0.661 MB	2.148	3	2,986	589
Bonanza (BA)	7,919	1,973	35,805	738	6.177 MB	7.388	20	641,108	30,785
TripAdvisor (TA)	145,316	1,759	151,340	24,315	8.489 MB	2.389	3	8,373	2,954
WikiElec (WE)	2,384	6,129	81,378	22,369	233.2 MB	24.374	120	26,277,607	9,767,363
BookCrossing (BC)	77,802	185,955	363,250	70,402	327.8 MB	3.288	6	1,112,973	344,357
CiaoDVD (CD)	21,019	71,633	1,518,033	107,447	1684 MB	35.088	200	5,581,587,857	154,596,407
AmazonRating (AR)	2,146,057	1,230,915	4,954,292	788,966	6443 MB	3.401	6	27,355,430	8,493,874
Epinions (EP)	120,492	755,760	13,348,412	319,908	18393 MB	31.197	200	169,361,642,036	942,128,969
Amazon (AZ)	21,176,522	9,874,211	71,699,784	10,977,347	145606 MB	5.325	12	714,670,270	151,078,721

signed networks [21]. For these rating data, we set high ratings (i.e., 3 and above out of 5) as positive and low rating as negative. The total numbers of signed butterflies are shown in the **Balanced** column and the **Unbalanced** column, and the sizes of the SBE index are shown in the Index Size column. In the dataset Epinions, even though there are over 170 billion butterflies, the size of the SBE index does not exceed 20 GB.

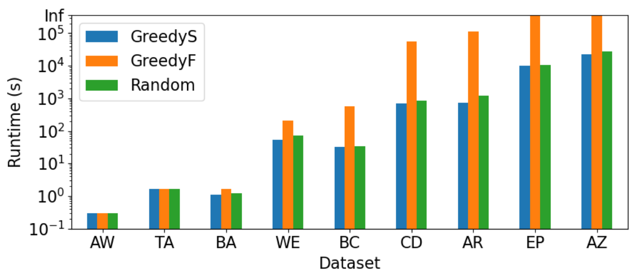
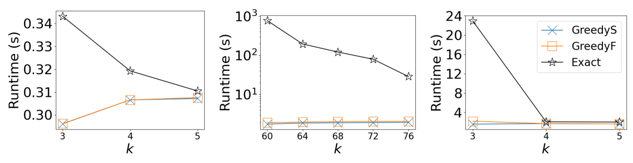
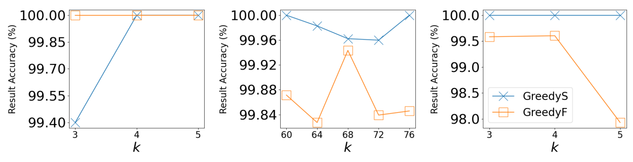


