Mining Butterflies in Streaming Graphs

by

Aida Sheshbolouki

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2023

© Aida Sheshbolouki 2023

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Jian Pei Professor, Department of Computer Science, Duke University
Supervisor(s):	M. Tamer Özsu University Professor, David R. Cheriton School of Computer Science, University of Waterloo
Internal Members:	Semih Salihoglu Associate Professor, David R. Cheriton School of Computer Science, University of Waterloo
	Yaoliang Yu Associate Professor, David R. Cheriton School of Computer Science, University of Waterloo
Internal-External Member:	Lukasz Golab Professor, Department of Management Sciences, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis includes material, which I learned and developed during my studies and publishing in peer-reviewed venues. The published articles[1] are co-authored and supervised by Prof. M. Tamer Özsu.

[1]Chapter 3 is published in [295, 296, 297, 298], Chapter 4 is published in [295, 298], and Chapter 5 is published in [296, 297].

Abstract

This thesis introduces two main-memory systems *sGrapp* and *sGradd* for performing the fundamental analytic tasks of biclique counting and concept drift detection over a streaming graph. A data-driven heuristic is used to architect the systems. To this end, initially, the growth patterns of bipartite streaming graphs are mined and the emergence principles of streaming motifs are discovered. Next, the discovered principles are (a) explained by a graph generator called *sGrow*; and (b) utilized to establish the requirements for efficient, effective, explainable, and interpretable management and processing of streams. *sGrow* is used to benchmark the stream analytics, particularly in the case of concept drift detection.

sGrow displays robust realization of streaming growth patterns independent of initial conditions, scale and temporal characteristics, and model configurations. Extensive evaluations confirm the simultaneous effectiveness and efficiency of *sGrapp* and *sGradd*. *sGrapp* achieves mean absolute percentage error $\leq 0.05/0.14$ for the cumulative butterfly count in streaming graphs with uniform/non-uniform temporal distribution and a processing throughput of 1.5×10^6 data record per second. The throughput and estimation error of *sGrapp* are 160× higher and 0.02× lower than baselines. *sGradd* demonstrates an improving performance over time, achieves zero false detection rates when there is not any drift and when a drift is already detected, and detects sequential drifts in zero to a few seconds after their occurrence regardless of drift intervals.

Acknowledgements

I would like to thank all the great people who made this thesis possible.

Many thanks to the person who continuously supported me, who taught me how to manage my time, who encouraged me to pursue my goals and boosted my confidence and motivation through the journey, who resembled a speaking encyclopedia to which I always referred my questions, who introduced me to the world of big data, who showcased the meaning of open-mindedness, kindness, wisdom and intelligence, and thoughtfulness, who is the one and only one prof. Tamer Özsu.

I would like to express my gratitude to my thesis committee members, professors Jian Pei, Semih Salihoglu, Yaoliang Yu, and Lukasz Golab, for taking time to review my thesis and providing constructive and insightful comments. I have learned various topics about graph data processing, machine learning, and data streams from their courses, publications, and discussions.

I was privileged to be a member of Data Systems Group and am thankful for many learning opportunities: from seminars and weekly lunch talks to meeting researchers, to serving on the lab committee and practicing management skills, to coordinating lunch talks and practicing communication skills, to organizing talks and meetings and practicing diversity matters, to working in a shared and open-space workplace, to celebrating the success of others and sharing happiness, to learning from groups chats on state-of-the-art topics, and many other teaching and touching occasions.

It was an honor to complete my Ph.D. at the University of Waterloo, working and living among the great members of faculties, staff, and students of different departments, offices, and centers. In particular, I am thankful to all members of David R. Cheriton School of Computer Science and the Faculty of Mathematics.

I would like to thank my friends, my support group. To name a few: Abeer Khan, Amine Mhedhbi, Anil Pacaci, Banafsheh Lashkari, Behnaz Jamasb, Besat Kassaie, Brad Glasbergen, Chang Ge, Elhab Bidaki, Enamul Haque, Karl Knopf, Kerem Akillioglu, Khaled Ammar, Mahsa Paknezhad, Mana Raziyan, Manoj Sharma, Maryam Omidvar, Melica Mirsafian, Michael Abebe, Mustafa Kurkmaz, Nafisa Anzum, Negar Arabzadeh, Peng Shi, Royal Sequeira, Sara Atapour, Shadi Ghasemitaheri, Shaya Pourmirza, Shubhankar Mohapatra, Shufan Zhang, Siddhartha Sahu, Tejasvi Kashi, and Zeynep Korkmaz.

And thanks to my grandparents, my parents, and my siblings. Their love has always been the magic of my life.

Dedication

This thesis is dedicated to my parents and sisters.

Table of Contents

List of Figures

List of Tables

Nomenclature

Streaming Graph, denoted as $\mathfrak{R} = \langle r^1, r^2, ...\rangle$, is an unbounded sequence of partially ordered **streaming graph records** (sgr), where each sgr $r^m = \langle v_i^m, v_j^m, \omega_{ij}^m, \tau^m\rangle$ with index m denotes an edge e_{ij} between i-vertex v_i and j-vertex v_j, weight ω_{ij}, and timestamp τ.

Degree of vertex v_i, denoted as $deg(i) = |N(v_i)|$, is the size of its neighbourhood $N(v_i) = \{v_j \mid e_{ij} \in E\}$.

A **window**, denoted as $W_k := [W_k^b, W_k^e)$, with index k is a range of width $|W_k|$.

A burst-based **graph snapshot**, denoted as $G_{N_b,t} = (E, V)$, is a graph formed by vertices and edges in a window of N_b bursts at time t.

A **Caterpillar**, denoted as $\rtimes$, is a three-path sub-structure and a **butterfly**, denoted as $\bowtie$, is a closed four-path sub-structure.

Butterfly Densification Power-law (BPL), denoted as $B(t) \propto f(|E|^\eta)$, $\eta > 1$, states that the butterfly count at time step t follows a power law function of the edge/sgr counts.

Strength Diversification, denoted as $\mu_S^1 < \mu_2^2$, $Y_2^1(S) < Y_2^2(S)$, $1 < CV^1(S) < CV^2(S)$, states that, given two consecutive burst-based graph snapshots at times t_1 and t_2, the probability distribution of butterfly vertex strengths $Pr(S)$ gets broader and more skewed since the average μ_S, excess kurtosis $Y_2(S)$, and coefficient of variation $CV(S)=\sigma_/\mu$ of strengths increase.

Steady Strength Assortativity, denoted as $Y_2^1(\delta) < Y_2^2(\delta)$, $CV^1(\delta) \approx CV^2(\delta)$, $CV(\delta) \approx 1$, $F_i^1 = F_i^2$, $i = 1,..,4$, $r^s > 0.1$, holds when the **strength assortativity localization factor**, denoted as $r^s = 1 - F_1$, is fixed over two consecutive burst-based graph snapshots at times t_1 and t_2, since the probability distribution of strength difference of butterfly edges $Pr(\delta)$ gets broader and more skewed while remaining fixed-shaped as

- the excess kurtosis $Y_2(\delta)$ of strengths differences δ increases,
- their coefficient of variation $CV(\delta)=\sigma_\delta/\mu_\delta$ remains fixed to 1, and
- the proportion of δs in four regions of the distribution F_i does not change.

sGrow model is a generative function with four parameters: connection probability ρ, the maximum batch size M, the slide parameter β, and the range of random walk's length $[L_{min}, L_{max}]$.

sGrapp framework is a butterfly counting algorithm with two parameters: the number of bursts per window N_b and BPL exponent α. *sGrapp* estimates the cumulative butterfly count at the end of window W_K as the summation of the followings

- $\hat{B}_{k-1}$
- the number of butterflies introduced by window W_k, denoted as $B_G^{W_k}$
- the approximate number of inter-window butterflies $\hat{B}^{interW}$

sGradd framework is a concept drift detection algorithm with one parameter: the number of bursts per window N_b. *sGradd* maps butterflies in the streaming graph to unipartite vertices (which resemble phase oscillators with a phase θ_v and frequency Ω_v); and then detects drifts at time step t by tracking the evolution of synchronization of phases using a quantity called order parameter $O[t] = \frac{((\sum_{v\in V} sin\theta_v)^2+(\sum_{v\in V} cos\theta_v)^2)^{\frac{1}{2}}}{|V|}$.

Chapter 1

Introduction

A wide variety of real-world datasets include data records that are related to each other; for example, citation datasets, transaction datasets, and social network data. Such datasets are best modeled as graphs, where the data records are represented as a set of vertices connected by edges capturing the relationships among entities. The graph model treats both the entities (vertices) and relationships (edges) as first-class objects. The model can also describe several types of many-to-many inter-dependencies among data records as well as their compositions (Table 1.1).

Due to the aforementioned high representational ability, graphs have long been used to represent datasets where it is important to explicitly capture relationships. Management and processing of graph datasets have always been driven by the characteristics of the datasets and/or workloads (often specified by the applications). In most modern applications (e.g. product order transactions, World Wide Web feeds, and social networks), graphs are not static, but change over time. A particular type that is of interest in this thesis is where the graph emerges over time as the entities and the relationships among them are established and the corresponding data records with fine-grained temporal information (i.e. timestamps) stream into a processing unit. These are called *streaming graphs* whose main characteristic is that they are unbounded and the full graph is never available to algorithms processing them. The continuous rapid temporal evolutions lead to unbounded/unknown stream length and non-stationary distributions of the underlying data snapshots. In this context, the temporal evolutions usually occur with respect to the most recent graph topology (i.e. update events are not global); the evolving streaming rates lead to non-uniform inter-arrivals; and multiple generative sources (as well as factors such as transmission delays) cause out-of-order arrival of data records to a processing unit which has no control over the arrival order or data rate [251, Stream Data

Relation Type	(Sub-)Graph Type
symmetric	undirected or bi-directed graphs
asymmetric	directed or oriented graphs
order and transitive	directed triangles or feed forward loops
anti-symmetric	non-bidirectional graphs
identity and reflexive	self-loops
non-binary	hyper-graphs
multiple	multi-dimensional graphs
full	complete graphs
empty	isolated vertices
Composed Relations	**(Sub-)Graph Type**
composition	multi-hop paths with different edge types
transpose	reversed edge directions
product	loops
intersection/union/difference	graph matching

Table 1.1: Data record inter-dependencies represented by graph data model.

Management], [130, 205]). A streaming graph is different from aggregated temporal graphs (or graph streams) that are a sequence of graph snapshots (representing a dynamic graph with an entirely available structure that undergoes temporal changes). Moreover, weight addition patterns [229], streaming context [251, Big Data Processing], and data-driven semantics [18] lead to burstiness in streaming record arrivals. A real example is the case of user-product interactions in Alibaba e-commerce services that incurred a processing rate of 470 million event logs per second during a peak interval [251, Big Data Processing]. The streaming graph model captures the characteristics of these real-world datasets. The model assumes that the graph is built incrementally as data records arrive. Each arrival consists of a timestamp assigned by the generative source and a payload that indicates the vertex/edge that is generated and additional information such as edge weight.

This thesis presents an approach towards data-driven algorithm/system design for explainable and interpretable streaming graph analytics. The data-driven approach refers to exploratory analysis of streaming graphs for in-depth understanding and identifying the structural and temporal organizing principles of real-world streaming graphs to design effective and efficient processing algorithms. The explainable and interpretable approach is concerned with performing iterative and stateful tasks over streaming graphs, such that the operations are explainable and the outputs are interpretable.

1.1 Thesis Scope

The thesis has two foci:

1. **Bipartite graphs.** The streaming graph record (sgr)s in many real-world applications capture the interactions that naturally occur in a bipartite mode. These bipartite streaming graphs represent heterogeneous connections between two disjoint sets of vertices (in this thesis refered as i-vertices and j-vertices). For instance, affiliation graphs that model the membership of people in groups, authorship graphs that model the links between authors and their works, text graphs that model the occurrence of words in documents, feature graphs that model the assignment of features to entities, and user-item graphs with items spanning different domains such as social networks (users-hashtags [365]), web-based services (users-websites, multimedia services, and products [316, 163, 310, 342, 329]), financial systems (users-donation campaigns [11]), transportation systems (users-registered vehicles [165]), and communication systems (users-phone calls [353]). It has been shown that all complex networks have an underlying bipartite structure [324, 141]. Even those networks that are naturally unipartite, e.g. social networks, have an inherent bipartite structure driving the topological structure of the unipartite version [243, 324, 141, 323]. Moreover, bipartite graphs provide full representation without information loss for interactions that naturally occur in one mode (compressed datasets as unipartite graphs [376]), or multiple modes (high order interconnections as hyper-graphs [161, 13, 336]).

   A natural question that arises is why the bipartite graph cannot be projected into a unipartite graph (via a common approach that connects vertices with shared neighbours [339]) and then apply existing methods for data mining over unipartite graphs? The answer is that projecting the graph based on just shared neighbours is misleading, and counting on it is inefficient since the projected graph displays different patterns [52] due to following reasons. First, the projected unipartite graph loses fine-grained pattern information [194], since the one-to-many relationship information is projected to pairwise relationships and the projection is not bijective. Second, the projected unipartite graph will have significantly more edges than the bipartite graph since each $i-(j-)$vertex v with degree d_v produces $\frac{d_v(d_v-1)}{2}$ homogeneous edges. That is, the number of edges in the original bipartite graph is $\Sigma_v d_v$ while in the projected graph it is $\Sigma_v \binom{d_v}{2}$. It has been shown [194] that projection can lead to an edge inflation of 200×. In the case of streaming bipartite graphs that already have a high number of edges, the projection will exacerbate the computational footprint. Finally, the patterns that emerge in the projected unipartite graph are not reliable signals of the original bipartite graph since the edge inflation artificially changes the patterns. For instance, it has been shown that the clustering coefficient is high in the projected mode [244, 141] and unipartite projection misleads the community detection analysis [143, 39]. Moreover, one-mode projection is a separate line of research. This problem scope is devoted to devising metrics for the similarity of vertices of each mode such that connecting similar vertices does not affect the structural patterns such as degree distribution [384]. Therefore, it is important to devise techniques to directly study bipartite structures.

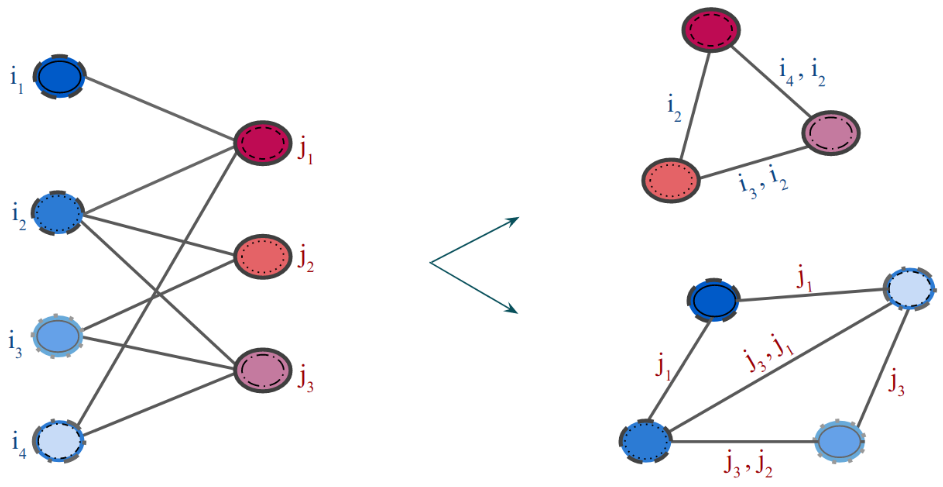
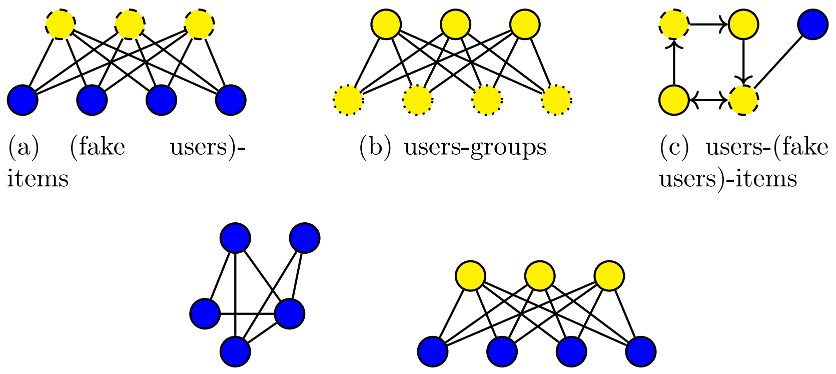


Figure 1.1: Projecting a bipartite graph to two unipartite graphs.

2. **Butterfly motifs.** The particular subgraph of focus throughout the thesis is *butterfly* (a complete subgraph between two pairs of distinct vertices). Similar to the triangles in unipartite graphs, butterflies are the simplest and most local form of a cycle in bipartite graphs. Butterflies are identified as one of the main topological drivers of structural features such as transitivity and degree assortativity, whose understanding is critical for improving the studies and models of spreading phenomena on social networks with bipartite backbone graphs [324]. Moreover, butterflies are of great importance in measuring properties such as cohesion, network stability, and error tolerance [378]. For instance, cohesion can be measured by counting the number of butterflies adjacent to vertices or by the clustering coefficient computed based on the fraction of paths of length three, which are adjacent to each vertex and form butterflies. Recently, various butterfly-based data models and analytic algorithms for cohesive subgraph detection in heterogeneous information networks have


(a) (fake users)-items (b) users-groups (c) users-(fake users)-items

(d) items (e) users-items

Figure 1.2: Bipartite graphs in fraud detection and recommendation systems.

been proposed [117, 149, 14, 370, 106, 220]. Study of cohesive substructures such as butterflies in the streaming setting is challenging due to stateful analytics ($O(n)$ memory/space for at least finding butterflies, while requiring sublinear computational efficiency) and requires specialized techniques for processing and management of data records. Moreover, butterfly-based processing impacts different applications including the following cases in user-item data streams [283].

- *Analytics for anomaly detection.*
  - (Fake user)-item graphs for detecting cases when a number of users are hired to complete transactions to promote target items (Figure 1.2(a)).
  - User-group graphs for detecting online gambling abnormal behaviours when users join gambling groups for exchanging abnormal amount of money (Figure 1.2(b)).
  - User-(fake user)-item graphs for detecting fraudulent money transfers (Figure 1.2(c)).
- *Analytics for recommender systems.*
  - Extract item-item similarity graphs over which a random walk is performed until reaching an item with different category for recommendation to a shopping basket (Figure 1.2(d)).
  - Find community of similar users for offering collaborative items (Figure 1.2(e)).
- *Analysis for promoting sustainable life styles.* United Nations has developed 17 Sustainable Development Goals (SDG) which are "the blueprint to achieve a

better and more sustainable future for all. They address the global challenges we face, including poverty, inequality, climate change, environmental degradation, peace, and justice"[1]. Particularly, SDG 12 titled *sustainable consumption and production*[2] is aimed to promote sustainable life styles. For instance, in case of user-item streams, one approach towards the mentioned goal is raising awareness about conscious consumption. Since a butterfly represents a pair of users interacting with two common items, analysis of butterfly interconnnections unveils hidden orders of user preferences and/or item perceptions. Such analyses can be conducted from different perspectives to gain insight on consumption trends and inform policy makers or product managers. These perspectives include the following cases:

– fast fashion brands are known to be not eco-friendly and the popularity of the low quality new items produced in such ways can reveal consumption patterns. This can be done through analysis of users interacting with new items or perception of new-items;
– re-using products is a sustainable consumption approach and the extent to which items are circulated among users/markets can reveal consumption patterns. This can be done through analysis of reusable items purchased/used by same users.
– long-lasting products and eco-friendly services can sometimes be costly and incur sharing of the expenses. The extent to which expenses (for certain items) are shared can reveal another consumption pattern. This can be done through analysis of user interactions with same certain items.

1.2 Streaming Graph Analysis

The emergence patterns of butterflies as the meso-scale temporal building blocks in bipartite streaming graphs are studied in two phases. Meso-scale refers to the intermediate granularity of subgraphs between microscopic subgraphs such as vertices and edges and macroscopic subgraphs covering most of the vertices in the graph such as the largest connected components. In the first phase, subgraph pattern mining is performed using vertex-centric methods (where the butterfly vertices are explored with respect to their

[1]United Nations sustainable development, un.org/sustainabledevelopment/

[2]"Sustainable consumption and production is about doing more and better with less. It is also about decoupling economic growth from environmental degradation, increasing resource efficiency and promoting sustainable lifestyles."

type, degree, support, and/or timestamp) and edge-centric methods (where the distribution of the inter-arrival of butterfly edges is explored). In the second phase, subgraph pattern mining is performed using vertex-centric methods (where the butterfly vertices are explored with respect to their type and weighted degree (strength)) and edge-centric methods (where the distribution of strength difference of butterfly edges is explored). In both phases, a landmark window is used for *eager* computations (where sgrs are appended to the window), or *lazy* computations (where batches of sgrs are appended to the window).

The first phase [297, Section 3.2] shows that butterflies are temporal motifs with bursty emergence patterns. Due to these emergence patterns, the number of butterflies is significantly and continuously higher than that of random (null) graphs. The quantitative emergence pattern is formulated as the **butterfly densification power law** (BPL) which states that the number of butterflies at time t follows a power law function of the number of edges at time t. Another finding is that the bursty butterfly formation is contributed by vertices with degree above the average of unique vertex degrees (hubs) and timestamp in the first 25% of ordered set of already seen timestamps (old hubs). The second phase [298, Section 5.1] discovers a phenomenon called **scale-invariant strength assortativity of streaming butterflies**, a co-occurrence of three patterns: butterfly densification, strength diversification, and steady strength assortativity. The confounding data-driven semantics are explained in the domain of user-item interactions as these patterns relate to three graph theory concepts: burstiness, rich-get-richer, and core-periphery. These laws influence the algorithms developed in this thesis.

The main challenge is performing the stream mining while simultaneously maintaining effectiveness and efficiency. As discussed in the following, this is addressed with respect to the data and techniques used throughout the analyses.

Data. Effective exploration relies on real-world bipartite streaming graphs with timestamps and weights. The focus of this research is user-item streams. The sequences of user-item interactions in web-services are typically associated with a weight that can be an explicit value such as rating, or an implicit value denoting the multiplicity of interactions between a pair of vertices. Moreover, the time-labeled interactions are continuously generated with a non-stable rate giving rise to emergence of an unbounded dynamic structure. The edge weights and fine grained temporal information enable exploring the temporal and connectivity patterns. Publicly available data are used in which the timestamp and weight are explicitly given in the data records (common in rating graphs). Implicit weights that are computed by aggregating multiple edges between two vertices are not considered, since such aggregations require aggregating the timestamps as well, which in turn manipulates the temporal properties and makes the temporal analysis unreliable.

Effective exploration also relies on configurable random graph models as null graph models to generate synthetic streams with known properties. A null graph model [297, Section 3.1] is proposed to better understand and explain what is happening in real world graph streams through the comparisons and contradictory case investigations. The model extends the popular and widely adopted Barabasi-Albert model [38] (e.g. used in [159, 221, 236]) to generate bipartite streaming gaph with respect to a given real graph such that the synthetic stream has (roughly) the same dynamic structural statistics but the timestamps are static.

Techniques. Effective exploration also relies on data mining approaches which supports the data characteristics (e.g. does not incur information loss, edge inflation, and artificial graph patterns). This enables explainable, interpretable, and reliable pattern discovery. A new metric, called *strength assortativity localization factor* [298, Section 4], is introduced. This metric enables simple and effective (fair and accurate), statistical temporal analysis (e.g. strength assortativity) for network inference in graphs having dynamic streaming rate, abundant (bi)cliques, different scales, and multiple/skewed (strength) distributions. It is based on tracking the localization of a low-dimensional vector, embedding data distributions in graph snapshots within/across streams.

Efficient exploration with a main-memory processing scheme requires incremental approaches to tackle the unboundedness and partial access to the graph structure. To this end, several windowing schemes are used for setting the slide size/frequency and window size/elements. Precisely, for the purpose of exploratory analyses, landmark windows are used and the analyses are performed after appending of either one sgr or a variable-length batch of sgrs. Also, an exact batched processing algorithm [297, Section 3.2] is devised for butterfly enumeration. This algorithm follows a vertex-centric approach that does not require accessing two-hop neighbours (i.e. it is not triple/wedge-based) and can be computed by looping over either i-vertices or j-vertices depending on their average degree. Therefore, it is suitable for large graphs with high average degrees.

1.3 Streaming Graph Modelling

This component of the thesis involves modelling/explaining the growth patterns in streaming graphs. As previous studies [29, 199, 108] describe, the graph models providing micromechanics or high-order generative process of graph structure are generally deemed as the explanation for the patterns observed in real-world graphs. Current works study and model the generative patterns of static or aggregated temporal graphs commonly optimised for down stream analytics or ignore (1) multi-partite/non-stationary data distributions, (2) *emergence* patterns (not just existence) of building blocks, and (3) streaming paradigms such as unbounded/time-sensitive updates, evolving streaming rates, and out-of-order/bursty records (e.g., [10, 133, 359, 14, 201, 25, 329, 373]). The thesis introduces a **s**treaming **grow**th model, called *sGrow* [298, Section 6], which includes a set of microscopic mechanisms to explain the discovered patterns (Figure 1.3). Microscopic mechanisms also known as 'local rules' determine how new edges connect to the rest of the graph. *sGrow* suits the following cases:

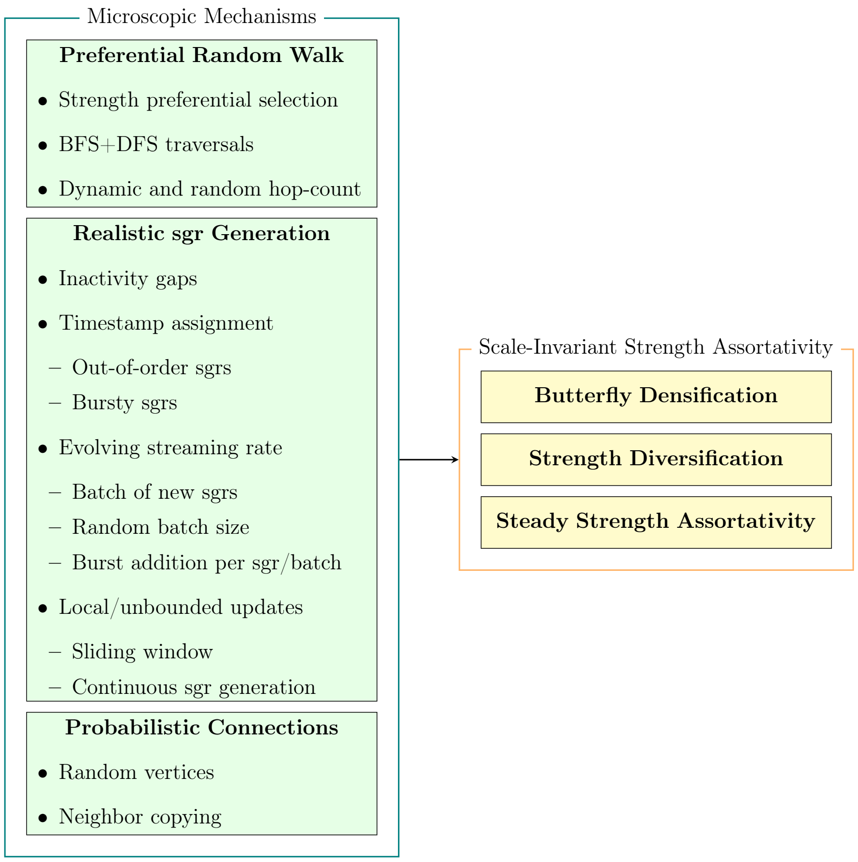


Figure 1.3: Introduced microscopic mechanisms for explaining butterfly emergence patterns by *sGrow* model.

- Streaming graph benchmarks by generating configurable realistic data streams supported by a reference guide for parameter configuration and stress testing analysis.

- Machine learning benchmarks by providing annotated data streams which are synthesized by realistic instance injection and suit both testing and training purposes.
- Development of streaming algorithms and models (e.g. concept drift models) by providing microscopic mechanisms and characteristic patterns.

The main challenge is reproducing the explored realistic patterns in an effective, efficient, explainable, and configurable approach. As discussed in the following, this is addressed with respect to the functions and parameters of the introduced algorithms.

Functions. An effective, efficient, and explainable growth model requires pinpointing the generative origins and modelling them via accurate and scalable micro-mechanics. *sGrow* incorporates techniques for iterative addition of bursts of sgrs which satisfy streaming graph model, preserve realistic patterns of butterfly emergence quantitatively and qualitatively, and make the stream generation scalable. Moreover, *sGrow* enables generating sequence of bipartite edges attributed with timestamps and weights, isolated/out-of-order edges, and four-vertex graphlets.

Parameters. A configurable model requires designing a parameterized algorithm which robustly realizes realistic growth patterns independent of initial conditions, scale and temporal characteristics, and model configurations. *sGrow* is parameterized with user-specified configurations for the scale, burstiness, level of strength assortativity, probability of out-of-order records, generation time, and time-sensitive connections.

1.4 Streaming Graph Analytics

The third component of the thesis involves designing analytics algorithms as part of frameworks for two cases in streaming graphs: butterfly counting and concept drift detection.

1.4.1 Butterfly Counting

The results from the previous research component confirm that butterflies are temporal motifs in bipartite streaming graphs. On the other hand, as noted in various studies (e.g.

[301, 256, 51]), motif counting is a fundamental problem in large-scale network analysis. Therefore, in this part of the thesis, the problem of butterfly counting in bipartite streaming graphs is studied. This benefits many applications where studying the cohesion in graph data is of particular interest. Examples include investigating the structure of computational graphs or input graphs to the algorithms, as well as dynamic phenomena and analytic tasks over complex real graphs (Figure 1.4). Butterfly counting is computationally expensive, and known techniques do not scale to large graphs; the problem is even harder in streaming graphs.

Following a data-driven methodology, the thesis introduces *sGrapp* [297, Section 4], a **s**treaming **gr**aph **app**roximation framework for butterfly counting. *sGrapp* uses a novel window-based stream processing, which adapts to the temporal distribution of the stream. The window management mechanism is general and conforms to any real stream with no assumption about the order and number of arriving sgrs. This mechanism provides load-balanced windows for efficient analytical workloads and also enables accurate continous/temporal analysis which are based on comparing the analysis over different windows of a stream as well as analysis over different streams having different temporal distributions.

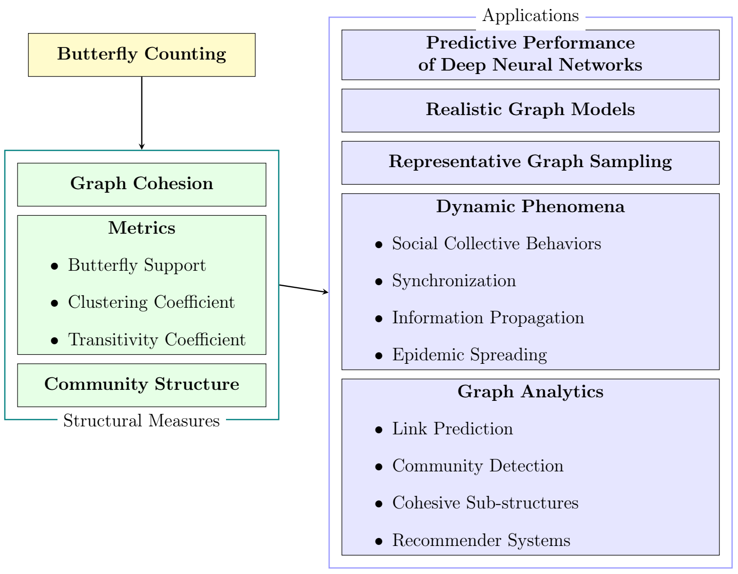


Figure 1.4: Example applications of butterfly counting.

The main challenge is simultaneously achieving efficiency and effectiveness. Exact butterfly counting is feasible only when the entire graph is available to the processing algorithm. As discussed in the following, this is addressed with respect to the butterfly emergence patterns and the windowing methods.

Patterns. An efficient streaming algorithm for butterfly counting can only deal with a subset of the stream at any given point in time. Also, a precise streaming algorithm demands taking into account all existing butterflies regardless of how long they take to form and how much memory is available. According to the discovered patterns and based on BPL, an estimate is provided for the count of a certain type of butterflies which are computationally challenging. Also, optimisations are introduced based on learning accurate values for the exponent of BPL.

Windowing. An efficient and accurate streaming algorithm for butterfly counting depends on appropriate windowing approaches for setting the window size/slide such that no butterfly is missed in counting. In window-based algorithms such as those in this thesis, care is required as butterflies may be split across windows, affecting the butterfly count – it is important to take into account the butterflies that may fall between windows. Moreover, when counting the number of multiple-window-spanning butterflies, it is important to quantify them based on BPL. Based on the discovered patterns, a window management method is introduced to deal with the bursty emergence patterns of butterflies. The proposed approach uses burst-based tumbling windows that can adapt to the temporal distribution of the real streams with no assumption about the order and number of arriving sgrs. The benefits are two-fold:

- providing load-balanced windows for efficient analytical workloads; and
- enabling accurate comparison of graph snapshots of the same or different streams.

1.4.2 Concept Drift Detection

Concept Drift (CD) occurs when a change in a hidden context can induce changes in a target concept. CD is a natural phenomenon in streaming data due to the non-stationary setting. Understanding, detection, and adaptation to CD in streaming data is vital for effective and efficient analysis/analytics as reliable outputs depend on adaptation to fresh inputs. Also, a variety of practical use-case scenarios reside in streaming setting and incur CD. This thesis defines CD in streaming graphs and introduces *sGradd*, a **s**treaming **gra**ph framework for **d**rift **d**etection.

The main challenge is, again, simultaneously achieving efficiency and effectiveness, while detecting and understanding the drifts with an explainable and unsupervised method. Moreover, the evaluation methodology should be accurate and reliable. As discussed in the following, this is addressed with respect to the framework design and its performance evaluations.

Design. An accurate detection algorithm demands stateful operations over windows of sgrs and this is computationally expensive. On the other hand, high velocity and dynamic streaming rate of sgrs necessitate a rapid drift detection. In the thesis, transient concepts in streaming graphs are defined. CD is defined in the case of transient, interconnected, and sequence of data instances forming a streaming graph which serves as the input to any online adaptive analytic task (in both supervised and unsupervised mode). Accordingly, *sGradd*, a modular framework, is introduced with data management and drift detection components based on the butterfly patterns. The components are composed of a collection of explainable, unsupervised, and adaptive techniques for understanding and detecting drifts in hidden contexts that are reflected in target transient concepts. The introduced techniques display initial and improving performance (with respect to accuracy and latency of detections) over the timeline of sgr arrivals.

Evaluation. An accurate evaluation demands drift labels and precise recognition of true/false and missed/delayed detections. *sGrow* is used to generate streams with different drift patterns (reoccurring versus gradual drifts) and intervals (close versus far drifts). *sGrow* generates sgrs through adding bursts such that the stream reproduces realistic subgraph emergence patterns; Therefore, it simulates a drift in a hidden context (generative process) rather than an explicit drift in the target concept (subgraph interconnnectivity patterns). When the drifts are close to each other, there is a concern about evaluation of both accuracy and latency since the detections can be delayed to a time point after the next drifts [173]. In such situations, it is not certain whether a detection is a duplicate false detection or it is a delayed detection corresponding to previous drifts. To address this concern, the accuracy rates and latency of the sequential drifts are considered simultaneously for close drifts and individually for far drifts.

1.5 Thesis Organization

The rest of thesis is organised as follows. Chapter 2 includes the definition and notations. Chapters 3, 4, and 5-6 further explain the algorithms and results introduced by the streaming graph analysis, modelling, and analytics, respectively. Chapter 7 concludes the thesis with a summary and future directions.

Chapter 2

Background and Definitions

This chapter includes definitions and notations used for describing the data, processing approaches, and subgraphs. Table 2.3 present the notations based on i-vertices. Similar notations hold for j-vertices where applicable.

2.1 Streaming Graph Model

Definitions 1, 2, 3, and 4 describe the streaming graph data model introduced in this thesis and [252, 253], as part of *S-Graffito* project[1].

Definition 1 (Streaming Record) *A streaming record (sr) r is a pair (τ, p) where τ is the event (application) timestamp of the record assigned by the data source, and p defines the payload of the record.*

Definition 2 (Streaming Graph Record) *A streaming graph record (sgr) is a streaming record (Definition 1) denoted as a quadruple $r^m = \langle v_i^m, v_j^m, \omega_{ij}^m, \tau^m \rangle$, where m is the sgr index, and the payload $p = \langle v_i^m, v_j^m, \omega_{ij}^m \rangle$ indicates an edge with weight ω_{ij}^m between vertices v_i^m and v_j^m.*

Definition 3 (Streaming Graph) *A streaming graph is an unbounded sequence of sgrs denoted as $\mathfrak{R} = \langle r^1, r^2, \cdots \rangle$ in which each record r^m arrives at a particular time t^m ($t^m \leq t^n$ for $m < n$).*

[1] dsg-uwaterloo.github.io/s-graffito/

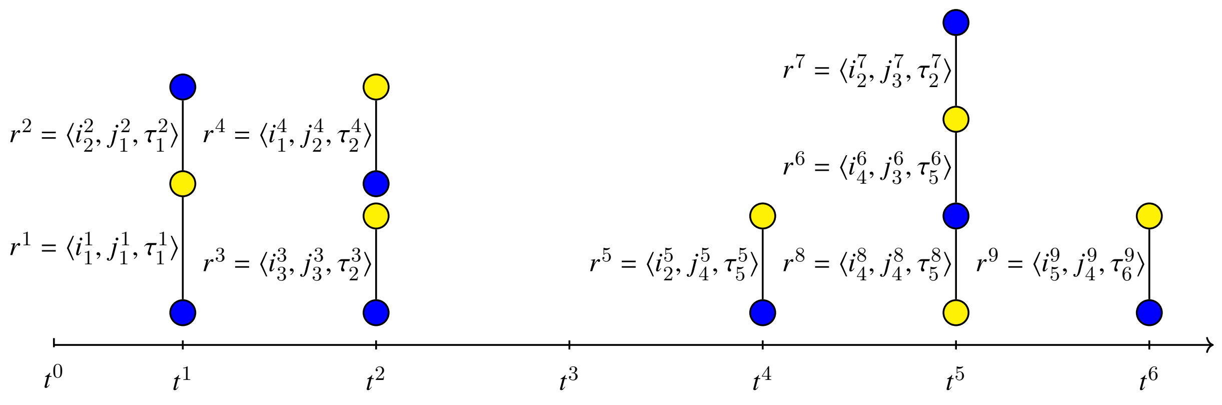

$r^7 = \langle i^7_2, j^7_3, \tau^7_2 \rangle$
$r^2 = \langle i^2_2, j^2_1, \tau^2_1 \rangle$
$r^4 = \langle i^4_1, j^4_2, \tau^4_2 \rangle$
$r^6 = \langle i^6_4, j^6_3, \tau^6_5 \rangle$
$r^1 = \langle i^1_1, j^1_1, \tau^1_1 \rangle$
$r^3 = \langle i^3_3, j^3_3, \tau^3_2 \rangle$
$r^5 = \langle i^5_2, j^5_4, \tau^5_5 \rangle$
$r^8 = \langle i^8_4, j^8_4, \tau^8_5 \rangle$
$r^9 = \langle i^9_5, j^9_4, \tau^9_6 \rangle$
t^0
t^1
t^2
t^3
t^4
t^5
t^6

Figure 2.1: Streaming graph records arriving at sequential time points $t^1 - t^6$

Definition 4 (Burst) *A burst is the batch of sgrs with same timestamp and arrival time.* $b = \{r^m \mid \nexists r^n : \tau^m = \tau^n, t^m = t^n, r^n \notin b\}$.

Multiple generative sources or transmission delays cause out-of-order arrival of sgrs to a processing unit which has no control over the arrival order or data rate. Therefore, a burst is defined as the batch of sgrs with same timestamp which arrive at the computational system *together*. It is not defined as *all* sgrs with same timestamp. Bursts can be repeated over time. For instance, Figure 2.1 illustrates a stream with nine sgrs:

$r^1 = (p^1, \tau^1_1)$, $r^2 = (p^2, \tau^1_1)$, $r^3 = (p^3, \tau^3_2)$, $r^4 = (p^4, \tau^4_2)$, $r^5 = (p^5, \tau^5_5)$, $r^6 = (p^6, \tau^6_5)$, $r^7 = (p^7, \tau^2_5)$, $r^8 = (p^8, \tau^8_5)$, and $r^9 = (p^9, \tau^9_6)$

In this example six bursts exist:

$b_1 = \{r^1, r^2\}$, $b_2 = \{r^3, r^4\}$, $b_3 = \{r^5\}$, $b_4 = \{r^6, r^8\}$, $b_5 = \{r^7\}$, and $b_6 = \{r^9\}$

Where b^2 and b^5 include a same timestamp τ_2 as r^7 is a late arrival.

The sequence of sgrs is considered to be ordered by arrival times rather than timestamps. This helps to take into account late arrivals and enables defining a stream as a sequence of arriving bursts.

2.2 Window Management

Definitions 6,7, and 8 describe existing approaches to manage a window as described in Definition 5. Definitions 9, 10, and 11 describe the introduced burst-based windows in the thesis. Tables 2.1 and 2.2 present instances of these windows operating over the streaming graph shown in Figure 2.1. The content of any of the defined windows form a graph snapshot, which is described in Definition 12.

Definition 5 (Window) *A window indexed by k, W_k, over a streaming graph is a finite multi-set of sgrs denoted as a range $[W_k^b, W_k^e)$, where W_k^b and W_k^e are the beginning and end borders.*

Definition 6 (Time-based Sliding Window) *A time-based sliding window with window size $|W_k|$ and slide parameter β is a window (Definition 5) that slides every β time units and at any time point t, $W_k^e = \lfloor t/\beta \rfloor \cdot \beta$ and $W_k^b = W_k^e - |W_k|$.*

Definition 7 (Tumbling Window) *A tumbling window is a time-based sliding window (Definition 6) that has a constant slide interval equal to the window size and covers disjoint intervals as $(|W_k| = \beta, \forall k)$ and $(W_{k+1}^b = W_k^e, W_{k+1}^e = W_{k+1}^b + |W_{k+1}|)$.*

Definition 8 (Landmark Window) *A landmark window is a window (Definition 5) that progresses as new sgrs are added and the window size increases. The beginning border is fixed and the window size is incremented as $W_{k+1}^b = W_k^b$ and $W_{k+1}^e = W_k^e + |W_{k+1}|$.*

Definition 9 (Burst-based Sliding Window) *A burst-based sliding window with a parameter N_b is a window (Definition 5) that progresses as N_b bursts (Definition 4) are added and window size changes randomly as a random number of sgrs are retired from the window.*

Definition 10 (Burst-based Tumbling Window) *A burst-based tumbling window with a parameter N_b is a window (Definition 5) that covers disjoint intervals. It progresses as N_b bursts are added and window size changes to the number of new sgrs. i.e. It is a tumbling window (Definition 7) with dynamic slide interval and window size.*

Definition 11 (Burst-based Landmark Window) *A burst-based landmark window with a parameter N_b is a landmark window (Definition 8) that progresses as N_b new bursts are added and the window size increases by the number of new sgrs.*

Table 2.1: Instances of time-based, count-based, tumbling, and landmark windows. Windows capture the sgrs arriving according to Figure 2.1.

Window Instance	Window Content
Time-based sliding $\beta = 1$, $\lvert W \rvert = 4$	
$W_1 = [t_0, t_4)$	$\{r^1, r^2, r^3, r^4\}$
$W_2 = [t_1, t_5)$	$\{r^2, r^3, r^4, r^5, r^6\}$
$W_3 = [t_2, t_6)$	$\{r^4, r^5, r^6, r^7, r^8\}$
Tumbling $\lvert W \rvert = \beta = 2$	
$W_1 = [t_0, t_2)$	$\{r^1, r^2\}$
$W_2 = [t_2, t_4)$	$\{r^3, r^4\}$
$W_3 = [t_4, t_6)$	$\{r^5, r^6, r^8, r^7\}$
Landmark	
$W_1 = [t_0, t_1)$	$\{\}$
$W_2 = [t_0, t_2)$	$\{r^1, r^2\}$
$W_3 = [t_0, t_3)$	$\{r^1, r^2, r^3, r^4\}$
$W_5 = [t_0, t_5)$	$\{r^1, r^2, r^3, r^4, r^5\}$
$W_6 = [t_0, t_6)$	$\{r^1, r^2, r^3, r^4, r^5, r^6, r^8, r^7\}$
$W_7 = [t_0, t_7)$	$\{r^1, r^2, r^3, r^4, r^5, r^6, r^7, r^8, r^9\}$

Definition 12 (Graph Snapshot) *A graph snapshot is a pair of vertex and edge sets $G = (V, E)$ forming a graph at time point t by the sgrs within a corresponding window W_k. For simplicity, graph snapshot denoted as $G_{W,t}$ and its corresponding window W_G are used interchangeably throughout the thesis.*

When the graph snapshot is unipartite, edges are denoted as $E = \{e_{vn} = (v, n, w_{vn})\}$. When the graph snapshot is bipartite, vertices are two disjoint sets of i- and j-vertices $V = V_i \cup V_j$, $V_i \cap V_j = \emptyset$ and edges are denoted as $E = \{e_{ij} = (v_i, v_j, w_{ij})\} \subseteq V_i \times V_j$. The set of one-hop neighbours of a vertex v is denoted as $N(v) = \{n \mid e_{vn} \in E\}$. Neighbours of an i-vertex v_i are called j-neighbours denoted as $N_j(v_i)$. Similar notation stands for i-neighbours of a j-vertex v_j, denoted as $N_i(v_j)$.

2.3 Subgraphs

Definitions 13,14, and 16 describe existing subgraphs (illustrated in Figure 2.2) and a graph property used in this thesis. Definition 15 introduces an extension of Definition 14.

Table 2.2: Instances of burst-based windows.
Windows capture the sgrs arriving according to Figure 2.1.

Window Instance	Window Content
Burst-based sliding $N_b = 1$	
$W_1 = [t_0, t_2)$	$\{r^1, r^2\}$
$W_2 = [t_0, t_3)$	$\{r^2, r^3, r^4\}$
$W_3 = [t_0, t_5)$	$\{r^1, r^2, r^4\}$
$W_4 = [t_0, t_6)$	$\{r^1, r^4, r^5\}$
$W_5 = [t_0, t_6)$	$\{r^1, r^4, r^7\}$
Burst-based tumbling $N_b = 1$	
$W_1 = [t_0, t_2)$	$\{r^1, r^2\}$
$W_2 = [t_2, t_3)$	$\{r^3, r^4\}$
$W_3 = [t_3, t_5)$	$\{r^5\}$
$W_4 = [t_5, t_6)$	$\{r^6, r^8\}$
$W_5 = [t_5, t_6)$	$\{r^7\}$
$W_6 = [t_6, t_7)$	$\{r^9\}$
Burst-based landmark $N_b = 1$	
$W_1 = [t_0, t_2)$	$\{r^1, r^2\}$
$W_2 = [t_0, t_3)$	$\{r^1, r^2, r^3, r^4\}$
$W_3 = [t_0, t_5)$	$\{r^1, r^2, r^3, r^4, r^5\}$
$W_4 = [t_0, t_6)$	$\{r^1, r^2, r^3, r^4, r^5, r^6, r^8\}$
$W_5 = [t_0, t_6)$	$\{r^1, r^2, r^3, r^4, r^5, r^6, r^8, r^7\}$
$W_6 = [t_0, t_7)$	$\{r^1, r^2, r^3, r^4, r^5, r^6, r^8, r^7, r^9\}$

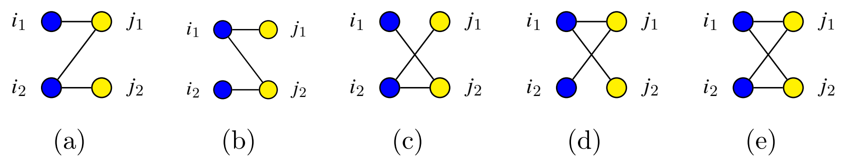


Figure 2.2: (a, b, c, d) Caterpillar and (e) butterfly sub-structures [14].

Definition 13 (Caterpillar) *A caterpillar* $\rtimes$ *is a three-path between two i-vertices* v_{i_1}, *and* v_{i_2} *and two j-vertices* v_{j_1}, *and* v_{j_2}. *Two pairs of i- and j-vertices can form four different caterpillars. i.e.* $\rtimes = \{e_{i_1,j_1}, e_{i_2,j_1}, e_{i_2,j_2}\} \vee \{e_{i_1,j_1}, e_{i_1,j_2}, e_{i_2,j_2}\} \vee \{e_{i_1,j_2}, e_{i_2,j_2}, e_{i_2,j_1}\} \vee \{e_{i_1,j_1}, e_{i_1,j_2}, e_{i_2,j_1}\}$ *(Figure 2.2(a)-(d)).*

Definition 14 (Butterfly) *A butterfly* $\bowtie^{i_1,i_2}_{j_1,j_2}$ *is a* $2,2$*-biclique between two i-vertices* v_{i_1}, v_{i_2} *and two j-vertices* v_{j_1}, v_{j_2}. *It is a closed four-path* $\bowtie^{i_1,i_2}_{j_1,j_2} = \{e_{i_1,j_1}, e_{i_2,j_1}, e_{i_2,j_2}, e_{i_1,j_2}\}$ *formed by adding an edge to a caterpillar (Figure 2.2(e)).*

Definition 15 (Young Butterfly) *A young butterfly* $\bowtie$ *is a butterfly with j-vertices having a timestamp within the last* x *percentage of seen unique timestamps in the stream, i.e.* $\bowtie = \{\bowtie^{i_1,i_2}_{j_1,j_2} \mid \exists r^m, r^n : v_{j_1} \in r^m, v_{j_2} \in r^n, (\tau^m, \tau^n) \in [\tau^{t-\lceil xt \rceil}, \cdots, \tau^{t-1}, \tau^t]\}$.

Considering young butterflies (i.e. restricting the set of j-vertices), enables case studies where the freshness of input data is important and/or the goal is to perform processing over transient data records rather than all seen data records (streaming processing). In the thesis, $x = 25\%$. Setting $x = 100\%$ would be equivalent to considering all seen vertices. The set of unique timestamps in the stream grows over time and consequently the set of j-vertices within the x percentage grows. Choosing a low percentage helps to keep the size of this set balanced particularly when the streaming rate is high.

Definition 16 (Vertex Strength) *Vertex strength (shortly* strength*) is the total weight of edges connected to a vertex denoted as* $S_i = \Sigma_{v_j \in N_j(v_i)} \omega_{ij}$, $S_j = \Sigma_{v_i \in N_i(v_j)} \omega_{ij}$ *[305, 40, 41].*

Table 2.3: Frequent notations used to describe data.

Notation	Description
r^m	Streaming graph record (sgr) with index m
v_i^m	i-vertex corresponding to sgr r^m
τ^m	Timestamp of sgr r^m
t^m	Arrival time of sgr r^m, a computational time point
ω_{ij}^m	weight of the edge between v_i and v_j corresponding to sgr r^m
$\mathfrak{R} = \langle r^1, r^2, ...\rangle$	Streaming graph
$V_i = \{v_i\}$	Set of i-vertices
E	set of edges
$deg(i)$	Degree of vertex v_i
d_i	Average degree of i-vertices
$N(v)$	One-hop neighbourhood of a vertex v
e_{vn}	A unipartite weighted edge between vertices v and n
e_{ij}	A bipartite edge between vertices v_i and v_j
N	Number of vertices in Barabasi-Albert model
m	Initial number of vertices in Barabasi-Albert model
m_0	Number of connections of each new vertex in Barabasi-Albert model
$W_k := [W_k^b, W_k^e)$	A window with index k as a range of width $\|W\|$
$G_{W,t}$	A graph snapshot formed by window W at time t
G_{N_b}	A graph snapshot formed by a burst-based window
β	Slide parameter for a sliding window
b	A burst
$\bar{B}$, $maxB$	Average/maximum seen burst size
N_b	Number of bursts per window
$\rtimes$	Caterpillar
$\bowtie_{j_1,j_2}^{i_1,i_1}$	Butterfly
$\bowtie$	Young butterfly
$\wedge$	Number of wedges (two-paths)
þ	Structural pattern

Chapter 3

Streaming Graph Analysis

Bipartite graphs are rich data structures with prevalent applications and characteristic structural features. However, less is known about their growth patterns, particularly in streaming settings. Frequent subgraphs (motifs [232] or graphlets [281]) as the building blocks of graphs [232] play an important role in understanding the structure of graphs [8, 281, 51, 215, 312, 347, 256, 207, 221, 184, 359, 156, 331, 299, 297]. Network motifs are "patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks" [232]. Identifying the motifs helps characterize the graph and also benefits applications that are based on subgraph-centric programming model (i.e. operates on subgraphs rather than vertices or edges) and can be optimized by indexing the network motifs. That is, network motifs represent the regularities in the graph data and are helpful in building indexes over frequent and regular graph structures (structural indexing) [350, 369, 290]. Butterflies are known to be motifs in static graphs, however their temporal emergence patterns are not well studied. This chapter presents investigations into the emergence patterns of butterflies in streaming graphs and on the underlying contributors to these patterns. The goal is to understand *How do butterflies as the building blocks of bipartite streaming graphs emerge?* This question is answered in two phases: Phase 1: showing that butterflies *are* the building blocks (temporal motifs) across the timeline of sgr arrivals in bipartite streaming graphs and identifying their emergence patterns [297]; and Phase 2: further explorations to explain the identified patterns and discover all of their contributing factors [298]. Both phases involve systematic and extensive experimental analysis of real and synthetic graphs. This is the first empirical study of how streaming graph substructures emerge.

Table 3.1: Frequent notations in this chapter.

Notation	Description
η	Butterfly densification power-law exponent for all butterflies
$N_{hub}(t)$	Number of hubs at time t
$B(t)$	The number of butterflies since the initial time point until t
B_i	Butterfly support of vertex i
S_i	Vertex strength of V_i
r	Assortativity coefficient
$Pr(\delta)$	Probability distribution of strength difference of butterfly edges
$Pr(S_i)$	Probability distribution of strength of butterfly i-vertices
μ_i	Average strength of butterfly i-vertices
μ_δ	Average strength difference of butterfly edges
σ_δ	Standard deviation of strength difference of butterfly edges
F_i	The i^{th} element of embedding vector of $Pr(\delta)$
r^s	Strength assortativity localization factor
CV	Coefficient of variation
Y_2	Excess kurtosis

3.1 Data and Methods

Analyses are conducted on a machine with 15.6 GB native memory and Intel Core $i7-6770HQCPU$@$2.60GHz*8$ processor. All algorithms are implemented in Java (OpenJDK versions $1.8.0-252$ in Phase 1 and 11.0.11 in Phase 2).

3.1.1 Data

The set of real-world bipartite streaming graphs and the synthetic streams generated by a proposed null graph model are described in the following.

Real-world graphs. The focus of research is the organizing principles in bipartite streams such as user-item streams. The sequences of user-item interactions are typically associated with a weight that can be an explicit value such as rating, or an implicit value denoting the multiplicity of interactions between a pair of vertices. Moreover, the time-labeled interactions are continuously generated with a non-stable rate giving rise to emergence of an unbounded dynamic structure. The edge weights and fine grained temporal information enable exploring the temporal and connectivity patterns. The data used are publicly available data in which the timestamp and weight are explicitly given in the data records (common in rating graphs). Implicit weights computed by aggregating multiple edges between two vertices are not considered since such aggregations require aggregating the timestamps as well, which in turn manipulates the temporal properties and makes the temporal analysis unreliable. All datasets are available at public repositories KONECT [191][1] and Netzschleuder[2]. These datasets include naturally occurring bipartite interactions as a set of records including the user ID, item ID, rating, and timestamp. The rating values are in the set $\{1, 2, 3, 4, 5\}$ in all datasets except for WikiLens with ratings in $\{0, 0.5, 1, .., 4.5, 5\}$. In WikiLens, the ratings are rounded and those ratings equal to 0 are replaced with 1 to convert the rating scale to $1 - 5$ (for fair comparison with other datasets). Tables 3.2 and 3.3 provide the statistics of the data streams used in the first and second phases of explorations, respectively (notations are described in Table 2.3). These datasets cover graphs with different structural properties (e.g. edge density, average vertex degree, and wedge (i.e. two-path) count) and characteristics (e.g. number and average size of bursts) which make them suitable for deep analysis. For instance, Ciao and Amazon have low average degree of both i- and j-vertices, while they are bursty streams. Epinions[3] has higher average degree of i-vertices compared to that of j-vertices with a very high number of wedges (the building blocks of butterflies), and it is a bursty stream with large bursts. WikiLens has high average degree of i-vertices but it is not bursty. ML100k has high average degree of i- and j-vertices and high number of wedges and it is roughly as bursty as ML1m and Yahoo which have higher average degree of i- and j-vertices and higher number of wedges.

Synthetic graphs. In Phase 1 of explorations, synthetic random graphs are used in addition to the real-world graphs to bolster the analysis over real-world graphs. In fact synthetic graphs are configurable and have known structural properties that ease the understanding of their patterns. These synthetic graphs are used to better understand and explain what is happening in real-world graphs through the comparisons and contradictory case investigations. These synthetic graphs are generated with respect to the three real-world graphs (Epinions, MovieLens100k, and MovieLens1m) in that the synthetic graphs and the corresponding real-world graphs have (roughly) the same structural statistics (e.g. the number of vertices and edges and the degree). The structure of these synthetic random graphs is generated according to the attachment mechanism of Barabasi-Albert (BA) model [38], which is a popular and widely adopted model for generating scale-free graphs. Given the total number of vertices N, the initial number of vertices m_0 and the number of connections of new vertices m ($m \leq m_0$) as inputs, the BA graph model applies the rich-

[1]http://konect.uni-koblenz.de/networks/

[2]networks.skewed.de

[3]In literature, Epinions appears as both unipartite and bipartite graphs. In this thesis the bipartite graph is used.

Table 3.2: Datasets used in Phase 1 of explorations.

	$\|V_i\|$	$\|V_j\|$	$\|E\|$	d_i	d_j	N	m	N_b
Epinions	$22,164$	$296,277$	$922,267$	41	3			$4,318$
BA+Epinions stamps	$22,514$	$21,455$	$922,254$	41	43	$22,515$	41	$4,318$
BA+random stamps	$22,514$	$21,455$	$922,254$	41	43	$22,515$	41	$921,159$
ML1m	$6,040$	$3,706$	$1,000,210$	166	270			$458,455$
BA+ML1m stamps	$6,106$	$6,022$	$999,901$	164	166	$6,107$	166	$458,312$
BA+random stamps	$6,106$	$6,022$	$999,901$	164	166	$6,107$	166	$994,467$
ML100k	943	$1,682$	$100,000$	106	59			$49,282$
BA+ML00k stamps	995	982	$99,905$	100	100	966	106	$49,254$
BA+random stamps	995	982	$99,905$	100	100	966	106	$996,555$
ML10m	$69,878$	$10,677$	$10,000,054$	143	937			$7,096,905$
Edit-FrWiki	$288,275$	$3,992,426$	$46,168,355$	160	11			$39,190,059$
Edit-EnWiki	$262,373,039$	$266,665,865$	$266,769,613$	70	12			$134,075,025$

Table 3.3: Datasets used in Phase 2 of explorations.

	$\|V_i\|$	$\|V_j\|$	$\|E\|$	d_i	d_j	N_b	$\bar{B}$	$\wedge$
Ciao	$17,615$	$16,121$	$72,665$	4.1	4.5	$4,919$	14.8	$4,896,641$
Epinions	$120,492$	$755,760$	$13,668,320$	113.4	18	501	$27,282$	$69,245,866,714$
WikiLens	326	$5,111$	$26,937$	82.6	5.2	$26,239$	1	$6,316,744$
ML100k	943	$1,682$	$100,000$	106	59.4	$49,282$	2	$18,367,254$
ML1m	$6,040$	$3,706$	$1,000,210$	165.6	269.9	$458,455$	2.2	$602,009,923$
Amazon	$2,146,057$	$1,230,915$	$5,838,041$	2.7	4.7	$3,329$	$1,753.7$	$627,186,651$
Yahoo	$1,000,990$	$624,961$	$256,804,235$	256.5	410.9	$105,331,405$	2.4	$4,627,224,528,654$

get-richer preferential attachment rule to generate a unipartite scale-free random graph. Precisely, this graph model creates an initial complete graph with m_0 vertices and keeps adding $N - m_0$ new vertices to this initial graph. The new vertices are connected to m existing vertices with higher probability of attachment dictated by the attachment rule. The BA preferential attachment rule states that the probability is determined based on the degree of the vertex, therefore the higher the degree (i.e. the older the vertex), the higher the probability of attachment. BA model produces growing unipartite graphs with no timestamps. Therefore, in the following, the model is extended to generate bipartite and scale-free streaming graphs with respect to a given real-world graph such that the structure is dynamic but the timestamps are static.

1. **Create Unipartite BA graph** – The input parameters to the BA model (i.e. N, m, and m_0) should be set such that the average degree of i-vertices and the number

of total edges ($|E|$) in real-world and synthetic graphs are (roughly) the same. That is because of the edge-centric nature of the intended analysis. Therefore, the m and m_0 are set as equal to the average degree of i-vertices (i.e. users) in the real-world graph and the value of N is determined in a way that it satisfies the equation for the number of edges in BA graph, that is $m_0(m_0-1)/2+(N-m_0)m = |E|$. Given the input parameters, the edge list of the scale-free unipartite directed graph is generated.

2. **Project the graph to bipartite mode** – A common approach to project a bipartite graph $BG = (V, E_{ij})$ to unipartite modes $G_i = (V_i, E_i,)$ and $G_j = (V_j, E_j)$ is to connect a pair of vertices if they have a common neighbour (Figure 1.1). That is, $(v_{i_m}, v_{i_n}) \in E_i$ iff $\exists v_j \in V_j : (v_{i_m}, v_j) \in E_{ij} \ \& \ (v_{i_n}, v_j) \in E_{ij}$ and the same connection rule for j-vertices. Accordingly, a reverse-engineering technique can be used for projecting the unipartite graphs to bipartite mode. Precisely, given a unipartite BA graph G_i with N_i or N_j vertices (assuming the vertices as i- or j-vertices), the bipartite mode BG is generated by the procedure below:

   (a) Assign N_j labels $\{L_k | 1 \leq k \leq N_j\}$ to arbitrary edges in G_i.

   (b) Create a set of N_j j-vertices.

   (c) Project each edge $(v_{i_m}, v_{i_n}) \in E_i$ with label L_k into two edges (v_{i_m}, v_{j_k}) and (v_{i_n}, v_{j_k}).

   Clearly, this procedure can yield a bipartite BA graph with a pre-specified number of i- and j-vertices. Therefore, it can mimic the number of vertices in the real-world graph exactly. However, the number of edges in the output bipartite BA graph does not match that of the unipartite BA graph and if we create a unipartite BA graph with specific number of edges, then the number of i-vertices would be affected accordingly. Therefore, this projection method can not yield bipartite BA graphs that have specific number of edges and vertices at the same time and solely adjusting the number of edges will affect the number of vertices. On the other hand, the intended analysis are edge-centric, therefore it is important to create synthetic bipartite graphs with the same number of edges as the real-world graphs.

   To address this problem, a simple projection method is followed. Given the list of directed edges in the unipartite BA graph, the sources of edges are treated as i-vertices and the destinations as the j-vertices. Hence, the BA graph is projected to bipartite mode with the same number of edges as that of the unipartite and the corresponding real-world graph. The number of i-vertices in the projected bipartite BA graph (equal to the N of unipartite BA graph) is very close to that of the real-world graph. In spite of different number of j-vertices in the projected and real-world graphs, this projection method is preferable as it solves the aforementioned issue. Moreover, this method preserves the scale-free characteristic of the uni-partite graph since the j-degree (i-degree) distribution in bipartite graph is equivalent to the in-degree (out-degree) distribution of vertices in the unipartite graph and the j-degree distribution is scale-free.

3. **Assign timestamps to the synthetic edges** – Given the timestamps of the real-world graph and the bipartite structure of the corresponding random graph, timestamps are assigned to the edges in two ways:

   (a) Each BA edge is randomly assigned a timestamp within the range of timestamps of the corresponding real-world graph and the resulting graph is called *BA+random stamps*.

   (b) The un-ordered timestamps of the corresponding real-world graph are assigned to arbitrary BA edges and the resulting graph is called *BA+real stamps*. This method guarantees same temporal distribution for the edges of BA and real-world graphs and supports fair comparisons.

All the edge lists (real and synthetic) are sorted based on the timestamps to simulate the streaming graph records in the analysis. When there are duplicate edge arrivals (i.e. multiple connections between two vertices), the duplicates are ignored and only the first edge is considered.

3.1.2 Methods

Exact Butterfly Counting. Analyses in both Phase 1 and Phase 2 rely on enumerating butterflies. It is important to calculate the exact number of butterflies to make sure that the analysis is correct and the identified patterns are reliable. Therefore, an exact butterfly counting algorithm (Algorithm 1) is introduced to count the occurrence numbers in sequential graph snapshots. Given a bipartite graph snapshot $G_{W,t}$ at a time point t, the goal is to compute $B(t)$ as the number of all quadruples $\langle v_{i_1}, v_{i_2}, v_{j_1}, v_{j_2} \rangle$ in $G_{W,t}$ such that they form a butterfly (Definition 14). Algorithm 1 follows a vertex-centric approach that does not require accessing two-hop neighbours (i.e. it is not triple-based) and can be computed by looping over either i-vertices or j-vertices depending on their average degree (denoted by d_i and d_j). The algorithm takes a vertex v_{i_1} (provided that $d_i \leq d_j$) and considering each pair of j-neighbours v_{j_1} and v_{j_2}, identifies any vertex v_{i_2} which is a common i-neighbour of v_{j_1} and v_{j_2} to form a butterfly (Figure 3.1). Sub-lists are used to avoid

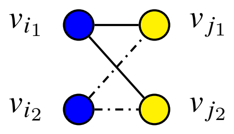


Figure 3.1: Schematic figure for the introduced algorithm countButterflies(G)

iterating over repeated j-neighbours (lines 4-6 in Algorithm 1) and the common neighbours are identified by iterating over the lower degree j-vertex (line 8 in Algorithm 1). Quadruples are added to a hash-set (line 10 in Algorithm 1) whose size determines the butterfly count (line 11 in Algorithm 1). The algorithm is extended to compute the butterfly support of each vertex as the number of butterflies incident to the vertex (Algorithm 2). In Phase 1,

Algorithm 1: Exact Butterfly Counting

```
1  Function countButterflies(G)
       Input: $G_{W,t} = \langle V_i \cup V_j, E_{ij} \rangle$, Static graph
       Output: $B(t)$, The number of butterflies in G
2      $Butterflies \leftarrow \emptyset$ $jneighbours \leftarrow \emptyset$ $vi_2s \leftarrow \emptyset$ for $i_1 \in V_i$ do
3          $jneighbours \leftarrow N_{i_1}$
4          for $index1 \in [1, size(jneighbours)]$ do
5              $j_1 \leftarrow jneighbours[index1]$
6              for $index2 \in [index1 + 1, size(jneighbours)]$ do
7                  $j_2 \leftarrow jneighbours[index2]$
8                  $vi_2s \leftarrow N_{j_1} \cap N_{j_2}$
9                  if $\langle i_1, j_1, i_2, j_2 \rangle \notin Butterflies$ then
10                     $Butterflies.add(\langle i_1, i_2, j_1, j_2 \rangle)$

11     $B(t) \leftarrow size(Butterflies)$
```

an eager computation model is adopted where the exact number of butterflies is computed after each edge is added. That is, Algorithm 1 runs over a landmark window (Definition 8). This is done in the time period 0 to 5000 (i.e. first 5000 sgrs) due to the computational overhead of the algorithm. Note that the frequency distribution of edge insertions occurring in time-intervals of variant sizes follows the same shape. This means that the distribution with respect to scaling across time scales is invariant (i.e. self-similar [333]). Therefore, we can rely on the analysis of a fraction of the subsequent streaming edges. To compare

Algorithm 2: Butterfly Support

```
1  Function ButterflySupport(G)
       Input: $G = \langle V_i \cup V_j, E_{ij} \rangle$, static graph
       Output: $vSupport$, butterfly support of vertices
2      $vSupport \leftarrow \emptyset$, $Butterflies \leftarrow \emptyset$, $jNeighbors \leftarrow \emptyset$, $vi2s \leftarrow \emptyset$
3      for $v_{i1} \in V_i$ do
4          $jNeighbors \leftarrow N_j(i_1)$
5          for $index1 \in [1, size(jNeighbors)]$ do
6              $j_1 \leftarrow jNeighbors[index1]$
7              for $index2 \in [index1 + 1, size(jNeighbors)]$ do
8                  $j_2 \leftarrow jNeighbors[index2]$
9                  $vi_2s \leftarrow N_i(j_1) \cap N_i(j_2)$
10                 for $i_2 \in vi_2s$ do
11                     if $\langle i_1, j_1, i_2, j_2 \rangle \notin Butterflies$ then
12                         $Butterflies.add(\langle i_1, i_2, j_1, j_2 \rangle)$
13                         $vSupport.put(i_1, vSupport.get(i_1) + 1)$
14                         $vSupport.put(j_1, vSupport.get(j_1) + 1)$
15                         $vSupport.put(i_2, vSupport.get(i_2) + 1)$
16                         $vSupport.put(j_2, vSupport.get(j_2) + 1)$
```

the numbers with that of a random graph as a null graph for checking the motifs, only the corresponding BA graph with the same real timestamp is used. This enables fair comparison of structural evolution of real-world and synthetic random graphs.

In Phase 2, Algorithm 1 is used to list the butterflies over sequential graph snapshots corresponding to a burst-based landmark window (Definition 11). The emergence of a certain number of butterflies is studied in different streams with different structural/temporal properties. That is, the prefix of streams is considered until the arrival of up to $\approx 6.5 \times 10^6$ butterflies, which covers the entire stream in WikiLens with 26220 bursts and a prefix of 10000, 9600, 460, 2000, and 15000 bursts in ML1m, Ml100k, Epinions, Amazon, and Yahoo, respectively. In Ciao, the entire stream is checked, which has 4900 bursts and $\approx 6.4 \times 10^5$ butterflies (Table 3.4). The corresponding timeline of burst arrival is divided into 20 equally distanced points and at each point the butterflies in the burst-based graph snapshot are studied. In the analyses, it is important to care about the value and the trend of data points; the number of graph snapshots (here 20) simply changes the smoothness of the plots and does not affect the results since streams with different distribution of timestamps are checked and the scale of graph snapshots differs in various streams. The number of edges/butterflies in each burst varies in different graphs depending on the burstiness of the stream.

Table 3.4: Statistics of the 20^{th} graph snapshot in real-world streams. $|E^{20}|$ is edge count, N_b^{20} is burst count, and butterfly count is denoted as $\bowtie^{20}$.

	$\lvert E^{20}\rvert$	N_b^{20}	$\bowtie^{20}$
Ciao	$72,574$	$4,900$	$636,440$
Epinions	$296,665$	460	$6,418,862$
WikiLens	$26,918$	$26,220$	$6,556,913$
ML100k	$18,696$	$9,600$	$6,492,834$
ML1m	$22,795$	$10,000$	$6,678,784$
Amazon	$2,194,798$	$2,000$	$6,496,236$
Yahoo	$42,105$	$15,000$	$6,496,563$

Strength Assortativity Measurement. The tendency of vertices to connect to similar vertices with respect to one of their quantitative/qualitative attributes is called *assortativity/homophily* [241]. For instance, *degree assortativity* refers to the tendency of vertices with similar degrees to connect. In a graph with degree (dis)assortative mixing pattern, high degree vertices are connected to high (low) degree vertices. In a graph with perfect (dis)assortativity, vertices connect only to same (different) degree vertices. In addition to connectivity insights (the primary goal in Phase 2 of explorations), assortativity provides information about the dynamic behaviour and robustness of the graph [318, 100]. For instance, degree disassortative complex networks compared to degree assortative networks exhibit higher epidemiological threshold leading to easier immunization, while assortative networks get higher resilience to systemic risk by degree-targeted immunization policies [100] (Noldus and Van Mieghem [247] describe this in a complete survey). The epidemiological threshold is defined as the critical ratio among the propagation rate and recovery rate of a disease above which epidemics ensue and immunization is the policy to stop the propagation process. Assortativity is usually studied with respect to vertex degrees. A previous study [204] has shown that studying the assortativity by considering just the degree does not completely uncover the organizational patterns in the structure of graphs. Leung and Chau [204] have introduced the weighted assortativity coefficient to measure the tendency of having a high-weighted edge between vertices with similar degrees. However, the goal of analyses in the phase two in this chapter is measuring the tendency of having an edge between vertices with similar strength (i.e. measuring strength assortativity to discover the connection patterns with respect to weights as well as degrees), particularly in butterflies. In the following, first the requirements are established for an effective measurement of strength assortativity that can accommodate analysis of meso-scale, bipartite, and temporal structures; next, a new metric is introduced for strength mixing patterns called *strength assortativity localization factor*.

The assortativity coefficient (r) [241] is a common metric for assortativity [382, 324, 322]. Assuming that we are interested in quantifying the tendency of vertices to connect to each other based on the similarity of their attribute K, r is computed as the pearson correlation of K of linked vertices and lies in the range $-1 \leq r \leq 1$. Positive (negative) r signals (dis)assortativity and $r = 0$ denotes random mixing. Another approach to study assortativity is to compute the average K of nearest-neighbours for each vertex and then aggregating the values by restricting the class of vertices with $K = k$. It is denoted as $\langle K_n \rangle$ which is a function of K. An increasing (decreasing) $\langle K_n \rangle$ signals (dis)assortativity. This can be inferred by checking the sign of the slope of a linear fit in the log-log plot of $\langle K_n \rangle$ as a function of K. In the following, the effectiveness of r and $\langle K_n \rangle$ in quantifying the strength assortativity of butterflies is investigated.

The evolution of two distributions are considered over sequential graph snapshots:

- $Pr(\delta)$, the probability distribution of strength difference for connected butterfly vertices which is computed as $Pr(\delta) = \frac{F(\delta)}{\Sigma F(\delta)}$, where $F(\delta)$ is the number of butterfly edges with strength difference δ and the sum runs over the range of δ values, and

- $Pr(S_i)$, the probability distribution of strength for butterfly i-vertices which is computed as $Pr(S_i) = \frac{F(S_i)}{\Sigma F(S_i)}$, where $F(S_i)$ is the number of butterfly i-vertices with strength S_i. The same notations hold for j-vertices and $Pr(S_j)$.

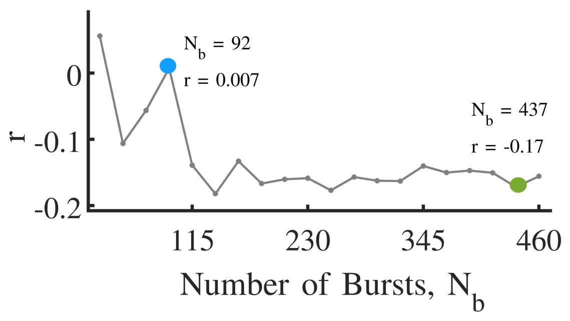
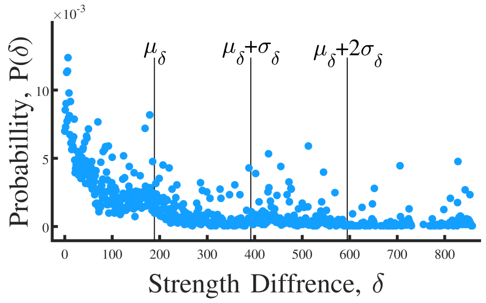
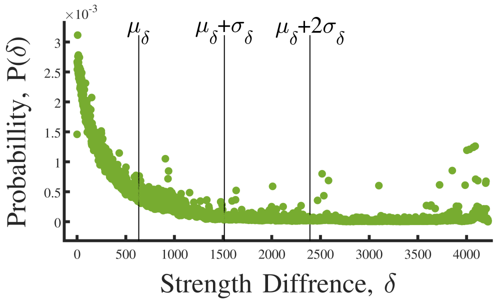


Figure 3.2: Assortativity coefficient over timeline of burst arrival in Epinions stream.

As a running example, the real-world stream Epinions is used with 20 equally-distanced points in the timeline of burst arrivals. At each point (N_b), r is calculated for the strengths of linked butterfly vertices in the corresponding graph snapshot G_{N_b} (Figure 3.2). Also, $Pr(\delta)$ is considered at two points corresponding to the arrival of 92 bursts (Figure 3.3(a)) and 437 bursts (Figure 3.3(b)). At $N_b = 92$, the probability that a butterfly edge has strength difference below the average strength difference μ_δ is $Pr(\delta \leq \mu_\delta) = 0.67$. However, the assortativity coefficient is $r = 0.007$ suggesting no (dis)assortativity (i.e. random connection of butterfly vertices with no tendency to connect to (dis)similar vertices). Also, at $N_b = 437$, majority of butterfly edges fall in the region behind μ_δ with probability $Pr(\delta \leq \mu_\delta) = 0.71$, while $r = -0.17$ suggests strength disassortativity.



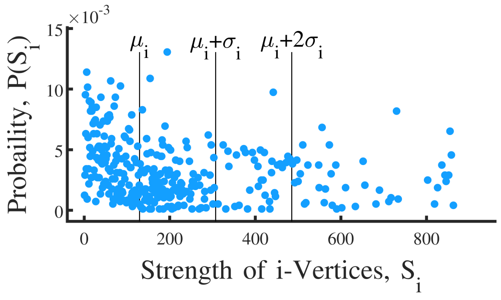
(a) $Pr(\delta)$ at $N_b = 92$



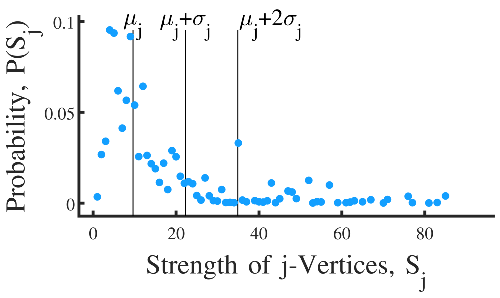
(b) $Pr(\delta)$ at $N_b = 437$

Figure 3.3: Distribution of strength differences of connected butterfly vertices.

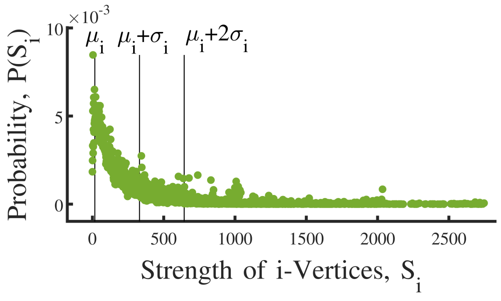
The reason behind this confusing behaviour of r is its bias toward the distribution of strength of i- and j-vertices with respect to their average. To clarify, consider $Pr(S_i)$ and $Pr(S_j)$ at these two time points (Figure 3.4(a),(b),(c),(d)). At $N_b = 92$, the probability that a butterfly i(j)-vertex has strength less than or equal to the average strength of butterfly i(j)-vertices $\mu_i(\mu_j)$ is almost equal to the probability that a butterfly i(j)-vertex has strength greater than the average strength of butterfly i(j)-vertices ($Pr(S_i \leq \mu_i) = 0.57$ and $Pr(S_j \leq \mu_j) = 0.54$). Therefore, many strength deviations from the mean strength, particularly for j-vertices, would be zero, making the coefficient an insignificant value close to zero ($r = 0.007$). At $N_b = 437$, a large majority of butterfly i(j)-vertices have strength above the average strength of butterfly i(j)-vertices ($Pr(S_i > \mu_i) = 0.9$, $Pr(S_j > \mu_j) = 0.8$), therefore their high deviations from the mean lowers the coefficient. In summary, the assortativity coefficient reflects the global correlation between $Pr(S_i)$ and $Pr(S_j)$ (two separate distributions). The assortativity coefficient fails to capture the pairwise correlations between strength of connected i- and j-vertices forming butterflies.



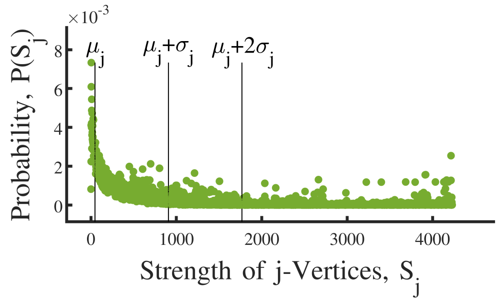
(a) $Pr(S_i)$ at $N_b = 92$



(b) $Pr(S_j)$ at $N_b = 92$



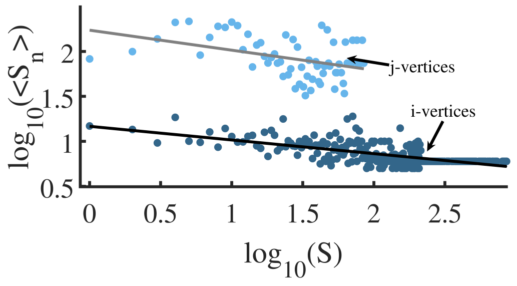
(c) $Pr(S_i)$ at $N_b = 437$



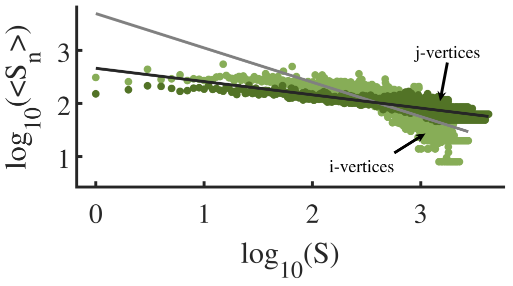
(d) $Pr(S_j)$ at $N_b = 437$

Figure 3.4: Distribution of strength of butterfly i-vertices and j-vertices.

Next, the neighbourhood-based approach for studying assortativity is examined. Figure 3.5, shows the nearest-neighbour average strength of vertices with strength S [260] at $N_b = 92$ and $N_b = 437$. At first glance, the decreasing trend suggests strength disassortativity: the higher the strength of a vertex, the lower the average strength of its neighbours and vice versa. However, we should consider the skewed $Pr(S_i)$ and $Pr(S_j)$ with high-strength vertices. Suppose that low-strength vertices are connected to many low-strength vertices and one high-strength vertex. In this case, the average strength of neighbours for these low-strength vertices would be high although the majority of neighbours have similar low strengths. That is, relatively few vertices that have high strengths (because of many connections and/or connections with high weights) skew the average strength of their low-strength neighbours and hence mislead the assortativity interpretation. This again highlights the issue of measuring strength assortativity in graphs with broad and skewed strength distribution. To conclude, using conventional assortativity metrics is not reliable for analyzing the strength assortativity of butterflies in bipartite streaming graphs since



(a) $N_b = 92$



(b) $N_b = 437$

Figure 3.5: Nearest-neighbour average strength of vertices with strength S.

- r is vertex-centric and reflects the global strength correlations rather than pairwise strength correlations. In particular, in case of computing r for butterflies, each butterfly vertex contributes duplicate values because of two adjacent edges; therefore, vertices with strength equal/close to the mean ($S - \mu \approx 0$) decrease the overall correlation, regardless of the strength of their neighbor.

- r is designed for unipartite graphs and using it in bipartite graphs can bias the outcome by the strength distributions of i- and j-vertices.

- The neighbourhood-based approach can be misleading in case of graphs with broad and skewed strength distributions since high-strength vertices have outlier impacts and make the interpretation difficult.

Informed by the above discussion, an appropriate measure for the tendency of vertices to connect to vertices with similar strength that is applicable to butterfly edges should satisfy the following properties:

- It should directly reflect the probability distribution of strength differences rather than the global correlations in the distribution of strengths.

- It should not be designed based on neighbour information since in case of skewed distribution of strengths, it would be biased by the outlier vertices.

- It should enable comparison of strength assortativity for sequential graph snapshots in the same stream as well as comparison of strength assortativity of graph snapshots in different graph streams.

The goal is to quantify and compare the distribution of strength differences in low dimension to enable temporal analysis over sequential graph snapshots of streams. A common approach for comparing distributions (usually degree distributions) is Kolmogrov-Smirnov test. However, this is sensitive to the distribution range and is not ideal for analyzing sequential graph snapshots and different graph streams. The Degree Distribution Quantification and Comparison (DDQC) approach [20] quantifies the degree distribution of a graph based on $4 \times 2^{\beta}$ regions in the degree distribution and uses this quantification for comparison. The regions are determined in two steps: first, the degree distribution is divided into four regions covering the intervals between five subsequent points: min(degree), $\mu - \alpha\sigma$, μ, $\mu + \alpha\sigma$, and $max(degree)$, where μ is the mean degree and σ is the standard deviation of degrees and α is a configurable parameter. Next, each region is divided into 2^{β} equal sub-regions, where β is the second configurable parameter. Given these regions in the probability distribution, a vector is constructed with $4 \times 2^{\beta}$ elements each representing the summation of probabilities in a corresponding region.

The probability distribution of strength difference of connected butterfly vertices $Pr(\delta)$ is considered given a graph snapshot G_{N_b}. Using graph snapshots corresponding to a burst-based landmark window (Definition 11) enables fair comparison of different graphs with different temporal characteristics. Inspired by the DDQC approach, $Pr(\delta)$ is divided into four regions based on the mean and standard deviation of δs (μ_δ and σ_δ, see Figure 3.3). As long as the first region covers the low δs, the number/coverage of other regions for the tail of right-skewed distribution is not important in mixing pattern analyses. Accordingly, the probability distribution is summarized as an embedding vector F with four elements ($\Sigma_{i=1,..,4} F_i = 1$). Each element corresponds to a region as below:

$$F_1 = \Sigma Pr(\delta), \forall \delta \leq \mu_\delta \tag{3.1}$$

$$F_2 = \Sigma Pr(\delta), \forall \mu_\delta < \delta \leq \mu_\delta + \sigma_\delta \tag{3.2}$$

$$F_3 = \Sigma Pr(\delta), \forall \mu_\delta + \sigma_\delta < \delta \leq \mu_\delta + 2\sigma_\delta \tag{3.3}$$

$$F_4 = \Sigma Pr(\delta), \forall \delta > \mu_\delta + 2\sigma_\delta \tag{3.4}$$

The vector F provides fine-grained information. Additionally, to express the strength assortativity as an scalar for simple network inference in temporal analyses, the **strength assortativity localization factor** is defined as $r^s = F_1 - 0.5$ to track the localization of δs (F) on the region behind mean (F_1). r^s lies in the range $[-0.5, 0.5]$. A positive r^s

highlights strength assortativity (i.e. vertices with similar strengths tend to connect to each other), negative r^s highlights strength disassortativity (vertices with dissimilar strengths tend to connect to each other) and a zero value corresponds to random strength mixing. $r^s = 0.5$ denotes perfect strength assortativity and $r^s = -0.5$ denotes perfect strength disassortativity.

3.2 Analysis, Phase 1

3.2.1 Temporal Motifs

Real-world graphs display rapid temporal evolution of the number of butterflies (Figure 3.6). To further investigate the growth pattern of butterfly frequency in these graphs, ten polynomial functions of degree one to ten are examined to fit the data points of temporal butterfly frequency evolution (black lines in Figure 3.6) and the best fitting function is picked (Table 3.5). The best fitting function satisfies three conditions:

- It has the lowest Root Mean Square Error (RMSE).
- It has the highest coefficient of determination (R^2).
- It is a non-decreasing function.

RMSE quantifies the estimation error, while R^2 quantifies the linear correlation between the estimated fitting function and the data points. Figures 3.7, 3.8, and 3.9 illustrate the best fitting function and its estimation errors (residuals) used in calculation of the RMSE. Note that high RMSE values are due to the increasing function giving rise to high residuals. The RMSE of different graphs are not compared; instead, the RMSE of different fitting functions for each graph are compared. Therefore, the absolute value of RMSE is not as important as its relative value for different functions. All the plots are properly fitted to polynomial functions of degree above 5 (best fitted to 5th, 7th, 9th and 10th degrees - Figures 3.7, 3.8, and 3.9). This is termed as the *butterfly densification power-law* (BPL, following the power-law terminology [201]): the number of butterflies at time point t (i.e. $B(t)$) follows a power law function of the number of edges at t (i.e. $B(t) \propto f(|E(t)|^{\eta}), \eta > 1$). Moreover, the outstanding frequency of butterflies in the real-world graphs compared to that of random graphs suggests that butterflies are network motifs *across the time line.*

Table 3.5: R^2 and RMSE of polynomial fitting functions of degree $d = 1$ to $d = 10$ for butterfly count in three real-world streams.
Filled cells decode increasing function and best fits are highlighted in gray cells.

R^2 RMSE	$d = 1$	$d = 2$	$d = 3$	$d = 4$	$d = 5$	$d = 6$	$d = 7$	$d = 8$	$d = 9$	$d = 10$
Epinions	0.9947 $1.481e^4$	0.9951 $1.435e^4$	0.9951 $1.432e^4$	0.9975 $1.028e^4$	0.9977 9751	0.9977 9716	0.9978 9598	0.9984 8130	**0.9987** **7409**	0.9987 7386
ML100k	0.931 $2.31e^6$	0.9977 $4.18e^5$	0.9978 $4.167e^5$	0.9978 $4.126e^5$	0.9983 $3.673e^5$	0.9983 $3.584e^5$	**0.9993** **$2.286e^5$**	**0.9993** **$2.286e^5$**	0.9997 $1.552e^5$	0.9997 $1.552e^5$
ML1m	0.8751 $2.119e^6$	0.9951 $4.196e^5$	0.9953 $4.111e^5$	0.9977 $2.895e^5$	0.9989 $1.976e^5$	0.9989 $1.961e^5$	0.999 $1.94e^5$	0.999 $1.937e^5$	0.999 $1.933e^5$	**0.999** **$1.933e^5$**
ML10m	0.8943 $3.223e^6$	0.9983 $4.034e^5$	0.999 $3.149e^5$	0.9992 $2.841e^5$	0.9993 $2.701e^5$	0.9993 $2.699e^5$	**0.9993** **$2.605e^5$**	0.9994 $2.493e^5$	0.9996 $1.868e^5$	0.9997 $1.781e^5$
Edit-FrWiki	0.9228 $8.09e^4$	0.9932 $2.408e^4$	0.9932 $2.397e^4$	0.9953 $1.998e^4$	**0.9966** **$1.693e^4$**	0.9968 $1.653e^4$	0.9979 $1.319e^4$	0.9988 $1.01e^4$	0.9988 9928	0.9989 9725
Edit-EnWiki	0.971 1990	0.9879 1288	0.9879 1285	0.9903 1150	0.9918 1060	0.9928 990	0.9951 821.3	0.9957 769.9	**0.9964** **696.5**	0.9967 671.7

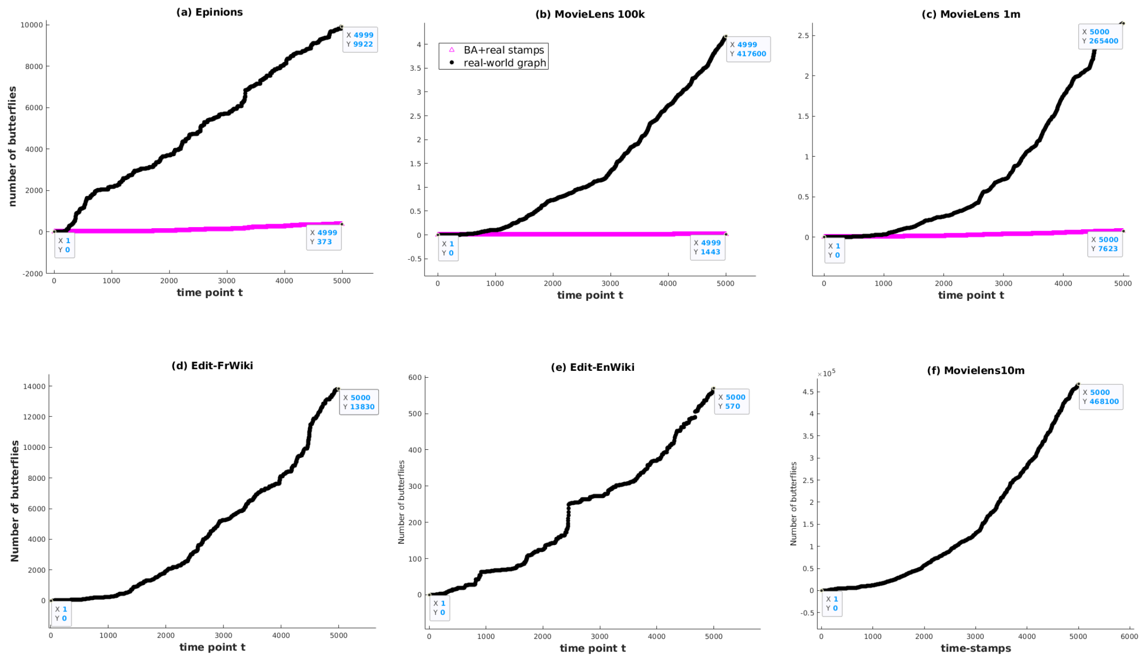


Figure 3.6: Temporal evolution of butterfly frequency.

3.2.2 Butterfly Emergence Patterns - Densification

Bursty Butterfly Formation. To study how butterflies as motifs are formed over time, the distribution of inter-arrival time of pair of edges forming a butterfly is studied. That

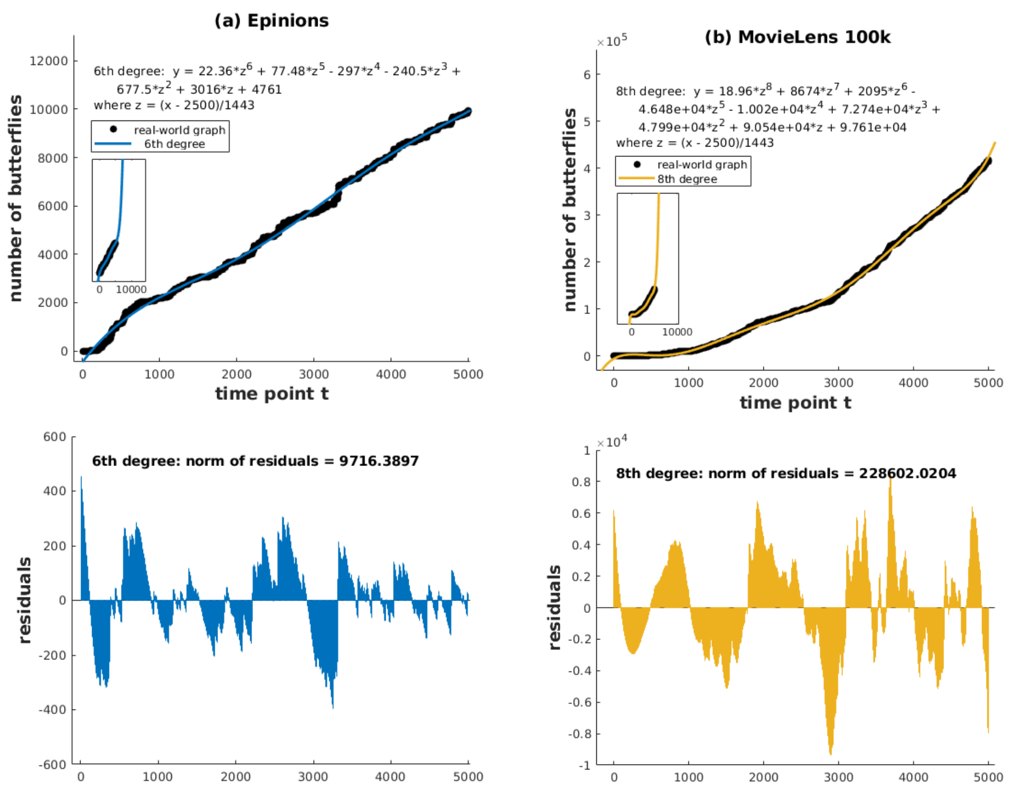


Figure 3.7: (top) Best Fitting functions for the temporal evolution of butterfly frequency and (bottom) the residual errors of the estimated fitting function in Epinions and ML100k streams.

is, for any pair of edges $\langle e_1, e_2 \rangle$ with timestamps τ_1 and τ_2 that co-exist in a butterfly, the inter-arrival time is $|\tau_1 - \tau_2|$. A lazy computation model is adopted to compute the inter-arrival distribution once at time point $t = 5000$ (i.e. after adding 5000 sgrs).

The distribution of inter-arrival values is skewed to the right (Figures 3.10 and 3.11). The left peaks and the heavy tail of the distribution reveal different patterns. The leftmost peaks highlight that many butterflies are formed by edges with close timestamps. On the other hand, according to Figure 3.6, the number of butterflies increase significantly over time. This suggests that *butterflies are formed in a bursty fashion.*

Next, the vertices that form the butterflies are investigated to see

- whether the bursty butterfly generation is contributed by hubs (i.e. vertices with

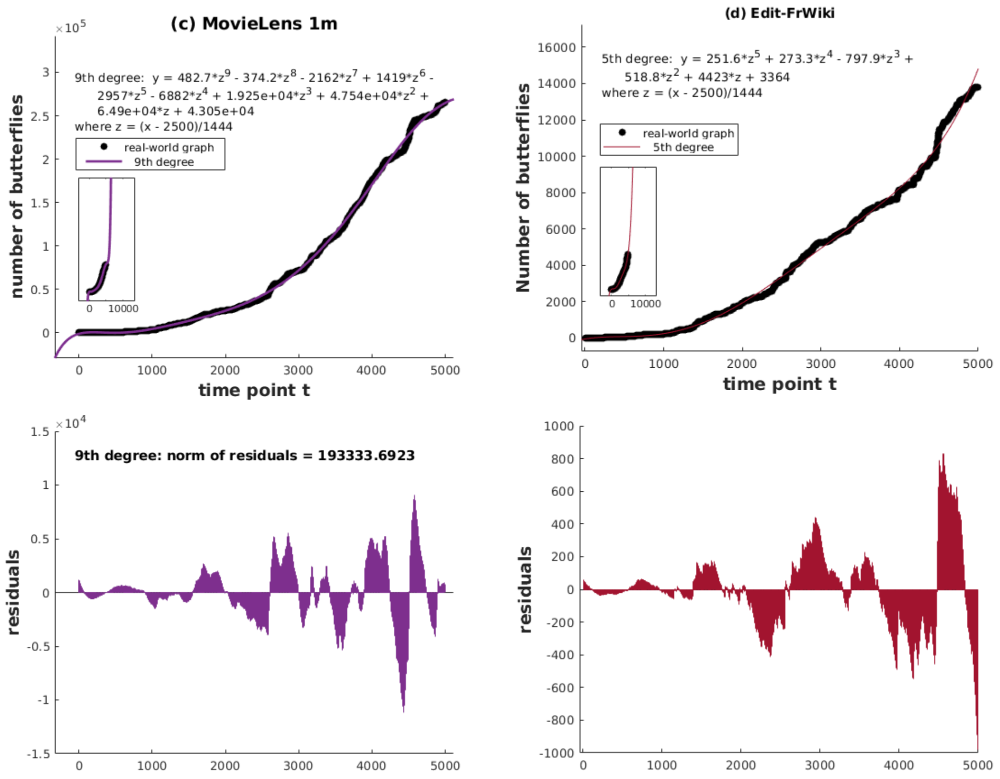


Figure 3.8: (top) Best Fitting functions for the temporal evolution of butterfly frequency and (bottom) the residual errors of the estimated fitting function in ML1m and Edit-FrWiki streams.

degree above the average of unique vertex degrees) or normal vertices; and

- if hubs are the main contributors, are they young, old, or both?

Hubs' contribution to butterfly emergence. The followings are studied to test the hypothesis that butterflies are contributed by hubs.

- The probability of forming butterflies by hubs
- The correlation between degree and support of vertices
- The connection patterns of hubs

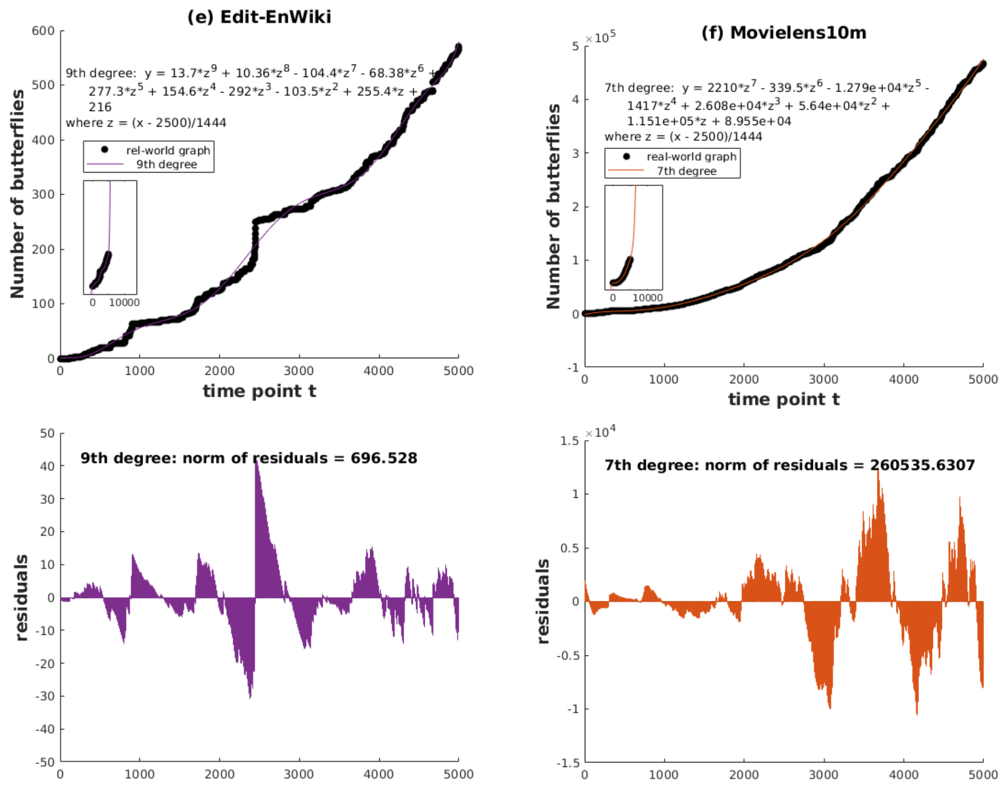


Figure 3.9: (top) Best Fitting functions for the temporal evolution of butterfly frequency and (bottom) the residual errors of the estimated fitting function in Edit-EnWiki and ML10m streams.

The probability of forming butterflies by hubs – Butterflies formed at time $t = 0$ to $t = 5000$ are enumerated and the fraction of butterflies formed by zero to four hubs (Table 3.6) and the fraction of butterflies formed by zero, one, or two i-/j-hubs (Table 3.7) are checked. It is evident that, butterflies mostly include one or, with higher probability, two hubs which are usually i-hubs.

The correlation between degree and support of vertices – The correlation between degree $deg(i)$ and butterfly support B_i is studied, where B_i is defined as the number of butterflies incident to each vertex. The correlation computed over the i-vertices and j-vertices is referred to as i-correlation and j-correlation (similarly computed), respectively. The Pearson correlation coefficient is computed at $t = 5000$ for all the $|V_i|$ or $|V_j|$ seen i-(j-)vertices in the graph snapshot. A positive correlation coefficient means $deg(i)$ and B_i

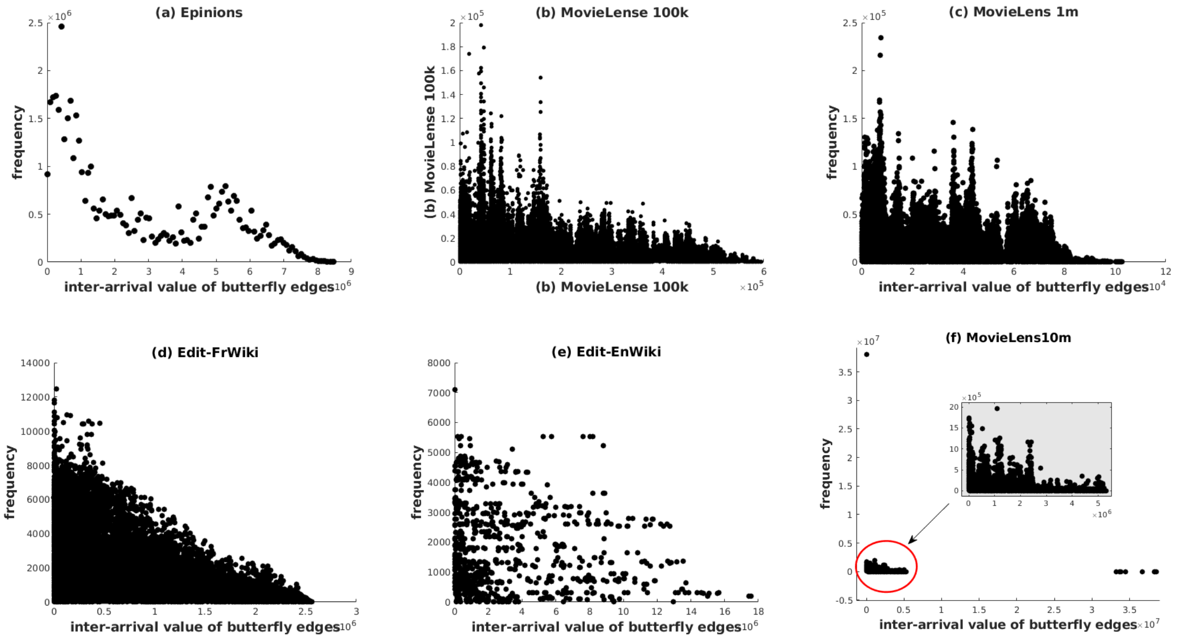


Figure 3.10: Distribution of inter-arrival time of edges forming butterflies in real-world graphs.