Figure 6: Performance of Greedy Heuristics Algorithms with Default Parameters



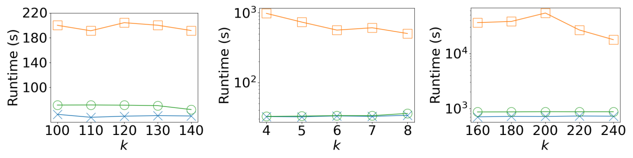
(a) AW (b) BA (c) TA

Figure 7: Performance Compared to Exact Solutions

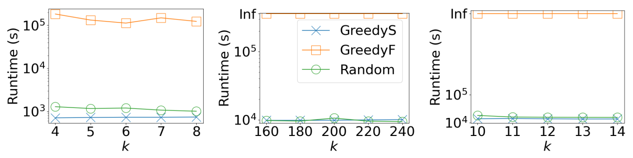


(a) AW (b) BA (c) TA

Figure 8: Effectiveness Compared to Exact Solutions

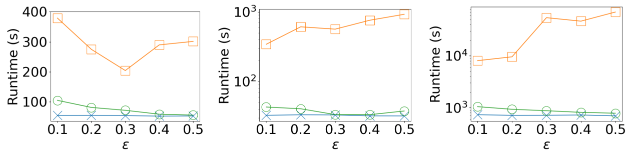


(a) WE (b) BC (c) CD

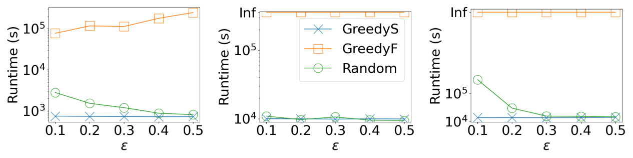


(d) AR (e) EP (f) AZ

Figure 9: Performance of GreedyS and GreedyF Varying k



(a) WE (b) BC (c) CD

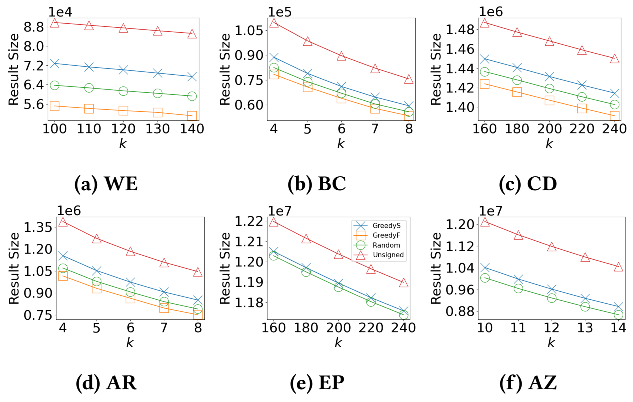


(d) AR (e) EP (f) AZ

Figure 10: Performance of GreedyS and GreedyF Varying ϵ

Table 3: Parameter Selections on WE: Varying k, $\epsilon = 0.3$

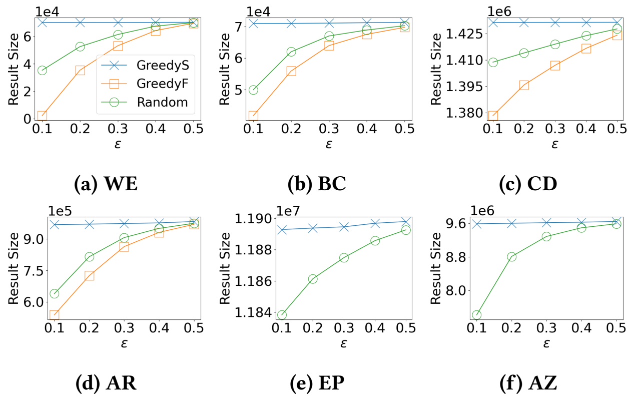
k	$\|E^+\|$	$\|E^-\|$	Balanced	Unbalanced	Avg. deg
15	74,542	10,234	22,783,447	3,035	32.94
30	73,036	9,387	22,745,255	2,211	35.93
60	70,184	8087	22,608,951	2,761	40.19
120	64,231	5,852	22,022,821	0	43.95
180	57,850	4,312	21,060,828	0	46.30
240	50,729	3,152	19,683,048	0	47.59



(a) WE (b) BC (c) CD

(d) AR (e) EP (f) AZ

Figure 11: Effectiveness of GreedyS and GreedyF Varying k



(a) WE (b) BC (c) CD

(d) AR (e) EP (f) AZ

Figure 12: Effectiveness of GreedyS and GreedyF Varying ϵ

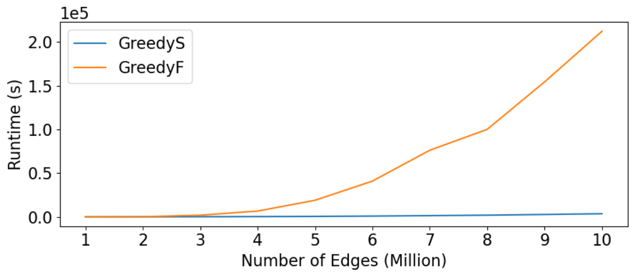


Figure 13: Scalability Test on Epinions with $k = 4$, $\epsilon = 0.3$

Table 4: Parameter Selections on WE: Varying ϵ, $k = 120$

ϵ	$\lvert E^+\rvert$	$\lvert E^-\rvert$	Balanced	Unbalanced	Avg. deg
0	64,378	5,817	22,025,322	0	43.91
0.2	64,343	5,847	22,024,512	0	43.94
0.4	64,399	5,856	22,043,184	9,867	43.99
0.5	64,425	5,903	22,054,310	17,481	44.02
0.6	67743	5852	22,269,633	387,069	44.46
0.8	70,130	16,504	25,701,195	9,005,633	48.59
1	70,471	17,036	22,022,821	9,574,670	48.62

Experiments of maximum balanced (k, ϵ)-bitruss searching algorithms are conducted by varying parameters k and ϵ. For each dataset, the default value of k (shown in column k) are decided depending on the density, which can be reflected by the average degree (shown in column Avg. deg). All algorithms are implemented[6] in C++11 and performed on a server with Intel Xeon Gold 6240R CPU @2.40GHz and 1TB of memory. If an experiment does not terminate in 100 hours, we terminate it and denote its runtime as Inf.

6.2 Performance Evaluation of Signed Butterfly Counting

Figure 5 reports the runtime of signed butterfly counting algorithms countBL and countI. The results showed that count outperforms countBL on all datasets, and the difference is larger for denser graphs. Dataset CD and EP, which are the densest, showed a runtime difference of more than 10 times. These findings suggest that the countI algorithm is more efficient and scalable, especially for larger and denser graphs. The efficiency of signed butterfly counting plays a vital role in our maximum balanced (k, ϵ)-bitruss searching algorithm. On dataset EP, for instance, the process of signed butterfly counting occupies about 40% total runtime of GreedyS.

6.3 Evaluation of the Exact Solution

Due to the high time complexity of the Exact, we evaluate the effectiveness and performance of Exact, GreedyS, and GreedyF algorithms only on the three small datasets (AW, BA, and TA) with specific values for k and ϵ. According to Figure 8, GreedyS and GreedyF had result sizes very close to the exact result size, but GreedyS was usually closer. This indicates that the greedy heuristic algorithms usually provide near-optimal solutions on real datasets. Figure 7 displayed the runtime results of Exact, GreedyS, and GreedyF on the three datasets, with varying k and the same ϵ as in Figure 8. Unlike Exact, which has exponential time complexity, GreedyS and GreedyF had much lower time costs, further highlighting their efficiency and practicality in real-world scenarios.

6.4 Evaluation of Greedy Heuristic Solutions

The runtime of algorithms GreedyS, GreedyF, and Random on 6 datasets with default k and $\epsilon = 0.3$ are visible in Figure 6. The results are consistent to our analysis that GreedyS has lower time complexity than GreedyF. Due to the additional time needed to compute followers of edges on each iteration, GreedyF is much slower in large and dense dataset, and does not terminate in 100 hours on the EP and AZ datasets. While both GreedyS and Random algorithms share the same time complexity, it is worth noting that GreedyS tends to outperform Random in terms of runtime. This efficiency can be attributed to the fact that GreedyS prioritizes the removal of edges associated with a high proportion of unbalanced butterflies, leading to a more rapid reduction in the size of E_C.

6.4.1 Evaluation of Greedy Heuristics Algorithms with Varying k. The evaluation of algorithms GreedyS and GreedyF with varying k and $\epsilon = 0.3$ on six datasets is shown in Figure 9 and Figure 11. Increasing k signifies denser communities, resulting in smaller

[6]https://github.com/qixiaoz/BalancedBitruss

result sizes for both algorithms across all datasets due to fewer edges satisfying support requirements. Higher k also improves our pruning technique's efficiency, reducing runtime. In Figure 11, we display the edge sizes of the unsigned k-bitruss for various k values, significantly larger than the signed balanced results, emphasizing the distinction between the two by excluding edges from unbalanced environments. For the EP dataset, `GreedyF` did not complete due to time constraints, suggesting computational challenges with larger datasets. However, on the remaining four datasets, `GreedyF` consistently performed the worst, indicating that discarding edges without considering unbalanced butterflies is not a prudent strategy.

6.4.2 *Evaluation of Greedy Heuristics Algorithms with Varying ϵ.*

Algorithm performance (including `GreedyS`, `GreedyF`, and `Random`) is assessed with varying ϵ on six datasets, as depicted in Figure 10 and Figure 12. `GreedyS` maintains stable runtime with changing ϵ, whereas `Random` experiences increased processing times with decreasing ϵ. Conversely, `GreedyF` generally exhibits increased runtime and result sizes as ϵ rises, attributed to edge appearance and disappearance due to unbalanced support changes. Lower ϵ values cause `GreedyF` to have smaller results than `GreedyS`. Result fluctuations in `GreedyF` are influenced by edge order for follower computation and edges with just one follower, crucial for `GreedyF`'s early termination.