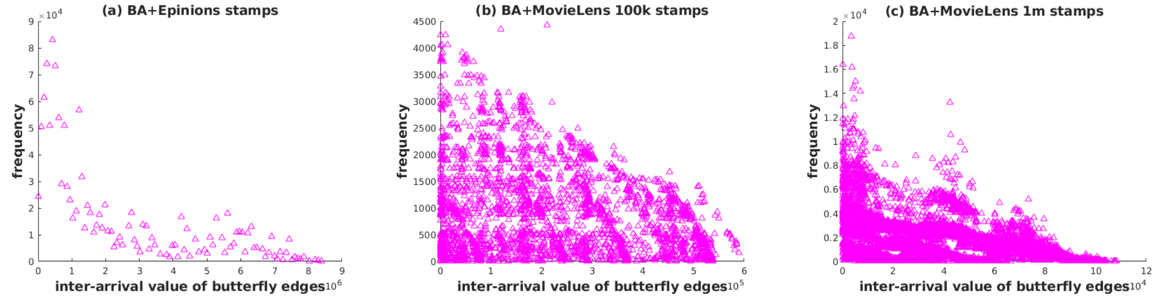


Figure 3.11: Distribution of inter-arrival time of edges forming butterflies in BA+real stamps graphs.

increase or decrease together, while a negative correlation means increasing one quantity implies decreasing the other one. Values close to 1 demonstrate strong correlation. As provided in Table 3.8, there is a strong positive correlation between the degree and the support of vertices in real-world graphs. i.e. the higher the degree, the higher the butterfly support and vice versa. This highlights the impact of hubs in the emergence of enormous

Table 3.6: Fraction of butterflies including zero, one, two, three, or four hub(s) after adding 5000 sgrs.

Fraction	0 hub	1 hub	2 hubs	3 hubs	4 hubs
Epinions	0.09	0.29	0.55	0.07	0
BA+Epinions stamps	0.11	0.44	0.39	0.06	0
ML100k	0.07	0.35	0.48	0.09	0.01
BA+ML100k stamps	0.24	0.28	0.28	0.15	0.05
ML1m	0.07	0.38	0.48	0.07	0
BA+ML1m stamps	0.01	0.33	0.6	0.06	0
ML10m	0.09	0.34	0.37	0.17	0.03
Edit-Frwiki	0.08	0.29	0.53	0.1	0
Edit-Enwiki	0.1	0.48	0.41	0.01	0

Table 3.7: Fraction of butterflies including zero, one, or two i-hub(s) or j-hub(s) after adding 5000 sgrs.

Fraction	0 i-hub	1 i-hub	2 i-hubs	0 j-hub	1 j-hub	2 j-hubs
Epinions	0.11	0.35	0.54	0.85	0.13	0.02
A+Epinions stamps	0.19	0.56	0.25	0.7	0.25	0.05
ML100k	0.10	0.46	0.44	0.75	0.21	0.04
BA+ML100k stamps	0.48	0.39	0.13	0.37	0.41	0.23
ML1m	0.1	0.43	0.47	0.84	0.15	0.01
BA+ML1m stamps	0.01	0.36	0.63	0.9	0.1	0
ML10m	0.25	0.54	0.21	0.47	0.33	0.2
Edit-Frwiki	0.11	0.35	0.54	0.81	0.18	0.01
Edit-Enwiki	0.1	0.5	0.4	0.97	0.03	0

number of butterflies in the real-world graphs.

The connection patterns of hubs – The extent to which i-(j-)hubs dominate the edges over time is quantified by means of two equivalent measures:

- the fraction of i-(j-)hub connections (denoted by $\frac{\sum_{i=1}^{N_{hub}(t)}(deg(hub_i))}{E(t)}$) normalized over the number of hubs at time point t (denoted by $N_{hub}(t)$), and
- the average degree of i-(j-)hubs (denoted by $\frac{\sum_{i=1}^{N_{hub}(t)}(deg(hub_i))}{N_{hub}(t)}$) normalized over the total number of edges at time point t (denoted by $|E(t)|$).

Both quantities are calculated by $\frac{\sum_{i=1}^{N_{hub}(t)}(deg(hub_i))}{E(t)*N_{hub}(t)}$ at any given time point t. An eager

Table 3.8: Correlation between the butterfly support and the degree of i-vertices (i-correlation) and j-vertices (j-correlation).

	i-correlation	j-correlation
Epinions	0.86	0.73
BA+Epinions stamps	0.56	0.72
ML1m	0.98	0.92
BA+ML1m stamps	0.92	0.89
ML100k	0.95	0.93
BA+ML100k stamps	0.63	0.88
ML10m	0.83	0.93
Edit-Frwiki	0.91	0.85
Edit-Enwiki	0.89	0.62

computation model is adopted to compute this value when a new edge is added. The time point t can be interpreted as the number of edges added to the graph since the initial time point $t = 0$.

While the number of edges added to the graph increases, the normalized fraction of i-(j-)hub connections (average degree of i-(j-)hubs) decreases over time in both real-world and BA graphs (Figures 3.12 and 3.13). Also, unlike real-world graphs, i- and j-hubs emerge later in the BA graphs (originated by the BA's preferential attachment rule), and the average degree of hubs in early time points is higher in real-world graphs than that of BA graphs. This is due to the bursty characteristic of graph stream (i.e. arrival of a bunch of edges with same time-stamp and same i- or j- vertex). In summary, early in the stream, the BA graphs have lower number of hubs with lower degrees compared to the real-world graphs. Figure 3.6 also illustrates the low number of butterflies in BA graphs earlier in the stream when there are no hubs in these graphs or the average hub degree is low. On the other hand, real world graphs have high number of hub connections and high number of butterflies. These observations again verify the contribution of hubs to the emergence of butterflies; When the number of hubs and the average degree of hubs are both low, the number of butterflies is also low (as seen in BA graphs). Also, when the number of hubs and their average degree is high, the number of butterflies is high (as seen in real-world graphs).

Contribution of hubs' age to butterfly emergence. The followings are studied to test the hypothesis that butterflies are contributed by old hubs.

- The evolution of young and old hubs

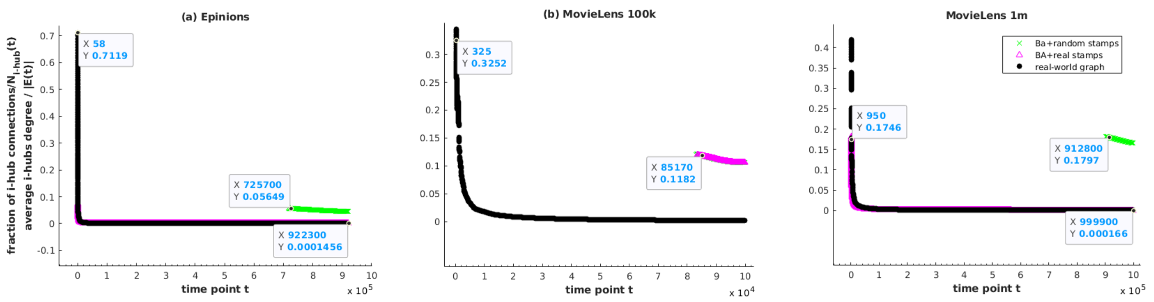


Figure 3.12: [Best viewed in colored.] Temporal evolution of the normalized fraction of i-hub connection (average i-hub degree).

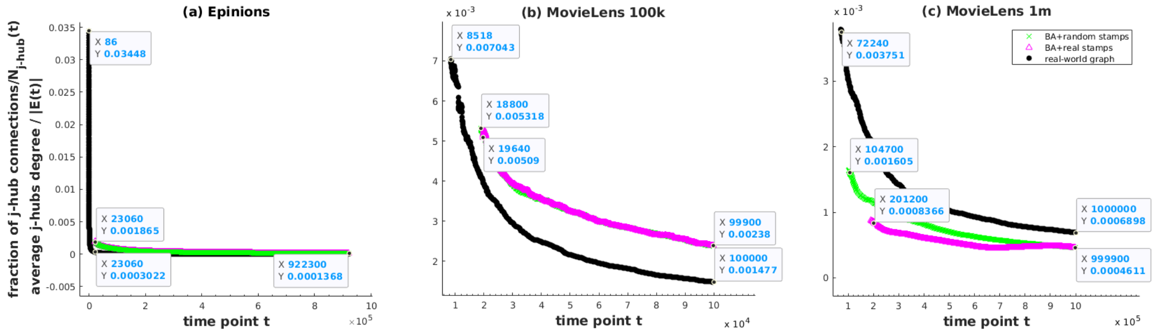


Figure 3.13: [Best viewed in colored.] Temporal evolution of the normalized fraction of j-hub connection (average j-hub degree).

- The inter-arrival of butterfly edges

The evolution of young and old hubs – As mentioned before, the i-(j-)hub are defined as any i-(j-)vertex whose degree is above the average of unique i-(j-)degrees in the graph. Accordingly, young (old) hubs are defined as any hub whose timestamp is in the last (first) 25% of ordered set of already seen timestamps. The vertex timestamps are determined as the timestamp of the sgr by which the vertex has been added to the graph for the first time. For instance, if a vertex i arrives via the inserting edges $e_1 = \langle i, j_1 \rangle$ and $e_2 = \langle i, j_2 \rangle$, the timestamp of vertex i is set to the timestamp of e_1, which has arrived before e_2 (assuming subscript identify order of arrival). A lazy computation model is adopted to compute the number of young/old i-(j-)hubs using a burst-based landmark window (Definition 11), where the computation is done over a growing graph generated by the edges in the append-only window following each expansion. Window expansion lengths are set to cover $0.1 * N_b$ unique timestamps in each window in Epinions, ML100k, ML1m, and ML10m. In the larger streams Edit-EnWiki and Edit-FrWiki, this value is equal to $0.01 * N_b$.

As shown in the Figure 3.14, young i-hubs and/or j-hubs are formed in the real-world graphs over time, while in BA graphs with random timestamps the number of young i-(j-)hubs is always zero. The timestamp of hubs in real graphs are randomly assigned to other vertices in BA graphs with real timestamps, therefore the old hubs are identified as young hubs that should be ignored. Figure 3.15 demonstrates that old hubs increase over time in BA graphs, which is not always the case for real-world graphs. Moreover the number of old hubs in real world graphs is less than that of BA graphs.

The inter-arrival of butterfly edges – Finally, the heavy tail of the inter-arrival distribution is studied, which is over-represented in BA graphs (Figure 3.11). The heavy tail is related to the butterfly edges with high inter-arrival times. These highly frequent butterfly edges with high inter-arrivals reflect the connection between the young vertices and old vertices. A hypothesis that young vertices are ordinary vertices and old ones are hubs (i.e. old hubs signify the bursty butterfly emergence) is proved due to the following. Young hubs can exist, however they are not the hubs dominating the butterflies.

- Hubs are main contributors to butterfly emergence; and
- The hubs forming the butterflies cannot be young hubs as BA graphs would be contradiction; BA graphs do not have young hubs (Figure 3.15), while they have many butterfly edges with high inter-arrival (Figure 3.6). Therefore, butterflies cannot originate from young hubs.

3.2.3 Summary

Butterflies are network motifs across the time line of sgr arrivals since the number of butterflies increases significantly over time in real-world streaming bipartite graphs, and at each time point the number of butterfly occurrences in real-world graphs are significantly higher than random graphs. This emergence of butterfly inter-connections is formulated as the *butterfly densification power law*, stating that the number of butterflies at any time point t is a power law function of the size of stream prefix seen until t. In terms of *how* these very large number of butterflies emerge over time, studies reveal the contribution of hubs in the streaming graphs. Further investigation of the impact of hubs in terms of their age reveal that the older hubs contribute more to the densification of butterflies.

3.3 Analysis, Phase 2

3.3.1 Butterfly Emergence Patterns - Strength Assortativity

Figure 3.16 shows the growth of butterfly count in real-world streams. To quantify this growth, the butterfly rate of each graph snapshot is defined as the number of butterflies in the graph normalized by the number of edges. The average butterfly rate (plus/minus the standard deviation) is computed over the sequential snapshots. The average butterfly rate is greater than 1 in all streams (Figure 3.16), as the number of butterflies in each graph snapshot is far higher than the number of edges and it can be calculated by a super-linear function of the number of edges (e.g. follows a power law $f(|E|^{\eta})$, $\eta > 1$ and the slope of the plots for butterfly count versus $|E|$ in the log-log scale is greater than 1 indicating that the number of butterflies grows super-linearly with respect to the number of edges in the sequential graph snapshots.) In some graphs the super-linearity starts after some time.

$Pr(\delta)$, the probability distribution of strength-difference of connected vertices in butterflies, is computed for the graph snapshots in the streams. Each probability distribution is embedded in a vector F. Figures 3.17 and 3.18 demonstrate the evolution of F elements and their corresponding strength assortativity localization factor (r^s) over the timeline of burst arrivals. In all streams, butterfly edges have strength-difference less than equal to the average strength-difference (μ_δ) with probability $Pr(\delta \leq \mu_\delta) \approx 0.7$ (F is localized on F_1). The tail of $Pr(\delta)$ for all graphs is heavier in the region $[\mu_\delta,\ \mu_\delta + \sigma_\delta]$ with probability of $Pr(\mu_\delta < \delta \leq \mu_\delta + \sigma_\delta) \approx 0.25$ (according to F_2 values) and gets lighter at the end. This demonstrates that the majority of butterfly edges are formed by vertices with similar strengths at all time points. Also, the strength assortativity localization factor is $0.15 \leq r^s \leq 0.2$ in all graphs at almost all time points (Figure 3.18).

Figure 3.19 shows the evolution of three statistical quantities for $Pr(\delta)$:

- mean μ_δ
- coefficient of variation $CV = \sigma_\delta / \mu_\delta$
- excess kurtosis $Y_2 = (N^{-1} \Sigma_{\delta_i} (\delta_i - \mu_\delta)^4 / \sigma_\delta^4) - 3$

Figure 3.20 shows the evolution of the same quantities for the probability distribution of strength of i- and j-vertices forming a butterfly, denoted as $Pr(S_i)$ and $Pr(S_j)$. CV, also known as relative standard deviation (RSD), enables measuring the degree of variation (dispersion) over distributions with different mean values. A high-variance distribution has CV>1 and a low-variance distribution has CV<1. Distributions such as exponential distribution with equal mean and standard deviation have CV=1. The excess kurtosis Y_2 enables measuring the heaviness of the tail of distribution relative to a normal distribution (which has Y_2=3). A heavy-tailed distribution has a positive Y_2 (called a *lepto-kurtic* distribution) and a light-tailed distribution has a negative Y_2 (called a *platykurtic* distribution). Distributions such as family of normal distributions have zero Y_2 (called *meso-kurtic*). The mean and standard deviation of strength-differences are equal to each other and evolve synchronously (Figure 3.19, CV≈1 for sequential G_{N_b}). On the other hand, the tail of right-skewed $Pr(\delta)$ gets heavier and the distribution gets broader (Figure 3.19, Y_2 increases). In Ciao, the tail gets lighter initially and then gets heavier. Moreover, all of the graphs have right-skewed $Pr(S)$ which gets broader and more skewed over time with the tail of strength distribution becomes heavier/longer over time (Figure 3.20, Y_2 and CV>1 increase). These observations make the steady behavior of strength assortativity more interesting: despite the fact that new high-strength vertices form butterflies and $Pr(\delta)$ gets broader, the relative standard deviation of δs does not change significantly and the strength assortativity localization factor r^s remains steadily positive. This implies that these graphs obey non-trivial mixing patterns.

The following concurrent mixing patterns hold in the real-world streams as butterflies emerge over time:

1. *Butterfly densification.* The number of butterflies grows over time and at each time point it is a super-linear function of the number of edges.

2. *Strength diversification.* $Pr(S)$ of butterflies is initially meso-kurtic and gets more right-skewed as the right tail grows heavier/longer (Y_2 starts from 0 and rises to extremely high values). The dispersion of strengths increases over time (CV>1 increases) as the standard deviation increases and the mean decreases.

3. *Steady strength assortativity.* The strength assortativity localization factor r^s is fixed at a positive value over time due to the fixed-shaped yet growing distribution of strength-differences of butterflies. $Pr(\delta)$ is initially meso-kurtic and gets more right-skewed as the right tail grows heavier/longer (Y_2 starts from 0 and rises to extremely high values). However, the dispersion of strength-differences does not change (CV≈1) due to synchronous evolution of mean and standard deviation. Also, the proportion of δs in different regions of $Pr(\delta)$ is constant (stable F elements). Therefore, the shape of the distribution is stable although the range expands.

The following graph concepts explain the data-driven semantics of the observed patterns in the domain of user-item rating streams.

- Burstiness. User-item interactions can be viewed as human-initiated events which introduce two levels of burstiness: individual-level and group-level. The former relates to the interactions of each user with several items at each time point or sequential time points with negligible differences. Barabasi [18] has shown that human-initiated events are driven by the queuing processes of human decision making leading to non-Poisson inter-event statistics. Such bursty interactions lead to formation of many wedges incident to each user/item. The latter relates to the concurrent interactions of several users at each time point. Such bursty interactions lead to merging the individual-level wedges and densification of butterflies. The significance of this continuous burstiness can change over time due to different circumstances leading to peak hours. For instance, Alibaba has reported that customer purchase activities during a heavy period in 2017 resulted in generation of 320 PB of log data in a six hour period [251, Big Data Processing]. There are other studies [229, 371], [251, Stream Data Management] showing that weighted bipartite streaming graphs display bursty patterns since eight additions in temporal graphs follow bursty patterns and data streams are commonly characterized as bursty.

- Strong-get-stronger. Online platforms utilise filters such as trends, best sellers, mostly viewed, hot/top categories, newly added, as well as timely promotions, point collection rewarding strategies, and (advertised) recommendations. These systematically lead to the increasing popularity and visibility of the items with most interactions and encouraging the users to interact more and become more active. Such interaction mechanics are similar to the rich-gets-richer argument, where the richness denotes the vertex strength. The butterflies are formed incident to such highly connected and high strength users/items (strong vertices) leading to butterfly densification and diversification of strengths.

- Core-periphery. Popular items attract the active users and in another view, active users mostly engage with trending items or make items trending/popular. This is similar to the mesoscale phenomenon 'core-periphery' [154, 99] also called 'rich club' [375, 95] stating that high-degree vertices tend to connect to each other and create a core attracting the new connections. Such core sets of vertices with high degrees/strengths in user-item streams create numerous edges between strong users and items with high butterfly support leading to assortativity patterns of butterfly vertices.

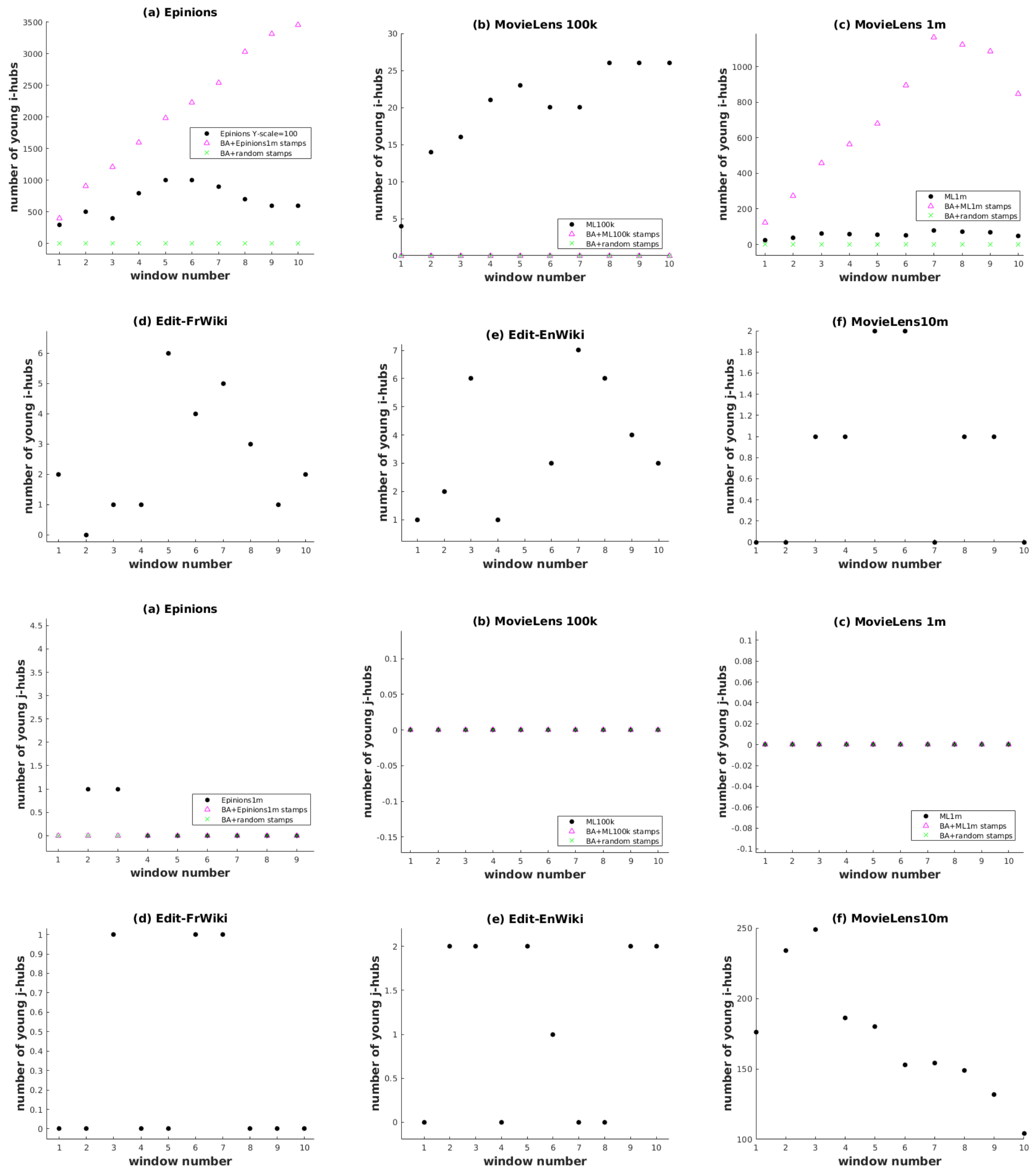


Figure 3.14: The number of young (top 6) i-hubs and (bottom 6) j-hubs after arrival of each batch of sgrs.

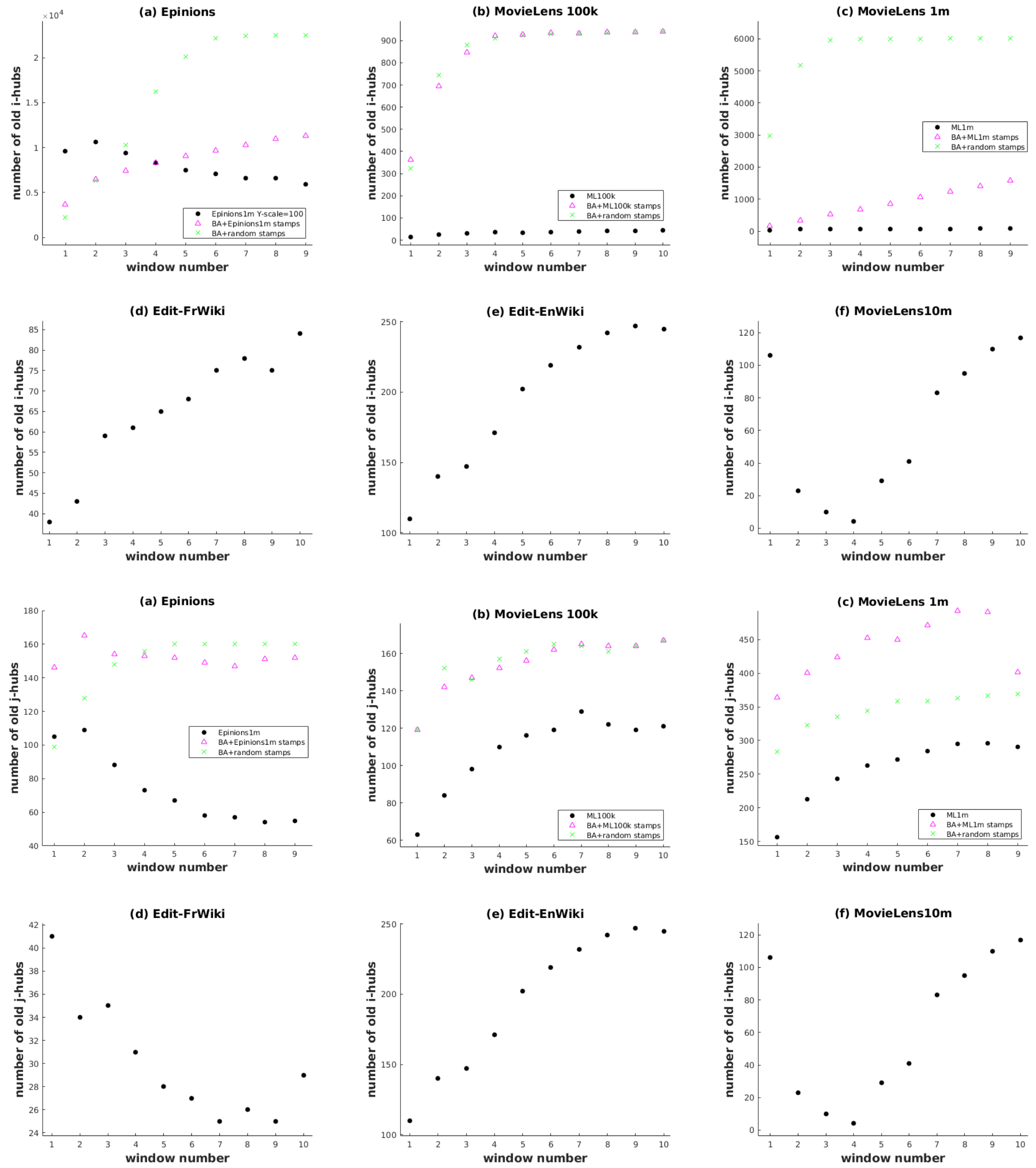
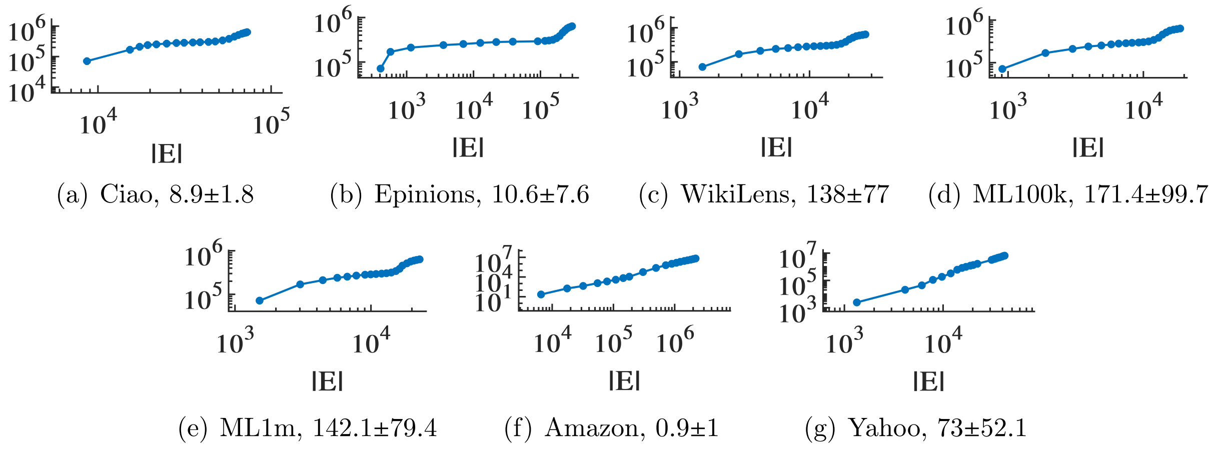


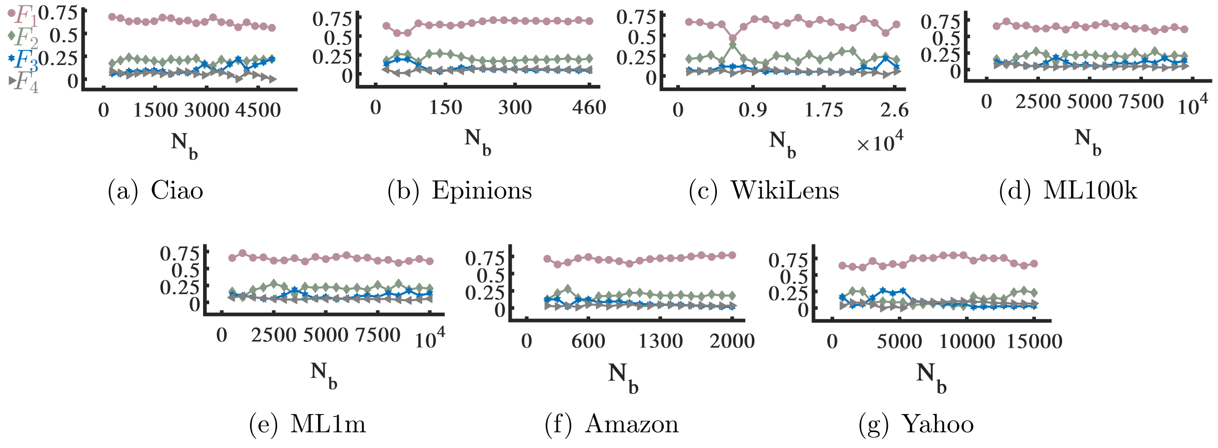
Figure 3.15: The number of old (top 6) i-hubs and (bottom 6) j-hubs after arrival of each batch of sgrs.



(a) Ciao, 8.9±1.8 (b) Epinions, 10.6±7.6 (c) WikiLens, 138±77 (d) ML100k, 171.4±99.7

(e) ML1m, 142.1±79.4 (f) Amazon, 0.9±1 (g) Yahoo, 73±52.1

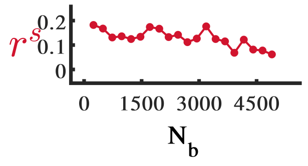
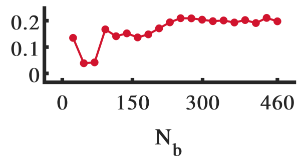
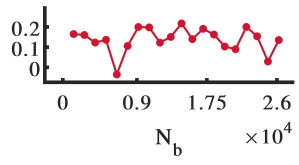
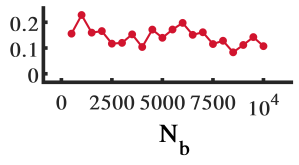
Figure 3.16: Butterfly count versus edge count in real-world streams with various average butterfly rates.



(a) Ciao (b) Epinions (c) WikiLens (d) ML100k

(e) ML1m (f) Amazon (g) Yahoo

Figure 3.17: F elements over the timeline of burst arrivals in real-world streams.


(a) Ciao

(b) Epinions

(c) WikiLens

(d) ML100k

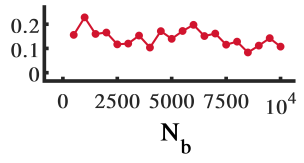
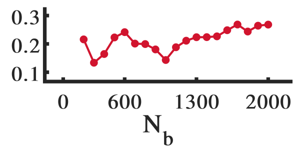
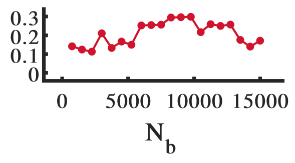
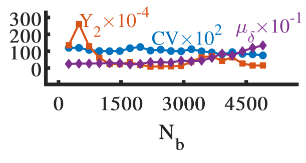
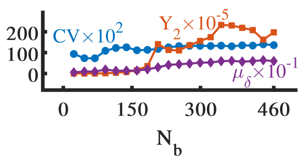
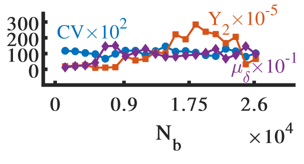
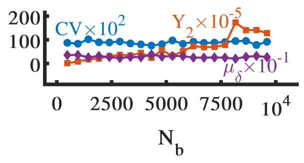

(e) ML1m

(f) Amazon

(g) Yahoo

Figure 3.18: Strength assortativity localization factor (r^s) of butterflies over the timeline of burst arrivals in real-world streams.


(a) Ciao

(b) Epinions

(c) WikiLens

(d) ML100k

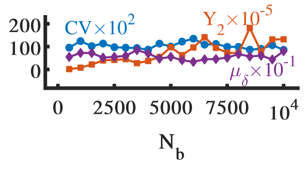
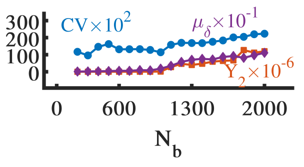
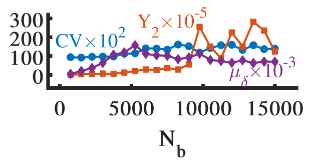
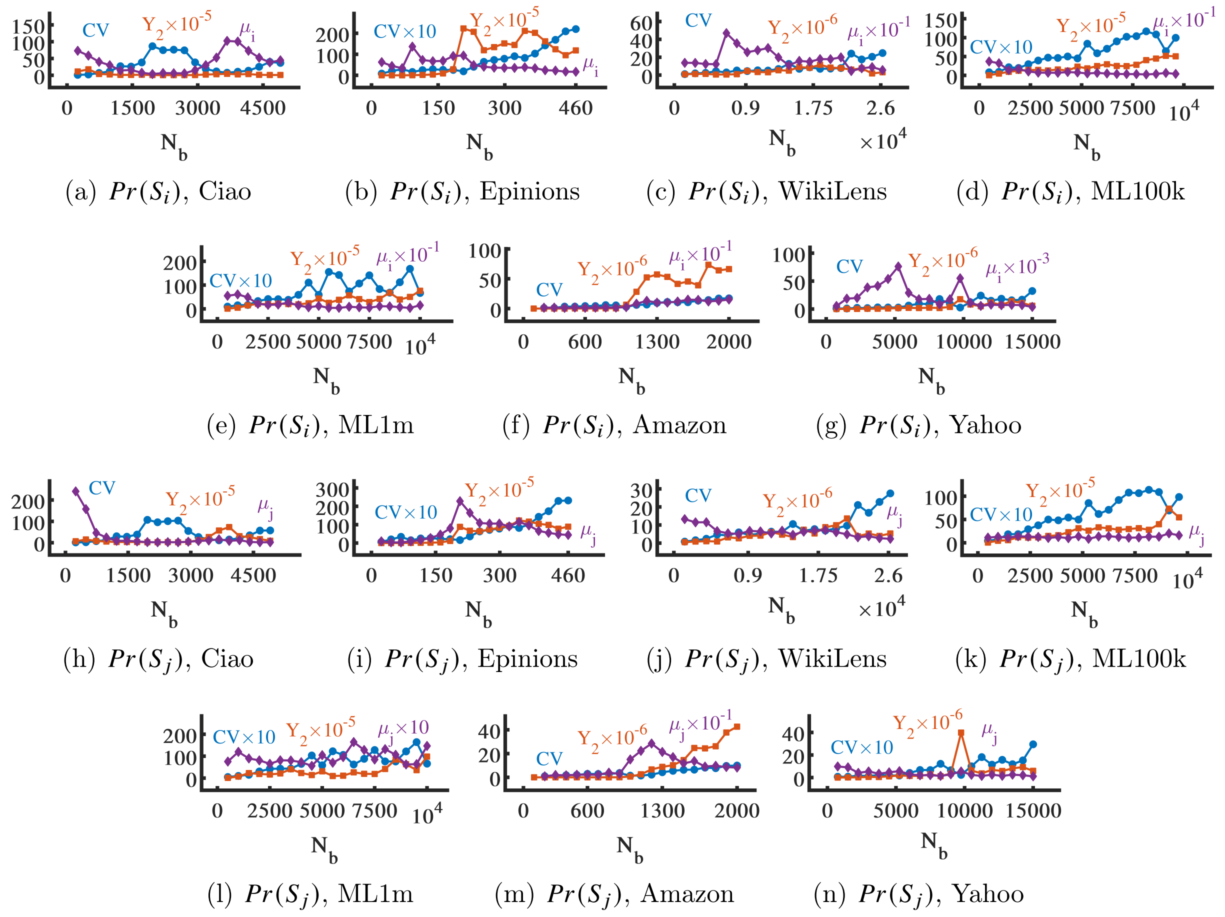

(e) ML1m

(f) Amazon

(g) Yahoo

Figure 3.19: Coefficient of variation (circles), excess kurtosis (squares), and mean (diamonds) of butterfly strength-differences over the timeline of burst arrivals in real-world streams.



(a) $Pr(S_i)$, Ciao (b) $Pr(S_i)$, Epinions (c) $Pr(S_i)$, WikiLens (d) $Pr(S_i)$, ML100k

(e) $Pr(S_i)$, ML1m (f) $Pr(S_i)$, Amazon (g) $Pr(S_i)$, Yahoo

(h) $Pr(S_j)$, Ciao (i) $Pr(S_j)$, Epinions (j) $Pr(S_j)$, WikiLens (k) $Pr(S_j)$, ML100k

(l) $Pr(S_j)$, ML1m (m) $Pr(S_j)$, Amazon (n) $Pr(S_j)$, Yahoo

Figure 3.20: Coefficient of variation (circles), excess kurtosis (squares), and mean (diamonds) of strengths of butterfly (a-g) i-vertices and (h-n) j-vertices over the timeline of burst arrivals in real-world streams.

3.3.2 Summary

Given two sequential graph snapshots G_{N_b,t_1} and G_{N_b,t_2}, the followings hold.

- The number of butterflies grows according to a super-linear function of the number of edges (e.g. follows a power law function $f(|E|^{\eta})$, $\eta > 1$ and $B_{G_{N_b,t_1}} < B_{G_{N_b,t_2}}$).
- $Pr(S)$ gets broader and more skewed ($\mu_S^1 < \mu_2^2$, $Y_2^1(S) < Y_2^2(S)$, $1 < CV^1(S) < CV^2(S)$).
- $Pr(\delta)$ gets broader and more skewed while remaining fixed-shaped ($Y_2^1(\delta) < Y_2^2(\delta)$, $CV^1(\delta) \approx CV^2(\delta)$, $CV(\delta) \approx 1$, $F_i^1 = F_i^2$, $i = 1, .., 4$, $r^s > 0.1$).

The co-occurrence of these patterns is counter-intuitive and interesting. As the stream and the number of butterflies grow rapidly, diversity of strengths for butterfly vertices increases and strong (high-strength) vertices get stronger and obtain weak neighbours with the increasing of variance of strength differences. Therefore, an increasing trend of disassortativity is expected. However, the majority of butterfly edges are formed by vertices with similar strength and this assortativity remains at a fixed level regardless of stream size or butterfly count. This phenomenon is referred to as *scale-invariant strength assortativity of streaming butterflies*, which is originated by the three parallel mixing patterns of butterflies.

Chapter 4

Streaming Graph Modelling

Explainable modelling of (bipartite) streaming graphs requires identifying the growth patterns and devising local rules to pinpoint the generative origins. Chapter 3 discussed analysis of real-world bipartite streaming graphs and presented the investigations into the emergence patterns of butterflies as the building blocks of these streams, concluding in a characteristic phenomenon called "scale-invariant strength assortativity of streaming butterflies". This chapter focuses on the next step of devising local rules. Local rules (also referred to as micro-mechanisms) are attachment mechanisms based on connections to vertices and their neighbourhoods [326]. The goal is to answer the question *What is the generative process underlying this phenomenon*? The question is addressed in four phases: Phase 1: reviewing the existing and seminal local rules to investigate their ability to explain the patterns and identify new local rules; Phase 2: analysis of the synthetic streams generated by the candidate local rules; Phase 3: introducing a streaming growth model called *sGrow* to implement the micro-mechanisms reproducing the realistic growth patterns; and Phase 4: Evaluating the performance of the introduced model. *sGrow* is the first model that explains real-life patterns effectively and reproduces them efficiently. Table 4.1 lists the frequent notations in this chapter.

4.1 Related Works

To explain the co-occurrence of butterfly densification, strength diversification, and steady strength assortativity in streaming graphs, the local rules and graph models leading to skewed distributions, degree correlation, and emergence of large numbers of cliques are reviewed in the following.

Table 4.1: Frequent notations in this chapter

Notation	Description
t	time step in *sGrow*
m	Number of new edges at each iteration, batch size in *sGrow*, Degree of new vertices in preferential attachment
M	Maximum batch size in *sGrow*, a parameter
L	length of Preferential Random Walk in *sGrow*, a parameter
ω	A random integer for operation selection in *sGrow*
p	Connection probability in Connecting Nearest-neighbour Model
p_h	Probability of selecting a host vertex in Butterfly Model
p_{step}	Probability of traversing a random walk in Butterfly Model
p_l	Probability of connections in Butterfly Model
p_e	Probability of edge removal in Duplication Divergence Model
p, p_b	Forward/backward probability in Forest Fire model

4.1.1 Graph Patterns

Graph patterns characterize a microscopic, mesoscopic, or macroscopic property of a graph (depending on the granularity of the reporting pattern, i.e. vertices/edges, neighbourhoods and motifs, or the entire topology) and can be viewed as either static or dynamic (depending on the underlying graph being a static snapshot or an evolving structure). Examples of static patterns include small diameter accompanied by high clustering coefficient (CC) [340], degree (anti)correlation [241], community structure [128], and power-laws (PL) such as degree distribution PL [38], weight PL [229], and snapshot/vertex strength PL [229, 41]. Examples of dynamic patterns include gelling points [229], increasing average degree, shrinking/controlled diameter, edge densification [201, 202, 119], and bursty weight addition [229]. Table 4.2 provides instances of different patterns partitioned across dynamism and granularity.

4.1.2 Growth Models

Two well-studied network growth mechanisms, preferential attachment and copying, form the basis of many generative models (e.g. [105, 97, 59, 9, 29, 142, 188, 185, 147, 259]) and are widely adopted in the development of graph management approaches (e.g. [159, 85, 221, 168, 213, 236, 351]). Both mechanisms are commonly applied in graph models based on the conception of adding a new vertex at each time step during an iterative process. Preferential attachment leads to skewed distributions, while copying mechanism

Table 4.2: Graph Patterns.

Patterns	Granularity	Dynamism	References
Small diameter & High CC	Macroscopic	Static	[340]
Degree Correlation	Microscopic	Static	[241]
Community structure	Mesoscopic	Static	[128]
Degree Distribution PL	Microscopic	Static	[38]
Weight PL	Microscopic	Static	[229]
Snapshot/Vertex Strength PL	Mesoscopic	Static	[229, 41]
Gelling points	Macroscopic	Dynamic	[229]
Increasing Average Degree	Microscopic	Dynamic	[201, 202]
Shrinking/Controlled Diameter	Macroscopic	Dynamic	[201, 202, 119]
Edge Densification	Microscopic	Dynamic	[201, 202]
Bursty Weight Addition	Microscopic	Dynamic	[229]

leads to degree correlation [326] and emergence of large numbers of cliques when applied explicitly [188] or implicitly and among other mechanisms [201]. In the following, these mechanisms and their alternatives and extensions are reviewed.

Barabasi-Albert model [38] starts with a small clique with m_0 vertices and applies the preferential attachment by connecting the new vertex to $m \leq m_0$ existing vertices selected randomly with probability proportional to their degrees. The preferential attachment rule has also been extended to strength-driven preferential attachment (SPA) where each new vertex is connected to m existing vertices randomly selected with probability proportional to their strength [42, 41, 204]. It has been shown that preferential attachment is induced by the following microscopic mechanisms. All of these mechanisms imply that the probability that a vertex receives a new edge is proportional to its degree, therefore they amount to preferential attachment and lead to scale-free structures [16].

- Copying [182, 188]: at every time step, a new vertex is connected to a constant number of vertices and the end point of each new edge is a randomly selected vertex with probability p or a neighbour of a prototype vertex with probability $1-p$.

- Edge redirection [186]: at every time step, a new vertex is added and a directed edge from the new vertex to a randomly selected vertex is created with probability $1 - p$, or the edge is redirected to the ancestor of the randomly selected vertex with probability p.

- Random walks [325]: at every time step, a new vertex is connected to a random vertex and the vertices reachable from it through breadth-first traversal with probability p until no new target is found.

- Attaching to edges [107]: at every time step, a new vertex is connected to two connected vertices.

The original version of copying, mentioned above, copies a neighbour of a randomly selected vertex with some probability at each time step. Other works have modified it as follows.

- Butterfly model [229] mixes copying and random walk mechanisms: at every time step, with probability p_{host} a new vertex picks a random vertex called host and with probability p_{link} forms edges with the vertices reachable from the host through a probabilistic random walk with traversal probability p_{step}. This model exhibits shrinking diameter, stabilized next-largest weakly connected component size, and edge densification.

- Growing network model with copying [185] connects the new vertex to a randomly selected vertex as well as its neighbours which leads to sparse ultra-small graphs with logarithmic growth of the average degree with respect to the number of vertices while the diameter equals 2.

- Duplication divergence model [326] removes the copied neighbours with some probability leading to power law decay of clustering coefficient as a function of degree.

- Nearest neighbours model [326] connects the new vertex to one randomly selected vertex and copies one neighbour with some probability leading to clustering coefficient power law and correlation between average neighbour degree and vertex degree.

- Forest Fire model [201, 202] applies the copying process by recursively connecting each new vertex to a randomly selected vertex (called ambassador) and certain numbers of its randomly selected out- and in-neighbours with forward probability p and backward probability p_b. This process leads to heavy-tailed in- and out-degree distributions due to an implicit preferential attachment, community structures due to neighbour copying mechanism [41], edge densification due to many internal-edge establishments, and shrinking diameter due to shortcut-edge establishments.

4.2 Analysis of Microscopic Growth Mechanisms

As discussed in previous section, preferential attachment leads to skewed distributions, and copying mechanism (particularly with the implementation scheme of Forest Fire model) leads to degree correlation and emergence of cliques and edge densification as well. Therefore, in an attempt to explain the origins of the observed patterns in real-world streams, the properties of synthetic streams generated by these local rules are investigated.

4.2.1 Data

Weighted bipartite streaming graphs are synthesized such that the graph structure grows according to the Forest Fire (FF) and strength preferential attachment (SPA) models. To this end, directed graphs are created via the growth models and the source vertices are treated as the i-vertices and destination vertices as the j-vertices. For the timestamp assignment, the time step at which new vertices are connected to existing vertices are used and for the weight assignments, random integers in the range $[1, 5]$ (the same weight scale as in real-world streams) are used. In FF model, when the backward-burning probability p_b is fixed and the forward-burning probability p increases, the graphs become denser and more clique-like with low diameter [202]. Therefore, graphs with fixed $p_b = 0.3$ and $p = 0.15$ (sparse region), 0.4 (transition region), and 0.7 (dense region) are generated. Experiments show that most of the edges are burned (visited) after checking the neighbours of the ambassador vertex, therefore no further edge is checked. This reduces computations and also allows addition of new external links beside the internal densification. In the SPA model, $m \in \{10, 50, 100\}$ is used since the average degree of vertices in real-world streams are mostly below 100 (Table 3.3). The same analysis approach as for real-world streams in Chapter 3 is used to investigate the emergence patterns of butterflies in the synthetic streams quantitatively (by checking the growth patterns of butterfly count) and qualitatively (by checking the assortativity patterns of butterflies and the confounding distributions).

4.2.2 Analysis

Figures 4.1-4.6 show the mixing patterns in FF and SPA streams. In the following, these patterns are investigated.

The butterfly count has a slow growth in FF streams and a speedy growth in SPA streams. The average butterfly rate is less than 1 in FF streams and higher than 500 in

SPA streams. That is, the growth of butterfly count with respect to the number of edges in the sequential graph snapshots is sub-linear in FF streams and extremely super-linear in SPA streams.

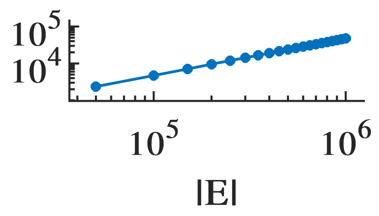
As the FF graph grows, in the transition region, the assortativity level fluctuates, and, in the sparse and dense regions, it changes trivially (0.02).

Although in FF stream butterflies emerge such that the range of $Pr(S_i)$ and $Pr(S_j)$ get broader and more skewed over time, strength-difference of butterfly edges retain the same distribution. $Pr(\delta)$ remains unchanged with a low dispersion (CV<1) as the graph grows in each region since the mean, and standard deviation are fixed and the tail changes slightly over time. Therefore, it is not surprising that r^s is stable. Moreover, in the denser graphs with more butterflies, assortativity patterns vanish ($r^s \rightarrow 0$ as p increases).

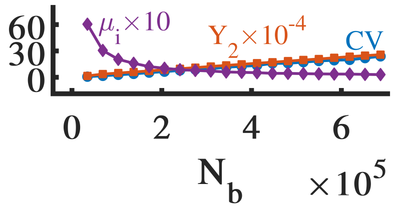
The evolution of r^s and the corresponding F elements in SPA streams shows that, for small values of m, there is no assortativity pattern. For $m = 100$, the graph snapshots display weak assortativity ($0.05 \leq r^s < 0.1$). The statistics of the corresponding $Pr(\delta)$s and that of $Pr(S_i)$s and $Pr(S_j)$s show that, as the graph grows, for all values of m, diversity of the strengths of i- and j-vertices do not change significantly (small change in μ_i, μ_j, and corresponding CV and Y_2). The strength-differences continuously follow a skewed distribution with a short tail as most δs remain around the mean ($F_1 + F_2 \approx 0.85$ and CV<1) and the skewness does not grow to very high numbers.

4.2.3 Summary

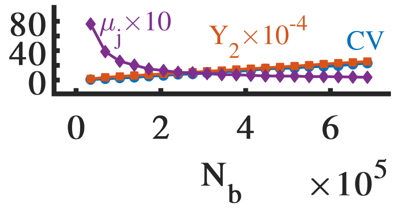
FF streams follow strength diversification but not butterfly densification and steady strength assortativity. In FF streams, as the new vertices attach to random vertices (ambassadors) and reach high-degree vertices (hubs) through copying the neighbours of ambassador, new butterflies emerge and the diversity of strength of butterfly vertices increases. When the probability of neighbour copying p is low (sparse regions), the new vertex establishes fewer connections, therefore, the probability of connecting to the high-strength hubs is lower, and also the strength of the new vertex remains low. As a result, many edges have low strength-difference, $Pr(\delta)$ is broader, and strength assortativity localization factor is positive. On the other hand, when p is high, although the number of butterflies is higher, the connections are established between pairs of vertices with both low and high strength-difference, since the ambassadors and their neighbours are selected uniformly at random. As a result, $Pr(\delta)$ has a lower variance and the strength assortativity displays randomness. Moreover, the butterfly count is a sub-linear function of the number of edges over time even when the graph displays edge densification (in dense regions).



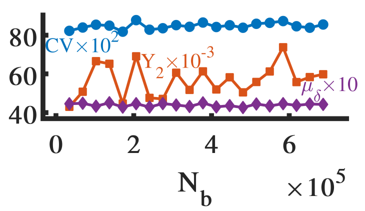
(a) Butterfly count



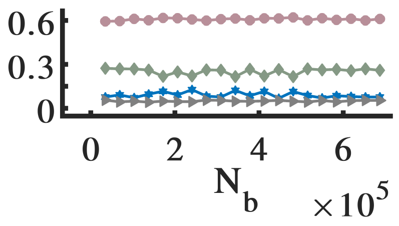
(b) CV, μ_i, Y_2 of $P(S_i)$



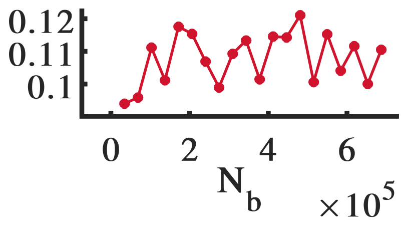
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

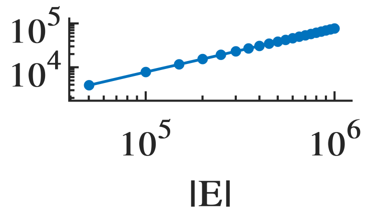


(e) F_1, F_2, F_3, F_4

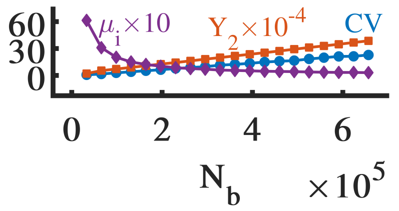


(f) r^s

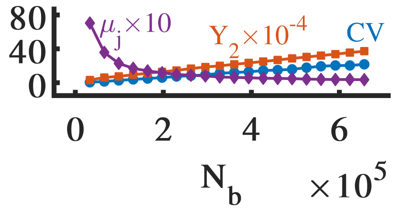
Figure 4.1: Mixing patterns of butterflies in FF stream with p=0.15 and average butterfly rate 0.04±0.



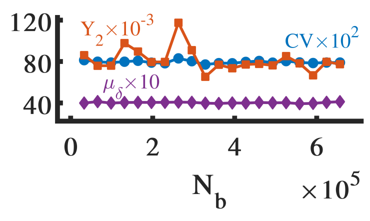
(a) Butterfly count



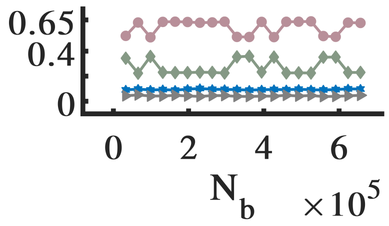
(b) CV, μ_i, Y_2 of $P(S_i)$



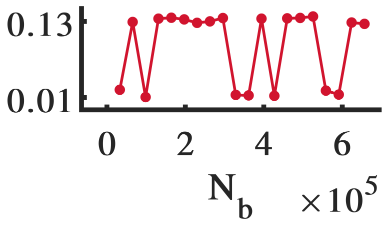
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

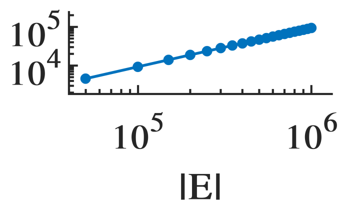


(e) F_1, F_2, F_3, F_4

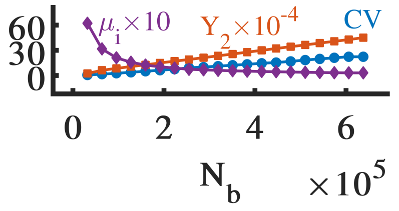


(f) r^s

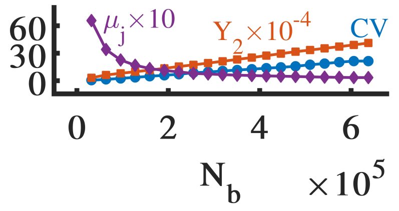
Figure 4.2: Mixing patterns of butterflies in FF stream with p=0.4 and average butterfly rate 0.08±0.



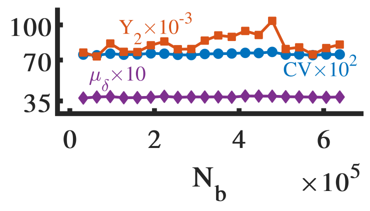
(a) Butterfly count



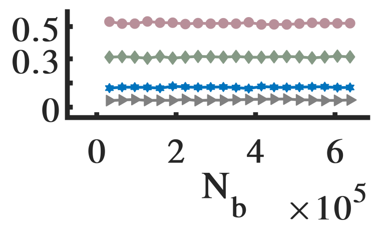
(b) CV, μ_i, Y_2 of $P(S_i)$



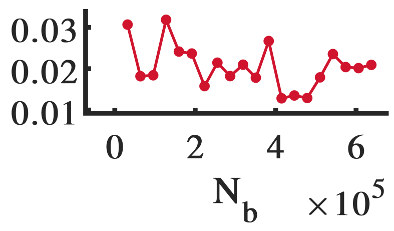
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

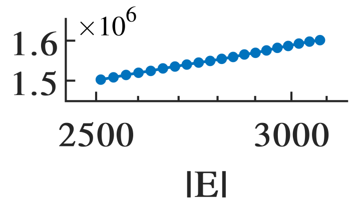


(e) F_1, F_2, F_3, F_4

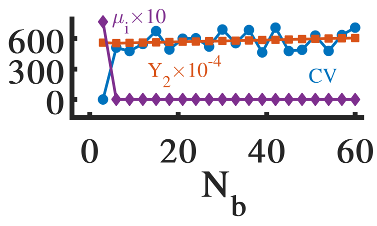


(f) r^s

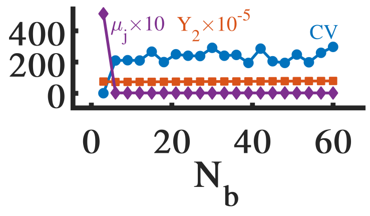
Figure 4.3: Mixing patterns of butterflies in FF stream with p=0.7 and average butterfly rate 0.09±0.



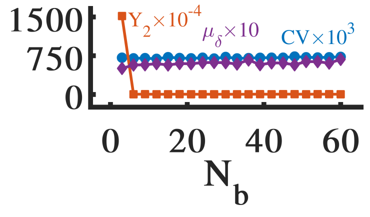
(a) Butterfly count



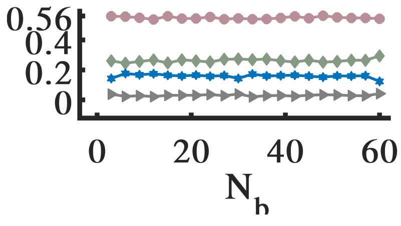
(b) CV, μ_i, Y_2 of $P(S_i)$



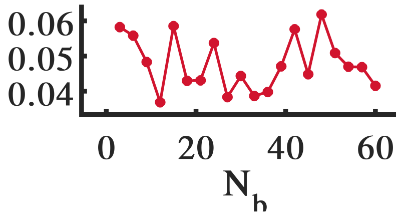
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

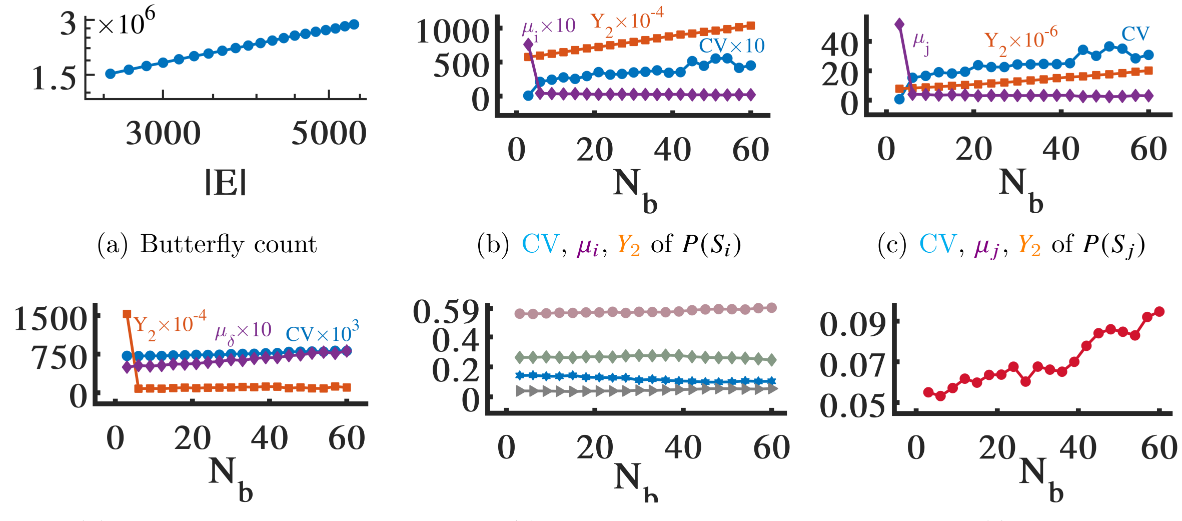


(e) F_1, F_2, F_3, F_4



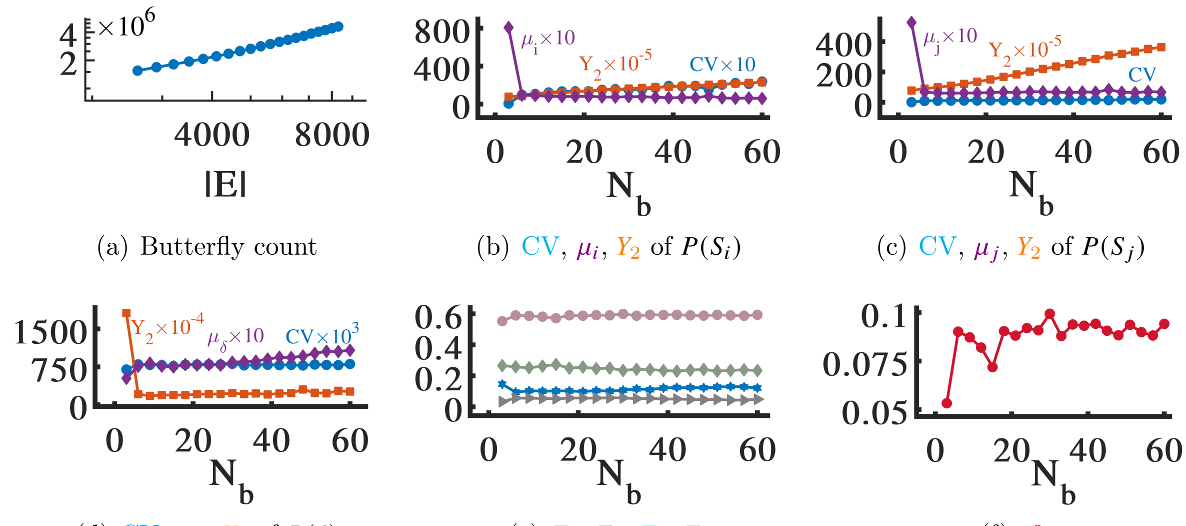
(f) r^s

Figure 4.4: Mixing patterns of butterflies in SPA stream with m=10 and average butterfly rate 556.6±24.5.


(a) Butterfly count (b) CV, μ_i, Y_2 of $P(S_i)$ (c) CV, μ_j, Y_2 of $P(S_j)$

(d) CV, μ_δ, Y_2 of $P(\delta)$ (e) F_1, F_2, F_3, F_4 (f) r^s

Figure 4.5: Mixing patterns of butterflies in SPA stream with m=50 and average butterfly rate 558±22.4.


(a) Butterfly count (b) CV, μ_i, Y_2 of $P(S_i)$ (c) CV, μ_j, Y_2 of $P(S_j)$

(d) CV, μ_δ, Y_2 of $P(\delta)$ (e) F_1, F_2, F_3, F_4 (f) r^s

Figure 4.6: Mixing patterns of butterflies in SPA stream with m=100 and average butterfly rate 548.1±17.

	Butterfly Densification	Strength Diversification	Steady Stregth Assortativity
SPA	✓	×	×
FF	×	✓	×

Table 4.3: Mixing patterns of butterflies in SPA and FF streams.

SPA streams follow butterfly densification but not strength diversification and steady strength assortativity. In SPA streams, as new low-strength vertices attach to m vertices with the highest strengths (strong vertices), many butterflies are formed around the high-strength vertices with a rate much higher than that of real-world streams. When the number of connections per new vertex m is higher, the probability of attachment to low-strength vertices is higher since the number of strong vertices is limited, therefore the number of edges among low-strength vertices increases. As a result, the graphs display weak strength assortativity when average degree is high. When m is low, the number of edges with high strength-difference is higher compared to the case with high m, although they don't exceed edges with low strength-difference. The diversity of $Pr(S)$ and $Pr(\delta)$ does not increase significantly in either cases.

As summarized in Table 4.3, FF streams with implicit degree-driven preferential attachment and neighbour copying yield graphs with increasing diversity of strengths of butterfly vertices, however the quantity of butterflies and their mixing schemes do not preserve realistic patterns. SPA streams with pure strength-driven preferential attachment lead to graphs with rapidly growing butterfly density, however the mixing patterns do not match realistic patterns. This highlights the essence of a growth model which has both strength-driven preferential attachment and neighbour copying flavours to ensure a balanced butterfly densification and incremental strength diversity. Further considerations regarding the integration of these two mechanisms with other effective mechanisms are also required to create realistic streams. The next section resolves this.

4.3 sGrow

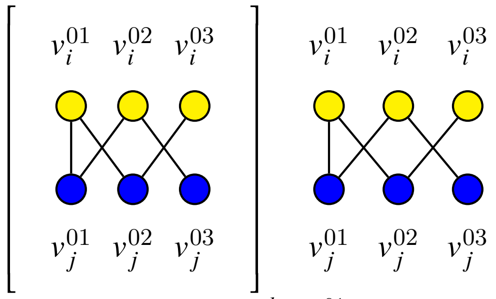
Burstiness, strong-gets-stronger, and core-periphery are the semantic concepts explaining the butterfly emergence patterns (Chapter 3) and also the strength preferential attachment and neighbour copying are the microscopic mechanisms explaining the butterfly densification and strength diversification (Section 4.2). In the following, these concepts and growth mechanisms are integrated with further mechanisms introduced in a streaming growth model, called *sGrow* [298]. This model explains the co-occurrence of the three realistic emergence patterns of butterflies in streaming graphs such that all four-vertex graphlets emerge in the graph, the sgrs are realistic (i.e. preserve streaming data characteristics), and the stream properties are configurable.

The introduced mechanisms (Figure 1.3) solve the drawbacks of existing works. For instance, a preferential random walk and copying micro-mechanisms feature strength preferential selection of vertices whose neighbours are copied with probabilistic connections. These lead to butterfly densification as well as emergence of four-vertex graphlets and also increase strength diversity of butterfly vertices. Moreover, the introduced random walk utilizes BFS and DFS traversals with dynamic and random number of hops, which balances the butterfly emergence patterns. The realistic sgr generation techniques (such as inactivity gaps, and techniques for timestamp assignment and evolving streaming rate) yield graphs satisfying streaming paradigms such as burstiness and out-of-order arrivals. These realistic sgr generation techniques also impact the temporal patterns of butterfly emergence (for instance timestamp of sgrs converting a caterpillar to a butterfly influences the number of butterflies in the corresponding burst). The local/unbounded graph updates, including the sliding window mechanism, avoid pure preferential attachment to a few hubs (avoids perfectly disassortative mixing) and helps with the steady pattern of strength assortativity of butterflies.

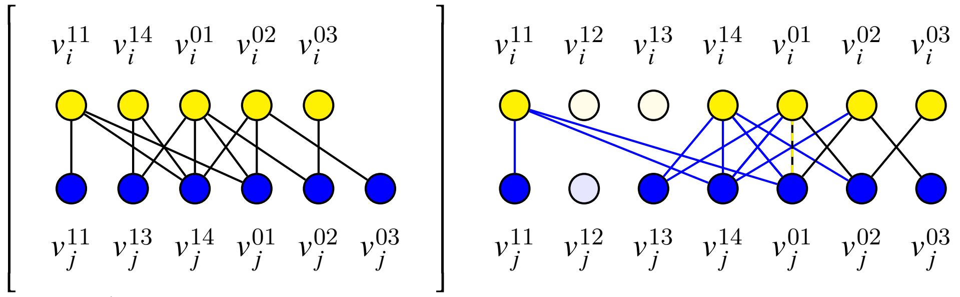
4.3.1 Overview

An overview of *sGrow* is provided in this section (Algorithm 3 - Figure 4.7). A time-based sliding window (Definition 6) is used to generate a sequence of sgrs (Definition 2) that constitute the synthetic weighted bipartite streaming graph (Definition 3). In the following, $G_{W,t}$ refers to the computational graph snapshot formed by the sgrs within the window. The output stream is a sequence of sgrs denoted as $\mathfrak{R}$. The time step t is the computational time point used for controlling the window and the timestamp τ is the sgr's time-label which follows the timestamp scale of an initial graph snapshot. A five-point scale $[1, 5]$ (similar to that of real-world streams) is used to generate weights.

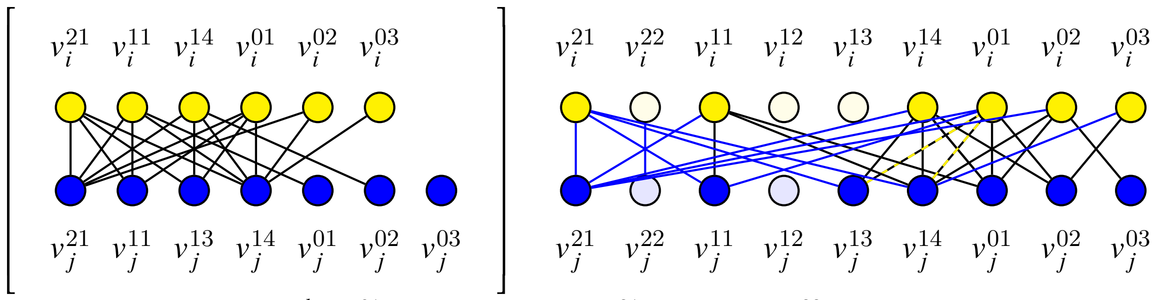
$G_{W,t}$ and $\mathfrak{R}$ are initiated with an initial graph snapshot $G_{W,t_0} = (V_0, E_0)$. The window's beginning border W^b is set to the first timestamp in G_{W,t_0} (Algorithm 3, lines 1 - Figure 4.7(a)). At each time step t, m (a random number in $[0, M)$, where M is a parameter) new sgrs $r^{l=1,..,m}=\langle v_i^l, v_j^l, \omega_{ij}^l, \tau\rangle$ with new vertices are created and added to $\mathfrak{R}$ and $G_{W,t}$ (Figure 4.7(b)(c)). The shared timestamp is one plus the last timestamp in $G_{W,t}$ and the weights are random integers $\omega_{ij}^l \in [1, 5]$ (Algorithm 3, line 4). To connect these new isolated edges to the rest of sgrs, the following procedure is followed. A random integer $\omega \in [-1, 5]$



(a) t=0, $W^b = r^{01}.\tau$



(b) t=1, $W^b = r^{01}.\tau + \beta$, L=1, m=4 (r^{11} incurs no-op, r^{12} & r^{13} incur removal, r^{14} adds burst.)



(c) t=2, $W^b = r^{01}.\tau + 2\beta$, L=2, m=2 (r^{21} adds burst, r^{22} incurs no-op.)

Figure 4.7: (left) The computational graph G and (right) the stream $\mathfrak{R}$ at the end of time steps t=0, 1, 2 with β=2, ρ=0.4, M=5, and $L \in [1, 2]$. New edges are blue and PRW edges are yellow dashed. Timestamps and weights are not depicted and it is assumed that the edges in G_0 expire from the window as their timestamps are below the W^b.

Algorithm 3: sGrow(ρ, M, β, $[L_{min}, L_{max}]$)

```
Data: $G_{W,t_0}$: an initial graph
Input: $\rho$: connection probability, $M$: maximum number of new edges, $\beta$: slide
       parameter, $[L_{min}, L_{max}]$: range of PRW's length
Output: $\mathfrak{R}$, sequence of streaming graph records
1  $G \leftarrow G_{W,t_0} = (V_0, E_0)$, $\mathfrak{R} \leftarrow E_0$, $\tau \leftarrow$ 1+ last timestamp in $G_{W,t_0}$, $t \leftarrow 0$, $W^b \leftarrow$
     first timestamp in $G_{W,t_0}$
2  while true do
3      $t \leftarrow t + 1$
4      Add $m \in [0, M)$ new sgrs $r^{l=1,..,m} = \langle v_i^l, v_j^l, \omega_{ij}^l, \tau \rangle$ with $\omega_{ij}^l \in [1, 5]$ to $\mathfrak{R}$ and
         $G_{W,t}$
5      for each $r^{l=1,..,m}$ do
6          $\omega \leftarrow$ a random integer in $[-1, 5]$
7          switch $\omega$ do
8              case -1 do
9                  Remove $v_i^l - v_j^l$ from $\mathfrak{R}$ and $G_{W,t}$.
10             case 0 do
11                 No operation
12             otherwise do
13                 $u_j^0 \leftarrow$SPS($V_j$)
14                 $L \leftarrow$ a random integer in $[L_{min}, L_{max}]$
15                 $(PRW_i, PRW_j) \leftarrow$ PRW($u_j^0$, $false$, $G_{W,t}$, $L$)
16                 $addBurst(v_i^l, v_j^l, PRW_i, G, \mathfrak{R}, \rho)$
17                 $addBurst(v_i^l, v_j^l, PRW_j, G, \mathfrak{R}, \rho)$
18                 $\tau \leftarrow \tau + |\frac{(\omega'-5)(\omega'-4)(\omega'-3)}{2}|$
19     Remove any newly added vertex $v_i^l$ and $v_j^l$ with less than 2 neighbours from
         $G_{W,t}$
20     $\tau \leftarrow \tau + 1$
21     $W^b \leftarrow W^b + \beta$
22     if $t = \beta$ then
23         Remove any edge with timestamp less than $W^b$ from $G_{W,t}$
24         $t \leftarrow 0$
```

is generated based on which, one of the three operations (edge removal, no-operation, and burst addition) is performed for each of the m new sgrs in parallel (Algorithm 3, lines 6-18). This random integer is aimed to randomize the order of selected operations. The range is set as $[-1, 5]$ to incur a probability of 5/7 for additions, and 1/7 for removals and no-operation. The intuition is the observation in real-world graphs where edge additions dominate edge removals. This range is found to best match the characteristics of real-world streams. Additions and removals happen as described in Section 4.3.2.

- ω=−1, the connection between v_i^l and v_j^l is removed from $\mathfrak{R}$ and $G_{W,t}$ (Algorithm 3, line 9). This edge removal introduces isolated vertices if the vertices do not acquire neighbours from the current or next batch (v_i^{12}, v_j^{12}, and v_i^{13} in Figure 4.7.b).

- ω=0, nothing happens (Algorithm 3, line 11). This no-operation introduces a gap of inactivity between streaming records to form bursts (Definition 4) and also introduces isolated edges. If the current sgr r^l is not connected to the subsequent sgrs in current batch ($r^{l+1,\dots,m}$) or the following batches, it will remain isolated (r^{22} in Figure 4.7.c).

- ω>0, a j-vertex u_j^0 in $G_{W,t}$ (v_j^{01} in Figure 4.7.b and v_j^{14} in Figure 4.7.c) is randomly selected via *Strength Preferential Selection* (SPS, described in Section 4.3.3). Next, a *Preferential Random Walk* (PRW, described in Section 4.3.4) starting from u_j^0 is performed in $G_{W,t}$. The number of hops in PRW (i.e. walk length) is a random integer L in the parameter range $[L_{min}, L_{max}]$. Using PRW as a backbone, bursts of new sgrs between r^l and the rest of sgrs in $G_{W,t}$ and $\mathfrak{R}$ are established (as described in Section 4.3.5). The strength preferential attachment and neighbour copying are necessary to ensure a balanced butterfly densification and incremental strength diversity. Therefore, these are incorporated into PRW, which serves as a backbone of vertices for adding bursts of sgrs through connecting the new isolated sgrs to these vertices and their neighbours. Since the vertices are connected through a random walk, graphlets emerge efficiently. Since at each iteration, a random number of PRW vertices are selected based on strength preferential probability, the graphlets effectively obey the observed patterns in real-world streams. Moreover, this process includes adding sgrs according to a parameterized probability as well as bounding the random length of PRW according to a parameterized range, which enable configurable sgr creation. After adding the last edge with weight ω', the timestamp τ is incremented as a function of ω': τ=τ+$|(\omega' - 5)(\omega' - 4)(\omega' - 3)/2|$. This function creates a timestamp interval as soon as generation of a sgr with low weight (i.e. $\omega' \leq 2$), therefore it helps characterize the burstiness of the stream (Algorithm 3, lines 13-18). This is based on an observation in real-world streams that the last sgr in each burst has a low weight. It is noteworthy that there are two levels of burstiness: (1) the bursts initiated by each r^l as described here, and (2) the bursts created concurrently by m sgrs $r^{l=1,..,m}$ which can be assumed as multiple generative sources.

The aforementioned procedure takes place for all m new sgrs in parallel[1]. After that, any newly added vertex with less than two neighbours is removed from $G_{W,t}$ (Algorithm 3, line 19 - v_i^{22} and v_j^{22} in Figure 4.7.c). This removal of new isolated vertices/edges is done to retain a connected computational graph, yet the old vertices whose adjacent edges are discarded by the window (as described below) become isolated in $G_{W,t}$ (v_j^{03} in Figure 4.7(c)). Also, the stream may hold isolated vertices/edges (v_i^{22}-v_j^{22}, v_i^{12}, v_j^{12}, and v_i^{13} in Figure 4.7(c)). Next, the timestamp is incremented by one (Algorithm 3, line 20). The window slides as W^b is incremented by β; the edges with timestamps out of the sliding window are removed from the graph after each β time steps; and the time step is reset to zero (Algorithm 3, lines 21-24). This sliding window mechanism is used to avoid pure preferential attachment to old vertices in global scale and create time-sensitive and local connections leading to emergence of young hubs (high degree vertices) to support the observations of real-world stream analysis in 3. The generation process happens continuously and $\mathfrak{R}$ streams-out as the sgrs are generated. This process can be restricted to continue until a desired number of sgrs S are generated ($|\mathfrak{R}| = S$) and then return the stream $\mathfrak{R}$.

4.3.2 Data Structures

The following data structures and basic graph/stream operators are used to implement *sGrow*'s algorithms.

- A vertex is an object with three attributes: ID, Strength, and timestamp τ. A new i(j)-vertex $v_i(v_j)$ is assigned an integer ID equal to the current number of i(j)-vertices, a strength initialized to zero, and a timestamp equal to that of the edge by which this vertex is added. Dot notation is used to refer to attributes of an object, e.g. $v_i.ID$ denotes the ID of the vertex v_i.

- An edge/sgr between vertices v_i and v_j is an object with four attributes: i-vertex (object v_i), j-vertex (object v_j), timestamp (integer τ), and weight (integer ω).

[1]This algorithm is implemented in a single machine architecture; a distributed version is doable but not considered here.

- The connections of graph $G_{W,t}$ are stored by two hash-map data structures to map each vertex ID to the hash-set of its immediate neighbours: *iNeighbours*$= \{(v_j.ID : N_i(v_j))\}$ and *jNeighbours*$= \{(v_i.ID : N_j(v_i))\}$. Hash-sets are used to avoid storing multiple edges between two vertices in the computational graph. Also, hash-map provides fast access to the neighbourhoods.

- The sgrs in $\mathfrak{R}$ are stored in a vector that retains the edges in the order of their additions which include out-of-order sgrs with respect to the timestamps as explained in Section 4.3.5.

When a new edge is added/removed to/from the graph or stream these data structures are updated accordingly and also the strengths of the vertices at the either ends of the edge are incremented/decremented by the weight of the edge.

4.3.3 Strength Preferential Selection

Function $SPS(V)$ is invoked in Algorithms 3. This function selects a random vertex in the set V according to *strength preferential probability* $\Lambda_v = \frac{v.strength}{\Sigma_{v' \in V} v'.strength}$ [42, 41]. Vertices in V are concurrently added to a list with multiplicity equal to their strength. Next, the list is shuffled and a random element v_0 in the list is selected as the output vertex.

4.3.4 Preferential Random Walk

Function $PRW(starter, isI, G, L)$ performs a random walk with L hops on a graph $G_{W,t}$. It starts from a $starter$ vertex whose type determines a boolean flag isI (true when $starter$ is an i-vertex and false otherwise). At each hop, a neighbour of the $starter$ vertex ($u_i \in N_i(starter)$, $u_j \in N_j(starter)$) is selected via strength preferential selection (invoking $SPS(N_i(starter))$, $SPS(N_j(starter))$). The selected neighbour (u_i, u_j) is added to a hash set of unique vertices (PRW_i, PRW_j) and is set as the $starter$ vertex. The starter flag is accordingly set and the hop counter is incremented by one. The next hop starts with the last added vertex. When the current selected neighbour is already in the hash set, if it is the last element, the walk continues to another neighbour of that vertex (in depth traversal) and if it is one of the previously selected vertices other than the last element, the walk continues in breadth traversal. Therefore, the walk is a combination of BFS and DFS with random preferential selection.

4.3.5 Burst Addition

Function $addBurst(v_i^l, v_j^l, PRW_i, G, \mathfrak{R}, \rho)$ is given in Algorithm 4. This function adds bursts (Definition 4) of sgrs to $G_{W,t}$ and $\mathfrak{R}$ based on the i-vertices in the PRW_i and new vertices v_i^l and v_j^l given a probability parameter ρ (Figure 4.8.a). The same procedure is followed to add bursts with respect to the j-vertices in the PRW_j. As illustrated in Figure 4.8, considering each i-vertex u_i in the PRW_i, the following connections are established:

Step 1. An edge between u_i and the newly added v_j^l is formed with the timestamp of v_j^l and a weight $\omega' \in [1, 5]$ (Algorithm 4, lines 3-4 – Figure 4.8.b). This edge connects edge v_i^l-v_j^l to the graph and also leads to emergence of $N_1 + N_2$ caterpillars (solid 3-paths in Figure 4.8.e,f), where N_1 and N_2 are the number of 1-hop (immediate) and 2-hop neighbours of u_i, respectively.

Step 2. With probability ρ, an edge between u_i and an existing j-vertex z_j, selected uniformly at random, is formed with timestamp $Min(u_i.\tau, z_j.\tau)$ and a weight $\omega' \in [1, 5]$ (Algorithm 4, lines 5-8 – Figure 4.8.c). Using a timestamp other than the current timestamp ($v_j^l.\tau$) introduces out-of-order sgrs (late arrival) since this sgr has a timestamp less than ($v_j^l.\tau$). It also helps balance the burst sizes since the current timestamp is not assigned to all edges in the current time step. This probabilistic edge leads to converting the caterpillars between u_i and z_j into butterflies at the generation time of either vertices (closed 4-path in Figure 4.8.g).

Step 3. With probability ρ, an edge between the newly added v_i^l and each u_i's immediate j-vertex neighbour n_j is concurrently formed with n_j's timestamp $n_j.\tau$ and a weight $\omega' \in [1, 5]$ (Algorithm 4, lines 9-12 – Figure 4.8.d). In other words, each of the adjacent links of u_i is copied with probability ρ. Since $n_j.\tau < v_i^l.\tau$, out-of-order sgrs join previous burst of sgrs with same timestamp (including the sgr incident to n_js). These probabilistic edges lead to converting the N_1 caterpillars that emerge in Step 1 into butterflies (closed 4-path in Figure 4.8.e). This step is used to generate streams with high number of sgrs per burst.

4.4 Performance Evaluations

The effectiveness and efficiency of *sGrow* are tested from three perspectives:

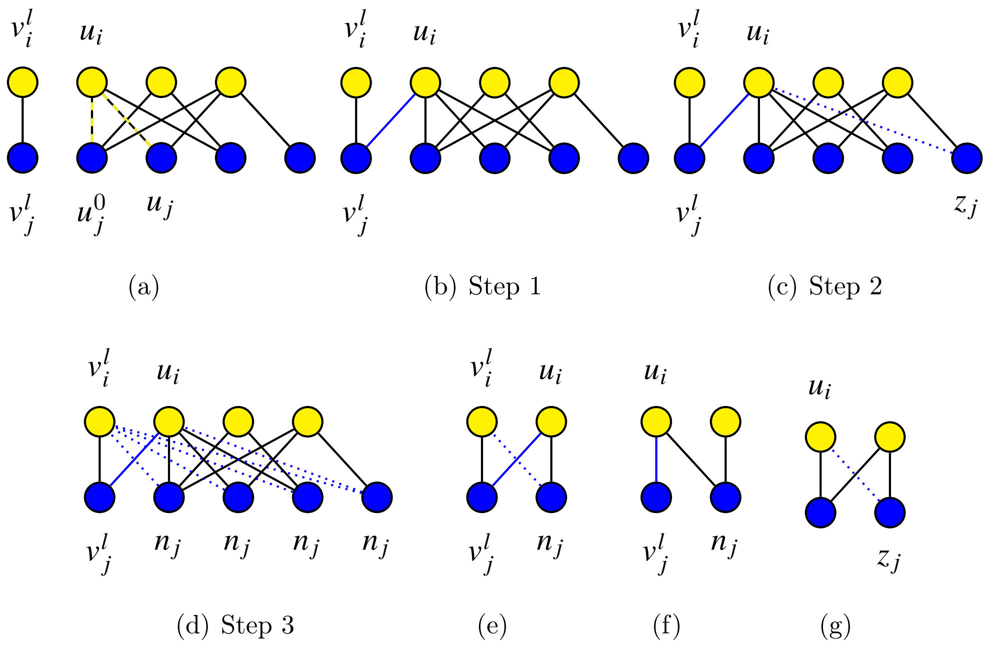


Figure 4.8: Four-vertex graphlets (e, f, g) and schematic burst addition steps (b, c, d) based on an i-vertex u_i in the PRW starting from u_j^0 designated by yellow dashed lines (a). New edges are blue and dotted lines denote probabilistic connections.

- Pattern reproduction (Section 4.4.1). The ability of *sGrow* to reproduce the realistic patterns is compared with baselines and examined under different levels of burstiness, initial graph snapshots, and butterfly counts.
- Stress testing (Section 4.4.2). The impact of introduced parameterized techniques on the effectiveness, efficiency, and burstiness of the generated stream by *sGrow* is examined.
- Computational complexity (Section 4.4.3). The computational complexity of *sGrow* is analyzed theoretically.

Data. In the following, the generative methods for *sGrow* and baseline streams are described.

sGrow streams. The streams generated by *sGrow* model are created with a prefix of 1000 sgrs from real-world streams (G_{W,t_0}) and the rest of the stream synthesized via

Algorithm 4: Add Burst

```
Function addBurst($v_i^l, v_j^l, PRW_i, G, \mathfrak{R}, \rho$)
    for each $u_i \in PRW_i$ do
        $\omega' \leftarrow$ a random integer in $[1, 5]$
        Add a new sgr $\langle u_i, v_j^l, \omega', v_j^l.\tau \rangle$ to $\mathfrak{R}$ and $G_{W,t}$
        if coin($\rho$) is Head then
            $z_j \leftarrow$ Select a random j-vertex
            $\omega' \leftarrow$ a random integer in $[1, 5]$
            Add a new sgr $\langle u_i, z_j, \omega', Min(u_i.\tau, z_j.\tau) \rangle$ to $\mathfrak{R}$ and $G_{W,t}$
        for each $n_j \in N_j(u_i)$ do                    // in highly bursty streams
            if coin($\rho$) is Head then
                $\omega' \leftarrow$ a random integer in $[1, 5]$
                Add a new sgr $\langle v_i^l, n_j, \omega', n_j.\tau \rangle$ to $\mathfrak{R}$ and $G_{W,t}$
```

sGrow model (Algorithm 3) with various parameter configurations and different number of butterflies. The generated streams are referred to as S-$\{G_0$-name$\}$. The streams used for pattern reproduction and stress testing purposes are generated as follows.

- Pattern reproduction. Epinions, Amazon, and Ciao are bursty streams with average burst sizes of b = 27282, 1753.7, and 14.8; Yahoo and ML1m are also bursty but with lower values b = 2.4 and 2.2; ML100k and WikiLens have the lowest burstiness with b = 2 and 1, respectively (Table 3.3). According to these burstiness profiles, the parameters are set to control the temporal distribution of sgrs. That is, to simulate a stream with high burstiness, M and $[L_{min}, L_{max}]$ are set to high values, which increase (1) the probability of creating high number of new edges at each time step (M) and (2) the burst size by generating backbone walks with more vertices ($[L_{min}, L_{max}]$). To simulate a stream with low burstiness (S-ML100k and S-WikiLens), the neighbourhood copying (Algorithm 4, lines 9-12 – step 3 in Section 4.3.5) is not performed. The default value of ρ is 0.3 and it is further adjusted by decreasing (increasing) to push the burst size towards lower (higher) values. β = 5 in all streams. The exact value of parameters are given in Table 4.4. All the reported results in these figures are based on the same streams.

- Stress testing. To evaluate the impact of *sGrow* techniques (batch of isolated edges, probabilistic connections, the random walk backbone, and the sliding window) on

Table 4.4: Parameters for generating sGrow streams.

	ρ	M	β	$[L_{min}, L_{max}]$
S-Ciao	0.3	100	5	$[2, 3]$
S-Epinions	0.2	300	5	$[3, 6]$
S-WikiLens	0.4	100	5	$[1, 3]$
S-ML100k	0.3	100	5	$[1, 3]$
S-ML1m	0.3	10	5	$[1, 2]$
S-Amazon	0.3	50	5	$[1, 2]$
S-yahoo	0.3	50	5	$[2, 3]$

effectiveness, efficiency, and burstiness, the corresponding parameters are evaluated statically and dynamically as follows:

– A diverse range of parameters $M \in \{100, 150, 200, 250, 300\}$, $\rho \in \{0.3, 0.4, 0.5, 0.6, 0.7\}$, $L_{min} \in \{1, 2, 3, 4\}$, $L_{max} \in \{3, 4, 5\}$, and $\beta \in \{5, 10, 15, 20\}$ are used to create S-Amazon stream. Each row of Tables 4.25, 4.26, and 4.28 illustrates the effect of one parameter on effectiveness, efficiency, and frequency distribution of burst sizes, respectively.

– S-Amazon stream is created with 10^7 sgrs such that after generating 5×10^6, one of the parameters switches from $M = 100$, $\beta = 5$, $L \in [15]$ and $\rho = 0.4$ to $M = 300$, $\beta = 20$, $L \in [45]$ and $\rho = 0.8$. Figure 4.27 illustrates the effect of the parameter switch on effectiveness and efficiency.

Baseline streams. The streams generated by baseline models are generated with a prefix of 1000 sgrs from Amazon stream and the rest of the stream synthesized via three graph models: Butterfly model (BM) [229], Connecting Nearest-neighbour model (CNM) [326], and Duplication Divergence model (DDM) [326]. These baseline streams are generated using random integers in $[1, 5]$ as weights and the time step at which new vertices join the graph as the timestamp of new sgrs. In all three models, the stream grows by adding a new vertex at each time step and the original models generate unipartite graphs. Either an i-vertex or a j-vertex is chosen to be added randomly with equal probability and the same procedure is followed for adding new j-vertices and new i-vertices.

- In CNM, the stream grows as new i(j)-vertices connect to existing j(i)-vertices. With probability $1 - p$, the new i-vertex is connected to a randomly selected j-vertex and with probability p, an edge between two randomly selected and disconnected vertices is established.

- In BM, the stream grows as new i(j)-vertices connect to existing j(i)-vertices residing on a random walk started from one or more host vertex. Each new i-vertex picks a host (a randomly selected j-vertex) with probability p_h and is assigned a probability p_{step} (a random value in $(0, 1]$). The new i-vertex visits the host as it is linked to the host with probability p_l and then with probability p_{step} visits a two-hop j-vertex on the random walk started from the host. This procedure continues until no new host is chosen.

- In DDM, the stream grows as new i(j)-vertices connect to existing j(i)-vertices by following duplication and divergence rules. In duplication phase, the new i-vertex v_i connects to all j-neighbours n_j of a randomly selected i-vertex $v_{i'}$ ($v_{i'}$ is duplicated) and during the divergence phase, one of the two edges $e_{i'n_j}$ or e_{in_j} is selected randomly and removed with probability p_e.

Metrics. The effectiveness of *sGrow* is evaluated and compared with real-world streams with respect to the three patterns of scale-invariant strength assortativity of butterflies. Qualitative evaluations are done by checking whether the patterns hold and quantitative evaluations are done by checking the error of r^s and F.

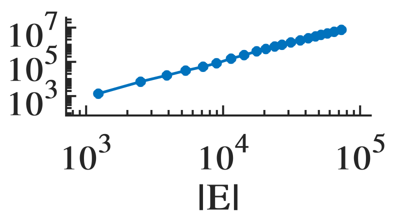
The efficiency of *sGrow* is evaluated by measuring the generation time (in seconds). Moreover, the burstiness of the *sGrow* streams is evaluated by investigating the frequency distribution of burst sizes.

Computing setup. Experiments are conducted on a machine with 15.6 GB native memory and Intel Core $i7 - 6770HQCPU@2.60GHz * 8$ processor. All *sGrow*'s algorithms and experiments are implemented and performed in Java (OpenJDK version 11.0.11).

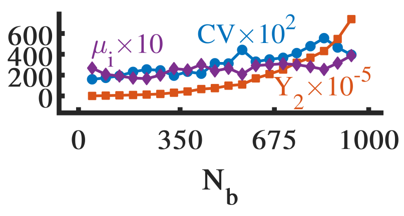
4.4.1 Pattern Reproduction

sGrow. The streams generated by *sGrow* obey the scale-invariant strength assortativity of butterflies. This is because all the synthetic streams preserve the realistic mixing patterns (Figures 4.9 - 4.15) regardless of the initial graph snapshot, butterfly count, and burstiness level as described below.

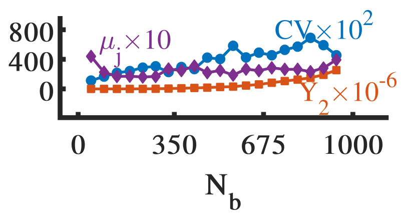
- Butterfly densification. The number of butterflies grows over time with an average butterfly rate higher than 1 in all the streams (Table 4.5) indicating that the butterfly count grows super-linearly with respect to the number of edges.



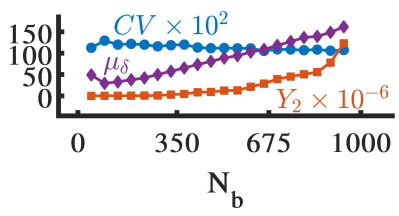
(a) Butterfly count



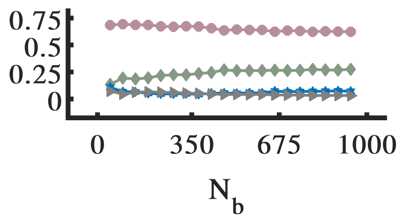
(b) CV, μ_i, Y_2 of $P(S_i)$



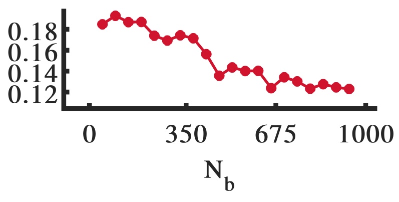
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

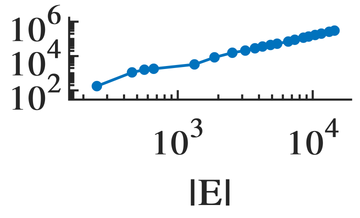


(e) F_1, F_2, F_3, F_4

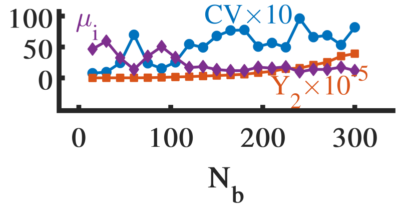


(f) r^s

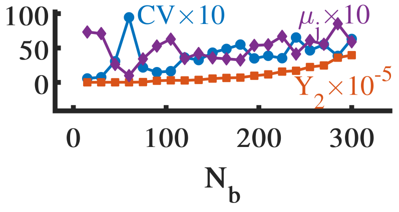
Figure 4.9: Mixing patterns of butterflies in Ciao stream.



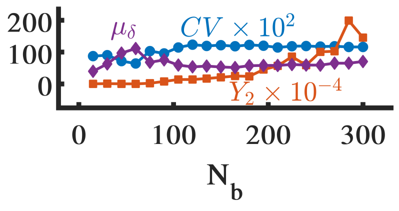
(a) Butterfly count



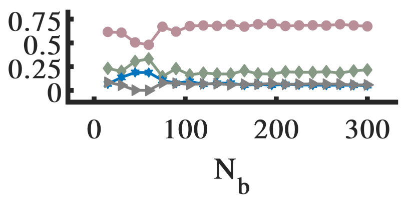
(b) CV, μ_i, Y_2 of $P(S_i)$



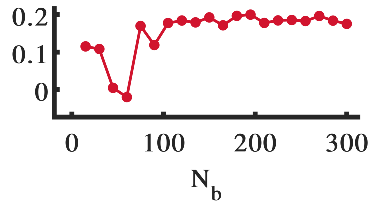
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

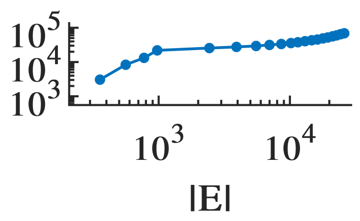


(e) F_1, F_2, F_3, F_4

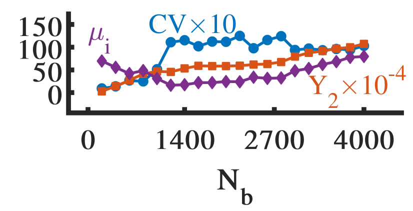


(f) r^s

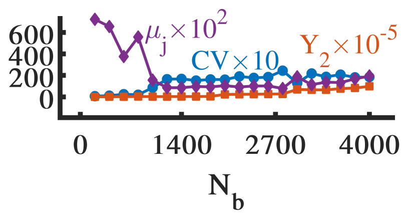
Figure 4.10: Mixing patterns of butterflies in Epinions stream.



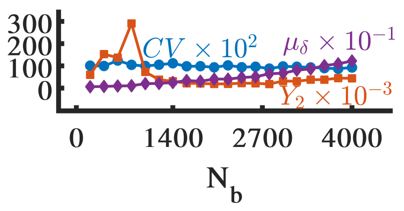
(a) Butterfly count



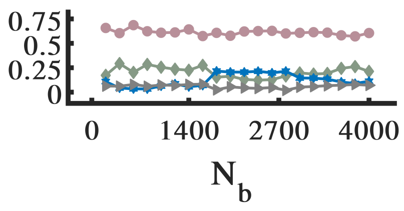
(b) CV, μ_i, Y_2 of $P(S_i)$



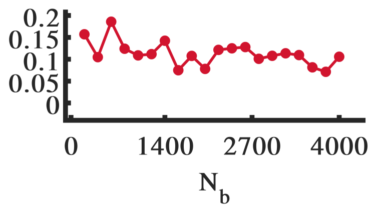
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

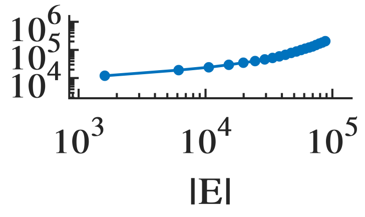


(e) F_1, F_2, F_3, F_4

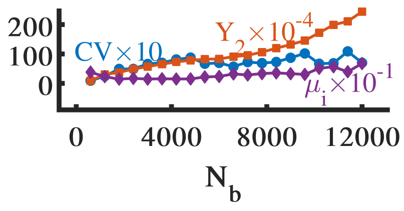


(f) r^s

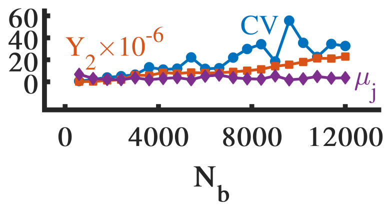
Figure 4.11: Mixing patterns of butterflies in WikiLens stream.



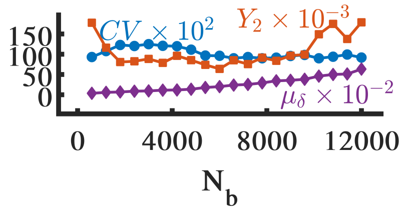
(a) Butterfly count



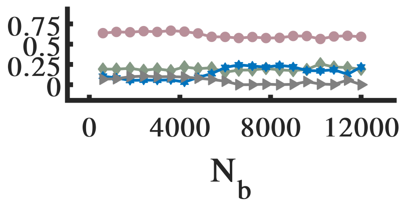
(b) CV, μ_i, Y_2 of $P(S_i)$



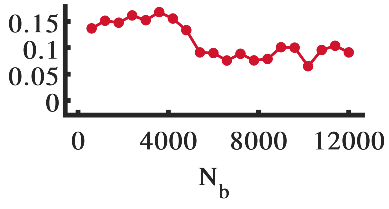
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

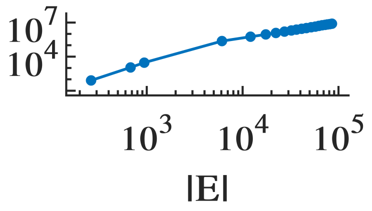


(e) F_1, F_2, F_3, F_4

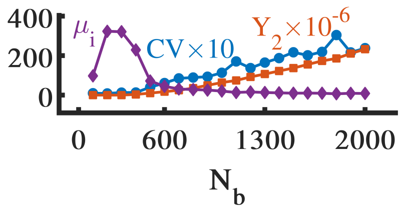


(f) r^s

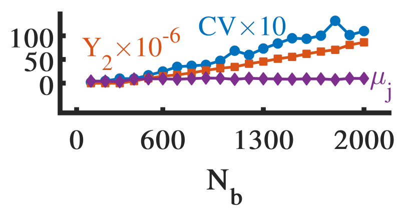
Figure 4.12: Mixing patterns of butterflies in ML100k stream.



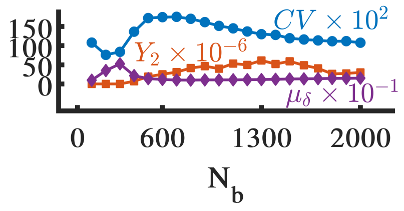
(a) Butterfly count



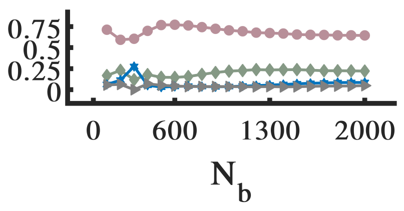
(b) CV, μ_i, Y_2 of $P(S_i)$



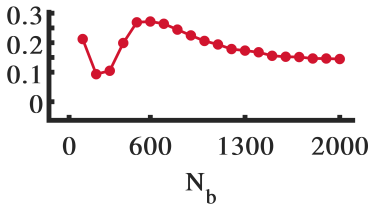
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

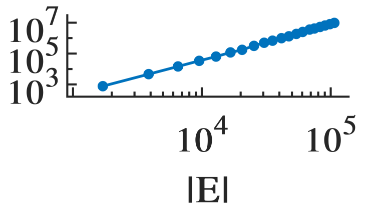


(e) F_1, F_2, F_3, F_4

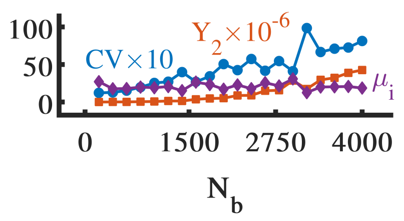


(f) r^s

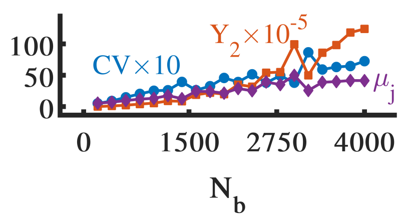
Figure 4.13: Mixing patterns of butterflies in ML1m stream.



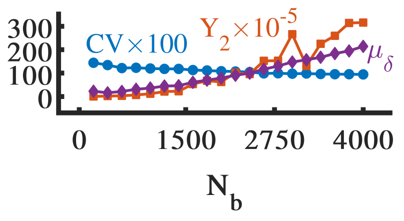
(a) Butterfly count



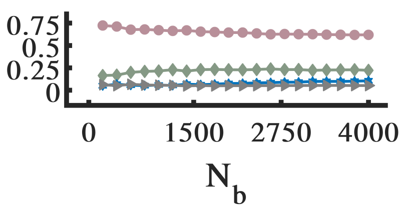
(b) CV, μ_i, Y_2 of $P(S_i)$



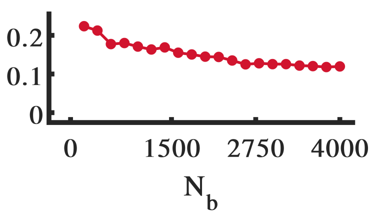
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

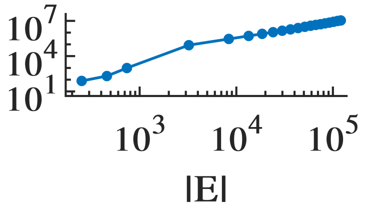


(e) F_1, F_2, F_3, F_4

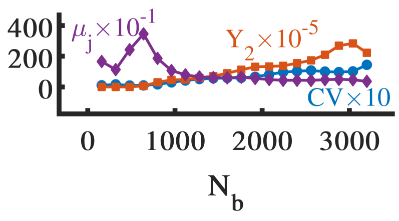


(f) r^s

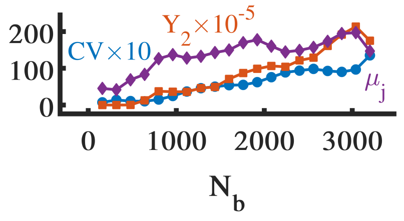
Figure 4.14: Mixing patterns of butterflies in Amazon stream.