6.5 Scalability Test

In this section, we conduct a scalability test to assess the runtime performance `GreedyS` and `GreedyF`, on the EP dataset. We run these two algorithms on the subgraphs by uniformly sampling the edges from EP by varying edge size from 1M to 10M, and keeping $k = 4$ and $\epsilon = 0.3$. The results in Figure 13 clearly demonstrate that `GreedyS` exhibits superior scalability compared to `GreedyF`.

6.6 Case Study

The TripAdvisor dataset (TA) is a compilation of a rating network extracted from TripAdvisor, where users can provide positive or negative ratings to hotels. In a case study involving a subset of this network, our objective is to validate the effectiveness of the maximum balanced (k, ϵ)-bitruss analysis. Using the `GreedyS` algorithm, we extract a balanced $(8, 0.3)$-bitruss, revealing a cohesive and stable community composed of users (lower layer) and rated hotels (upper layer). Within this community, each user-hotel rating is balanced within its nested group, as depicted in the left portion of Figure 14. Notably, our result contains no unbalanced butterflies. However, if we were to seek an unsigned 8-bitruss instead, the result would include 110 more ratings and 153 unbalanced butterflies, rendering the discovered community less stable. The right side of Figure 14 displays balanced bicliques containing at least two users within the same subgraph. These bicliques, however, have the effect of fragmenting the community and causing certain edges within it to be overlooked.

6.7 Evaluation of Parameter Selections

To understand how different values of k and ϵ impact outcomes, we analyzed the WE dataset using the `GreedyS` algorithm, exploring a range of k and ϵ values. The results, summarized in Table 3 and Table 4, provide valuable insights.

Table 3 shows that increasing k raises the average degree but reduces subgraph size. This indicates a preference for tightly connected edges as k increases. In Table 4, we observe that for ϵ values up to 0.5, unbalanced butterflies are minimal compared to balanced ones. However, beyond $\epsilon = 0.5$, unbalanced butterflies increase notably. Therefore, we recommend $\epsilon \leq 0.3$ and advise determining k based on graph density and specific user requirements.

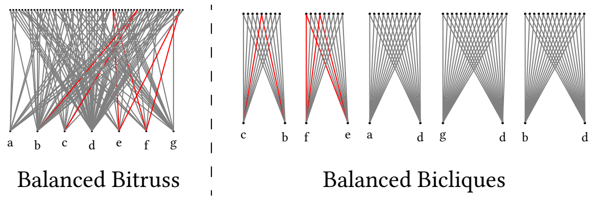


Figure 14: A Case Study on TripAdvisor Rating Network.

7 CONCLUSION

In this work, we formally define balanced (k, ϵ)-bitruss and the maximum balanced (k, ϵ)-bitruss search problem. We proved that this problem is NP-hard and that any nontrivial approximation solution to this problem is also NP-hard. To tackle this challenge, we developed novel strategies to speed up the counting of balanced and unbalanced butterflies and their updates after edge removals. We proposed two greedy heuristic algorithms, `GreedyS` and `GreedyF`. Our experimental results showed that `GreedyS` is more efficient and generates better results, while `GreedyF` focuses on the followers of edges instead of the unbalanced butterflies, making it a relatively shortsighted strategy. We conducted experiments on real-world datasets and found that our two greedy heuristic algorithms generate results that are very close to the exact results but with much less runtime. Therefore, we conclude that `GreedyS` is the preferred solution for our problem.

ACKNOWLEDGMENTS

Lei Chen's work is partially supported by National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16213620, CRF Project C2004-21GF, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Microsoft Research Asia Collaborative Research Grant and HKUST-Webank joint research lab grants. Yue Wang (corresponding author) is partially supported by China NSFC (No.62002235).

REFERENCES

[1] Aman Abidi, Lu Chen, Chengfei Liu, and Rui Zhou. 2022. On Maximising the Vertex Coverage for Top-k t-Bicliques in Bipartite Graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2346–2358.