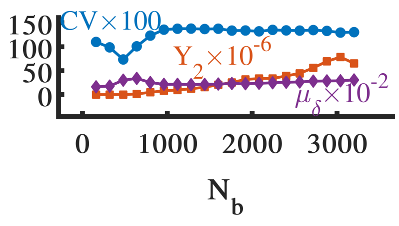
(a) Butterfly count



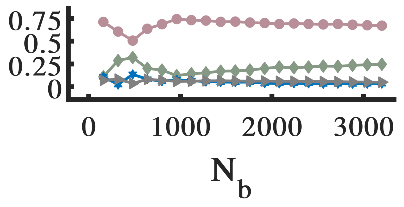
(b) CV, μ_i, Y_2 of $P(S_i)$



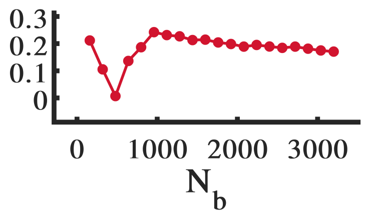
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$



(e) F_1, F_2, F_3, F_4



(f) r^s

Figure 4.15: Mixing patterns of butterflies in Yahoo stream.

- Strength diversification. The mean, relative standard deviation, and tail skewness/heaviness of $Pr(S)$ increases over time showing that butterfly vertex strengths diversify over time.

- Steady strength assortativity. The synchronous evolution of mean and standard deviation accompanied by increasing skewness/heaviness of the tail of $Pr(\delta)$ plus the stable values of F elements over time demonstrate that $Pr(\delta)$ is fixed-shaped yet growing. Moreover, the strength assortativity localization factor changes trivially over time and is positive with F_1 values between 0.5 and 0.7 indicating the steady strength-assortativity of butterflies over time.

Table 4.6 presents the mean absolute error of r^s and F elements in *sGrow* streams with respect to that of real-world streams over the sequential burst-based graph snapshots. In all synthetic streams the error is between 0.01 and 0.1. The low errors of F elements suggest that sequential graph snapshots in the *sGrow* streams have similar strength difference distribution to the corresponding real-world streams. Particularly the low error of F_1, r^s represents the similar strength assortativity localization factor. This indicates that *sGrow* reproduces the similar strength difference distribution and strength assortativity of butterfly edges as in the real-world streams.

Table 4.5: Average butterfly rate in synthetic streams generated by our model.

	average butterfly rate	$\lvert E^{20}\rvert$	N_b^{20}	$\bowtie^{20}$
S-Ciao	37.2440 ± 31.4987	$72,966$	940	$7,430,097$
S-Epinions	9.4358 ± 6.1519	$14,495$	300	$304,417$
S-WikiLens	6.3037 ± 5.6897	$25,957$	$4,000$	$69,302$
S-ML100k	2.191 ± 1.3045	$87,951$	$12,000$	$204,118$
S-ML1m	59.8963 ± 29.2066	$85,397$	$2,000$	$7,860,689$
S-Amazon	30.8994 ± 28.8517	$106,714$	$4,000$	$9,599,739$
S-Yahoo	51.1708 ± 27.891	$120,616$	$3,200$	$10,847,746$
BM $p_l = 0.3$, $p_h = 0.5$	$4.9294e-4 \pm 2.3263e-4$	106716	81448	30
BM $p_l = 0.5$, $p_h = 0.7$	$0.0035 \pm 8.9103e-04$	106716	48944	253
BM $p_l = 0.7$, $p_h = 0.9$	0.2534 ± 0.0453	106720	14660	26371
CNM $p = 0.3$	$0.0015 \pm 5.2659e-04$	106739	105841	94
CNM $p = 0.5$	$0.0064 \pm 9.4627e-04$	106739	105841	536
CNM $p = 0.7$	0.0423 ± 0.0068	106739	105841	4420
DDM $p_e = 0.3$	8.4597 ± 6.2345	106714	25524	2183130
DDM $p_e = 0.5$	1.2260 ± 0.8530	106714	42644	294543
DDM $p_e = 0.7$	0.0928 ± 0.0685	106714	69263	23342

Baselines. None of the baseline streams obey the scale-invariant strength assortativity of butterflies (Figures 4.16-4.24) since þ$_1$, þ$_2$, and þ$_3$ are not preserved.

- Butterfly densification. The number of butterflies grows slowly in the streams (Table 4.5). In BM streams, a butterfly forms when a new vertex connects to a host (selected with probability p_h) and its two hop neighbour with probability p_l, therefore as p_h and p_l increase, butterfly count increases. In BM stream with $p_l = 0.3$ and $p_h = 0.5$, at some graph snapshots there is no new butterfly added to the graph and butterfly count remains the same. However, the total number of butterflies in each graph snapshot is far below the number of edges. In CNM streams, butterflies have higher chance of emergence when potential edges are converted to edge with probability p compared to addition of new edges with probability $1 - p$, therefore as p increases butterfly count increases, albeit insignificantly. In DDM, with probability $1 - p_e$ butterflies emerge with the addition of each new vertex, which forms wedges with $degree(v_{i'})$ neighbour of $v_{i'}$. Consequently the lower p_e, the higher is the butterfly count. However, the average butterfly rate is still lower than that of *sGrow*, specifically S-Amazon with average butterfly rate 30.89994 and DDM with $p_e = 0.3$ having 8.4597. Moreover, the streams have low burstiness as N_b^{20} is higher than that of *sGrow*.

Table 4.6: Mean absolute error of r^s and F elements.

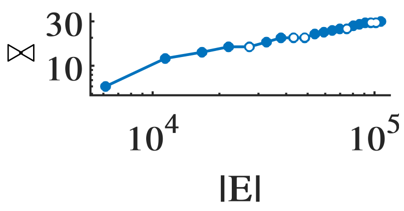
	F_1, r^s	F_2	F_3	F_4
S-Ciao	0.0286	0.05	0.0569	0.0199
S-Epinions	0.0318	0.0389	0.0219	0.0163
S-WikiLens	0.0528	0.0705	0.08	0.0194
S-ML100k	0.0376	0.0513	0.0642	0.0384
S-ML1m	0.0617	0.0445	0.049	0.015
S-Amazon	0.1064	0.0539	0.0476	0.094
S-Yahoo	0.0578	0.0782	0.0519	0.0316
BM $p_l = 0.5$, $p_h = 0.7$	0.1436	0.0855	0.0855	0.0270
BM $p_l = 0.7$, $p_h = 0.9$	0.1279	0.0768	0.0463	0.0156
CNM $p = 0.3$	0.1869	0.1234	0.0997	0.0423
CNM $p = 0.5$	0.1373	0.0593	0.0635	0.0179
CNM $p = 0.7$	0.1418	0.0773	0.0561	0.0105
DDM $p_e = 0.3$	0.0654	0.0654	0.0415	0.0201
DDM $p_e = 0.5$	0.0690	0.0427	0.0306	0.0289
DDM $p_e = 0.7$	0.0851	0.0634	0.0415	0.0129

- Strength diversification. The vertex strengths do not diversify over time as $Pr(S)$ does not get broader and skewed. In all streams CV of vertex strength of butterfly i- and j-vertices is greater than one and increases over time, however Y_2 and μ_S do not show an increasing trend.

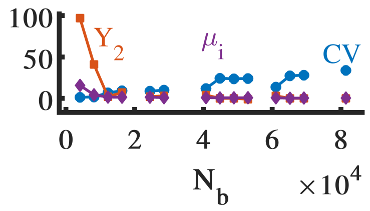
- Steady strength assortativity. The distribution of strength difference of connected butterfly vertices does not get broader, however it retains a fixed shape. Y_2 and μ_δ of $Pr(\delta)$ do not increase over time, although CV is fixed at ≈ 1 in BM with highest p_l and p_h and CNM with $p > 0.3$. In BM and CNM streams, the localization of δs below the mean (r^s) is not stable at values at least above 0.1 and fluctuates over time, although the streams display similar F values as Amazon stream (low errors as shown in Table.

4.4.2 Stress Testing

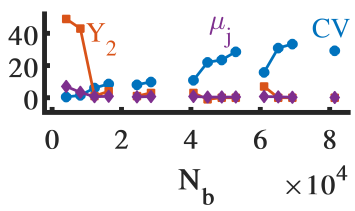
Effectiveness. Figure 4.25 shows the evolution of r^s as the number of butterflies grows to 10^7 in S-Amazon stream with different parameter configurations. Data points are not clustered by colours (corresponding parameters $[L_{min}, L_{max}]$, M, and β) in rows a, c, and d, however they are clustered and ordered by ρ. Also, $0.1 \leq r^s \leq 0.15$ regardless of



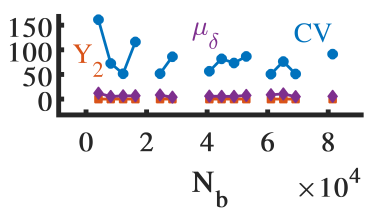
(a) Butterfly count



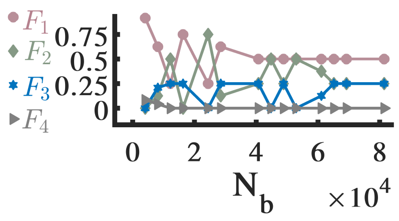
(b) CV, μ_i, Y_2 of $P(S_i)$



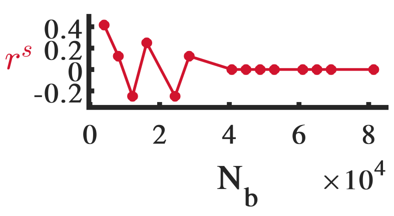
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

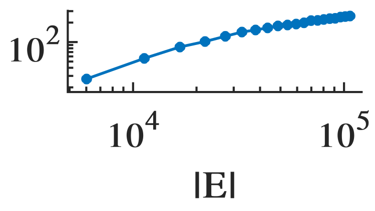


(e) F_1, F_2, F_3, F_4

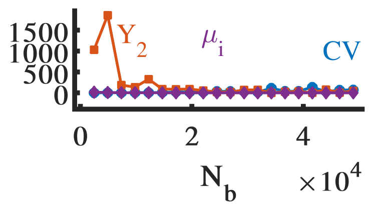


(f) r^s

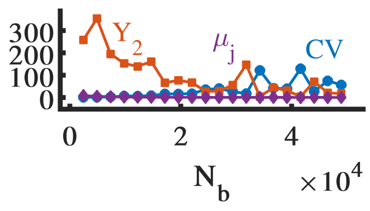
Figure 4.16: Mixing patterns of butterflies in BM stream with $p_l = 0.3$, $p_h = 0.5$.



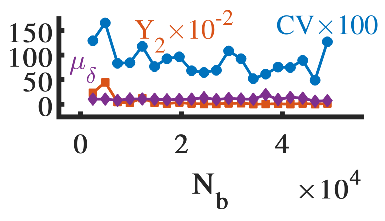
(a) Butterfly count



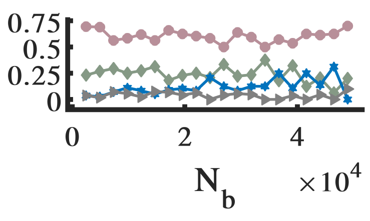
(b) CV, μ_i, Y_2 of $P(S_i)$



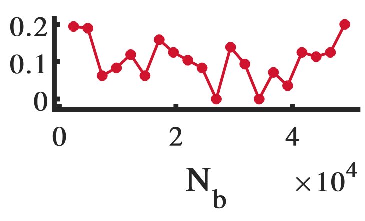
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

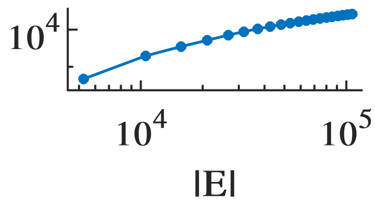


(e) F_1, F_2, F_3, F_4

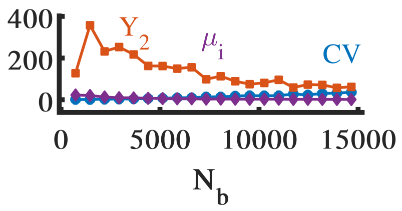


(f) r^s

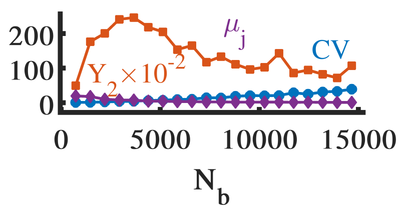
Figure 4.17: Mixing patterns of butterflies in BM stream with $p_l = 0.5$, $p_h = 0.7$.



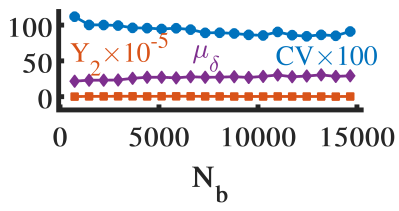
(a) Butterfly count



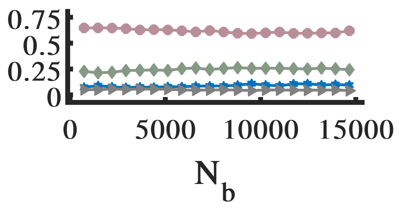
(b) CV, μ_i, Y_2 of $P(S_i)$



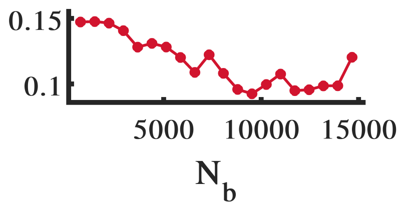
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

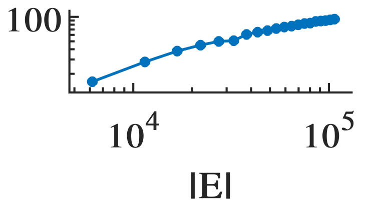


(e) F_1, F_2, F_3, F_4

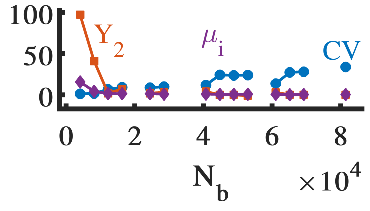


(f) r^s

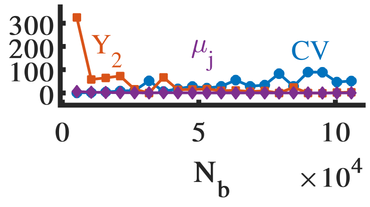
Figure 4.18: Mixing patterns of butterflies in BM stream with $p_l = 0.7$, $p_h = 0.9$.



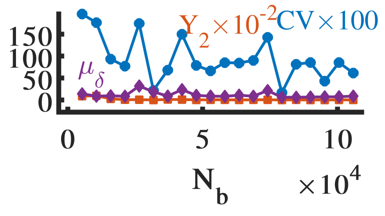
(a) Butterfly count



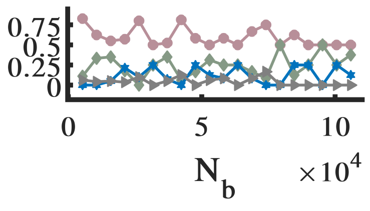
(b) CV, μ_i, Y_2 of $P(S_i)$



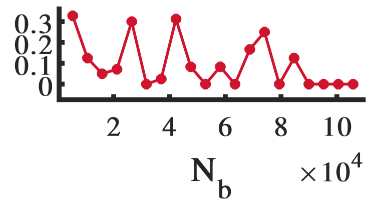
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

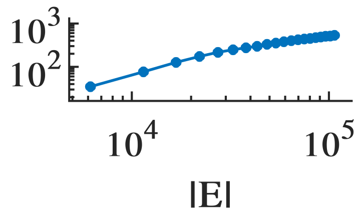


(e) F_1, F_2, F_3, F_4

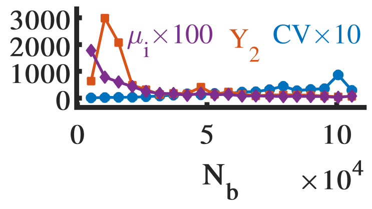


(f) r^s

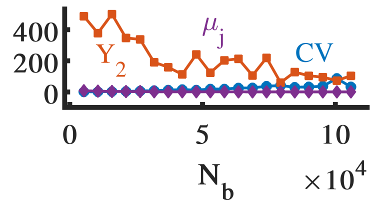
Figure 4.19: Mixing patterns of butterflies in CNM stream with $p = 0.3$.



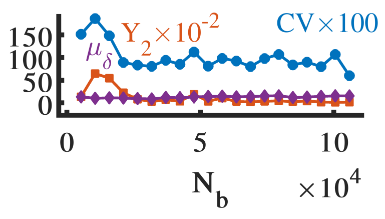
(a) Butterfly count



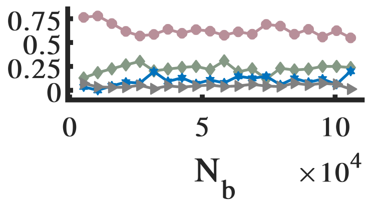
(b) CV, μ_i, Y_2 of $P(S_i)$



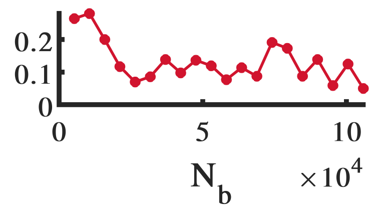
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

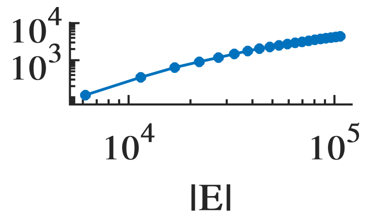


(e) F_1, F_2, F_3, F_4

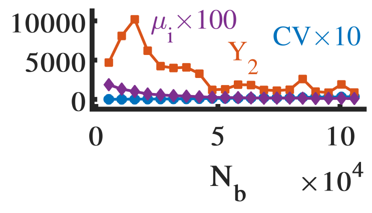


(f) r^s

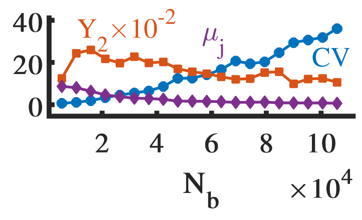
Figure 4.20: Mixing patterns of butterflies in CNM stream with $p = 0.5$.



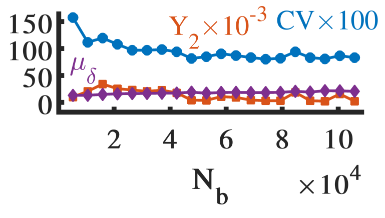
(a) Butterfly count



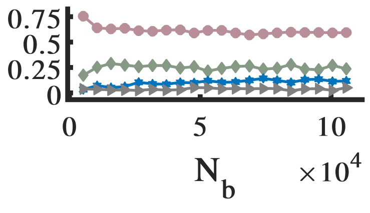
(b) CV, μ_i, Y_2 of $P(S_i)$



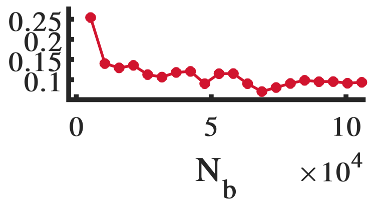
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

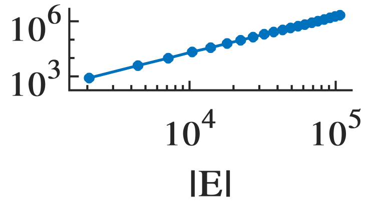


(e) F_1, F_2, F_3, F_4

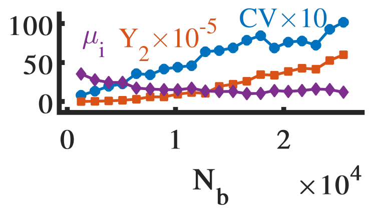


(f) r^s

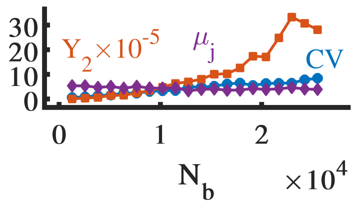
Figure 4.21: Mixing patterns of butterflies in CNM stream with $p = 0.7$.



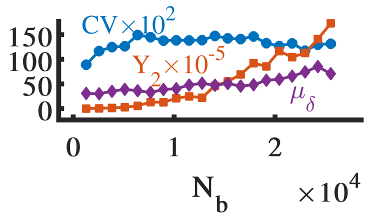
(a) Butterfly count



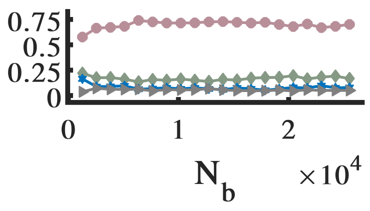
(b) CV, μ_i, Y_2 of $P(S_i)$



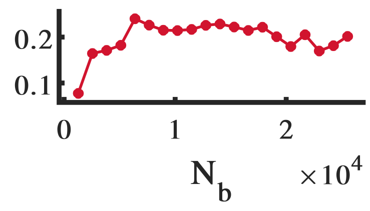
(c) CV, μ_j, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

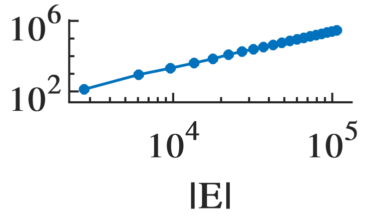


(e) F_1, F_2, F_3, F_4

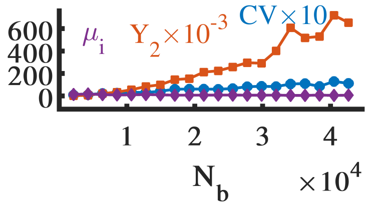


(f) r^s

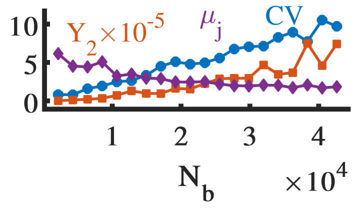
Figure 4.22: Mixing patterns of butterflies in DDM stream with $p_e = 0.3$.



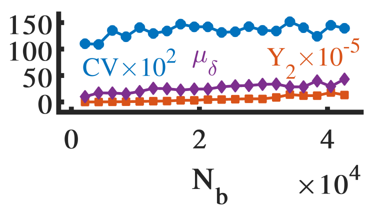
(a) Butterfly count



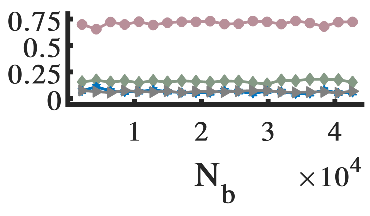
(b) CV, μ_i, Y_2 of $P(S_i)$



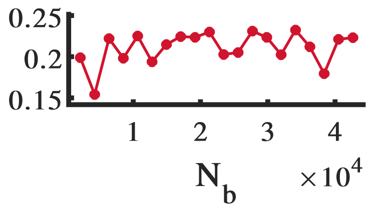
(c) CV, μ_i, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$

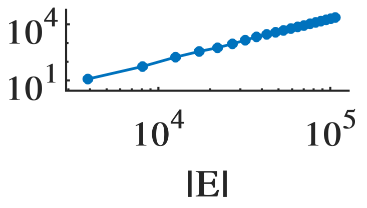


(e) F_1, F_2, F_3, F_4

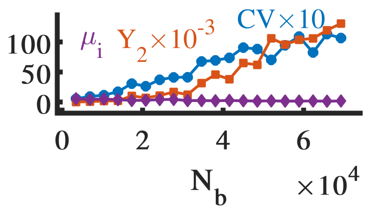


(f) r^s

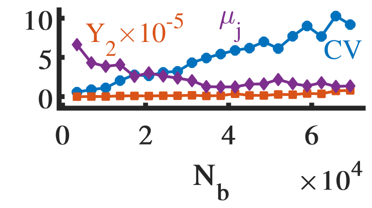
Figure 4.23: Mixing patterns of butterflies in DDM stream with $p_e = 0.5$.



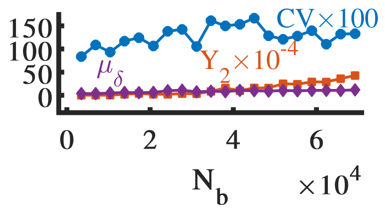
(a) Butterfly count



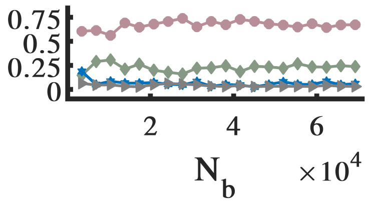
(b) CV, μ_j, Y_2 of $P(S_i)$



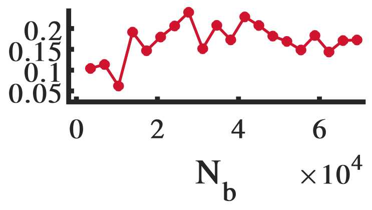
(c) CV, μ_i, Y_2 of $P(S_j)$



(d) CV, μ_δ, Y_2 of $P(\delta)$



(e) F_1, F_2, F_3, F_4



(f) r^s

Figure 4.24: Mixing patterns of butterflies in DDM stream with $p_e = 0.7$.

$[L_{min}, L_{max}]$, M, and β. The value of r^s is stable at a positive value, which is higher for lower connection probabilities ρ. As the connection probability ρ increases, the probability of establishing connections between the newly added vertices with low strength and high strength neighbours of the PRW vertices increases, therefore the assortativity level decreases. Figure 4.27, depicts the evolution of r^s as the number of butterflies grows to 10^7 in S-Amazon stream with parameter switch in the middle of stream generation. Compared to the stream with static parameters, the data points of streams with dynamic parameters follow the same pattern after the switch of $[L_{min}, L_{max}]$, M, and β. Increasing the value of ρ slightly decreases r^s, yet the steady state is retained after the switch.

Efficiency. Figure 4.26 displays the time for generation as the stream grows to 10^7 sgrs in S-Amazon stream with different parameter configurations. Data points are not clustered by colours (corresponding parameters $[L_{min}, L_{max}]$, M, and β) in rows a, c, and d, however they are clustered by ρ and ordered by both ρ and $[L_{min}, L_{max}]$. That is, the generation time is not impacted by M (Figure 4.26(c)) or β (Figure 4.26(d)), however it is affected by ρ and $[L_{min}, L_{max}]$. As the connection probability ρ increases, the generation time decreases since the number and the size of bursts created at each time step increases (Figure 4.26(b)). As the range of random walk length $L \in [L_{min}, L_{max}]$ increases, the generation time decreases since the size of bursts created at each time step increases (Figure 4.26(a)). Figure 4.27, depicts the time for generation as the stream grows to 10^7 sgrs in S-Amazon stream with parameter switch in the middle of stream generation. Switching M and β gradually decreases the slope of generation time curve, while switching L and ρ promptly and significantly changes the slope. This confirms that L and ρ significantly determine the generation time.

Burst Size. Figure 4.28 shows the frequency distribution of all burst sizes in S-Amazon with 10^7 sgrs generated by different parameter configurations. For all values of ρ and L_{max}, by increasing M, the range of burst sizes expands; Therefore, M significantly impacts burst sizes. Also, the maximum burs size increases as ρ and L_{max} increase.

4.4.3 Computational Complexity

In the following, it is shown that high degree/strength vertices and PRW hop count determine the computational cost of *sGrow*.

THEOREM 1 *The worst case computational complexity of* sGrow *in each window with graph* G=$(V_i \cup V_j, E)$ *and PRW parameter* L *is* $O(S_{max} + L(N^j_{max} + N^i_{max}))$*, where* S_{max} *is the maximum strength in* G *and* N^i_{max} *and* N^j_{max} *are the maximum number of i-neighbors and j-neighbors for vertices in* V_i *and* V_j.

PROOF 1 sGrow*'s computations at each window are dominated by burst additions as the initialisation, sgr addition/removals, and window sliding take* $O(1)$ *computation. The worst case computational complexity of burst additions is the following.*

$$O(SPS(.)) + O(PRW(.)) + O(2addburst(.)) \tag{4.1}$$

Let us assume that the maximum strength in G=$(V_i \cup V_j, E)$ *is* S_{max}*,* L *is the parameter for the PRW hop count, and the maximum number of i-neighbors and j-neighbors for vertices in* V_i *and* V_j *are* N^i_{max} *and* N^j_{max}*. Accordingly, we would have the following complexities:* $O(SPS(V_j)) = S_{max}$ *since the value assignments and corresponding operators, and the list shuffling take one unit of computations, and the outer for loop is parallel and the inner loop sequentially performs* S_{max} *computational units. We have* $O(PRW(u^0_j, false, G, L)) = (L/2{+}1)N^j_{max}{+}(L/2)N^i_{max}$ *since* $|PRW_j| = L/2{+}1$ *and* $|PRW_i| = L/2$*. Also,* $O(addburst(v^l_i, v^l_j, PRW_j, G, \mathfrak{R}, \rho)) = O(addburst(v^l_i, v^l_j, PRW_i, G, \mathfrak{R}, \rho)) =$

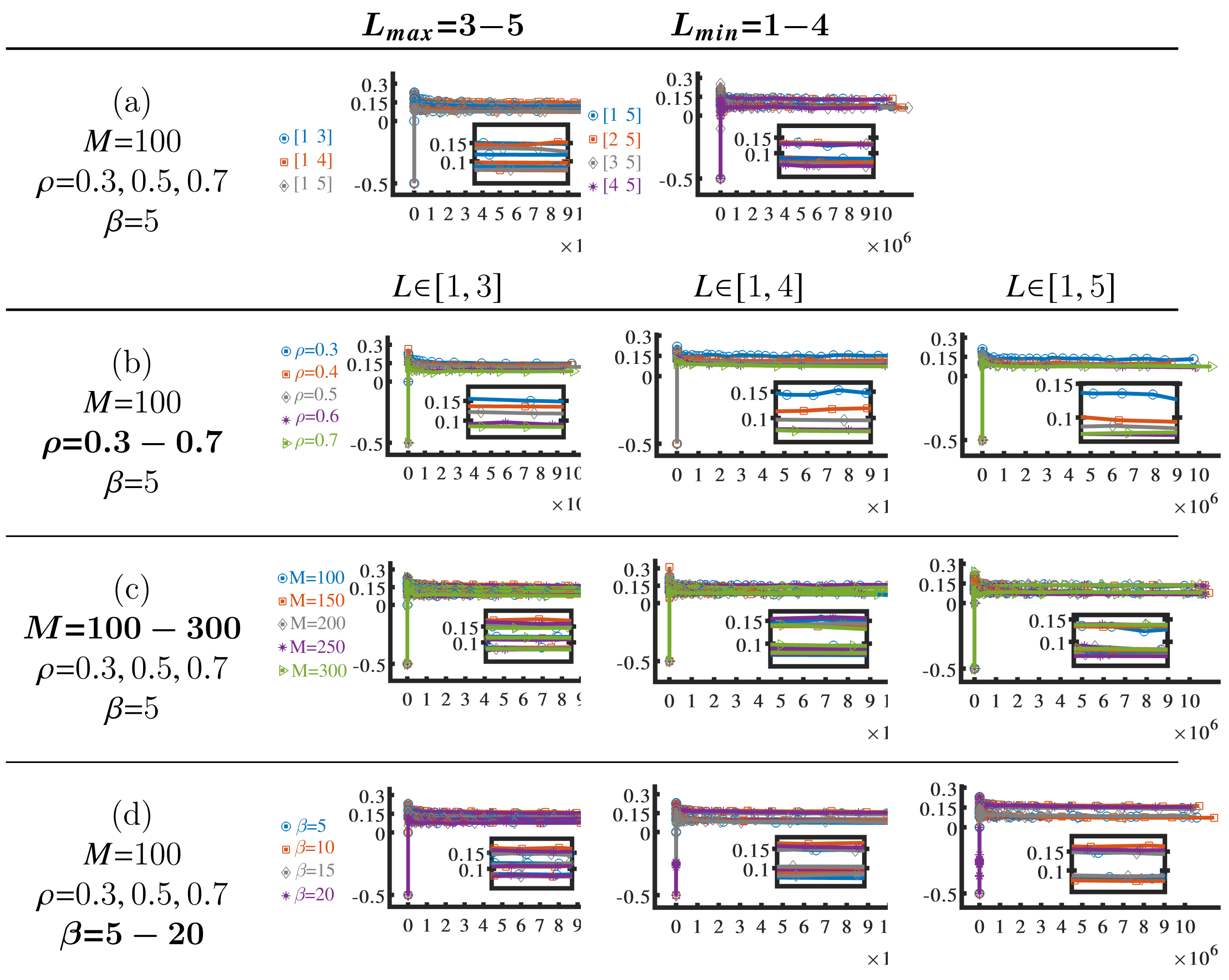


Figure 4.25: Strength assortativity localization factor versus butterfly count for S-Amazon.

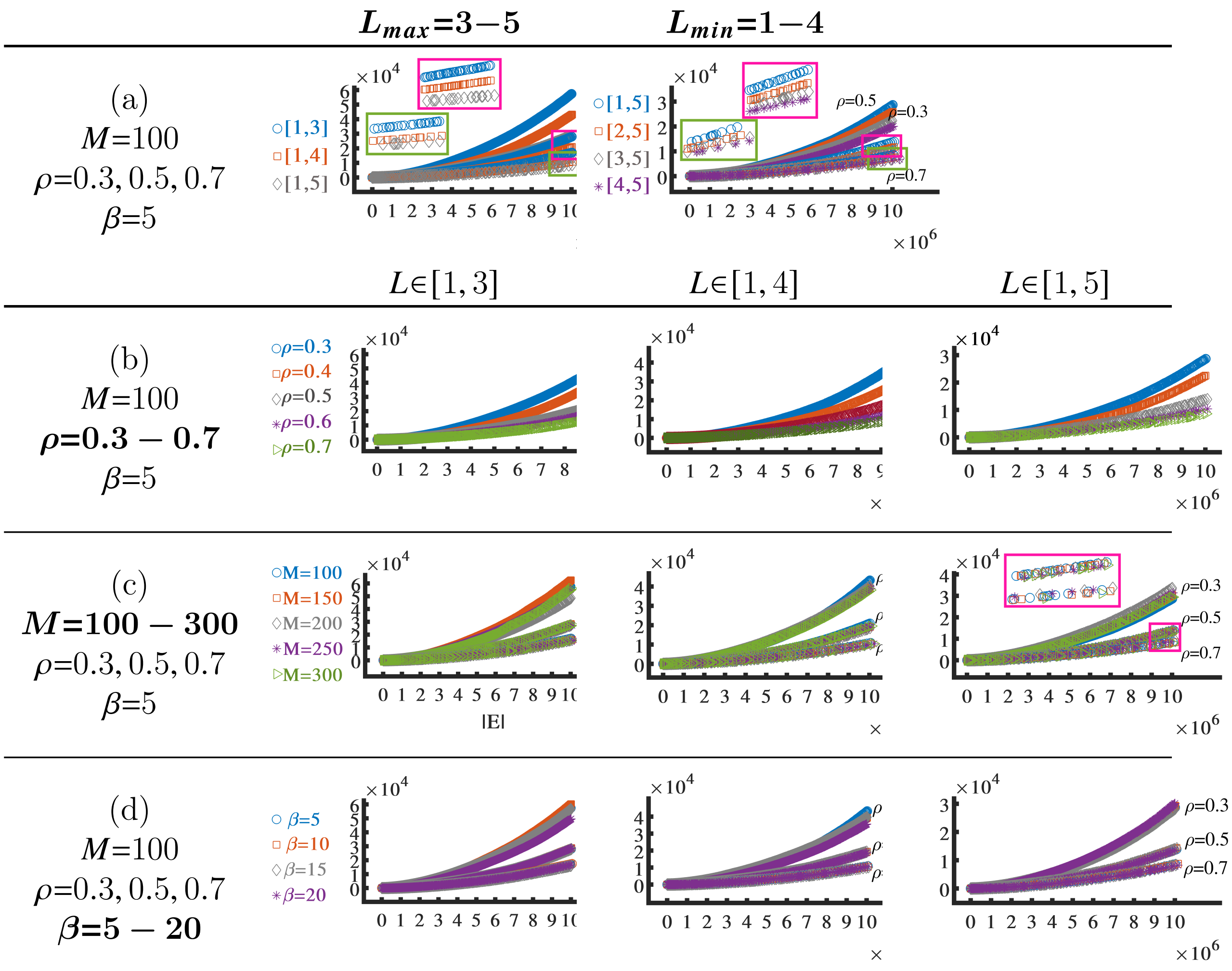


Figure 4.26: Generation time (s) versus the number of sgrs for S-Amazon.

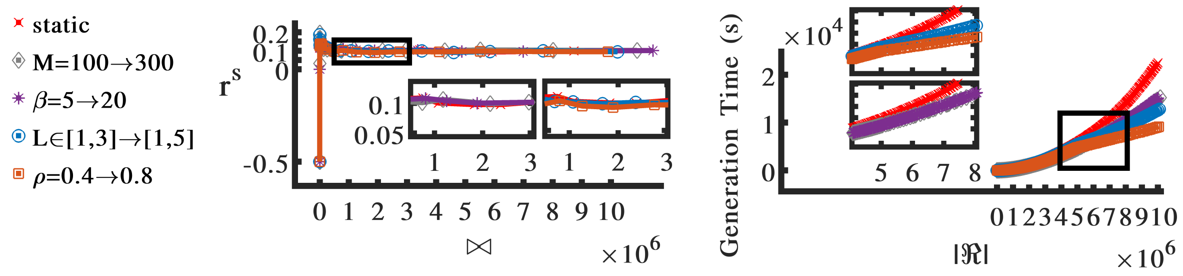


Figure 4.27: Strength assortativity localization factor versus butterfly count and generation time (s) versus the number of sgrs in S-Amazon. Parameters switch from $M = 100$, $\beta = 5$, $L \in [1, 5]$ and $\rho = 0.4$ to $M = 300$, $\beta = 20$, $L \in [4, 5]$ and $\rho = 0.8$ after generation of 5×10^6 sgrs.

$O(1)$ since the value assignments, and probabilistic connections are done in $O(1)$ and Step 3 is performed via a parallel loop. Therefore, the total complexity of burst additions would be $S_{max} + (L/2 + 1)N^j_{max} + (L/2)N^i_{max} + O(1)$, i.e. high degree/strength vertices and PRW hop count determine the computational cost.

4.5 Summary

Essence and features. Study of the existing local rules for graph growth that yield skewed distributions, degree correlation, and cohesive structures shows that implicit degree-driven preferential attachment and copying mechanisms or solely strength-driven preferential attachment with random assignment of timestamps to the edges can only partially preserve the observed patterns but are not effective enough to reproduce these patterns simultaneously. Therefore, a set of microscopic mechanisms, in the body of a proposed streaming growth model called *sGrow*, are introduced. These mechanisms are based on realistic sgr generation, probabilistic connections, and strength-driven preferential random walks which explain the emergence patterns of streaming butterflies. *sGrow* is designed as an iterative addition of bursts of sgrs which satisfies streaming data model, preserves realistic patterns of butterfly emergence quantitatively and qualitatively, and makes the stream generation scalable. Moreover, *sGrow* enables generating sequence of bipartite edges attributed with timestamps and weights, isolated/out-of-order edges, and four-vertex graphlets.

Use cases. *sGrow* suits the following cases:

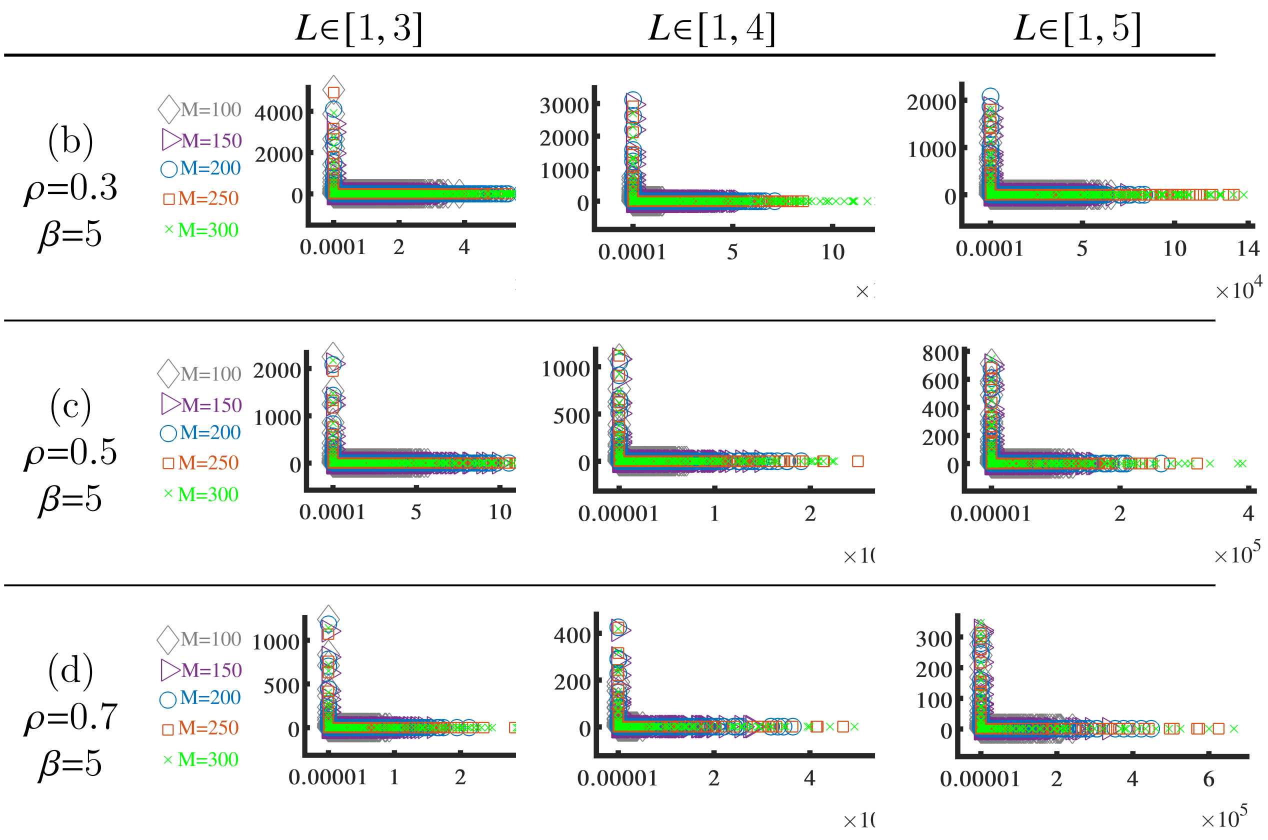


Figure 4.28: Frequency of burst sizes in S-Amazon.

- Streaming graph benchmarks. Performance evaluation of algorithms including but not limited to butterfly-based algorithms relies on considering the characteristics of input graphs [22, 25, 63]. Given the lack of streaming graph generators, *sGrow* assists in understanding the important graph characteristics as well as providing realistic *configurable* and scalable graphs for stress testing purposes.

- Machine learning benchmarks. Collecting/annotating the training/testing datasets for graph-based models in domains such as outlier detection and computer vision is challenging due to the nature of data (e.g. rare outliers and diverse image instances), and high cost of manually labeling the instances [367, 329, 258]. Solutions include building benchmark datasets via artificial instance injection [79, 367, 114] and applying weakly-supervised techniques [216, 329, 140]. *sGrow*'s analytical approach can be extended to such domains to inform the design of graph-based models with the temporal connectivity patterns and also generating realistic yet synthetic datasets to which the artificial instances are injected.

- Concept drift modeling. To improve the performance of online adaptive learning algorithms in stream-based recommender systems for web activities, it is important to consider the temporal evolution of modeled concepts due to a change in the distribution of log data or a change in the relation between data and target variable (i.e. concept drift) [15, 274, 125]. Considering a butterfly as two users with mutual preferences and two items with mutual perceptions, *sGrow* impacts modeling the parallel drift of concepts such as user preferences and item perception.

- Algorithm developments. Graph analytics and generative models utilise microscopic mechanisms and graph pattern for algorithm design [329, 297]. *sGrow*'s microscopic mechanisms and growth patterns benefit these cases as well.

Parameter Configuration. The introduced techniques with configurable parameters enable generating realistic bipartite streaming graphs with generation time sub-linear with respect to the number of sgrs. The more bursty the stream, the lower the generation time. In the following, a reference guide for configuring the parameters is elaborated.

- M: The upper bound for the random number of new batched sgrs added at each time step does not impact the strength assortativity patterns and the generation time. This parameter can be comfortably used to adjust the level of burstiness of the streaming graph without affecting the performance of the generative algorithm.

- ρ: The probability of creating an edge between each random walk vertex and a random vertex or an edge between each new sgr and neighbours of the random walk vertices impacts the level of strength assortativity but not its steady state. It also affects the generation time. This parameter can be used to trade off scalability and the level of strength assortativity. The default value is $\rho = 0.3$, yet increasing ρ would decrease the generation time and strength assortativity level. Values less than or equal to 0.7 ensure positive strength assortativity. Also, ρ determines the probability of out-of-order sgrs.

- $[L_{min}, L_{max}]$: The range for random length of PRW backbone used for establishing burst of sgrs does not impact the strength assortativity patterns. It affects the generation time. This parameter can be used to increase the scalability of the stream generation. The default range is $[1, 2]$ and shifting/expanding the range by increasing the lower and/or upper bound would decrease the generation time.

- β: The sliding window parameter used as the frequency and the size of sliding does not impact the strength assortativity patterns and generation time. This parameter can be comfortably used for creating streams in which sgrs are semantically time-sensitive and need a user-specified slide parameter. The default value is $\beta = 5$, while any other value can be set.

Evaluations. Comprehensive evaluations validate the efficacy of *sGrow* in realization of streaming growth patterns effectively and independent of initial conditions, scale and temporal characteristics, and model configurations. Analyses also verify the robustness of *sGrow* in generating streaming graphs based on user-specified properties for the scale and burstiness of the stream, level of strength assortativity, probability of out-of-order streaming records, generation time, and time-sensitive connections.

Chapter 5

Streaming Graph Analytics - Butterfly Counting

Enumerating and counting butterflies is important in measuring graph cohesion and clustering or community structure [14]. Clustering or community structure is measured by the transitivity/clustering coefficient that is computed as the fraction of caterpillars which form a butterfly [210, 363, 14]. Graph cohesion can be measured by the number of butterflies-per-vertex and by the local clustering coefficient. Study of such local structural measures unveils hidden ordering and hierarchies in graphs displaying structural deviations from uncorrelated random connections [78, 275, 242]. A recent study investigates the predictive performance of deep neural networks by means of clustering coefficient [354]. Other applications are realistic graph models [14, 179] and representative graph sampling [362]. The study of different phenomena in complex graphs such as social collective behaviours [102], synchronization [299, 379], information propagation [175], and epidemic spreading [268] rely on the clustering coefficient. Moreover, clustering coefficient plays an important role in graph analytics tasks such as link prediction [160] and community detection [363], and in general any graph processing algorithm relying on counting the mutual neighbours or Jaccard similarity. For example, in author-publication streams, it is important to count the butterflies to discover cohesive substructures with high clustering coefficient representing the research groups; in user-item and trader-stock streams, butterfly counting helps detect abnormal activities by means of calculating clustering coefficient or discovering the k-bitruss subgraphs. Also, the distribution of local clustering coefficient is used as a feature to uncover statistical differences between normal and fraudulent data in applications such as spam detection [45]. Furthermore, butterfly counting provides similarity measures for product recommendation over user-product streams. Other application areas include hashtag recommendation in user-hashtag streams of social networks and intrusion detection in source-destination data flows of telecommunication networks.

Motif counting is a fundamental problem in large-scale network analysis [301, 256, 51] and given the manifest of new data models such as streaming graphs, it is critical to study this problem in streaming contexts. Particularly, butterfly counting is computationally expensive, and known techniques do not scale to large graphs; the problem is even harder in streaming graphs. While there are many algorithms proposed for triangle counting in static and streaming unipartite graphs, counting the butterfly occurrences is still a challenging problem in bipartite streaming graphs due to the following reasons.

- Bipartite streaming graphs are triangle free and current approaches for triangle counting over unipartite streams (e.g. [45, 46, 72, 196, 303, 170, 304]) are not useful, because, as Sanei-Mehri et al. [286] describe, the bimodal structure should be considered for effectiveness.
- Exact butterfly counting algorithms (e.g. [330, 149, 301, 285, 331]) are not able to deal with the unboundedness and high velocity of temporal evolutions in graph streams.
- Simultaneously achieving effectiveness and efficiency is a pressing issue which requires special attention [117].

5.1 Butterfly Counting over Streaming Graphs

In view of the streaming graph properties discussed in Chapter 1, the precise problem definition is as follows: *Given a sub-sequence of streaming graph records ordered by their timestamps, how to compute the total number of butterflies in the entire emerging graph G at time point t – denoted as $B(t)$.*

Computing the exact value of $B(t)$ over a streaming graph is not feasible, since complete access to the graph is not possible. It is known [26, 33] that without knowing the size of the streaming input data, it is not possible to determine the memory required for processing the data, and unless there is unbounded memory, it is not possible to compute exact answers for this data stream problem. Exact counting is not even possible in massive static graphs [220, 68]. Even if it is assumed that enough memory is available to maintain the input graph corresponding to a subset of the stream from a certain starting point in time to the current time, the growing graph needs to be continually maintained. Moreover, the iterative and stateful algorithm that retains the intermediate results and state of the computation should be continually executed. This is equivalent to using landmark windows. These append-only windows are computationally expensive for exact butterfly counting. According to the following analysis, given a graph snapshot at t_1, a landmark window appending $|W|$ new edges and ΔV_i new i-vertices to the graph snapshot at t_2 incurs a lower bound for computations of $O((|V_{i,t_1}| + \Delta V_i)K_{i,t_1}K_{j,t_1})$ which is, in the worst case, on the order $O(c|V_{i,t_1}|^3 + |W||V_{i,t_1}|^2)$ making landmark windows computationally too expensive.

The alternative is approximation. One such approach is to use random sampling or sparsification [73, 285, 286], which requires determining the sampling probability, reservoir size, and scaling factor. The sampling process is done several times and can be a potential overhead, lowering the processing throughput. Moreover, to provide accurate results the sampling is performed over the entire prefix of the received streaming graph records and this has high memory and time overhead unless the sampling rate is increased, which results in low processing throughput. Consequently, providing both efficiency and effectiveness is a major challenge in sampling-based methods.

Another approximation approach is to use sliding windows over the incoming sgrs. Sliding windows are known as a natural approximation method over data streams [33]. Most existing streaming proposals (e.g. [73, 37, 74]) assume that (a) all the edges incident to a vertex arrive together (i.e. incidence streams), and (b) vertex degrees are bounded. Neither of these is likely to hold in real-life streaming graphs. In the following, a butterfly counting framework is introduced that can efficiently return an accurate answer over any stream without these unrealistic assumptions. It has been shown that an approximate butterfly count that bounds the relative error to $0 < \delta < 0.01$ needs to store the entire graph, which is possible in $\theta(n^2)$ where n is the number of vertices [286]. This is not feasible in streaming systems.

Computational Complexity of Exact Counting. An exact counting algorithm over streaming graphs requires a landmark window to collect all the streaming edges from a beginning time point to *now*. A multi-pass counting algorithm is executed over the graph snapshot in the window. In what follows, a landmark window with subsequent window expansions yields graph snapshots at two subsequent time points t_1 and t_2. The computational complexity of a butterfly counting algorithm with $O(\sum_{i_1 \in V_i} \sum_{j_1 \in N_{i_1}} deg(j_1))$ over a streaming graph that follows a densification power law is analyzed at these two time points. The lower bound of i-degree at time t_1 is denoted as K_{i,t_1}.

Given a graph snapshot at t_1, a landmark window appending $|W|$ new edges and ΔV_i new i-vertices to the graph snapshot at t_2 with lower bound i-degree at time t_1 denoted as

K_{i,t_1}, a butterfly counting algorithm with $O(\sum_{i_1 \in V_i} \sum_{j_1 \in N_{i_1}} deg(j_1))$ incurs a lower bound for computations of $\Omega((|V_{i,t_1}| + \Delta V_i) K_{i,t_1} K_{j,t_1})$ which is in worst case of $\Omega(c|V_{i,t_1}|^3 + |W||V_{i,t_1}|^2)$ due to the following.

1. It has been shown empirically that at any time t, the number of edges in the real world graph snapshots $E(t)$ follows a power law function of the number of vertices $|V(t)|$. This is known as *densification power law* denoted as $|E(t)| \propto |V(t)|^a$, where a is the exponent that generally lies strictly between 1 (asserting constant average degree over time) and 2 (corresponding to the extremely dense graphs where each vertex is, on average, connected to a constant fraction of all vertices)[201]. According to the densification power law, the graphs grow superlinearly and the vertex out-degrees grow over time. This natural pattern in bipartite graphs with directed edges from i-vertices to j-vertices is interpreted as the growing of average i-vertex degrees.

   Also, when the power law exponent is $a = 2$ (worst case), we would have

   $$b|V_{j,t_1}| \leq K_{i,t_1} \tag{5.1}$$

   $$b|V_{j,t_2}| \leq K_{i,t_2} \tag{5.2}$$

   where b is a constant $0 < b < 1$ and K_{i,t_1} and K_{i,t_2} are the lower bound i-degree at times t_1 and t_2. That is the shared minimum degree of i-vertices in the graph snapshot at times t_1 and t_2, respectively.

2. Now, consider a butterfly counting algorithm for static graphs, which has the computational complexity of $O(\sum_{i_1 \in V_i} \sum_{j_1 \in N_{i_1}} deg(j_1))$. The lower bound computational complexity of running the counting algorithm at t_1 and t_2 will be:

   $$\Omega(|V_{i,t_1}| K_{i,t_1} K_{j,t_1}) \tag{5.3}$$

   $$\Omega(|V_{i,t_2}| K_{i,t_2} K_{j,t_2}) \tag{5.4}$$

   and according to the densification power law,

   $$\Omega(|V_{i,t_2}| K_{i,t_1} K_{j,t_1}) < \Omega(|V_{i,t_2}| K_{i,t_2} K_{j,t_2}) \tag{5.5}$$

3. Furthermore, the expansion of a landmark window at time t_2 introduces $|W|$ new edges which include $\Delta V_i \leq |W|$ new i-vertices. That is,

   $$|V_{i,t_2}| = |V_{i,t_1}| + \Delta V_i \leq |V_{i,t_1}| + |W| \tag{5.6}$$

   Therefore, the lower bound of computations at t_2 will be $\Omega((|V_{i,t_1}| + \Delta V_i) K_{i,t_1} K_{j,t_1})$.

4. The worst case scenario can be analyzed when there are $|W|$ new i-vertices (right side of inequality 5.6). Accordingly, the worst case lower bound at t_2 will be $\Omega((|V_{i,t_1}| + |W|)K_{i,t_1}K_{j,t_1})$. When the power law exponent is $a = 2$ (Equation 5.1), the worst case lower bound at t_2 will become $\Omega(b(|V_{i,t_1}| + |W|)|V_{j,t_1}|K_{j,t_1})$. Since $|V_{j,t_1}|K_{j,t_1} \propto d|E^{t_1}| \propto d|V_{i,t_1}|^2$, where $0 < d < 1$, the worst case lower bound at t_2 will be $\Omega(c(|V_{i,t_1}|+|W|)|V_{i,t_1}|^2) = \Omega(c|V_{i,t_1}|^3 + |W||V_{i,t_1}|^2)$ where $0 < c < 1$.

In order to evaluate the computational footprint of a main memory algorithm that returns exact answers, it is necessary to consider the memory required to store the input and the state of computations as well as the processing latency. Although steaming graphs have unbounded size, suppose the memory requirement is relaxed with the (unrealistic) assumption that enough memory is available to store the streaming graph without discarding any graph elements. Performing exact butterfly counting over the dense real streams after adding each batch of edges to a graph snapshot at t_1 incurs a high lower bound of computational footprint on the order $\Omega(|V_{i,t_1}|^3)$. Interpreting this computational complexity according to the time and memory overhead per unit of computation and considering the growth patterns (i.e. growing number of vertices) elaborates the overhead of exact butterfly counting over streaming graphs. This highlights the essence of approximate counting methods which do not introduce expensive computations.

In the following, first, the related works on butterfly counting are reviewed in Section 5.2. This is followed by introducing a butterfly counting approach called *sGrapp* in Section 3.2. The design of *sGrapp* is informed by the graph mining insights from the Phase 1 of the streaming graph analysis. The computational complexity of *sGrapp* and its performance are evaluated in Section 5.4. *sGrapp* demonstrates superior performance in terms of estimation error for butterfly count and processing throughput compared to baseline algorithms. Table 5.1 lists the frequent notations in this chapter.

5.2 Related Works

The existing works in butterfly counting can be classified along three dimensions: graph characteristics (bipartite/unipartite), data location (disk-resident/in-memory) and graph availability (static/dynamic/streaming). Detailed coverage of each design point is beyond the scope of this thesis; in the following, the focus is on two particular design points that are most relevant: static bipartite graphs and streaming bipartite graphs.

Table 5.1: Frequent notations in this chapter.

Notation	Description
α	Butterfly densification power-law exponent for the inter-window butterflies
$B(t)$	The number of butterflies since the initial time point until t
B^{W_k}	Number of butterflies introduced by at least one vertex in W_k
$\hat{B}_k$	Estimation of number of butterflies at time $t = W_k^e$
$B_G^{W_k}$	Number of butterflies in graph corresponding to W_k
B^{interW}	Number of inter-window butterflies
$\hat{B}^{interW}$	Estimation of number of inter-window butterflies
K_{i,W_k}	the lower bound of degree of i(j)-vertices in W_k
$V_{i,W_k}/E_{W_k}$	Set of i-vertices/edges in $[W_k^b, W_k^e)$
E_k	Set of edges in the interval $[W_0^b, W_k^e)$
P	Sampling probability in FLEET algorithms
γ	Sub-sampling probability
M	Reservoir capacity in FLEET algorithms

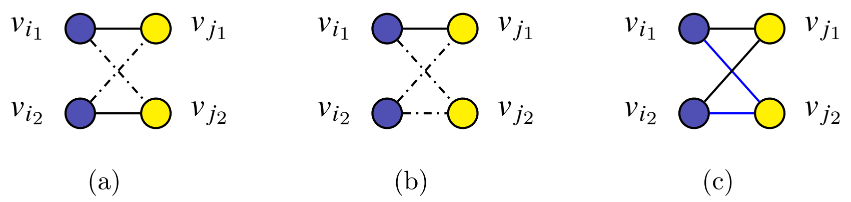


Figure 5.1: Butterfly counting methods.

5.2.1 Counting in Static Bipartite Graphs

The literature on counting (bi)cliques in static bipartite graphs (e.g. [330, 331, 285]) and static unipartite graphs (e.g. [334, 152]) is quite rich. A major challenge in this context is the massive size of these graphs. Some studies (e.g. [152, 92, 45, 158, 157, 254]) have focused on disk-resident data and optimised I/O access patterns for counting the exact number of cliques. Other studies consider in-memory algorithms and use random sampling so that the induced graph can fit in main memory for estimating the number of (bi)cliques [73, 285]. There are also studies (e.g. [176, 27]) that propose scaling out computation by parallelization.

Butterfly counting algorithms in bipartite graphs follow either vertex-centric or edge-centric processing. One straightforward edge-centric approach is to take each pair of disjoint edges $(e_{i_1,j_1}, e_{i_2,j_2})$ in the graph (Figure 5.1(a)) and check for the existence of the two other edges that complete the butterfly pattern. The complexity of this approach is $O(|E|^2)$, which is too expensive for graphs with a high number of edges. Another edge-centric approach [91] takes an edge e_{i_1,j_1} and examines the existence of the three complementary edges. That is, the algorithm checks the connections between neighbours of i_1 and neighbours of j_1 denoted as j_2 and i_2, respectively, to see whether they are connected by an edge e_{i_2,j_2} (Figure 5.1(b)). This approach can be implemented with an algorithm that has complexity $O(\sum_{\langle i_1,j_1\rangle\in E} Min(deg(i_1), deg(j_1)))$, but this is not appropriate for dense graphs with high number of edges and high average degrees. The state-of-the-art approach [330, 331, 285] is vertex-centric that takes a vertex v_i and traverses all two-hop neighbours to identify triples $\langle i_1, j_1, i_2\rangle$ and $\langle i_1, j_2, i_2\rangle$. That is, it finds all triples (i.e. two-paths) with common incident vertices (i.e. the same two-hop neighbour) and then combines them to get the number of all butterflies (Figure 5.1(c)). The complexity of this approach is $O(\sum_{i_1\in V_i}\sum_{j_1\in N_{i_1}} deg(j_1))$, which is challenging for graphs with high average i- and j-degrees as a result of traversing two hop neighbours [331].

5.2.2 Counting in Streaming Bipartite Graphs

In the streaming graph context, the literature is also rich for counting in unipartite graphs (e.g. [335, 334, 73, 37, 45, 74, 53, 67]). A state-of-the-art butterfly counting study over bipartite streaming graphs is FLEET [286], which introduces a suite of algorithms. FLEET1 samples the sgrs of a landmark window with probability P into a reservoir with fixed capacity M to bound the memory consumption and increments the butterfly count by the number of incident butterflies for each sampled edge. When the size of reservoir exceeds M, the sgrs are sub-sampled with probability γ and the butterfly count is set to the exact number of butterflies in the reservoir. The sampling probability is then multiplied by γ for the following sgrs. FLEET2 avoids re-computing the exact number of butterflies in the reservoir during the sub-sampling iterations. FLEET3 avoids re-computation and also updates the estimate before sampling the sgrs into the reservoir. FLEETSSW uses count-based sliding windows with limited graph size in each window, and FLEETTSW uses time-based sliding windows with fixed window length across windows. To overcome the variable number of sgrs inside each window, FLEETTSW assumes an upper-bound for the number of sgrs in a window on top of a first-in, first-out (FIFO)-based sampling scheme. As will be discussed in Section 5.3, there exist a number of inter-window butterflies in the stream; these are missed by the FLEET algorithms which use sliding windows. The FLEET variants which are based on landmark windows (FLEET1, FLEET2, and FLEET3) do not provide a proper trade-off between accuracy and processing throughput. Moreover, FLEET requires determining a sub-sampling probability and a normalization factor to scale-up the estimation computed over the sampled sgrs, and the specification of a time when the result is ready to be returned. FLEET requires a sufficiently large amount of memory to guarantee a desired level of accuracy.

5.3 sGrapp

Section 5.1 proved the impossibility of exact counting of butterflies in a streaming graph and discussed the constraints for a proper window-based approximation. This section describes a new approximate butterfly counting algorithm called *sGrapp* that uses tumbling windows. Tumbling windows are used in order to avoid double counting of repeated butterflies, as will be described – tumbling windows do not overlap when windows move, thus avoiding the double-counting problem.

It is possible to adopt a time-based tumbling window model with lazy computation and calculate the exact number of butterflies introduced by each window of disjoint edge insertions, W_k, at the end time of the window denoted by $B_G^{W_k}$, and increment the cumulative value accordingly: $B(t = W_k^e) = B(t = W_{k-1}^e) + B_G^{W_k}$. This processing is incremental. An important issue is that there may exist some butterflies that are formed by the edges with large inter-arrival times (heavy and long tail in Figure 3.10). These butterflies, referred to as *inter-window butterflies*, are not captured within one window (unless it is sufficiently large). However, setting the window length to a large value to cover the inter-window butterflies imposes a high computational footprint in terms of memory and time. This conflicts with the goal of using a windowed approach to lower this footprint by performing incremental processing over subsets of sgrs.

sGrapp addresses this issue by (a) using tumbling windows whose lengths are adaptive to the temporal distribution of sgrs (hence dealing with bursty arrivals) instead of using heavily-loaded and lengthy sliding windows, and (b) approximating the number of inter-window butterflies individually instead of aggregating their count with that of window W_k. These two ideas are the keys to simultaneously enabling efficiency and accuracy. Precisely, *sGrapp* estimates the number of butterflies from the beginning of the first window $t = W_0^b$ until the end of kth window denoted as $\hat{B}(t = W_k^e) = \hat{B}_k$ by counting the exact number of butterflies in the graph corresponding to the current window W_k denoted as $B_G^{W_k}$ and approximating the number of inter-window butterflies denoted as $\hat{B}^{interW}$. The estimated cumulative value would be $\hat{B}_k = \hat{B}_{k-1} + B_G^{W_k} + \delta(k \neq 0)\hat{B}^{interW}$, where the function $\delta(\cdot)$ returns 1 for true input and 0, otherwise. Note that the first window W_0 has no inter-window butterflies and hence the corresponding term would become zero by means of the

Algorithm 5: Adaptive windows

```
Data: $\mathfrak{R} = \langle r^1, r^2, \cdots \rangle$, sequence of sgrs
Input:
$N_b$, Number of unique timestamps per window
Output: $x$, Analysis output collection
1  $G \leftarrow \langle V = \emptyset, E = \emptyset \rangle$, $t \leftarrow 0$, $unqt \leftarrow \emptyset$, $x \leftarrow \emptyset$, $k \leftarrow 0$, $W_k^b \leftarrow \tau^0$
2  while true do
3      $r^t = (\tau^t, p) \leftarrow sgrIngest()$
4      if $r^t \neq \emptyset$ then
5          $unqt$.add($\tau^t$)
6          $G \leftarrow updateG(r^t, G)$
7      if $unqt.size() == N_b$ then
8          $x[t] \leftarrow analysis(G)$
9          $k \leftarrow k + 1$
10         $W_k^b \leftarrow \tau^t$
11         for $e \in G : e.timestamp \leq W_k^b$ do
12             $G \leftarrow Delete(e, G)$
13     $t \leftarrow t + 1$
```

delta function.

Adaptive burst-based tumbling Windows. This subsection introduces an adaptive window framework for butterfly approximation. A main challenge with time-based windows is *how to set the length of windows*. A common approach in stream processing is setting the length of a window using a predetermined value L ($|W_i| = L$, $\forall i$). This is sensible in online querying workloads where the user may be interested in results of a particular window size. However, this is not a suitable approach in analytical workloads, such as butterfly counting, where the objective is to count occurrences over the entire timeline of the graph evolution. The temporal distributions of edge arrivals (frequency distribution of sgr timestamps) are different in each window, meaning that the number of sgrs is not uniform across all time intervals. Therefore, a fixed window size would result in windows of sgrs that cover differing numbers of timestamps, which imposes unbalanced loads on the processing algorithms, particularly in the case of sgr arrivals with bursty characteristics and non-uniform temporal distribution.

To tackle this issue, an adaptive approach is introduced to set the window length. This approach determines the window length according to the timestamps of the stream and adapts to the temporal distribution of the stream (Algorithm 5) with no assumption about the order and number of arriving sgrs per time unit. Hence, streams with differing arrival rates and temporal distributions can be accommodated. Precisely, a burst-based tumbling window (Definition 10) is used, which includes a variable number of sgrs but a certain number of *unique* timestamps in the stream, N_b. That is, given the number of unique timestamps per window N_b, sgrs are ingested to the window (Algorithm 5, lines 3-6). When N_b timestamps are seen, the window is closed and the intended analysis is performed over the corresponding snapshot (Algorithm 5, line 8). The outputs of the analysis are streamed out correspondingly. Next, the window slides forward (Algorithm 5, lines 9-10) and the retired edges are deleted from the graph snapshot corresponding to the window (Algorithm 5, lines 11-12). In tumbling windows, all the sgrs are retired when the window slides, and the graph snapshot is renewed. The time step is incremented and the algorithm continues until there is a sgr (i.e. continuously in real world streams).

This may appear as a count-based window, but it is not. A count-based window would contain a fixed number of sgrs, while here only the number of unique timestamps (i.e. the number of bursts) in the window is fixed. Therefore, it is burst-based with adaptive width since the window borders adapt to the temporal distribution of the stream. In fact adaptive windowing would reduce to count-based windowing, if and only if the temporal distribution of stream is uniform and unique timestamps occur with equal frequency numbers. Therefore this windowing mechanism is general and conforms to real streams. Sequential adaptive windows cover the same fraction of distribution of the sgrs (load-balanced windows for efficient analytical workloads) and also enables comparing the analysis over different windows of a stream as well as analysis over different streams having different temporal distributions (time-based windows for the accuracy of temporal analysis).

Approximating the Number of Inter-Window Butterflies. This section explains how *sGrapp* approximates the $\hat{B}^{interW}$ and consequently $\hat{B_k}$. Algorithm 6 describes how *sGrapp* uses the adaptive windowing framework (Algorithm 5) to estimate the number of butterflies in the streaming graph. Since *sGrapp* uses tumbling windows, instead of checking the timestamp of windowed sgrs to decide on the retirement (Algorithm 5, lines 11-12), the processing graph is renewed (line 13 of Algorithm 6). As mentioned earlier, the total number of butterflies (Algorithm 6, line 10) is calculated as the summation of the following:

- Total number of butterflies computed at the end of previous window;

- The exact number of butterflies in the current window (Algorithm 6, line 9 - invoking Algorithm 1);
- The estimated number of inter-window butterflies contributed by current window.

```
Algorithm 6: sGrapp($N_b$, $\alpha$)
  Data: $\mathfrak{R} = \langle r^1, r^2, \cdots \rangle$, sequence of time-ordered sgrs
  Input: $N_b$, Number of unique timestamps (bursts) per window
  $\alpha$, Approximation exponent
  Output: $\hat{B}[t]$, Approximated number of butterflies at t
1 $G \leftarrow \langle V = \emptyset, E = \emptyset \rangle$, $t \leftarrow 0$, $unqt \leftarrow \emptyset$, $k \leftarrow 0$, $\hat{B} \leftarrow \emptyset$, $B_G^{W_k} \leftarrow 0$, $\hat{B}_k \leftarrow 0$, $E \leftarrow 0$
2 while true do
3     $r^t = (\tau^t, p) \leftarrow sgrIngest()$
4     if $r^t \neq \emptyset$ then
5         $unqt$.add($\tau^t$)
6         $G \leftarrow updateG(r^t, G)$
7         $E \leftarrow updateE(r^t, E)$
8     if $unqt.size() == N_b$ then
9         $B_G^{W_k} \leftarrow countButterflies(G)$
10        $\hat{B}_k \leftarrow \hat{B}_{k-1} + B_G^{W_k} + \delta(k \neq 0)E^\alpha$
11        $\hat{B}[t] \leftarrow \hat{B}_k$
12        $k \leftarrow k + 1$
13        $G \leftarrow \langle V = \emptyset, E = \emptyset \rangle$
14    $t \leftarrow t + 1$
```

According to the butterfly densification power law, the number of butterflies follows a power-law function of the number of existing edges in the graph. Moreover, recall the observation from analyses of real graphs that butterflies are formed by hubs. Thus, the number of inter-window butterflies is approximated as $\hat{B}^{interW} = |E(t = W_k^e)|^\alpha$, where $|E(t = W_k^e)|$ is the total number of edges since $t = W_0^b$ until $t = W_k^e$. The total number of added edges are updated at ingestion time (Algorithm 6, line 7) as E is increased when the sgr is an edge insertion and decreased when sgr is an edge deletion and α is the approximation exponent.

5.3.1 Optimised Algorithm: sGrapp-x

The approximation exponent used in *sGrapp* (Algorithm 6) is constant over sequential graph snapshots. However, the estimated number of butterflies using a static exponent can be over or under the true value in subsequent windows. To understand the underlying reason, recall the butterfly emergence patterns. Old hubs are the main contributors to butterflies; since the number of edges connecting to old hubs varies across different windows, the estimated number of inter-window butterflies should not increase linearly with respect to the number of sgrs. Therefore, *sGrapp* is optimised by changing the exponent over windows. To this end, *sGrapp* is modified to a semi-supervised algorithm that is called *sGrapp-x*.

The algorithm is provided with true value of butterflies for an initial subset of the stream (the percentage of the available ground truth is denoted by x). Based on the true value, the relative error $\frac{\hat{B_K}-B_K}{B_K}$ is computed in the corresponding window W_K (Algorithm 7, line 17). If the relative error is lower than a user-specified negative tolerance value (in the experiments -0.05 is used), that means there is an underestimation, therefore the exponent is increased by 0.005 (Algorithm 7, lines 12-13). Similarly the exponent is decreased in case the relative error is above positive tolerance value to avoid over-estimation in the next window (Algorithm 7, lines 10-11). The exponent is stabilized when the error is tolerable and after the supervised search for the exponent is finished. In summary, the optimised version of *sGrapp* is an adaptive algorithm using reinforcement learning that learns the most accurate approximation exponent for any given window parameter N_b in a subset of stream and generalizes the learned exponent to the rest of stream. *sGrapp-x* is semi-supervised with outstanding performance given limited ground truth.

5.4 Performance Evaluation

The effectiveness and efficiency of *sGrapp* and its optimised version *sGrapp-x* are tested from three perspectives:

- Accuracy (Sections 5.4.1, 5.4.2, and 5.4.4). The ability of *sGrapp* and *sGrapp-x* to approximate the butterfly count is compared with baselines and examined under different levels of burstiness, exponent values, and available ground truth.
- Throughput (Section 5.4.3). The processing latency as well as the number of processed elements by *sGrapp* and *sGrapp-x* are examined for each window individually and all windows cumulatively.

Algorithm 7: sGrapp-x(N_b^W, α, x)

```
Data: $\{r^t\}$, sequence of time-ordered sgrs
$B$, ground truths
Input: $N_b$, Number of unique timestamps (bursts) per window
$\alpha$, Approximation exponent
$x$, Percentage of the available ground truth
Output: $\hat{B}[t]$, Approximated number of butterflies at t
1  $G \leftarrow \langle V = \emptyset, E = \emptyset \rangle$, $t \leftarrow 0$, $unqt \leftarrow \emptyset$, $k \leftarrow 0$, $\hat{B} \leftarrow \emptyset$, $B_G^{W_k} \leftarrow 0$, $\hat{B}_k \leftarrow 0$, $E \leftarrow 0$,
    $error \leftarrow 0$
2  while true do
3      $r^t = (\tau^t, p) \leftarrow sgrIngest()$
4      if $r^t \neq \emptyset$ then
5          $unqt$.add($\tau^t$)
6          $G \leftarrow updateG(r^t, G)$
7          $E \leftarrow updateE(r^t, E)$
8      if $unqt.size() == N_b$ then
9          $B_G^{W_k} \leftarrow countButterflies(G)$
10         if $t < 0.01 * x * size(B)$ & $error > 0.05$ then
11             $\alpha- = 0.005$
12         if $t < 0.01 * x * size(B)$ & $error < -0.05$ then
13             $\alpha+ = 0.005$
14         $\hat{B}_k \leftarrow \hat{B}_{k-1} + B_G^{W_k} + \delta(k \neq 0)E^{\alpha}$
15         $\hat{B}[t] \leftarrow B_K$
16         if $t < 0.01 * x * size(B)$ then
17             $error \leftarrow \frac{\hat{B}_k - B_k}{B_k}$
18         $k \leftarrow k + 1$
19         $G \leftarrow \langle V = \emptyset, E = \emptyset \rangle$
20     $t \leftarrow t + 1$
```

- Complexity analysis (Section 5.4.5). The approximation properties of *sGrapp* are analyzed theoretically in terms of computational and error bounds.

Data. The real-world streams used in Phase 1 of the analysis in Chapter 3 are used in experiments (Table 3.2). The ground truths for *sGrapp-x* are obtained by running the exact counting Algorithm 1 over the streams. x is the percentage of the available ground truth. $x = 25$, 50, 75, and 100 are used throughout the experiments. Due to the computational expense of Algorithm 1, the truth values are collected over a limited number of sgrs: 72344 in Epinions, 12259 in ML100k, 21696 in ML1m, 21778 in ML10m, 75000 in Edit-EnWiki, and 75000 in Edit-FrWiki.

Metrics. The effectiveness of *sGrapp* and *sGrapp-x* is evaluated using Mean Absolute Percentage Error (MAPE), which is computed for windows with variable number of unique timestamps and different exponent values. The number of unique timestamps per window, N_b, varies in different streams and the value of N_b is set differently for each stream. The values of α and N_b are cross validated to explore the region including the best accuracy (lowest MAPE illustrated by the lightest color) for *sGrapp*. $MAPE = \frac{1}{n}\Sigma\frac{|B_k - \hat{B_k}|}{B_k}$, where B_k is the ground truth computed over the growing graph at $t = W_k^e$ by Algorithm 1 and $\hat{B_k}$ is the approximated value at $t = W_k^e$, and n is the number of windows. The data tips in the figures demonstrate the pair of α and N_b yielding the lowest MAPE.

The efficiency of *sGrapp* and *sGrapp*-100 is evaluated by averaging over 50 independent cases. The efficiency metrics are not reported for $x < 100$ since their efficiency is close to that of *sGrapp*-100. For each stream, the performance is studied for the parameter settings that yield the best accuracy (highlighted data points in Figures 5.2 and 5.7) to see the overhead of a highly accurate approximation. Note that parameter values do not affect the efficiency. The latency of *sGrapp* and *sGrapp*-100 is checked for each processing window (Figures 5.16 and 5.17). The window latency of all the streams (except the Epinions) is not decreasing. The window latency of each stream follows its temporal distribution pattern. Therefore, to omit the effect of temporal distribution, the performance is studied by considering both the processing time (latency) and the number of processed elements. To this end, at the end point of each window, the window throughput (i.e. the number of processed edges in the window divided by the elapsed time in seconds, Figures 5.20 and 5.21) is checked as well as the total throughput (i.e. the total number of processed edges since the first window until the end of the current window divided by the total elapsed time in seconds, Figures 5.18 and 5.19).

Computing setup. Experiments are conducted on a machine with 15.6 GB native memory and Intel Core $i7-6770HQCPU@2.60GHz*8$ processor. FLEET algorithms and *sGrapp* algorithms are implemented in Java (OpenJDK version 1.8.0 – 252).

Table 5.2: The approximation MAPE for different window lengths and exponents. Exponents are calculated as the probability of one or two i-hub plus the probability of one or two j-hubs at different time points in Epinions stream.

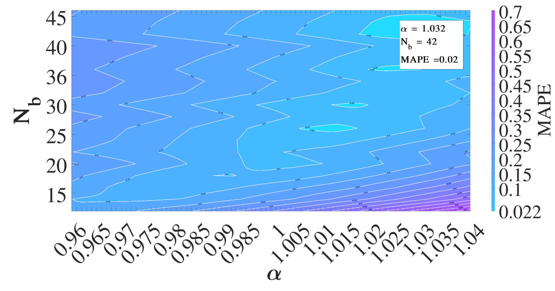
MAPE	$0.006 * N_b$	$0.007 * N_b$	$0.008 * N_b$	$0.009 * N_b$	$0.01 * N_b$
$\alpha = Pr(t = 1k) = 1.2178$	3.0036	2.5461	2.5005	2.2996	2.2602
$\alpha = Pr(t = 2k) = 1.077$	0.4472	0.4591	0.3318	0.2359	0.2632
$\alpha = Pr(t = 3k) = 1.1274$	1.0295	0.8281	0.8212	0.6954	0.7079
$\alpha = Pr(t = 4k) = 1.0806$	0.4778	0.3551	0.3574	0.2597	0.2864
$\alpha = Pr(t = 5k) = 1.0389$	0.14286	0.1016	0.0778	0.0864	0.0456
$\alpha = Pr(t = 6k) = 1.0296$	**0.0953**	**0.0723**	**0.524**	**0.0709**	**0.0315**
$\alpha = Pr(t = 7k) = 1.0438$	0.1760	0.1176	0.1054	0.1014	0.0597
$\alpha = Pr(t = 8k) = 1.0591$	0.2897	0.1950	0.2000	0.1525	0.1446
$\alpha = Pr(t = 9k) = 1.0546$	0.2553	0.1658	0.1713	0.1370	0.1188
$\alpha = Pr(t = 10k) = 1.0420$	0.1639	0.1189	0.0953	0.0959	0.0508

5.4.1 Effectiveness of sGrapp

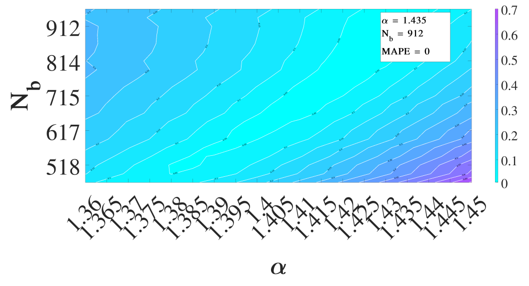
The approximation accuracy of *sGrapp* is not sensitive to window length and the exponent, since there exists a combination of approximation exponent and window length for each stream that yields appropriate MAPE (Figure 5.2). In fact, the best MAPE of *sGrapp* is significantly lower than 0.1 in all of the rating streams, demonstrating that *sGrapp* outputs a good approximate of actual butterfly count.

When the approximation exponent is high and the window is compact (bottom right corners in Figure 5.2), the error is high. In this case, *sGrapp* overestimates the number of inter-window butterflies due to high exponent value. Also, when the exponent is low and the window includes a large number of sgrs (top left corner in Figure 5.2), the error is high. The reason in this case is that *sGrapp* underestimates the number of inter-window butterflies. An appropriate parameter region to gain a reasonable accuracy is where α and N_b are both high or low (middle diameter from top right corner to bottom left corner in Figure 5.2). The best accuracy is always obtained for higher exponent values. For rating networks, an appropriate exponent value for *sGrapp* is $\alpha = 1.4$.

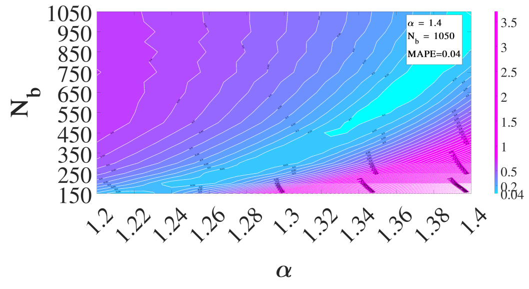
According to the contribution of hubs to the emergence of butterflies (Section 3.2), the value of approximation exponent is related to the probability of having at least one i-hub ($Pr(N^t_{iHub} \geq 1)$) plus the probability of having at least one j-hub ($Pr(N^t_{jHub} \geq 1)$) in the butterflies at time t, i.e. $\alpha = Pr(t) = Pr(N^t_{iHub} = 1) + Pr(N^t_{iHub} = 2) + Pr(N^t_{jHub} = 1) + Pr(N^t_{jHub} = 1)$ (Table 3.7). That is, the value of α can be determined based on



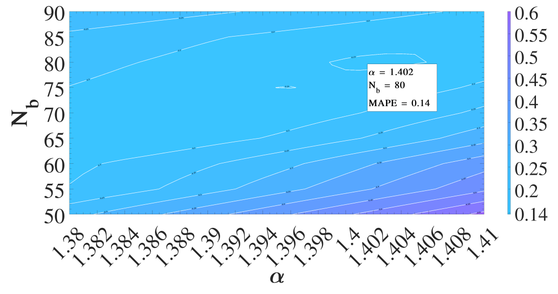
(a) Epinions



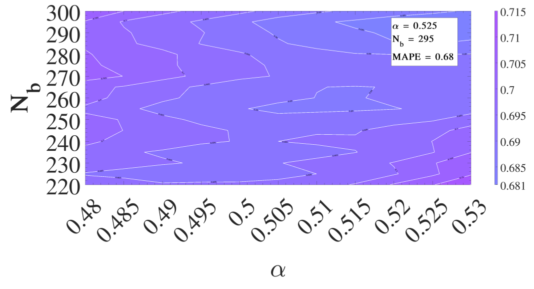
(b) MovieLens 100k



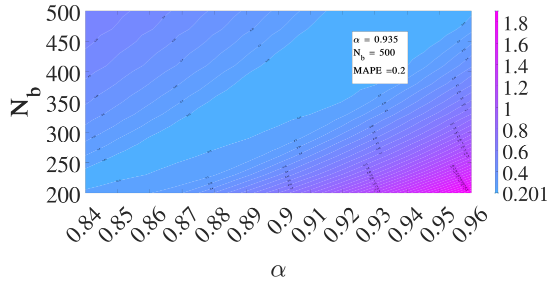
(c) MovieLens 1m



(d) MovieLens 10m

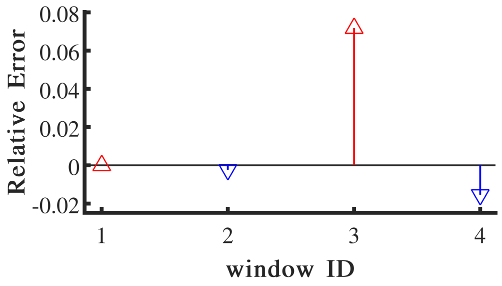


(e) Edit-EnWiki

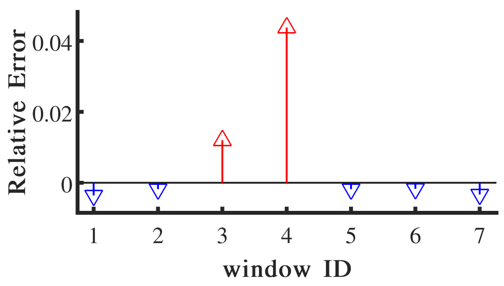


(f) Edit-FrWiki

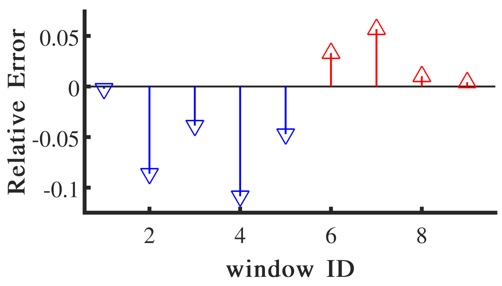
Figure 5.2: Accuracy of *sGrapp* for different values of α and N_b



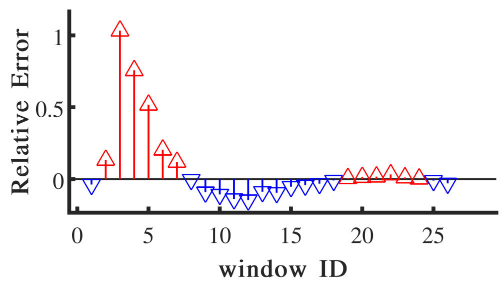
(a) Epinions, $\alpha = 1.03$, $N_b = 42$, $MAPE = 0.02$



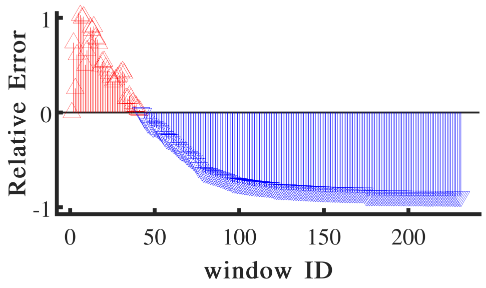
(b) ML100k, $\alpha = 1.4$, $N_b = 912$, $MAPE = 0$



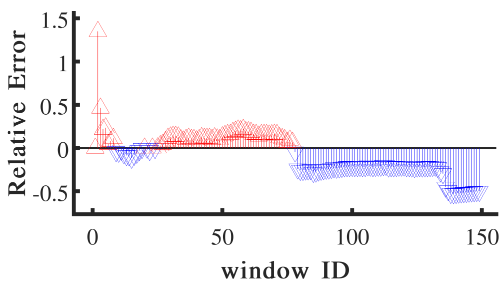
(c) ML1m, α=1.4, N_b=1050, $MAPE$=0.04



(d) ML10m, α=1.4, N_b = 80, $MAPE$=0.1



(e) Edit-EnWiki, α=0.5, N_b=295, $MAPE$=0.7



(f) Edit-FrWiki, α=0.9, N_b=500, $MAPE$=0.2

Figure 5.3: Relative Error of *sGrapp* over windows for the best obtained MAPE.

the probability of i- or j-hubs forming butterflies at a certain time point t. The time point t is likely a *tipping point* where the number of hub connections in the graph is stabilized (Figures 3.12 and 3.13). To check this, the value of $Pr(t)$ is calculated for $t \in \{1000, 2000, .., 9000, 10000\}$ in the Epinions graph stream. The value of MAPE is computed for sGrapp(N_b, α). $\alpha = Pr(t)$ and $N_b^W \in \{0.006N_b, 0.007N_b, 0.008N_b, 0.009N_b, 0.01N_b\}$. Table 5.2 reports the value of MAPE for the approximations with different exponent values and different fraction of unique timestamp per adaptive window. At $t = 6000$, where the exponent is equal to $\alpha = Pr(t = 6000) \approx 1.03$, the approximation error is the lowest. This time point is a *tipping point* where the fraction of average hub degree is steadily low afterward and high backward (Figures 3.12 and 3.13). Moreover, in Figure 5.2, the best accuracy is obtained when the exponent is equal to $Pr(t = 6000) = 1.03$.

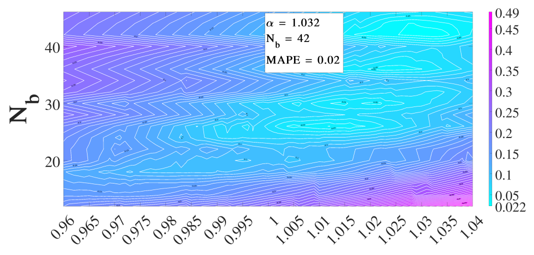
Next experiments investigate how *sGrapp*'s performance evolves over windows to track the origins of the accuracy gain. The most accurate α and N_b (highlighted data points in Figure 5.2) are picked and the signed value of relative error $\frac{|B_k - \hat{B_k}|}{B_k}$ for each window W_k is depicted in Figure 5.3. Depending on the value of N_b, the number of windows vary in different streams. Positive errors (depicted by red upward triangles) reflect over-estimations and negative errors (depicted by blue downward triangles) reflect under-estimations. In ML10m, Edit-EnWiki and Edit-FrWiki, the approximation begins with over-estimation and ends up with under-estimation. The underlying reason is the static exponent over sequential windows with different number of connections to the old hubs and consequently different number of inter-window butterflies.

5.4.2 Effectiveness of sGrapp-x

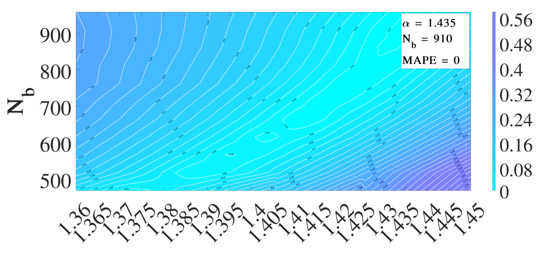
The accuracy of *sGrapp-x* is evaluated in terms of MAPE in the region that *sGrapp* displays the lowest errors in Figures 5.4 – 5.7. While this is not the optimal parameter region for *sGrapp-x*, it enables a fair comparison of *sGrapp* with its optimised version *sGrapp-x* to evaluate the effectiveness of the introduced optimisations. Note that, *sGrapp-x* begins with a given exponent value and ends up with a modified value after the supervision phase reaches an error below 0.05. Therefore *sGrapp-x* is fed with same input values of α and N_b as *sGrapp*. The values shown in Figures 5.4 – 5.7 reflect the inputs.

It is evident from these figures that *sGrapp-x* further improves the accuracy. To further explore the impact of the introduced optimisations on accuracy, Figures 5.4 – 5.7 are compared with Figure 5.2 according to three factors:

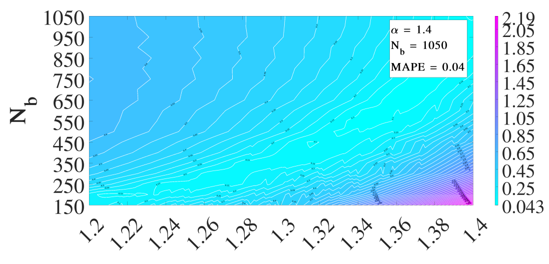
- Improving the minimum MAPE (Figure 5.8);



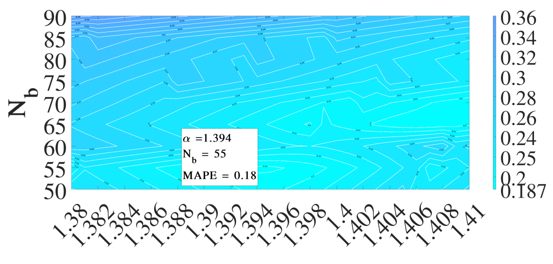
(a) Epinions



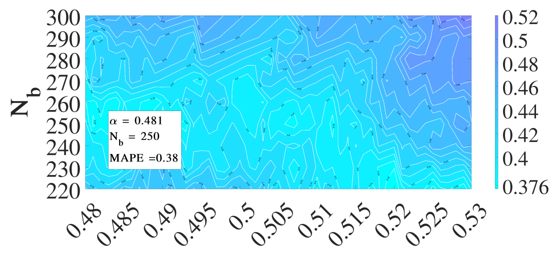
(b) ML100k



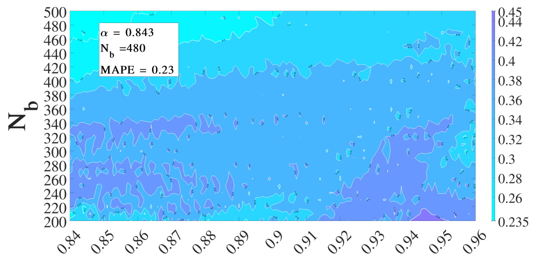
(c) ML1m



(d) ML10m

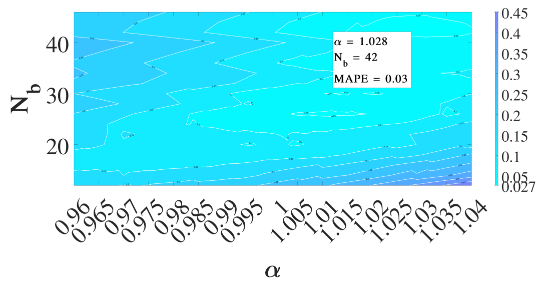


(e) Edit-EnWiki

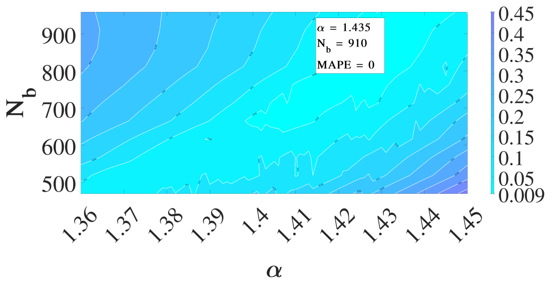


(f) Edit-FrWiki

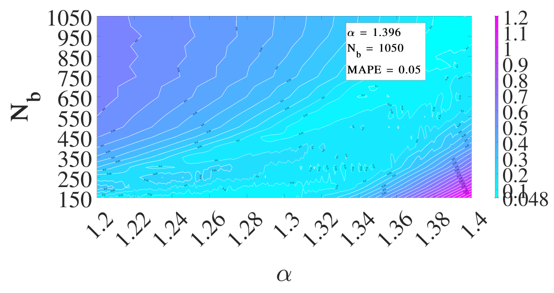
Figure 5.4: Accuracy of *sGrapp*-25 for different values of α and N_b.



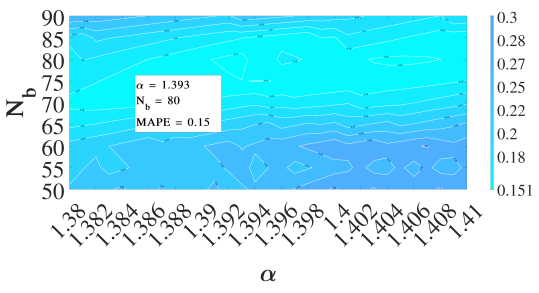
(a) Epinions



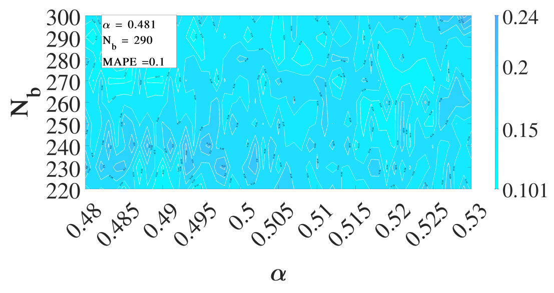
(b) ML100k



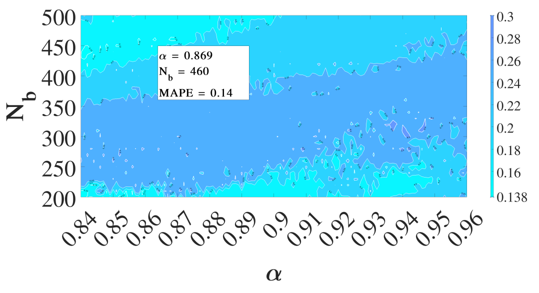
(c) ML1m



(d) ML10m

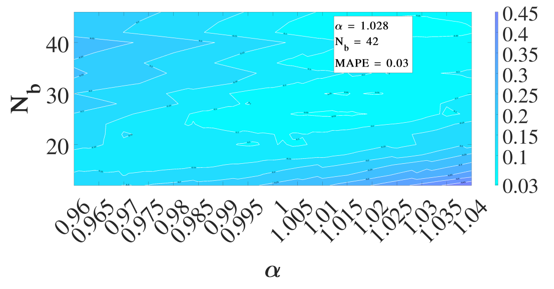


(e) Edit-EnWiki

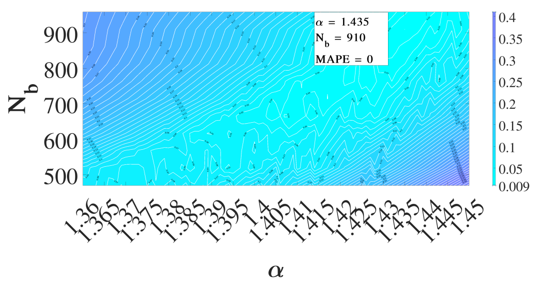


(f) Edit-FrWiki

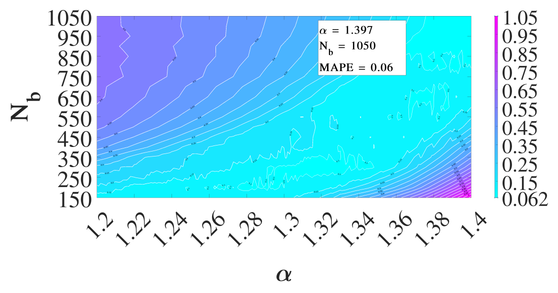
Figure 5.5: Accuracy of *sGrapp*-50 for different values of α and N_b



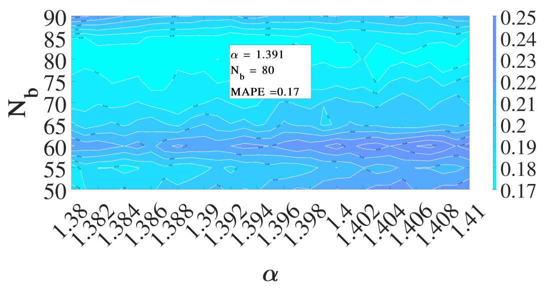
(a) Epinions



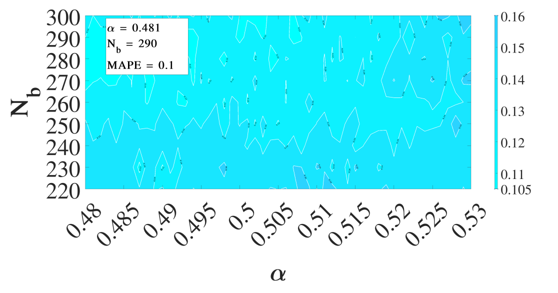
(b) ML100k



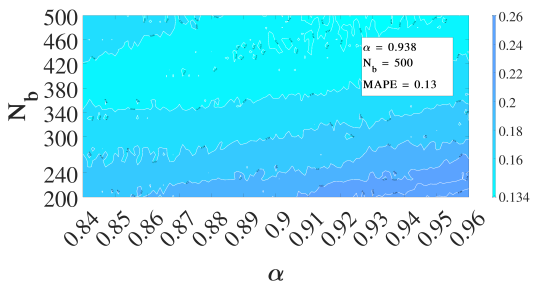
(c) ML1m



(d) ML10m

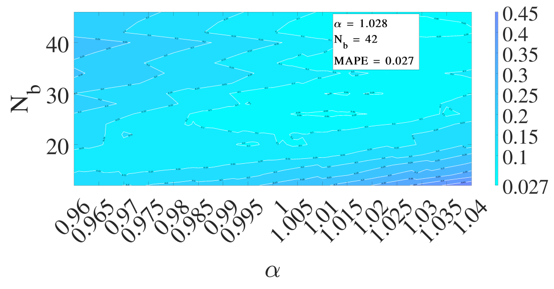


(e) Edit-EnWiki

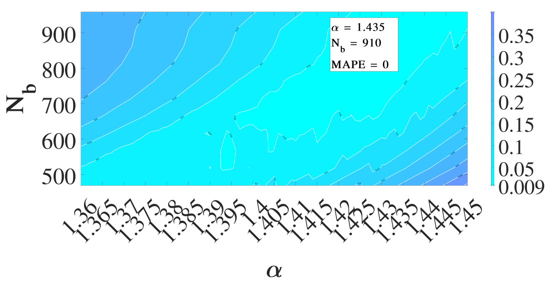


(f) Edit-FrWiki

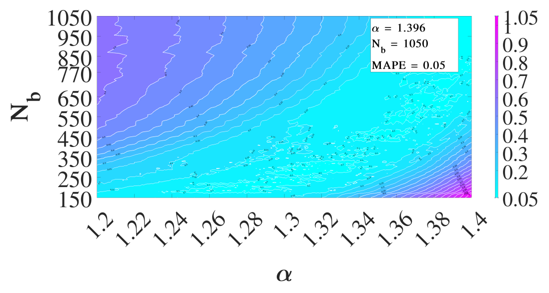
Figure 5.6: Accuracy of *sGrapp*-75 for different values of α and N_b.



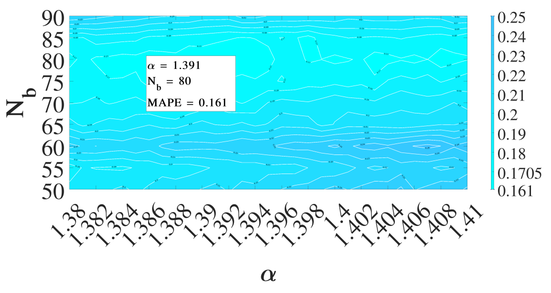
(a) Epinions



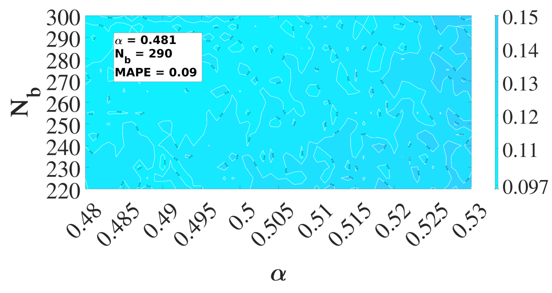
(b) ML100k



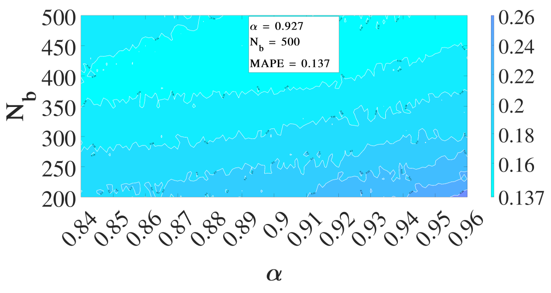
(c) ML1m



(d) ML10m



(e) Edit-EnWiki



(f) Edit-FrWiki

Figure 5.7: Accuracy of *sGrapp*-100 for different values of α and N_b.

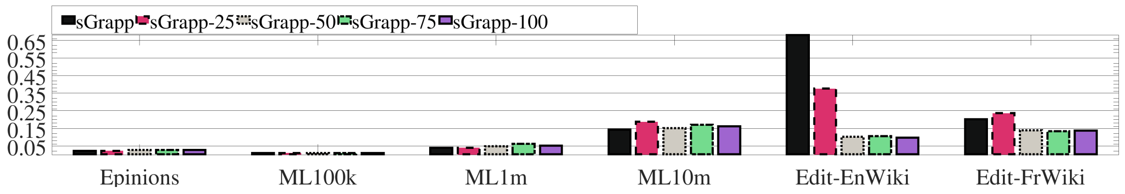


Figure 5.8: Minimum approximation MAPE.

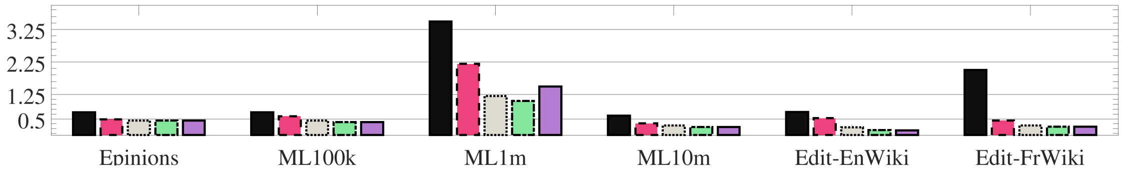


Figure 5.9: Maximum approximation MAPE.

- Improving the maximum MAPE (Figure 5.9);
- Expanding the coverage of MAPE$\leq$ 0.15 and MAPE$\leq$ 0.2 (Figures 5.10 and 5.11).