[2] Sinan G Aksoy, Tamara G Kolda, and Ali Pinar. 2017. Measuring and modeling bipartite graphs with community structure. *Journal of Complex Networks* 5, 4 (2017), 581–603.

[3] Pranay Anchuri and Malik Magdon-Ismail. 2012. Communities and balance in signed networks: A spectral approach. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 235–242.

[4] Tibor Antal, Pavel L Krapivsky, and Sidney Redner. 2005. Dynamics of social balance on networks. *Physical Review E* 72, 3 (2005), 036121.

[5] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*. 119–130.

[6] Dorwin Cartwright and Frank Harary. 1956. Structural balance: a generalization of Heider's theory. *Psychological review* 63, 5 (1956), 277.

[7] Yulin Che, Zhuohang Lai, Shixuan Sun, Yue Wang, and Qiong Luo. 2020. Accelerating truss decomposition on heterogeneous processors. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1751–1764.

[8] Chen Chen, Yanping Wu, Renjie Sun, and Xiaoyang Wang. 2021. Maximum signed θ-clique identification in large signed graphs. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[9] Chen Chen, Qiuyu Zhu, Yanping Wu, Renjie Sun, Xiaoyang Wang, and Xijuan Liu. 2022. Efficient critical relationships identification in bipartite networks. *World Wide Web* 25, 2 (2022), 741–761.

[10] Zi Chen, Long Yuan, Xuemin Lin, Lu Qin, and Jianye Yang. 2020. Efficient maximal balanced clique enumeration in signed networks. In *Proceedings of The Web Conference 2020*. 339–349.

[11] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16, 3.1 (2008), 1–29.

[12] Tyler Derr, Charu Aggarwal, and Jiliang Tang. 2018. Signed network modeling based on structural balance theory. In *Proceedings of the 27th ACM international conference on information and knowledge management*. 557–566.

[13] Tyler Derr, Cassidy Johnson, Yi Chang, and Jiliang Tang. 2019. Balance in signed bipartite networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1221–1230.

[14] Tyler Derr and Jiliang Tang. 2018. Congressional vote analysis using signed networks. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1501–1502.

[15] Ming Gao, Ee-Peng Lim, David Lo, and Philips Kokoh Prasetyo. 2016. On detecting maximal quasi antagonistic communities in signed graphs. *Data Mining and Knowledge Discovery* (2016).

[16] Christos Giatsidis, Bogdan Cautis, Silviu Maniu, Dimitrios M Thilikos, and Michalis Vazirgiannis. 2014. Quantifying trust dynamics in signed graphs, the s-cores approach. In *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 668–676.

[17] Yang Hao, Mengqi Zhang, Xiaoyang Wang, and Chen Chen. 2020. Cohesive subgraph detection in large bipartite networks. In *32nd International Conference on Scientific and Statistical Database Management*. 1–4.

[18] Fritz Heider. 1946. Attitudes and cognitive organization. *The Journal of psychology* 21, 1 (1946), 107–112.

[19] Zan Huang. 2010. Link prediction based on graph topology: The predictive value of generalized clustering coefficient. *Available at SSRN 1634014* (2010).

[20] Junghoon Kim, Hyun Ji Jeong, Sungsu Lim, and Jungeun Kim. 2023. Effective and efficient core computation in signed networks. *Information Sciences* 634 (2023), 290–307.

[21] Wataru Kudo, Mao Nishiguchi, and Fujio Toriumi. 2020. Gcnext: graph convolutional network with expanded balance theory for fraudulent user detection. *Social Network Analysis and Mining* 10 (2020), 1–12.

[22] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web*. 641–650.

[23] Rong-Hua Li, Qiangqiang Dai, Lu Qin, Guoren Wang, Xiaokui Xiao, Jeffrey Xu Yu, and Shaojie Qiao. 2018. Efficient signed clique search in signed networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 245–256.

[24] Pedro G Lind, Marta C Gonzalez, and Hans J Herrmann. 2005. Cycles and clustering in bipartite networks. *Physical review E* 72, 5 (2005), 056127.

[25] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2019. Efficient (α, β)-core computation: An index-based approach. In *The World Wide Web Conference*. 1130–1141.

[26] Wensheng Luo, Kenli Li, Xu Zhou, Yunjun Gao, and Keqin Li. 2022. Maximum Biplex Search over Bipartite Graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 898–910.

[27] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2020. Maximum biclique search at billion scale. *Proceedings of the VLDB Endowment* (2020).

[28] Julian O Morrissette. 1958. An experimental study of the theory of structural balance. *Human Relations* 11, 3 (1958), 239–254.

[29] Garry Robins and Malcolm Alexander. 2004. Small worlds among interlocking directors: Network structure and distance in bipartite graphs. *Computational & Mathematical Organization Theory* 10 (2004), 69–94.

[30] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. 2018. Butterfly counting in bipartite networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2150–2159.

[31] Aida Sheshbolouki and M Tamer Özsu. 2022. sGrapp: Butterfly approximation in streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 16, 4 (2022), 1–43.

[32] Jessica Shi and Julian Shun. 2020. Parallel algorithms for butterfly computations. In *Symposium on Algorithmic Principles of Computer Systems*. SIAM, 16–30.

[33] Kelvin Sim, Jinyan Li, Vivekanand Gopalkrishnan, and Guimei Liu. 2006. Mining maximal quasi-bicliques to co-cluster stocks and financial ratios for value investment. In *Sixth International Conference on Data Mining (ICDM'06)*. IEEE, 1059–1063.

[34] Renjie Sun, Chen Chen, Xiaoyang Wang, Ying Zhang, and Xun Wang. 2020. Stable community detection in signed social networks. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2020), 5051–5055.

[35] Renjie Sun, Yanping Wu, Chen Chen, Xiaoyang Wang, Wenjie Zhang, and Xuemin Lin. 2022. Maximal balanced signed biclique enumeration in signed bipartite graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1887–1899.

[36] Renjie Sun, Qiuyu Zhu, Chen Chen, Xiaoyang Wang, Ying Zhang, and Xun Wang. 2020. Discovering cliques in signed networks based on balance theory. In *Database Systems for Advanced Applications: 25th International Conference, DASFAA 2020, Jeju, South Korea, September 24–27, 2020, Proceedings, Part II 25*. Springer, 666–674.

[37] Vida Vukašinović, Jurij Šilc, and Risth Škrekovski. [n.d.]. Modeling acquaintance networks based on balance theory. *International Journal of Applied Mathematics and Computer Science* 24, 3 ([n. d.]), 683–696.

[38] Jia Wang, Ada Wai-Chee Fu, and James Cheng. 2014. Rectangle counting in large bipartite graphs. In *2014 IEEE International Congress on Big Data*. IEEE, 17–24.

[39] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex Priority Based Butterfly Counting for Large-scale Bipartite Networks. *PVLDB* (2019).

[40] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient bitruss decomposition for large-scale bipartite graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 661–672.

[41] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2023. Accelerated butterfly counting with vertex priority on bipartite graphs. *The VLDB Journal* 32, 2 (2023), 257–281.

[42] Yanping Wu, Renjie Sun, Chen Chen, Xiaoyang Wang, and Qiuyu Zhu. 2020. Maximum signed (k, r)-truss identification in signed networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 3337–3340.

[43] Zhaoming Wu, Charu C Aggarwal, and Jimeng Sun. 2016. The troll-trust model for ranking in signed networks. In *Proceedings of the Ninth ACM international conference on Web Search and Data Mining*. 447–456.

[44] Xianhang Zhang, Hanchen Wang, Jianke Yu, Chen Chen, Xiaoyang Wang, and Wenjie Zhang. 2022. Bipartite graph capsule network. *World Wide Web* (2022), 1–20.

[45] Jun Zhao, Renjie Sun, Qiuyu Zhu, Xiaoyang Wang, and Chen Chen. 2020. Community identification in signed networks: a k-truss based model. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2321–2324.

[46] Alexander Zhou, Yue Wang, and Lei Chen. 2021. Butterfly counting on uncertain bipartite graphs. *Proceedings of the VLDB Endowment* 15, 2 (2021), 211–223.

[47] Zhaonian Zou. 2016. Bitruss decomposition of bipartite graphs. In *Database Systems for Advanced Applications: 21st International Conference, DASFAA 2016, Dallas, TX, USA, April 16-19, 2016, Proceedings, Part II 21*. Springer, 218–233.