The minimum MAPE value in the studied parameter space is roughly the same for both *sGrapp* and *sGrapp-x* (x = 25–100) in all rating graph streams (Figure 5.8). *sGrapp-x* lowers the minimum MAPE with respect to *sGrapp* in Edit-EnWiki graph from 0.681 to 0.376 (via $x = 25$), 0.105 (via $x = 75$), 0.101 (via $x = 50$), and 0.097 (via $x = 100$); in Edit-FrWiki graph from 0.201 to 0.235 (via $x = 25$), 0.137 (via $x = 100$), 0.134 (via $x = 75$), and 0.130 (via $x = 50$). That is, the minimum MAPE is lowered ranging from 44.79% to 85.76% in Edit-EnWiki and 31.84% to 35.32% in Edit-FrWiki. The maximum MAPE related to the over-estimations (bottom right corners in Figures 5.4 – 5.7) is notably decreased in all graph streams (Figure 5.9). The most significant decrease corresponds to Edit-FrWiki stream with the highest change from 2 to 0.26 (via $x = 75, 100$) and Edit-EnWiki stream with highest change from 0.715 to 0.15 (via $x = 100$).

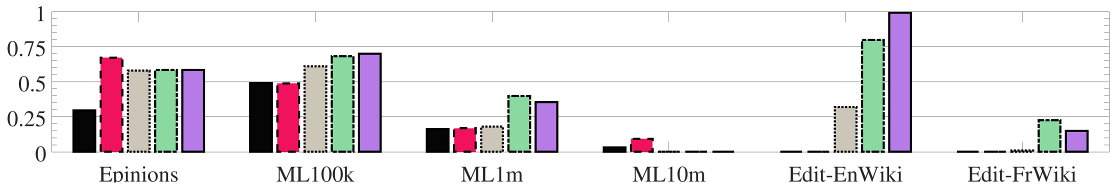


Figure 5.10: Probability of approximation with MAPE less than equal 0.15.

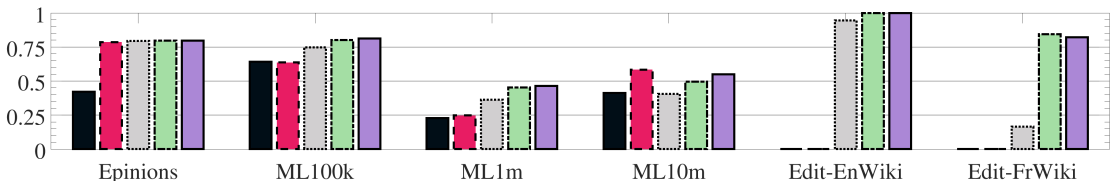
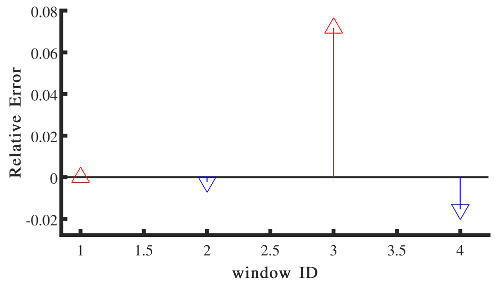
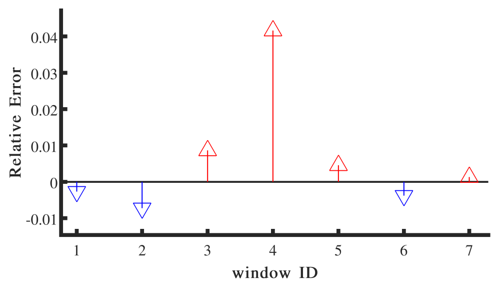


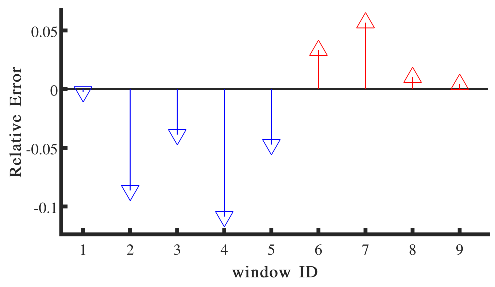
Figure 5.11: Probability of approximation with MAPE less than equal 0.2.



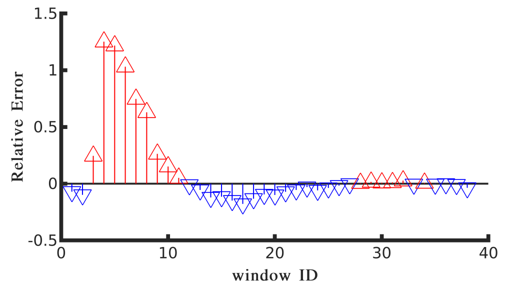
(a) Epinions, α=1.032, N_b=42, MAPE=0.022



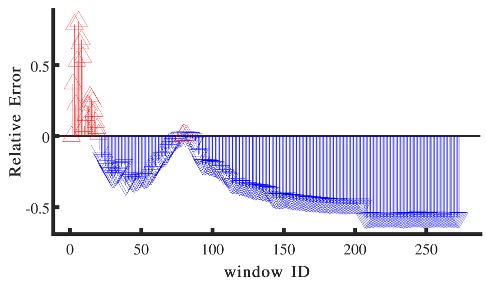
(b) ML100k, α=1.435, N_b=910, MAPE=0.009



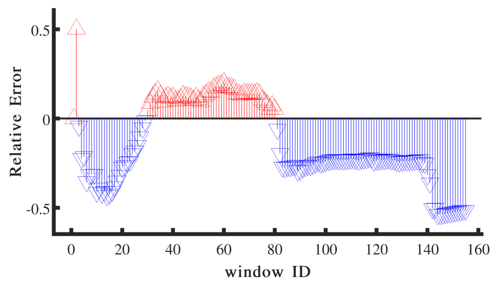
(c) ML1m, α=1.4, N_b=1050, MAPE=0.043



(d) ML10m, α=1.394, N_b=55, MAPE=0.187

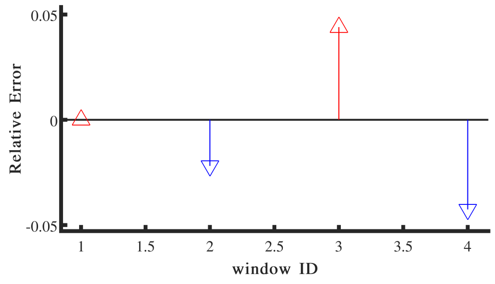


(e) EnWiki, α=0.481, N_b=250, MAPE=0.376

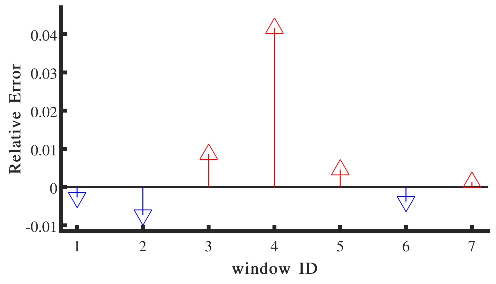


(f) FrWiki, α=0.843, N_b=480, MAPE=0.235

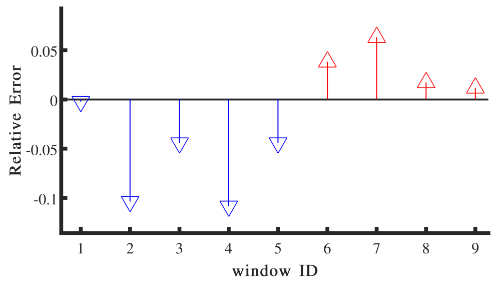
Figure 5.12: Relative Error of *sGrapp*-25 over windows for the best obtained MAPE.



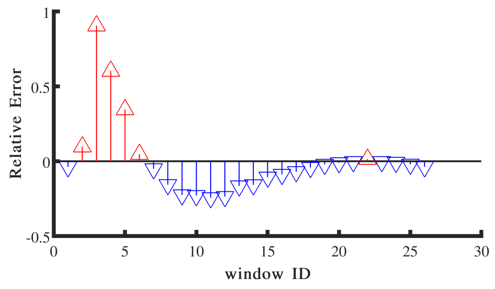
(a) Epinions, α=1.028, N_b=42, MAPE=0.027



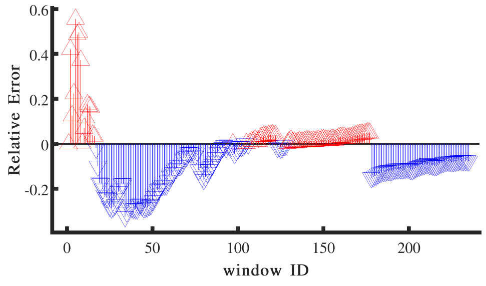
(b) ML100k, α=1.435, N_b=910, MAPE=0.009



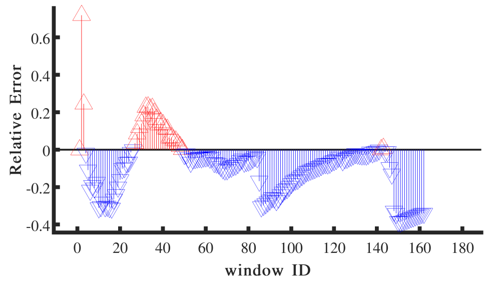
(c) ML1m, α=1.396, N_b=1050, MAPE=0.048



(d) ML10m, α=1.393, N_b=80, MAPE=0.151

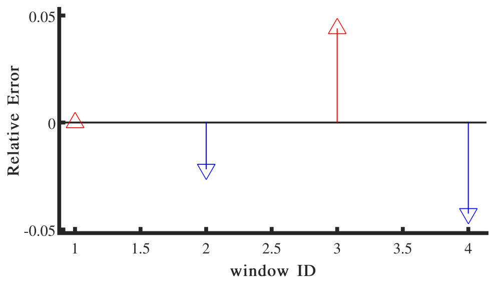


(e) EnWiki, α=0.481, N_b=290, MAPE=0.101

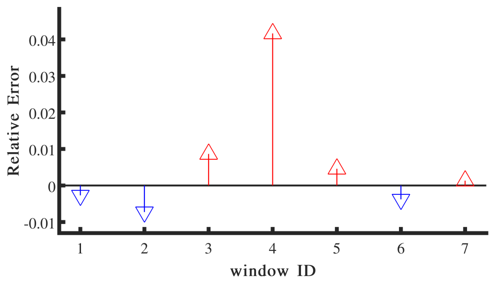


(f) FrWiki, α=0.869, N_b=460, MAPE=0.138

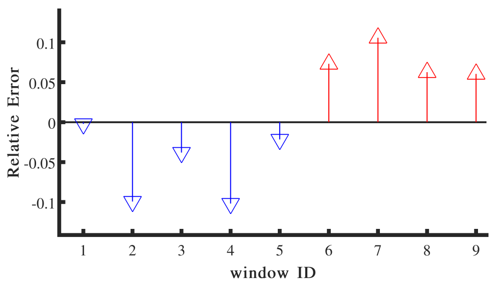
Figure 5.13: Relative Error of *sGrapp*-50 over windows for the best obtained MAPE.



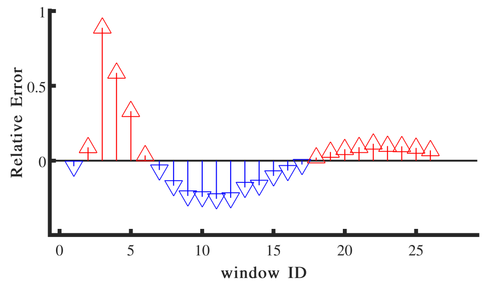
(a) Epinions, α=1.028, N_b=42, MAPE=0.027



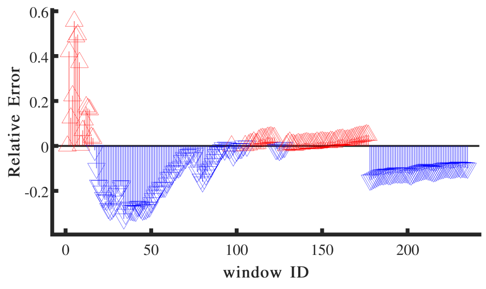
(b) ML100k, α=1.435, N_b=910, MAPE=0.009



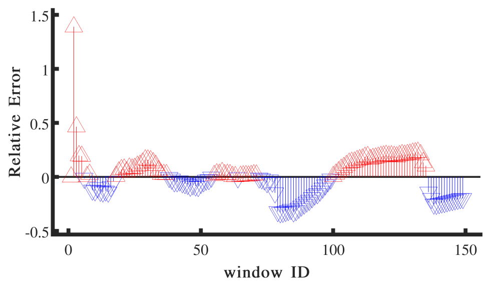
(c) ML1m, α=1.397, N_b=1050, MAPE=0.062



(d) ML10m, α=1.391, N_b=80, MAPE=0.169



(e) EnWiki, α=0.481, N_b=290, MAPE=0.105



(f) FrWiki, α=0.938, N_b=500, MAPE=0.134

Figure 5.14: Relative Error of *sGrapp*-75 over windows for the best obtained MAPE.

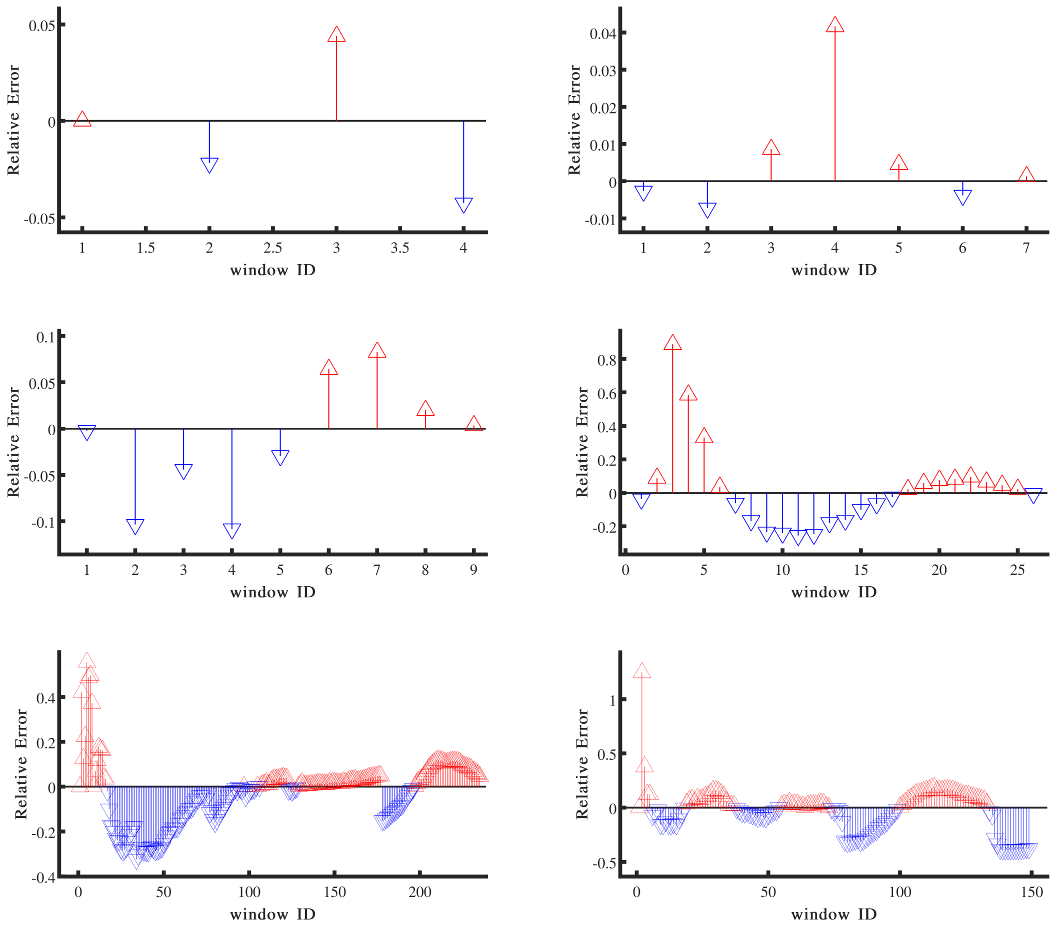


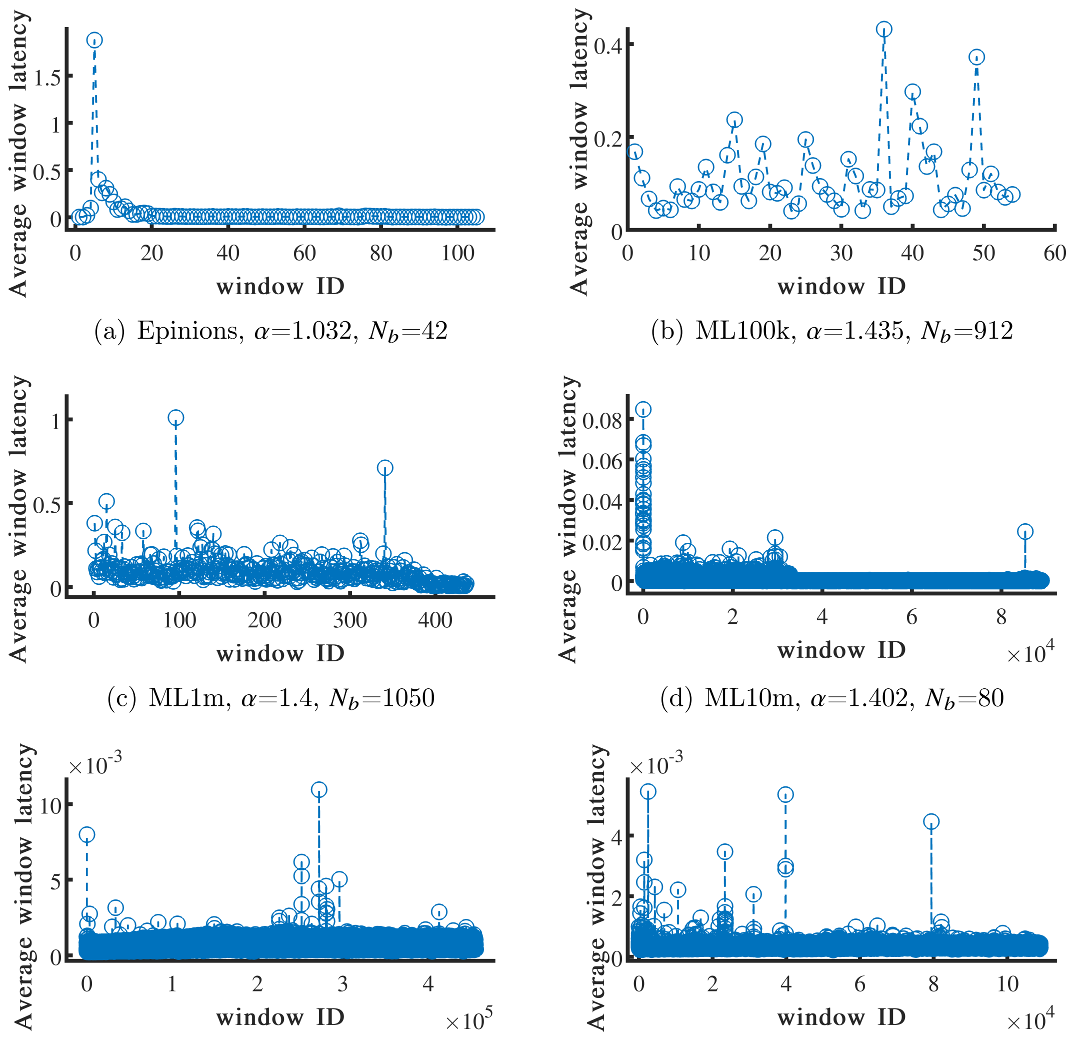
Figure 5.15: Relative Error of *sGrapp*-100 over windows for the best obtained MAPE.

Figures 5.10 and 5.11 present the probability of approximation with MAPE≤ 0.15 and MAPE≤ 0.2 ($\boldsymbol{Pr}$(MAPE $\leq 0.15(0.2)$)) by calculating the fraction of approximations that satisfy MAPE≤ 0.15 and MAPE≤ 0.2. That is the relative coverage of light blue areas in Figures 5.2 – 5.7. When the approximation MAPE is above 0.15 or 0.2, the corresponding bars are omitted in Figures 5.10 and 5.11. Since *sGrapp*-100 approximates the number of butterflies in Edit-EnWiki with highest MAPE equal to 0.15, the corresponding bar has a height of 1. *sGrapp*-25 improves the accuracy of *sGrapp* in MovieLens10m better than other *sGrapp-x* versions. For the other graph streams, when $x \geq 50$, *sGrapp-x* displays fairly well accuracy improvement as the probability of accurate approximation (i.e. average window error below 0.15 and 0.2) is amplified. As expected *sGrapp*-100 has the most improvement, however *sGrapp*-75 and *sGrapp*-50 are reliable improvement alternatives for Edit-FrWiki and the rest of graph streams, respectively. *sGrapp*-x, $x = 25, 50, 75$, and 100 can achieve the $\boldsymbol{Pr}$(MAPE $\leq 0.15(0.2)$) equal to 67.13% (78.53%), 60.94% (94.55%), 79.74% (84.27%), and 99.31% (100%). Most notably, *sGrapp*-50(75) increases $\boldsymbol{Pr}$(MAPE ≤ 0.2) from 0 to 94.55(100)% in Edit-EnWiki.

Next, the evolution of the signed value of relative error are checked over windows for the data points with the lowest *sGrapp-x* MAPE. As shown in Figures 5.12, 5.13, 5.14, and 5.15, dynamically changing the approximation exponent heals the under/over-estimation problem; Hence the average window error is diminished. *sGrapp-x* lowers the average approximation error of *sGrapp* to a value less than equal 0.05 in rating graphs and 0.14 in Wikipedia graphs for any given window length and approximation exponent.

5.4.3 Efficiency of sGrapp and sGrapp-x

The window throughput displays fluctuations due to variable number of sgrs in each window; however overall it is higher in later windows for both *sGrapp* and *sGrapp-100*. The total throughput of both *sGrapp* and *sGrapp-100* increases over time. As mentioned in previous section, the old hubs are the main contributors to the butterfly formation. Since old hubs occur in the early windows, the later windows mostly include butterfly vertices with lower degree. That is, there are fewer windowed butterflies in later windows than the inter-window butterflies. Therefore, the exact counting algorithm that computes the number of windowed butterflies finishes quicker. Also, rapid approximation of the inter-window butterflies plays the main role in reducing the processing time, enhancing the total throughput. An evidence is the throughput for MovieLens100k that has almost uniform temporal distribution: There is an increasing total throughput over windows. This is important since the number of sgrs in the windows is not decreasing while the throughput is



(a) Epinions, α=1.032, N_b=42

(b) ML100k, α=1.435, N_b=912

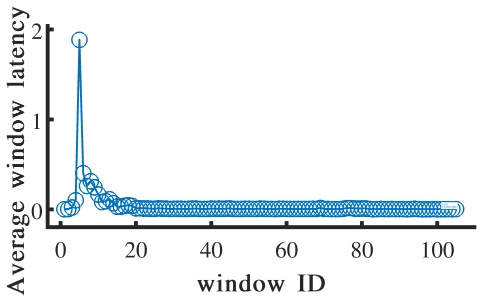
(c) ML1m, α=1.4, N_b=1050

(d) ML10m, α=1.402, N_b=80

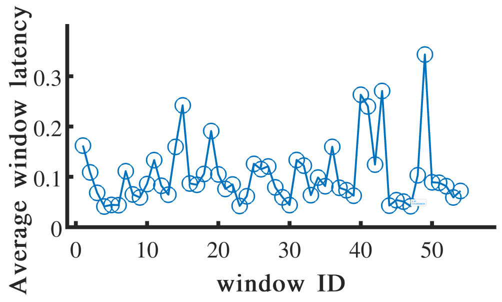
(e) EnWiki, α=0.525, N_b=295

(f) FrWiki, α=0.935, N_b=500

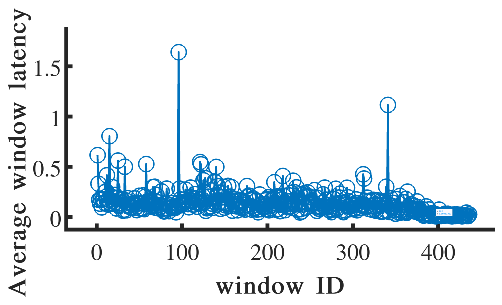
Figure 5.16: Average window latency (s) of *sGrapp*.



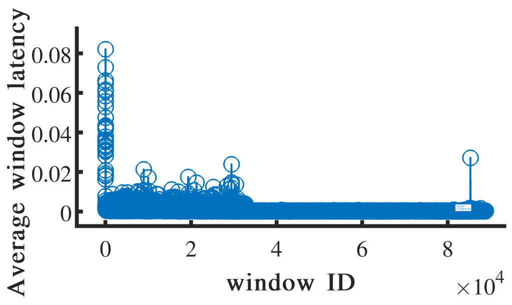
(a) Epinions, α=1.028, N_b=42



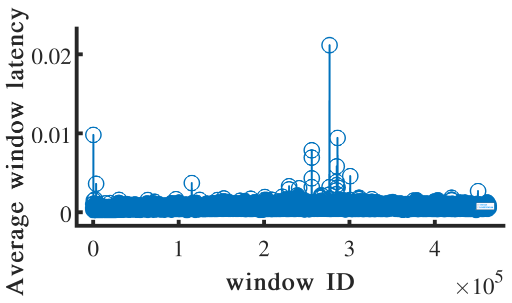
(b) ML100k, α=1.435, N_b=910



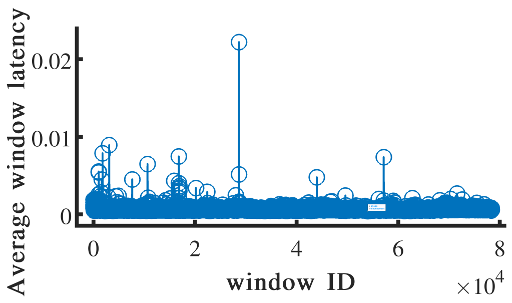
(c) ML1m, α=1.396, N_b=1050



(d) ML10m, α=1.391, N_b=80

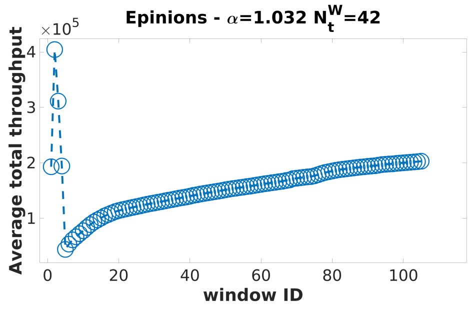


(e) EnWiki, α=0.481, N_b=290

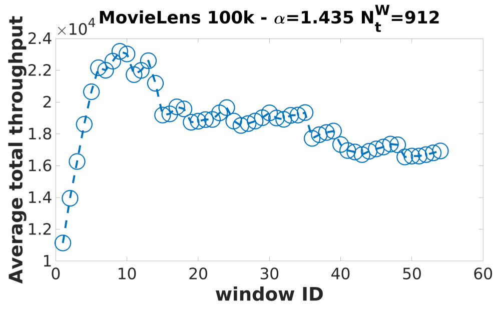


(f) FrWiki, α=0.927, N_b=500

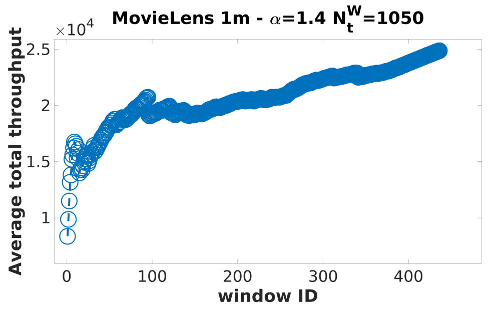
Figure 5.17: Average window latency (s) of *sGrapp*-100.



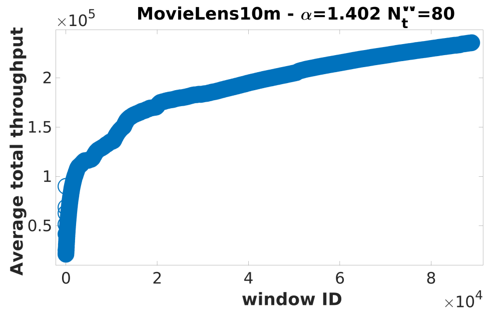
(a) Epinions, α=1.032, N_b=42



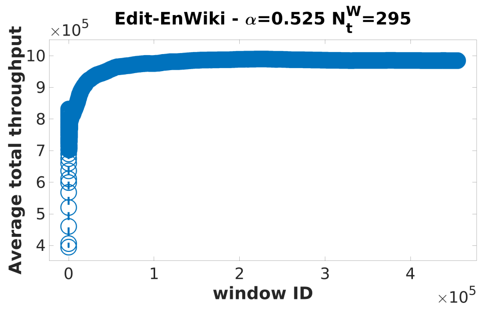
(b) ML100k, α=1.435, N_b=912



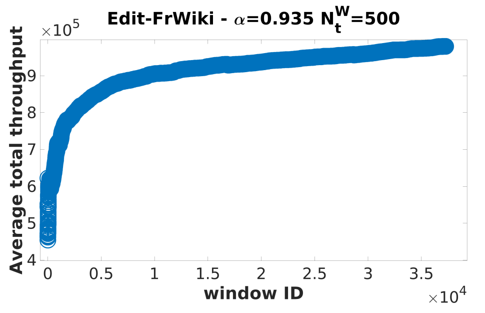
(c) ML1m, α=1.4, N_b=1050



(d) ML10m, α=1.402, N_b=80

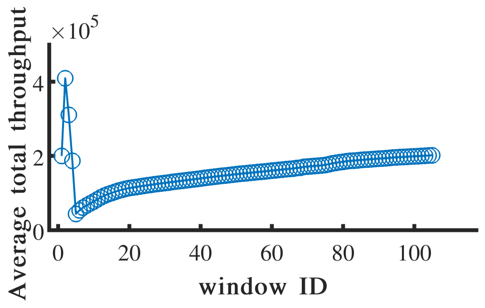


(e) EnWiki, α=0.525, N_b=295

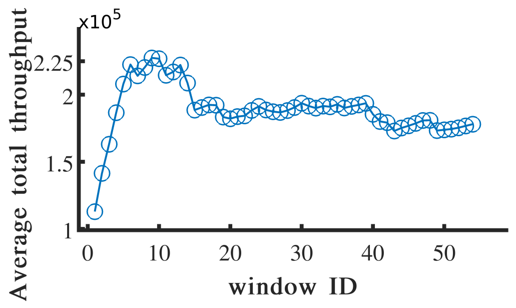


(f) FrWiki, α=0.935, N_b=500

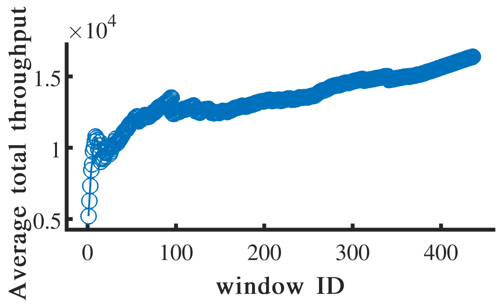
Figure 5.18: Average total throughput (edge/s) of *sGrapp* at the end of each window.



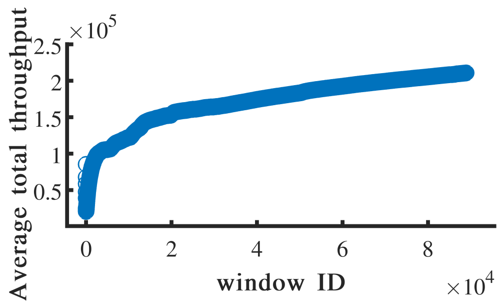
(a) Epinions, α=1.028, N_b=42



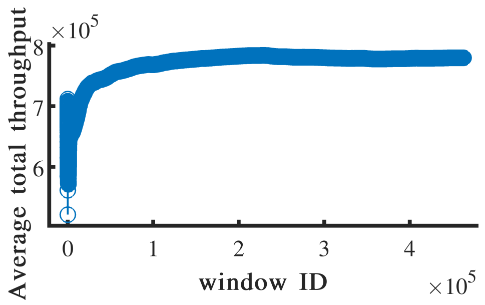
(b) ML100k, α=1.435, N_b=910



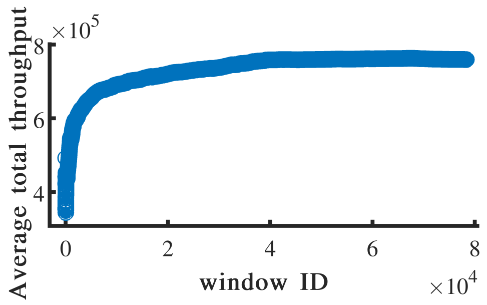
(c) ML1m, α=1.396, N_b=1050



(d) ML10m, α=1.391, N_b=80

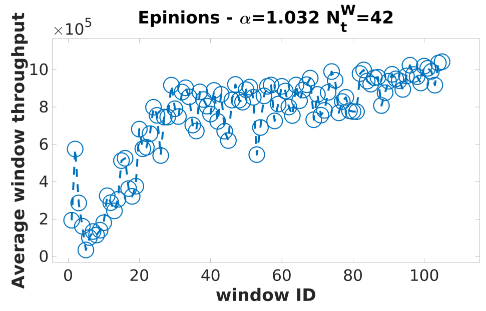


(e) EnWiki, α=0.481, N_b=290

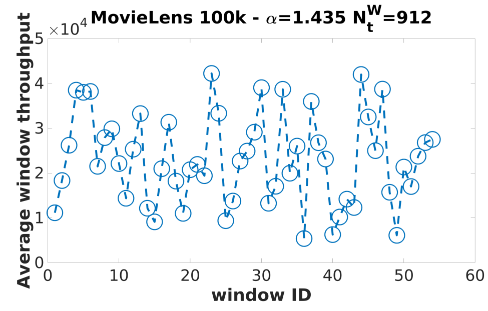


(f) FrWiki, α=0.927, N_b=500

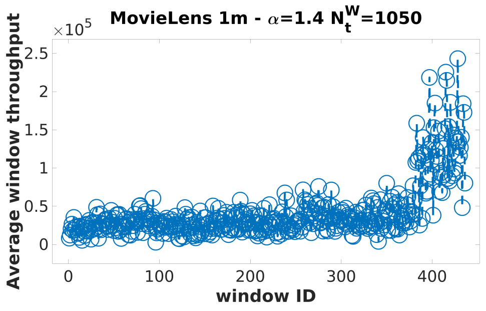
Figure 5.19: Average total throughput (edge/s) of *sGrapp*-100 at the end of each window.



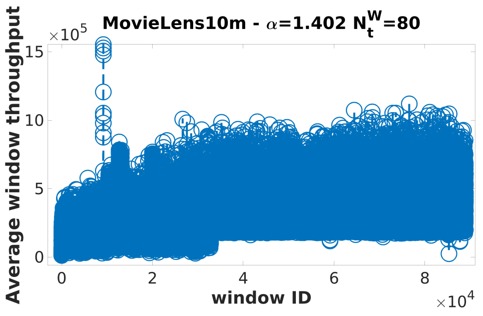
(a) Epinions, α=1.032, N_b=42



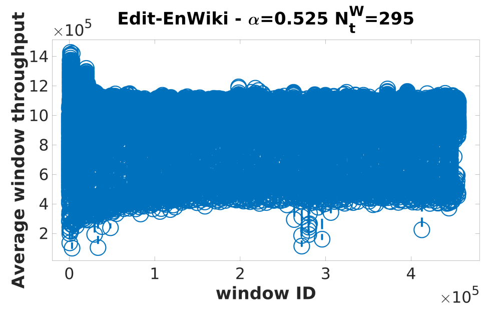
(b) ML100k, α=1.435, N_b=912



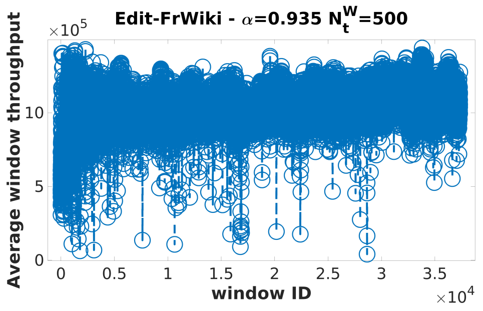
(c) ML1m, α=1.4, N_b=1050



(d) ML10m, α=1.402, N_b=80

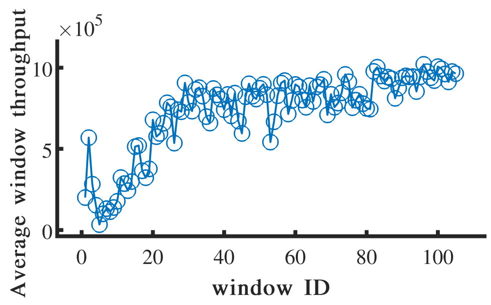


(e) EnWiki, α=0.525, N_b=295

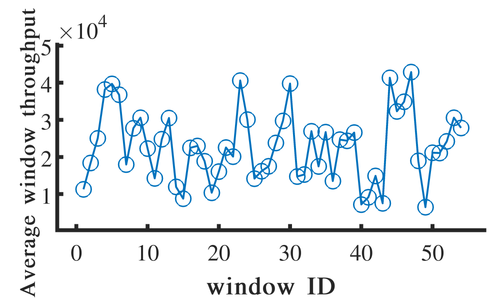


(f) FrWiki, α=0.935, N_b=500

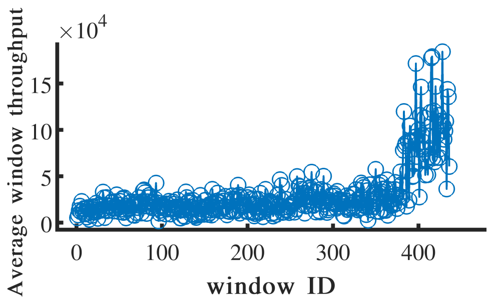
Figure 5.20: Average window throughput (edge/s) of *sGrapp* at the end of each window.



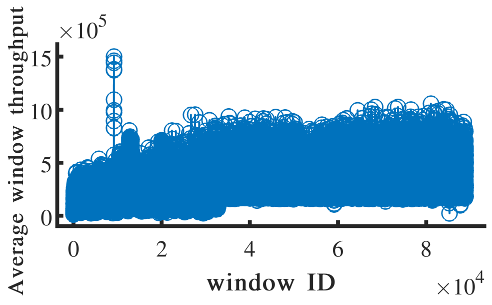
(a) Epinions, α=1.028, N_b=42



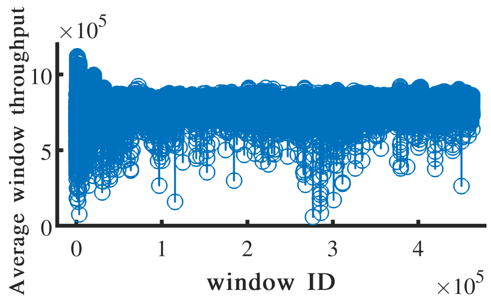
(b) ML100k, α=1.435, N_b=910



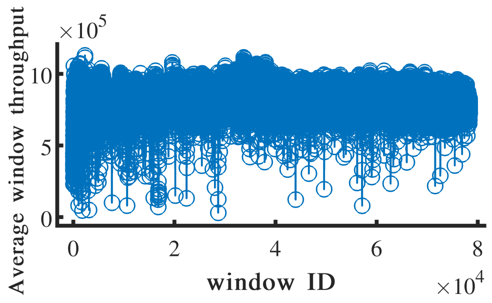
(c) ML1m, α=1.396, N_b=1050



(d) ML10m, α=1.391, N_b=80



(e) EnWiki, α=0.481, N_b=290



(f) FrWiki, α=0.927, N_b=500

Figure 5.21: Average window throughput (edge/s) of *sGrapp*-100 at the end of each window.

increasing. This confirms (a) the algorithm's power is independent of the structural/temporal characteristics of the input data and (b) the algorithm is efficient particularly in dense streams.

5.4.4 Comparison with Baselines

The effectiveness and efficiency of *sGrapp* suit are compared against those of FLEET suit. Experimental results of FLEET suit show that FLEET3, FLEET2 and FLEET1 have the best performance (in that order), therefore these are used as baselines. These three FLEET variants are based on landmark windows, while the other weaker FLEET variants FLEETSSW and FLEETTSW use sliding windows. While *sGrapp* has the α (approximation exponent) and N_b (number of unique timestamps per window) parameters, FLEET has the M (reservoir size) and γ (sub-sampling probability) parameters. Since the performance of FLEET algorithms is sensitive to its parameters, *sGrapp* algorithms are compared against the FLEET settings which achieve the best performance. The sub-sampling probability is set as $\gamma = 0.7$ as suggested by FLEET's authors [286].

When the reservoir size M is greater than the entire stream, latency is negatively impacted since sub-sampling does not occur and all the edges are added to the reservoir and for each new edge the exact butterfly counting is executed. Hence, for evaluating the accuracy over the prefix of a stream, reservoir size is set as $M = 0.01S$, where S is the size of available stream. For evaluating the efficiency, a range of values $M \in \{75k, 150k, 300k, 600k\}$ are also used to examine the throughput over the entire stream; these values are the ones offered in the original paper [286]. The approximation exponent values that yield the lowest MAPE in *sGrapp* are used; these do not necessarily yield the best MAPE in the optimised variant *sGrapp-x*. Since FLEET algorithms use different window semantics than *sGrapp*, virtual burst-based adaptive windows over FLEET algorithms are used to extract the estimated values at the end of virtual windows for accuracy evaluations only (not for efficiency tests). The same value of N_b is used for *sGrapp* and FLEET suits to compute MAPE: $N_b \in [42, 912, 1050, 80, 290, 500]$ for Epinions, ML100k, Ml1m, Ml10m, Edit-EnWiki, and Edit-FrWiki, respectively. For efficiency comparisons, the same value used in effectiveness experiments is used since the goal is to check the efficiency cost of the most accurate approximation. For each N_b, there exists an alpha yielding a high precision estimate. N_b does not affect accuracy.

Table 5.3 reports the total throughput over the entire streams for *sGrapp* and FLEET suits. Since FLEET1's throughput is very low, it is not included in this experiment. By

Table 5.3: Throughput of different algorithms for γ=0.7.

Throughput	FLEET2 M=75k	FLEET3 M=75k	FLEET2 M=150k	FLEET3 M=150k	FLEET2 M=300k	FLEET3 M=300k	FLEET2 M=600k	FLEET3 M=600k	*sGrapp*	*sGrapp*-100
Epinions	89575	137411	59336	53077	16912	16360	11028	10907	**182427**	166895
ML100k	3664	5652	4691	4717	3509	3424	4268	4378	8026	**8629**
ML1m	23490	23292	12038	7355	2383	1673	1004	857	**26698**	26487
ML10m	147665	72918	62905	23536	16719	5358	4410	2337	**234571**	228021
Edit-FrWiki	554741	155343	298019	57477	116917	16856	41051	6240	**1000861**	985265
Edit-EnWiki	**2564565**	719375	1373708	305347	911170	114806	324183	34283	1085185	1098382

Table 5.4: MAPE of different algorithms for γ=0.7 and M=0.1S and same N_b.

MAPE	FLEET1	FLEET2	FLEET3	*sGrapp*	*sGrapp*-25	*sGrapp*-50	*sGrapp*-75	*sGrapp*-100
Epinions	0.058	13.789	0.336	**0.022**	**0.022**	0.028	0.028	0.028
ML100k	0.959	2.287	0.399	**0.009**	**0.009**	**0.009**	**0.009**	**0.009**
ML1m	0.085	5.261	0.047	0.043	**0.043**	**0.053**	0.067	0.055
ML10m	0.156	0.839	**0.086**	0.143	0.247	0.162	0.180	0.170
Edit-FrWiki	1.575	49.165	57.563	0.201	0.313	0.217	**0.134**	0.137
Edit-EnWiki	2.689	467.747	178.702	0.684	0.494	0.161	0.141	**0.137**

increasing the size of reservoir the throughput of all FLEET algorithms decreases since the frequency of exact butterfly counting per edge increases. It is always the case that $M = 75k$ and $M = 600k$ yield the highest and the lowest throughput, respectively. *sGrapp* outperforms FLEET for every setting: minimum (maximum) ratios of *sGrapp* to FLEET throughput are 1.32 (16.7), 1.5 (2.5), 1.13 (31.1), 1.58 (100.3), 1.8 (160.4), and 0.4 (32) in Epinions, ML100k, ML1m, ML10m, Edit-FrWiki, and Edit-EnWiki, respectively. *sGrapp* and *sGrapp-x* outperform FLEET suit within a range of $[1.13, 160.4]$, with the performance improvement increasing as streams become larger (i.e. Edit-FrWiki, ML10m, and Edit-Enwiki).

Table 5.4 reports accuracy (in terms of MAPE) of *sGrapp* and FLEET suits over the subset of stream with available true values. *sGrapp* and *sGrapp-x* achieve MAPE values equal to 0.022, 0.009, 0.043, 0.143, 0.134, and 0.137 in Epinions, ML100k, ML1m, ML10m, Edit-FrWiki, and Edit-EnWiki which are significantly lower than those of FLEET – *sGrapp* errors are 0.38×, 0.02×, 0.91×, 1.66×, 0.08×, and 0.05× of FLEET for these graphs. Table 5.4 shows that for ML10m, FLEET3's accuracy is 0.057 better than *sGrapp*. Table 5.3 similarly dhows that FLEET3's throughput for the same dataset is up to 100× lower, explaining the high computational cost of FLEET3 in this specific dataset. FLEET3 updates the estimate for each new edge by enumerating butterflies incident to that edge. This increases the probability of detecting the incident butterflies by a factor of P (i.e. sampling probability), however the computations are much increased. This technique is more impactful in ML10m with high butterfly density. Butterfly estimate $\hat{B}$ is updated as

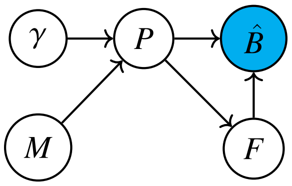


Figure 5.22: Impact of FLEET parameters on estimate.

soon as an edge arrives in FLEET3 or during the sampling and (or) sub-sampling phase in FLEET1 (FLEET2). In FLEET1, when P is not high or M is small and γ is low, $\hat{B}$ is not frequently updated and error goes up. In FLEET2, many butterflies are missed due to sampling. Moreover, FLEET has poor accuracy when the butterflies are distributed across the edges uniformly (e.g. Edit-EnWiki with a low butterfly density of 9.1×10^{-21} according to the statistics in [286]). The reason is that $\hat{B}$ is updated for some edges only.

In summary, the accuracy of FLEET algorithms are highly dependent on M, γ, and the frequency of updating $\hat{B}$, because $\hat{B}$ is updated with respect to the P; and P is updated as $p \leftarrow p * \gamma$ in each sampling round, which in turn increases $\hat{B}$ more. As depicted in Figure 5.22, M and γ (confounding variables) impact P and P impacts $\hat{B}$ directly through the formula and indirectly through the frequency of updates. A high frequency of butterfly counting and high sub-sampling come at the cost of low throughput. A large M comes at the cost of memory consumption as well as latency issues. FLEET suit cannot guarantee both efficiency and effectiveness at the same time. *sGrapp* does not suffer from the aforementioned issues since it does not rely on exact counting and sampling; rather it relies on counting the inter-window butterflies. *sGrapp* keeps the computational footprint of exactly counting the in-window butterflies low by means of the load-balanced adaptive windows and then, effectively estimates the number of inter-window butterflies which are the dominant butterflies based on the butterfly densification power law formalism.

5.4.5 Complexity Analysis

A previous study of space bounds [286] has shown that any butterfly counting algorithm, either randomized or deterministic, that returns an accurate (exact/approximate) answer (i.e. bounds the relative error to a small value $0 < \delta < 0.01$ for each computation round) requires storing the entire graph in $\theta(n^2)$ bits, where n is the number of vertices. On the other hand, it is not possible to determine the size of stream (i.e. n) in real world streaming graphs. Hence, it is not possible to determine the memory required for processing the data without knowing the size of data [26]. In the following, the properties of the introduced estimator are analyzed in terms of computational and error bounds.

Computational Bound

THEOREM 2 *The upper bound of computational complexity of sGrapp for each window* W_k *is* $O(\frac{K_{i,W_k}(K_{i,W_k}-1)}{2}K_{j,W_k}\mathcal{R}N_t^{W_k})$*, where* $\mathcal{R}$ *is the average stream rate and* $K_{i,W_k}(K_{j,W_k})$ *is the lower bound of degree of i(j)-vertices in* W_k*.*

PROOF 2 *sGrapp's computations at each window are dominated by the exact counting algorithm as calculating the number of inter-window butterflies is negligible and we ignore it as well as the summations. When i-vertices are the vertex set with lower average degree, the computational complexity of the core exact counting algorithm is the following.*

$$O(\sum_{i_1 \in V_i} \sum_{j_1, j_2 \in N_{i_1}} Min(deg(j_1), deg(j_2))) \tag{5.7}$$

Assume the lower bound i-degree and j-degree in the graph snapshot corresponding to the tumbling window W_k *be* K_{i,W_k} *and* K_{j,W_k}*, respectively. Accordingly, the computational complexity for this window would be* $O(\frac{K_{i,W_k}(K_{i,W_k}-1)}{2}K_{j,W_k}|V_{i,W_k}|)$*, where* V_{i,W_k} *denotes the set of i-vertices in the window* W_k*. Since the stream can include edges connecting already existing vertices, the total number of edges in* W_k*, denoted as* E_{W_k}*, is greater than equal the total number of i-vertices in* W_k*, i.e.* $|V_{i,W_k}| \leq |E_{W_k}|$*. Therefore,*

$$O(\frac{K_{i,W_k}(K_{i,W_k}-1)}{2}K_{j,W_k}|V_{i,W_k}|) \leq O(\frac{K_{i,W_k}(K_{i,W_k}-1)}{2}K_{j,W_k}|E_{W_k}|) \tag{5.8}$$

sGrapp *uses tumbling windows with adaptive lengths, therefore* $|E_{W_k}| \approx \mathcal{R}N_b^{W_k}$*, where* $\mathcal{R}$ *is the average stream rate (i.e. number of edges per timestamp) and* N_b^W *is the number of unique timestamps (bursts) in* W_k*. Hence, the upper bound of computational complexity of sGrapp for a tumbling window* W *at* t *is* $O(\frac{K_{i,W_k}(K_{i,W_k}-1)}{2}K_{j,W_k}\mathcal{R}N_b^{W_k})$*. Note that this stands for all sequential windows.*

Error Bound

LEMMA 1 *The exact number of inter-window butterflies at the end of each window* W_k*,* $\forall k > 0$*, denoted as* B^{interW} *is bounded as* $|E_{W_k}| - 2|V_{i,W_k}| \leq B^{interW} \leq \binom{|V_{i,W_k}|}{2}$*, where* V_i *is the set of all i-vertices in the* W_k*.*

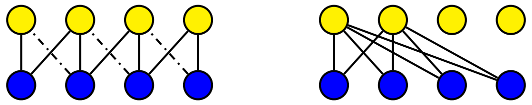


Figure 5.23: Schematic butterfly formation. i(j)-vertices are in the bottom (top).

PROOF 3 *The number of inter-window butterflies contributed by window W_k denoted as B^{interW}, is minimum when the W_k's edges E_{W_k} are uniformly distributed over vertices by connecting each i-vertex in W_k to at least 2 j-neighbors in W_k and previous windows forming a series of caterpillars (solid edges in Figure 5.23–left). In this case, according to the pigeonhole principle, the number of edges that complete the caterpillars (dashed edges in Figure 5.23–left) will determine the number of inter-window butterflies: $B^{interW} = |E_{W_k}| - 2|V_{i,W_k}|$. B^{interW} is maximum when all of the W_k's i-vertices are connected to two j-vertices such that at least one of them is not in W_k (Figure 5.23–right). (Note, when all of j-neighbors are in previous windows, there wouldn't be any in-window butterfly in W_k). In this case, the number of inter-window butterflies reduces to the number of ways we can choose two i-vertices from the entire set of i-vertices: $B^{interW} = \binom{|V_{i,W_k}|}{2}$. Therefore, $|E_{W_k}| - 2|V_{i,W_k}| \leq B^{interW} \leq \binom{|V_{i,W_k}|}{2}$.*

THEOREM 3 *The absolute error of* sGrapp *at the end of each window W_k is bounded as $\Sigma_{l=1}^{k}|E_l|^{\alpha} - \binom{|V_{i,W_k}|}{2} \leq Err \leq \Sigma_{l=1}^{k}|E_l|^{\alpha} - |E_{W_k}| + 2|V_{i,W_k}|$ where E_k, E_{W_k}, and V_{i,W_k} denote the number of edges in the interval $[W_0^b, W_k^e)$, the number of edges in the interval $[W_k^b, W_k^e)$, and the number of i-vertices in the interval $[W_k^b, W_k^e)$, respectively.*

PROOF 4 sGrapp *estimates the total number of butterflies at the end of each window W_k, $\forall k > 0$, as $\hat{B}_k = \hat{B}_{k-1} + B_G^{W_k} + |E_k|^{\alpha}$ with initial term $\hat{B}_0 = B_G^{W_0}$. Expanding this recursive equation would yield $\hat{B}_k = \Sigma_{l=0}^{k} B_G^{W_l} + \Sigma_{l=1}^{k} E_l^{\alpha}$. On the other hand, according to the lemma 1, the true value of the total number of butterflies at the end of each window W_k, $\forall k > 0$, denoted as B_k lies in the range $\Sigma_{l=0}^{k} B_G^{W_l} + E_k - 2|V_{i,W_k}| < B_k < \Sigma_{l=0}^{k} B_G^{W_l} + \binom{|V_{i,W_k}|}{2}$, where V_{i,W_k} is the set of all seen i-vertices in the interval $[W_k^b, W_k^e)$. Therefore, the absolute error of* sGrapp *$Err = |B_k - \hat{B}_k|$ falls in the range $\Sigma_{l=1}^{k}|E_l|^{\alpha} - \binom{|V_{i,W_k}|}{2} \leq Err \leq \Sigma_{l=1}^{k}|E_l|^{\alpha} - |E_{W_k}| + 2|V_{i,W_k}|$.*

5.5 Summary

Essence and features. Butterfly counting is a fundamental problem in mining bipartite streaming graphs. Butterfly counting is computationally expensive, and known techniques do not scale to large graphs; the problem is even harder in streaming graphs. The insights from Phase 1 of analysis of real-world streams shed light on developing a streaming graph approximation framework for butterfly counting, *sGrapp*, as follows.

- The streaming graph mining findings establish the requirements for an efficient and effective analytical stream processing framework. Accordingly, a window management method is introduced to deal with the bursty emergence patterns of butterflies. This approach uses burst-based tumbling windows that can adapt to the temporal distribution of the real streams with no assumption about the order and number of arriving streaming graph records. The benefits are two-fold:
  - Providing load-balanced windows for efficient analytical workloads.
  - Enabling accurate continous/temporal analysis which are based on comparing the analysis over different windows of a stream as well as analysis over different streams having different temporal distributions.
- Statistical analyses uncover the temporal organizing principles of butterflies that impact the identification of any potential butterfly that should be counted by the algorithm. Specifically, the study reveals the dominant contribution of old hubs with young neighbours on shaping butterfly structures over time. That is, the stream mining reveals the emergence of a certain type of butterflies (inter-window butterflies), which are computationally challenging to capture since they span a long time interval to form (as it takes a while before newly added vertices get connected to old hubs and the butterfly structure completes) and demand lengthy time-based windows to cover them (performance bottleneck of incremental butterfly approximation). The enumeration of these butterflies is separated from the rest of streaming butterflies and their count is estimated according to the BPL. This separation has two impacts:
  - Increases the accuracy due to capturing frequent butterflies.
  - Increases efficiency due to enumerating the rest of the streaming butterflies using the proposed exact counting core algorithm on top of an adaptive burst-based windowing method.
- The results of streaming graph mining (contribution of old hubs to butterfly emergence and their dynamic degree) explain the fluctuating over/under-estimation of

butterfly count. Accordingly, optimisations are introduced to enhance the approximate count particularly in dense graphs. *sGrapp* is optimised by changing the exponent of BPL over windows. To this end, it is modified to a semi-supervised algorithm that is called *sGrapp-x*. The performance is improved as the following.

- The minimum average window error is lowered for up to 85.76%.
- The maximum average window error is lowered for up to 87%.
- The probability of low approximation error (i.e. error below 0.15 and 0.2) increases up to 100% in the densest graphs.

Use cases. Similar to triangles in unipartite graphs, enumerating butterflies is crucial in understanding the structure of bipartite graphs. This benefits many applications where studying the cohesion in a graph shaped data is of particular interest. Examples include investigating the structure of computational graphs or input graphs to the algorithms, as well as dynamic phenomena and analytic tasks over complex real graphs.

Evaluations. *sGrapp* displays average window error below 0.05 in streams with almost uniform temporal distribution and its optimised variant yields average window error below 0.14 in streaming graphs with non-uniform temporal distribution. *sGrapp* can handle streams with both high number of edges and high average degree with a sub-linear memory footprint, which is lower than that of the baselines and processes 1.5×10^6 sgrs-per-second. Experimental analysis show that *sGrapp* achieves 160× higher throughput and 0.02× lower estimation error than baselines. Moreover, empirical analyses demonstrate that the performance of *sGrapp* is independent of its input data, hence can be applied to any real stream.

Chapter 6

Streaming Graph Analytics - Concept Drift Detection

Concept drift (CD) is a phenomenon that occurs when "changes in hidden context can induce more or less radical changes in target concept" [344]. Hidden context refers to insufficient, incomplete, or unobservable information about the dataset [113], and target concept refers to known and/or observable information which have direct impact on the output of a downstream task over this dataset. The task can be training a learner model or some form of data analytics/analysis; in both cases the processing task is required to adapt to the changes in its fresh input so as to generate relevant and reliable outputs. CD has been identified as the main cause of decreased effectiveness in many data-driven information systems such as data-driven decision support systems and early warning systems [218]. CD detection benefits applications including cases where it is important to detect a change in the streaming sensor data that affects the automatic monitoring of operation of an industrial plant, a smart home management system, or a robotic mobile system.

CD is natural in non-stationary settings where temporal or spatial evolutions in data characteristics or generative source(s) are the origins of the drift [173, 237]. Streaming data are good representatives of such non-stationary settings, where changes are reflected in data snapshots/samples created at different time points or locations over streaming data records generated by one or multiple generative sources. Another source of change can be sampling bias in supervised machine learning tasks where the generating distribution and/or labelling function of source training data (a source domain) is different from those of target/test data (a target domain) used for operation/evaluation of the trained model. This case is referred to as *domain adaptation* [49, 48, 50, 101, 43]. In this thesis, the former case is studied: detecting changes in streaming data records which are transient, sequential (partially ordered by their arrival time), and heterogeneously interconnected, forming a bipartite streaming graph. No assumptions are made on the downstream task(s) and the availability of data labels in the source and target domains. The goal is to answer the question *how to detect/understand concept drift in a bipartite streaming graph*. It is an important problem for the following reasons:

- Study of CD is vital in various application areas such as monitoring and control (for "real-time monitoring or control of some automated activity"), information management (for "organizing and personalizing information"), and analytics and diagnostics (for "characterizing health, well-being, or a state of humans, economics, or entities") [381].

- Current works mostly focus on supervised learning tasks where the training streaming data records are labelled and statistically independent (usually stream of data points or unipartite graphs). In these studies, CD is considered as a change in a statistical property of data, particularly the joint probability distribution of data and a target label. On the other hand, in many real-world scenarios, the data records are structured as biparte streaming graph records: heterogeneously interconnected and not statistically independent. Also, in the aforementioned applications, the processing task includes pattern recognition, anomaly detection, and clustering, where the ground truth is not always available and the task is unsupervised.

- Research in CD management covers three aspects [218]:

  - CD detection: identify changes to characterize and quantify the drift.
  - CD understanding: describe the drift event by providing information about the time, severity, and/or the contributing factors of drift.
  - CD adaptation: update a downstream task whenever a need for adaptation is identified.

  While many specialized techniques are introduced in the literature for online drift adaptation as part of supervised prediction tasks such as classification, limited number of works consider drift detection over a streaming graph as an independent component that can be combined with downstream analytics. Therefore, there is still a need for a general purpose CD management framework that

  - integrates with any downstream analytics (both supervised and unsupervised adaptive tasks as discussed above),

- incurs low computational overhead in terms of both time and memory, while accurately detecting the drifts,
- supports various drift types,
- provides information about the drifts' time and location,
- enables unsupervised drift detection, and addresses the lack of ground truth for drift events in real-world data streams, and
- does not require user-defined threshold for drift evaluations.

6.1 Drift Detection over Streaming Graphs

Detecting drifts in streaming graphs can be done by tracking either the changes of the generative source or the performance of a downstream task operating on the data. While the latter method depends on the task, the former depends on the data source, which may or may not be related to the task. For instance, certain tasks require specific data formats/availability, while different tasks using the same data can be designed for a certain goal. This thesis takes the former approach to detect a change in a transient concept by detecting a change in data source. The precise problem definition is as follows: *Given a sub-sequence of streaming graph records partially ordered by their arrival time steps, how to detect a concept drift as close as possible to its occurrence, while providing descriptive information about the drift.*

While different structures can be considered for transient concept, temporal motifs are better candidates since their abundance over the timeline of sgr arrivals benefits drift detection; less frequently occurring structures challenge the detection method. As shown in Chapter 3, butterflies are temporal motifs and their emergence patterns imply the generative patterns of sgrs. Therefore, butterflies can be utilised to effectively detect drifts over streaming graphs. Moreover, butterflies, as the smallest bipartite cohesive subgraphs, enable incremental processing during drift detection. As new sgrs are added to the graph snapshot, butterflies can be maintained without re-examining. Other bipartite structures (maximal subgraphs including several butterflies such as k-bitruss [332], k-wing [288], and $s(\alpha,\beta)_\tau$−core [151]) require dynamic maintenance. Therefore, butterflies can be utilised to effectively and efficiently detect drifts over streaming graphs.

Efficiency and effectiveness of the detection approach should be simultaneously achieved since:

Table 6.1: Frequent notations in this chapter.

Notation	Description
$P(\bowtie_v) = v$	Function mapping a young butterfly to a unipartite vertex v
$\bigcap(\bowtie_u, \bowtie_v)$	Set of common vertices in $\bowtie_u$ and $\bowtie_v$
Ω_v	Frequency of a phase oscillator vertex v
θ_v	Phase of a phase oscillator vertex v
$O[t]$	Order parameter values at sequential time points
S	Drift detection suffix size

- Accuracy of drift detection can be influenced by the amount of analyzed data records. For instance, detecting an abrupt drift does not require a lengthy window of data records compared to gradual, incremental, or reoccurring drifts which require a window spanning a longer time interval and larger amount of data to be examined.

- Accuracy of detections relies on the efficiency, since late detections can be misleading and detections should be as close as possible to the drift events.

On the other hand, similar to other analytic tasks, CD detection over streaming graphs is computationally expensive due to the volume and velocity of the data, as well as the iterativeness and statefulness of the processing task. As discussed in Section 5.1, computing exact answers for analytic tasks over streaming graphs is not feasible since (1) it is not possible to maintain the entire graph, intermediate results, and system states in memory due to unboundedness of the graph, and (2) exact answers demand performing the analytics over the entire stream. Even if we relax the memory constraint, analytics such as butterfly-based tasks enumerating butterfly cliques are compute-intensive, incurring a lower bound of computation of $O(n^3)$, where n is the growing size of vertices in the stream. Therefore, a butterfly-based drift detection algorithm requires design of efficient and effective sliding window management techniques.

In the following, first, the related works on concept drift management are reviewed in Section 6.2. Next, transient concept and concept drift in a streaming graph are defined, according to which a framework for streaming graph drift detection called *sGradd* is introduced and evaluated in Sections 6.3 and 6.4. *sGradd* demonstrates efficiency and effectiveness in detecting drifts with different types and occurrence intervals. Table 6.1 lists the frequent notations in this chapter.

6.2 Related Works

The related works are viewed through the lens of a modular framework for performing drift detection, understanding, and adaptation. The framework has three components:

- Data management for retrieving streaming data and memory management to decide how to retain data and system state in memory;
- Drift detection for identifying changes and corresponding metadata; and
- Drift adaptation for updating the downstream task.

Accordingly, the existing works are divided in two groups: *active* (Figure 6.1) and *passive* (Figure 6.2) approaches. In active approaches, streaming data is continuously ingested and windowed via data management component and then drifts are explicitly detected and explained via drift detection component, which triggers a signal for updating the downstream task via drift adaptation component. In passive approaches, a data model is learned in the data management component to extract the most important features of data for dimensionality reduction purposes, and the target predictions of downstream task. Based on the performance of this model (for instance, the learner's error), an implicit drift alert is signaled for drift adaptation. Since the focus in this thesis is on concept drift detection and understanding, in the following, only the data management and drift detection components in active and passive approaches are reviewed. More comprehensive review of the works on drift adaptation exist in literature [218, 125, 4].

6.2.1 Data management

Data records are continuously ingested and windowed through the window management sub-component and possibly fed into a learner model through the data model sub-component (green boxes in Figures 6.1 and 6.2).

Window management. While in most passive approaches, a model is learned over the stream of incoming data records using a landmark window, active approaches usually use a two-window method: a *reference window* and a *data window*. Contents of data window are evaluated when the window closes using the reference window as a baseline to determine whether or not a change has happened. While the data window covers the newly arrived data records, the reference window can be fixed [292, 219] or moving [173, 35]. Another windowing method in active approaches is to use one data window. Contents of each

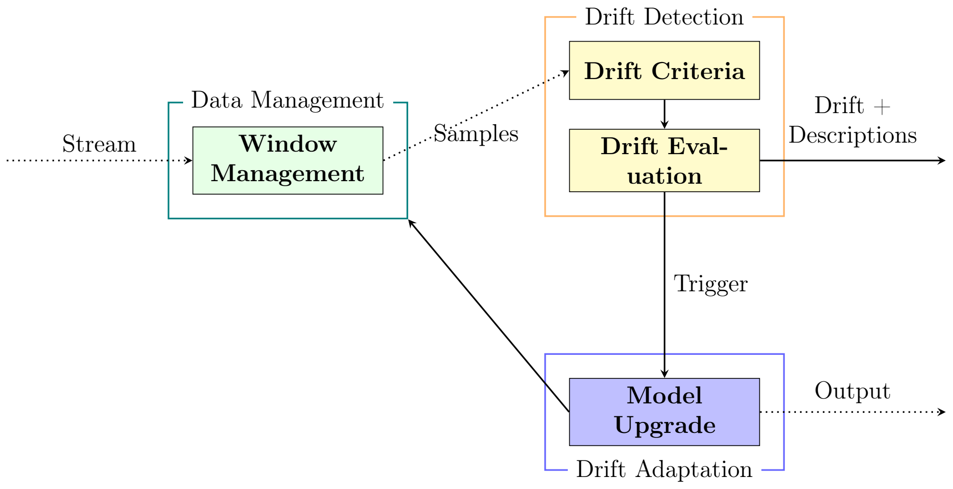


Figure 6.1: Active concept drift management.

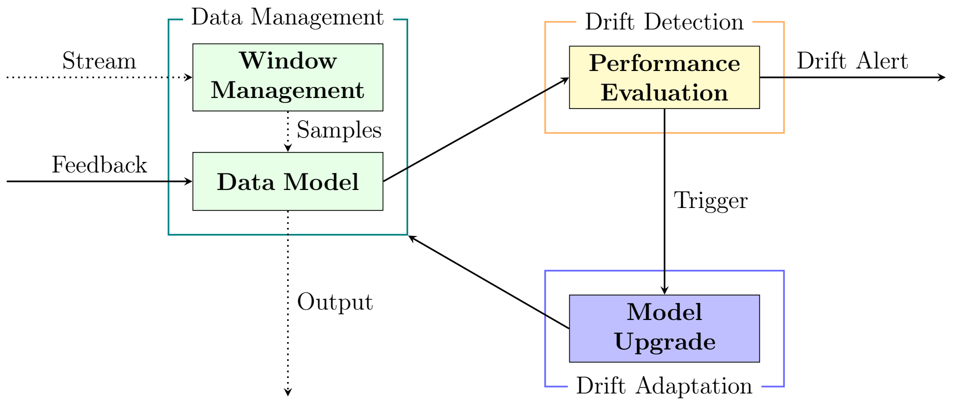


Figure 6.2: Passive concept drift management.

window instance are compressed to low dimensional embeddings. This results in a sequence of embeddings as the drift criterion, which is evaluated to detect a change [352, 358, 262]. Different techniques have been used in drift detection literature for maintaining windowed contents with respect to the window borders, window size, and sliding approach. Active approaches commonly use landmark windows [124], while others use sliding windows [246, 352, 358, 262] with static time-based or count-based sizes [34, 352, 358, 262] or dynamic and optimised sizes [60, 132]. When the window size is fixed, all/sampled streaming records are added/removed according to the size [352, 358, 262] or a weighting function is used to gradually remove elements with low weights [125]. Passive approaches update the learner upon the arrival of each new data record. These approaches do not remove the contribution of old data and slowly adapt to abrupt changes as adaptation only happens when new data records omit the old concepts. Window management raises a trade-off between efficiency and effectiveness of drift detection in both active and passive approaches as small windows can detect sudden/abrupt drifts, however gradual and slow drifts require sufficiently large window sizes.

Data model. In passive approaches, given a window of streaming data, a data model is learned which performs the target adaptive task (green boxes in Figure 6.2). The decrease in model's effectiveness determines the need for an adaptation (yellow box in Figure 6.2). For instance, when the online error rate of a classifier reaches a drift threshold, a model update is required [124, 212, 349, 122]. Some methods also consider a warning threshold to prepare a new model and replace it with the old model when the drift threshold is reached. Since there is no explicit drift detection in passive approaches, understanding the drift events (i.e. when, where, and how the drift has happened) is not possible and the adaptation can not be informed by the drift patterns. For instance, reoccurring concept drifts can be more efficiently adapted by reusing the trained models or re-training the poor performers only, especially in case of ensemble learning. Unnecessary updates can be avoided in case of insignificant drifts. Moreover, the adaptation can be triggered too often in noisy environments, or too slowly in case of gradual drifts, which requires lengthy windows that in turn increase the computational cost. In those cases where a user feedback is required for model updates, the drift adaptation is further delayed.

6.2.2 Drift detection

In active approaches, given the windows of streaming data, drift criteria is evaluated to decide whether or not a drift has occurred (yellow boxes in Figure 6.1). Hidden concepts that are recognized as the root source of drifts are not tractable/measurable, therefore concept drift is usually detected when a statistical property of data stream changes over time. The first formal definition of change detection in data streams [173] considers windows as data samples and compute their distribution distance to identify a drift using a hypothesis test method. This type of drift detection is also done using multiple hypothesis tests running in parallel or as a hierarchy of sequential tests [366, 356]. Other works on graph streams (sequence of graph snapshots arriving one at each time point) compute embeddings [358] or entropy of discriminative subgraphs [262, 352] over a batch of graph snapshots and detect a change over the sequence of embeddings/entropy values. The drift evaluation is done by means of a diversion dissimilarity measure [262] or a hypothesis test [358] and utilising a static user-defined threshold.

6.3 sGradd

In this section the concept of drift in a streaming graph is defined and a **s**treaming **gra**ph framework for **d**rift **d**etection called *sGradd* is proposed. The framework features the following properties:

- It has only one parameter, N_b, which determines the number of bursts in the window ingesting the sgrs. This parameter can be adjusted based on the expected level of burstiness in the stream. For bursty streams, $N_b = 1$ is a reasonable setting, while in case of low streaming rates, $N_b > 1$ is a more efficient setting.
- Equipped with efficient window management techniques, the framework balances the computational workload and conforms to real-world streams with varying streaming rates over time. It uses one data window only, and does not require a reference window. The data window is a burst-based sliding window which is projected to a predicate-based (logical) sliding window. Contrary to existing works that use fixed-size sliding windows with a fixed slide size of one step, in the proposed framework, the sizes of both windows adapt to the evolving streaming rate. Also, the burst-based sliding window moves forward as soon as it is projected by retiring all of its elements (i.e., it is a tumbling window). The predicate-based window moves forward after the drift detection, if a predicate over the stream burstiness is satisfied, by removing a fraction of its randomly selected elements.
- It employs an active approach with a data management component (similar to data modelling in passive approaches), which reduces time and memory consumption through the extraction of effective information from the sgrs and efficiently maintain this information as the system state. The drift evaluation is done by means

of a simple and efficient measure, and utilises a dynamic threshold adaptive to the number of detections without user input initialization.

6.3.1 Definitions

Definition 17 (Structural Pattern) *Structural pattern $\not{p}$ is a quantified pattern of vertex inter-connectivities in a graph snapshot, i.e.* $\not{p}(G_{W,t}) : G_{W,t} \mapsto \mathbb{R}$.

Definition 18 (Transient Concept) *Transient concept is a non-stable structural pattern in transient data records, i.e.* $\not{p}(G_{W,t}) \mid \exists (W_1, t_1), (W_2, t_2) : \not{p}(G_{W_1,t_1}) \neq \not{p}(G_{W_2,t_2})$.

Definition 19 (Concept Drift in Streaming Graphs) *Concept drift in a streaming graph is the event of a change in a transient concept over successive graph snapshots. In other words, given a certain pattern $\not{p}$, concept drift is the event of observing two successive windows W_1 and W_2 corresponding to the sequential time points t_1 and t_2, where $t_2 - t_1 \geq 1$ and $\not{p}(G_{W_1,t_1}) \neq \not{p}(G_{W_2,t_2})$.*

As shown in Figure 6.3, a concept drift can occur in four different modes according to the progress of drift [125]: gradual, reoccurring, incremental, or abrupt.

6.3.2 Overview

The *sGradd* framework (Figure 6.4) takes sgrs from a stream as input, maintains an incrementally updatable graph structure, and streams-out the drift signals as triggers for any online adaptive algorithm. It has a modular architecture with two main components for data management and drift detection.

Data Management. Given the arriving sgrs, in this stage data is managed through windowing and abstracting important information. The goal is to create efficient and effective workload for ultimate processing in the drift detection component. The main idea is to use a tumbling window to ingest raw data records and update information about the burstiness of the stream followed by a projection of the window into a more compact window to effectively reduce the computational overhead of drift detection. This component has two sub-modules:

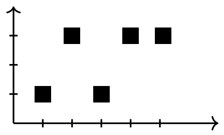
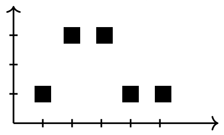
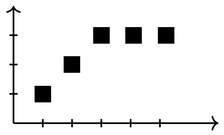
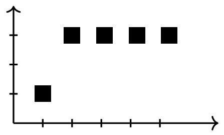
CD mode	Description
Gradual 	the transient concept changes gradually and frequently, while spanning a considerable time interval until a new concept is stabilized.
Reoccurring 	the transient concept switches to a new concept and then it is repeated.
Incremental 	the transient concept changes as incremental steps towards a new concept.
Abrupt 	the transient concept switches to a new concept suddenly at an trivial time interval.

Figure 6.3: Concept drift modes.

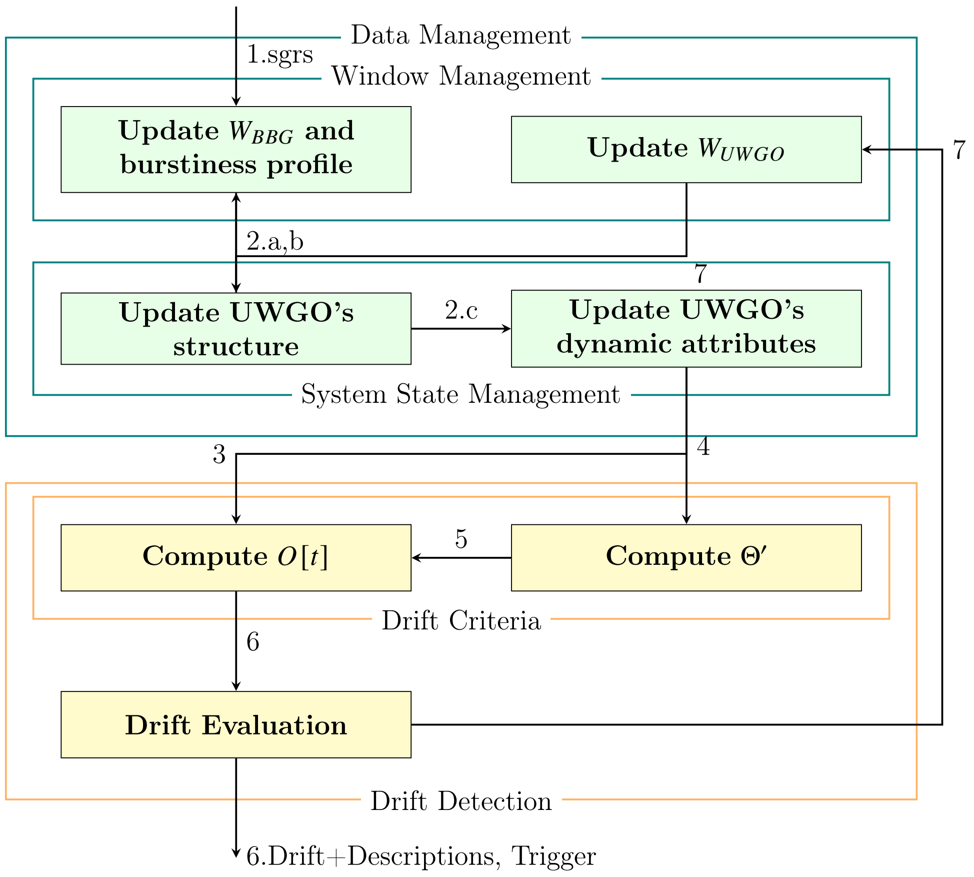


Figure 6.4: *sGradd*'s architecture.

- *Window Management*: This sub-module ingests the sgrs and uses two windows W_{BBG} and W_{UWGO} to manage the sgrs and system state, respectively. W_{BBG} is a **b**urst-based tumbling window with a corresponding **b**ipartite **g**raph snapshot. It accepts ingested sgrs (raw data records). W_{BBG} is projected to W_{UWGO}, which is a predicate-based window with a corresponding **u**nipartite **w**eighted **g**raph of **o**scillators. It updates the system state as described below.

- *System State Management*: The key temporal and structural features of the stream (burstiness properties and butterfly inter-connectivities) are extracted from W_{BBG} and embedded into UWGO to form system states. Drift detection component operates on UWGO and then a number of vertices are removed from UWGO if a predicate is satisfied. This removal completes a computation round[1].

UWGO is aimed to reduce the computational expenses due to the following reasons:

- The butterflies in W_{BBG} are mapped to vertices forming UWGO. The unipartite graph of UWGO resembles a network of oscillators, where each oscillator v has a phase $\theta_v \in [0, 2\pi]$ which oscillates with a frequency Ω_v. The phase of an oscillator embeds the neighborhood of the corresponding butterfly. Therefore, projecting BBG to UWGO implies projecting the butterfly neighborhoods to a latent space of phases in $[0, 2\pi]$.
- The contents of W_{BBG} are entirely retired as soon as they are projected to UWGO. This frees up memory and avoids redundancy. In another words, UWGO decreases the memory consumption by reducing the size of system state and retiring the processed sgrs.
- Similar to W_{BBG}, W_{UWGO} adapts to sgr streaming rate by adjusting its slide interval and slide size to the burstiness of stream. Therefore, UWGO balances the workload assigned to data management and drift detection components in the next round as it adapts to the stream burstiness.
- UWGO is constructed based on mapping butterfly structures to UWGO vertices. Butterflies do not change with the addition of new sgrs, therefore UWGO is maintained incrementally by adding new vertices/edges. In contrast, other bipartite structures such as k-bitruss [332], k-wing [288], and $s(\alpha, \beta)_\tau$−core [151] require dynamic maintenance in streaming settings since they are maximal subgraphs, which include butterflies. Using such structures would add the overhead of recomputing the UWGO structure.

Drift Detection. Given the UWGO snapshot, in this stage concept drifts are detected. Since butterflies are building blocks of the stream (Chapter 3), the transient concept (Definition 18) is considered to be the butterfly inter-connectivity pattern in the original stream. The goal is to detect a change in this concept (Definition 19) to figure out a change in the streaming graph. The main idea is to quantify and evaluate two evolution trends of the transient concept, an observed evolution trend and a predicted one. Briefly, the current and future information about the data are extracted and utilised to learn a drift in the past.

[1]In a distributed/parallel setting, each computation round can start with ingesting sgrs to a window $W_{BBG,K+1}$ in parallel to detecting drifts on a previous window $W_{UWGO,K}$, right after the projection is done. This reduces the interval between two execution of drift detection for as much as $O(Maximum(ingest(k+1), detect(k) + retire(k)))$.

This is done efficiently through the UWGO to learn the drifts as close as their occurrences. This component has two sub-modules:

- *Drift Criteria:* This sub-module computes the concept drift criteria (the density of butterfly inter-connections) over UWGO. For each UWGO snapshot, two instances of the drift criteria are quantified and appended to their corresponding sequence: one based on the observed butterfly inter-connectivities and another based on a prediction of future inter-connectivities.

- *Drift Evaluation:* The quantified drift criteria are evaluated to detect a drift in transient concept. The trend of evolutions in the two sequences of drift criteria are examined while considering the burstiness profile of the stream. At the detection of a change, a drift alert is signaled with descriptive information about the time and location of the drift.

Data Structure. The bipartite graph snapshots of BBG are implemented similarly to the object oriented data structure described in Section 4.3.2. The unipartite graph snapshots of UWGO are also implemented using an object oriented approach as following.

- A vertex is an object with three attributes: an integer ID, a double phase, and a double frequency. To assign the vertex ID, the vertex object v is created with zero values for all attributes and next the hash code of the object v sets the ID.

- An edge is an object with four attributes: a String ID concatenating the ID of vertices, two vertex objects and a double weight.

- A hash set retains vertices in V and two hash indexes are used to store $N(v)$ and fast retrieval of graph elements: One hash index retains $N(v)$ by mapping the vertex IDs to a hash set of neighbours; Another hash index retains E by mapping the edge IDs to the edge objects. New vertices are added to V and to $N(v)$ if their ID is absent, therefore the stored vertices have distinct IDs. When an sgr arrives, The ID of sgr vertices are added to $N(v)$ according to the aforementioned method and their corresponding hash set of neighbours are updated. Duplicate edges are not stored.

sGradd is precisely described in Algorithm 8.

Algorithm 8: sGradd(N_b)

```
Data: $\mathfrak{R} = \langle r^1, r^2, \cdots \rangle$, sequence of sgrs
Input: $N_b$, Size of BBG window
1  $BBG \leftarrow \langle V_1 = V_i \cup V_j = \emptyset, E_1 = \emptyset \rangle$, $UWGO \leftarrow \langle V_2 = \emptyset, E_2 = \emptyset \rangle$, $unqt \leftarrow \emptyset$, $k \leftarrow 0$,
     $t \leftarrow 0$, $currentB \leftarrow 1$, $\bar{B} \leftarrow 0$, $bursty \leftarrow false$, $maxB \leftarrow 0$, $b2v \leftarrow \emptyset$, $d \leftarrow 0$
2  while true do
3      $\langle currentB, \bar{B}, maxB, isBursty \rangle \leftarrow ingest(r^t, currentB, \bar{B}, maxB, isBursty)$
4      if ($\tau^t \notin unqt$ & $(|unqt| + 1)\% N_b = 0$ & $|unqt| > 1$) then
5          $unqt.add(\tau^t)$
6          $UWGO \leftarrow project(iVertexConnections, unqt, BBG, UWGO, b2v)$
7          System.gc()
8          $\langle O_1, O_2 \rangle \leftarrow$DriftCriteria($UWGO$, $t$)
9          $d \leftarrow DetectDriftv1(O_1, O_2, d, t, winNum)$ or
             $DetectDriftv2(\bar{B}, maxB, O_1, O_2, d, t, winNum)$
10         $\langle UWGO, k \rangle \leftarrow$slide($UWGO$)
11     else
12         $unqt.add(\tau^t)$
13     $t{+}{+}$
```

6.3.3 Data Management

The interleaved procedures of window management and system state management sub-modules are as follows (steps 1 and 2 and green boxes in Figure 6.4).

1) Arriving sgrs are continuously ingested into W_{BBG} (Algorithm 8, line 3 invoking Algorithm 9). Edges are added to BBG and the burstiness profile of the stream, which includes the following four quantities, is updated online.

   - $currentB$, the current burst size
   - $\bar{B}$, the average burst size
   - $maxB$, the largest seen burst size
   - $isBursty$, a boolean flag for burstiness of the arriving sgrs

   W_{BBG} is a burst-based tumbling window (Definition 10), same as the window used in *sGrapp* (described in Section 5.3). When N_b bursts are seen, the window closes and the following steps are performed (Algorithm 8, lines 4-10).

2) The full window W_{BBG} is projected to W_{UWGO} as follows (Algorithm 8, line 6 invoking Algorithm 10 – Figure 6.5).

2.a) The structure of UWGO is updated by identifying the young butterflies, mapping them to UWGO vertices, and connecting the UWGO vertices.

The young butterflies (Definition 15) $\bowtie \in W_{BBG}$ are identified using the exact butterfly listing algorithm in *sGrapp* suit (Algorithm 10, lines 3-15). Figure 6.5(b) illustrates the eight young butterflies identified among eleven butterflies (butterflies incident to j_1 are excluded):

$$\bowtie_{v_1} = \{e_{i_1,j_2}, e_{i_2,j_2}, e_{i_2,j_3}, e_{i_1,j_3}\}$$

$$\bowtie_{v_2} = \{e_{i_1,j_3}, e_{i_2,j_3}, e_{i_2,j_4}, e_{i_1,j_4}\}$$

$$\bowtie_{v_3} = \{e_{i_1,j_2}, e_{i_2,j_2}, e_{i_2,j_4}, e_{i_1,j_4}\}$$

$$\bowtie_{v_4} = \{e_{i_2,j_5}, e_{i_3,j_5}, e_{i_3,j_6}, e_{i_2,j_6}\}$$

$$\bowtie_{v_5} = \{e_{i_4,j_7}, e_{i_5,j_7}, e_{i_5,j_8}, e_{i_4,j_8}\}$$

$$\bowtie_{v_6} = \{e_{i_5,j_9}, e_{i_6,j_9}, e_{i_6,j_{10}}, e_{i_5,j_{10}}\}$$

$$\bowtie_{v_7} = \{e_{i_6,j_{11}}, e_{i_7,j_{11}}, e_{i_7,j_{12}}, e_{i_6,j_{12}}\}$$

$$\bowtie_{v_8} = \{e_{i_8,j_{13}}, e_{i_9,j_{13}}, e_{i_9,j_{14}}, e_{i_8,j_{14}}\}$$

UWGO vertices are created and connected to each other (Algorithm 10, line 16 invoking Algorithm 11). Each young butterfly is mapped to a unipartite vertex using a projection function $P : \bowtie \in W_{BBG} \mapsto v \in W_{UWGO}$. Two unipartite vertices are connected iff the corresponding butterflies share at least one i-(j-)vertex and the number of shared vertices determines the edge's weight (Algorithm 10, lines 5-19). i.e. for any UWGO vertices u and v,

$$e_{vn} = (v, n, |\cap(\bowtie_v, \bowtie_n)|) \in UWGO \iff v = P(\bowtie_v), n = P(\bowtie_n), |\cap(\bowtie_u, \bowtie_v)| \geq 1$$

where $\cap(\bowtie_u, \bowtie_v)$ is the set of shared i-(j-)vertices in $\bowtie_u$ and $\bowtie_v$. In user-item streams, i-vertex based connections reveal patterns of user preferences over new items and j-vertex based connections reveal patterns of new item perceptions. For example, in Figure 6.5(c), v_1 and v_2 are connected by an edge with weight equal to two since $\bowtie_{v_1}$ and $\bowtie_{v_2}$ share two i-vertices i_1 and i_2.

2.b) The local data structures are set to null. Also, all of the sgrs in W_{BBG} are retired to free up memory and avoid redundant updates to the system state (Algorithm 10, line 17).

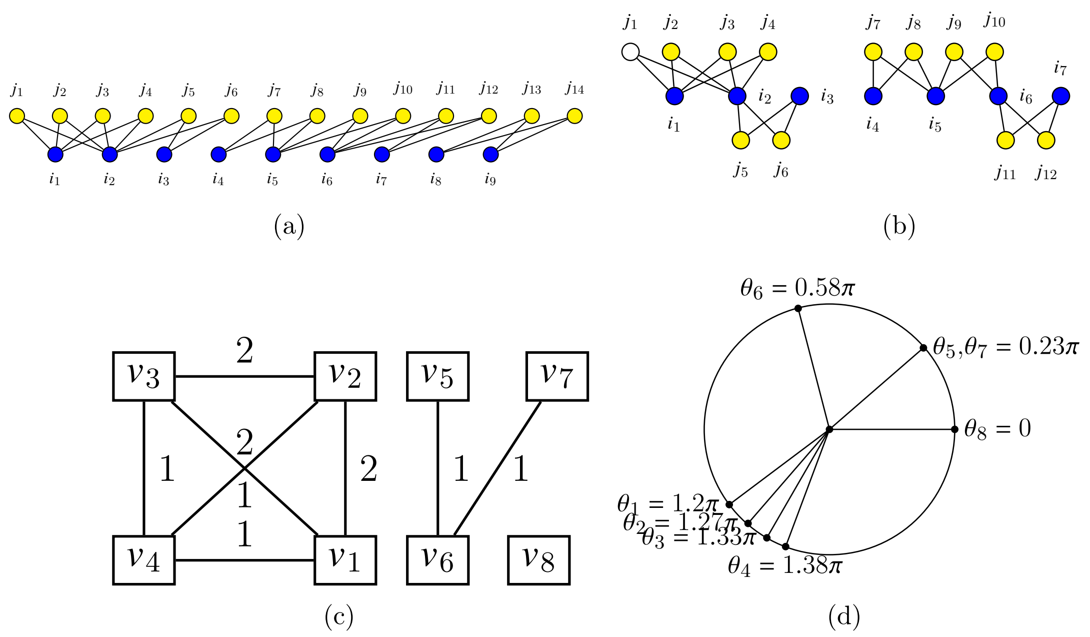


Figure 6.5: Projecting G_{BBG} to UWGO. (a) G_{BBG}. (b) Identified young butterflies with shared i-vertices. (c) Structure of UWGO. Where not clear, the label on the right side of an edge denotes the weight. (d) Phases of UWGO vertices.

2.c) The attributes of UWGO vertices (connected phase oscillators) are updated (Algorithm 10, lines 18-23). To this end, each vertex $v \in W_{UWGO}$ is assigned a frequency Ω_v sampled from a normal distribution with mean equal to one and a phase $\theta_v = (\sum_{n \in N(v)} H(n))\%2\pi$, where $H(n)$ is the hash code of the object[2] representing the vertex n. Isolated vertices would have $\theta = 0$ and vertices with same neighbourhoods would have same phases. Figure 6.5(d) shows example phases for oscillators, where $\bowtie_{v_8}$ has $\theta = 0$ since it is not connected to other butterflies, $\bowtie_{v_5}$ and $\bowtie_{v_7}$ have a same phase as they have one shared neighbour $\bowtie_{v_6}$, and the rest of butterflies except for $\bowtie_{v_6}$ have close phases due to similar neighbourhoods.

[2]The hash code is implemented using Java method hashCode() which must consistently return the same integer for 'equal' objects during one execution of a Java application. This method is not required to return distinct integers for unequal objects by general contract indicated in https://docs.oracle.com/. In *sGradd*'s implementation, this concern is resolved since the butterflies are distinct and their corresponding vertex objects are unequal.

Algorithm 9: Add sgrs into W_{BBG} and update burstiness profile of the stream

```
1  Function ingest($r^t$, $currentB$, $\bar{B}$, $maxB$, $isBursty$)
       Data: $r^t$, a sgr with index $t$
       Input:
       $currentB$, the current burst size
       $\bar{B}$, average burst size
       $maxB$, the largest seen burst size
       $isBursty$, a boolean flag for burstiness of stream
       Output:
       $\langle currentB, \bar{B}, maxB, isBursty \rangle$, updated burstiness profile
2      if $r^t \notin BBG$ then
3          $BBG.add(r^t = (v_i^t, v_j^t, \tau^t))$
4      if $unqt \ni \tau^t$ then
5          $currentB++$
6          $isBursty \leftarrow (currentB > \bar{B}$ & $currentB > maxB)$
7      else
8          $\bar{B} \leftarrow (\bar{B} \times |unqt| + currentB)/(|unqt| + 1)$
9          $currentB \leftarrow 1$
10     if $currentB > maxB$ then
11         $maxB \leftarrow currentB$
12     return $\langle currentB, \bar{B}, maxB, isBursty \rangle$
```

Algorithm 10: Project W_{BBG} to W_{UWGO}

```
1  Function project($iVertexConnection, unqt, BBG, UWGO, b2v$)
       Input:
       $iVertexConnection$, a boolean flag
       $unqt$, a hash set of seen unique timestamps
       $BBG = (V_i \cup V_j, E_{ij})$, the bipartite graph structure to capture arriving sgrs
       $UWGO = (V, E)$, the graph structure to maintain the system state
       $b2v$, A hash map of butterflies to UWGO vertices
       Output: $UWGO$, updated $UWGO$
2      $Butterflies \leftarrow \emptyset$, $jNeighbors \leftarrow \emptyset$, $vi_2s \leftarrow \emptyset$, $mergedList \leftarrow \emptyset$, $i2b \leftarrow \emptyset$,
        $j2b \leftarrow \emptyset$
3      forall $i_1 \in V_i$ do
4          $sorted \leftarrow sort(unqt)$
5          $sorted \leftarrow sorted[|unqt| - [|unqt|/4], .., |unqt| - 1]$
6          $youngJNeighbors \leftarrow \forall v_j \in N_j(i_1) : v_j.\tau \in sorted$
7          forall $index1 \in [1, |youngJNeighbors)|]$ do
8              $j_1 \leftarrow youngJNeighbors[index1]$
9              forall $index2 \in [index1 + 1, |youngJNeighbors)|]$ do
10                 $j_2 \leftarrow youngJNeighbors[index2]$
11                 $i_2vertices \leftarrow N_i(j_1) \cap N_i(j_2)$
12                 forall $i_2 \in i_2vertices,\ i_2 \neq i_1$ do
13                     $\bowtie \leftarrow [i_1.ID, j_1.ID, i_2.ID, j_2.ID]$
14                     if $\bowtie \notin Butterflies$ then
15                         $Butterflies.add(\bowtie)$
16                         $updateUWGOstructure(\bowtie, UWGO, iVertexConnection, b2v)$
17     $i2b \leftarrow \emptyset$, $j2b \leftarrow \emptyset$, $mergedList \leftarrow \emptyset$, $BBG \leftarrow \emptyset$
18     forall $v \in V$ do
19         forall $n \in N(v)$ do
20             $\theta_v += n.hashCode()$
21         $\theta_v = \theta_v \ \% \ 2\pi$
22         v.setPhase($\theta_v$)
23         $\Omega_v \leftarrow$ sample from a Gaussian distribution
24         v.setFrequency($\Omega_v$)
25     return $UWGO$
```

Algorithm 11: Update the structure of UWGO

```
1  Function updateUWGOstructure($\bowtie, UWGO, iVertexConnection, b2v$)
       Input:
       $\bowtie$, a butterfly
       $iVertexConnection$, a boolean flag
       $UWGO = (V, E)$, the graph structure to maintain the system state
       $b2v$, A hash map of butterflies to UWGO vertices
2      $mergedList \leftarrow \emptyset$
3      $i2b.put(\langle i_1 : \bowtie\rangle, \langle i_2 : \bowtie\rangle)$
4      $j2b.put(\langle j_1 : \bowtie\rangle, \langle j_2 : \bowtie\rangle)$
5      $v \leftarrow$ new UWGO vertex with $\theta = 0$, $\Omega = 0$, $ID = 0$
6      $v.setID(v.hashCode())$
7      $UWGO.add(v)$
8      $b2v.put(\langle \bowtie, v\rangle)$
9      if $iVertexConnection$ then
               // add all butterflies adjacent to $i_1$ and $i_2$ to mergedList
10         $mergedList.addAll(i2b.get(i_1.ID))$
11         $mergedList.addAll(i2b.get(i_2.ID))$
12     else
               // add all butterflies adjacent to $j_1$ and $j_2$ to mergedList
13         $mergedList.addAll(j2b.get(j_1.ID))$
14         $mergedList.addAll(j2b.get(j_2.ID))$
15     forall $\bowtie \in mergedList$ do
16         $u \leftarrow b2v.get(\bowtie)$
17         if $v \neq u$ then
18             $e \leftarrow$ new UWGO edge between $u$ and $v$ with weight=$|mergedList|$
19             $UWGO.add(e)$
```

6.3.4 Drift Detection

The sequential procedures of drift criteria and drift evaluation sub-modules are as follows (steps 3-7 and yellow boxes in Figure 6.4).

Drift Criteria. The degree of global synchronization (phase coherence) is calculated as a drift criterion (Algorithm 8, lines 2-5). The intuition is that, a UWGO vertex embeds a butterfly in BBG and its phase embeds the neighbourhood of the vertex. Therefore, global phase synchronization of UWGO vertices reflects the neighbourhood similarities in UWGO, which in turn reflects the density of butterfly inter-connections in BBG. The global phase synchronization in UWGO at time point t is quantified using a metric called order parameter [280]:

$$O[t] = ((\sum_{v \in V} sin\theta_v)^2 + (\sum_{v \in V} cos\theta_v)^2)^{\frac{1}{2}} / |V|$$

$0 \leq O[t] \leq 1$, the higher the value of $O[t]$, the greater the degree of synchronization. $O[t] = 1$ denotes a phase synchrony state where all of the vertices have a same phase. Two sequences of the order parameter, O_1 and O_2, as the current and future state of the transient concept are recorded over time:

3) The order parameter is first computed over UWGO's structure and phases as $O_1[t]$ (Algorithm 8, line 2). The phases in Figure 6.5(d) would result in $0_1[t] = 0.17$.

4) The next phases in UWGO ($\Theta' = \{\theta'_v\}$) are estimated. To this end, Kuramoto model [280] is used. This model provides a mathematically tractable and simple realization of dynamic process of phase oscillation on complex networks. Given the current/initial phases in the network $\Theta = \{\theta_v\}$, edge weights $\{w_{vn}\}$, and frequencies $\{\Omega_v\}$, Kuramoto model provides the phase evolution of a vertex v as $\frac{d\theta_v}{dt} = \Omega_v + \sum_{n \in N(v)} w_{vn} sin(\theta_v - \theta_n)$. This ordinary differential equation is solved using Runge Kutta method with $h = 0.01$ (Algorithm 8, line 3).

$$\theta'_v = \theta_v + \frac{h}{6}(K(1)_v + 2K(2)_v + 2K(3)_v + K(4)_v)$$

$$K(1)_v = \Omega_v + \Sigma_{n \in N(v)} w_{vn} \sin(\theta_n - \theta_v)$$

$$K(2)_v = \Omega_v + \Sigma_{n \in N(v)} w_{vn} \sin(\theta_n + \frac{h}{2}K(1)_n - \theta_v - \frac{h}{2}K(1)_v)$$

$$K(3)_v = \Omega_v + \Sigma_{n \in N(v)} w_{vn} \sin(\theta_n + \frac{h}{2}K(2)_n - \theta_v - \frac{h}{2}K(2)_v)$$

$$K(4)_v = \Omega_v + \Sigma_{n \in N(v)} w_{vn} \sin(\theta_n + hK(3)_n - \theta_v - hK(3)_v)$$

The model results in the following phases for the example UWGO snapshot in Figure 6.5: $\theta'_1 = 0.12$, $\theta'_2 = -0.07$, $\theta'_3 = -0.14$, $\theta'_4 = -0.16$, $\theta'_5 = 0$, $\theta'_6 = -0.2$, $\theta'_7 = 0.05$, $\theta'_8 = -0.18$.

5) The order parameter is again computed as $O_2[t]$ over the same structure of UWGO, yet predicted phases $\Theta' = \{\theta'_v\}$ as the phases of oscillators (Algorithm 8, lines 4-5). The computed phases in the running example have $O_2[t] = 0.87$.

Drift Evaluation. A drift is detected by evaluating the trend of evolutions of O_1 and O_2 (Algorithm 8, line 9).

6) Two versions are proposed for Drift Evaluation sub-module: V1 (Algorithm 15) and V2 (Algorithm 16). Both algorithms signal a drift when two conditions C_1 and C_2 are satisfied.

   C_1: a local maximum/minimum is observed in O_2.

   C_2: O_1 remains steadily fixed.

   The algorithms differ in the way they check for C_1 and C_2. Algorithm V1 implements C_1 as $(10^\alpha \mu_1 - 10^\alpha O_1[t])/10^\alpha < 10^{-\alpha}$, where

   - μ_1 is the average of last S values of O_1,
   - $O_1[t]$ is the most recent value in O_1, and
   - α is a dynamic value used for determining (a) a threshold and (b) a precision for difference of μ_1 and $O_1[t]$.

   Algorithm V1 implements C_2 as ($Nmore \geq S'$ or $Nless \geq S'$), where

   - $Nmore$ and $Nless$ denote the number of elements in the most recent suffix of S elements in O_2 that are greater and less than $O_2[t]$, respectively, and
   - S' is a fraction of S.

   Soon after a drift occurs, the structure of streaming graph perturbs and O_1 and O_2 experience frequent fluctuations, therefore C_1 and C_2 should adapt to these perturbations through proper setting of S, S', and α.

- S is determined based on a function of the number of detections, d (Algorithm 15, line 2). S decreases slowly as d increases, therefore μ_1, $Nmore$, and $Nless$ are computed over smaller suffix sizes to pass the fluctuated values.
- S' decreases with a rate of 0.05 as the number of detections increases in order to make C_2 easier and avoid missing drifts.
- α is calculated as the current number of detections plus two; therefore, as detections increase, C_1 gets more difficult in order to avoid false detections.

Algorithm V2 implements C_1 similar to V1 except that the threshold is higher ($5 \times 10^{-\alpha}$) and μ_1 is calculated over S' elements. Also, C_2 is the same as in V1, however S is determined based on a function of average burst size, maximum burst size seen, as well as d (Algorithm 16, line 2). This function makes S increasing and decreasing by a factor depending on the size of the burst arrivals. Overall, these strategies aim reaching a balanced state between sensitivity and robustness.

7) Finally, the predicate-based window W_{UWGO} retires a fraction of its randomly selected vertices if the stream is identified as bursty (i.e. when a burst arrives whose size is greater than both the maximum seen burst size and the average burst size (Algorithm 8, line 10 invoking Algorithm 17). The fraction of vertices for removal is determined by a function of maximum seen burst size to adapt the window size (equivalently the number of vertices in UWGO) to the stream burstiness. Consequently the workload imposed to the data management and drift detection components in the next round would be balanced across the timeline of burst arrivals.

6.4 Performance Evaluation

In the following, the effectiveness and efficiency of *sGradd* are tested from three perspectives:

- Accuracy (Sections 6.4.1 and 6.4.3). The ability of *sGradd* to effectively detect concept drifts is examined under different drift patterns and drift intervals.
- Latency (Section 6.4.2 and 6.4.3). The detection latency is examined under different drift patterns and drift intervals.
- Complexity analysis (Section 6.4.5). The computational complexity of *sGradd* is analyzed theoretically.

Algorithm 12: Drift Criteria

```
1 Function DriftCriteria(UWGO, t)
      Input:
      $UWGO = (V, E)$, the graph structure to maintain the system state
      $t$, current time step
      Output:
      $\langle O_1, O_2 \rangle$, updated sequences of order parameters for observed and estimated
      phases
2     $O_1[t] \leftarrow computeOrderParameter(UWGO)$
3     $\Theta' \leftarrow Kuramoto(UWGO, 0.01)$
4     $UWGO.setInitialPhases(\Theta')$
5     $O_2[t] \leftarrow computeOrderParameter(UWGO)$
6     return $\langle O_1, O_2 \rangle$
```

Algorithm 13: Compute Order Parameter

```
  Input:
  $UWGO = (V, E)$, the graph structure to maintain the system state
  Output: $o$, order parameter value
1 Function computeOrderParameter(UWGO)
2     forall $v \in V$ do
3         $sum_c += cosine(\theta_v)$
4         $sum_s += sine(\theta_v)$
5     $o = (sum_c^2 + sum_s^2)^{\frac{1}{2}}$
6     return $o$
```

Data. Simulated streaming graphs with ground truth about drift time and pattern are used in the experiments. While *sGradd* works over unbounded stream length, for the purpose of evaluations, synthetic streams with size $|\mathfrak{R}| = 10^6$ are used. Streams are generated by *sGrow*, given a prefix of 1000 sgrs from Amazon user-item stream. *sGrow* as a configurable model enables generating sgrs through adding bursts such that the streaming graph reproduces realistic subgraph emergence patterns. Therefore, it enables simulating a drift in a hidden context (generative process) rather than an explicit drift in the target concept (subgraph inter-connectivity patterns).

A drift is introduced to the generative process by increasing/decreasing two parame-

Algorithm 14: Compute the Kuramoto phases via Runge Kutta method

```
1  Function Kuramoto(UWGO, h)
       Input:
       $UWGO = (V, E)$, the graph structure to maintain the system state
       $h$, a numerical parameter
       Output: $\Theta'$, set of computed phases
2      forall $v \in V$ do
3          $sum \leftarrow 0$
4          forall $n \in N(v)$ do
5              $sum += w_{nv} sin(\theta_u - \theta_n)$
6          $K1_v \leftarrow \Omega_v + sum$
7      forall $v \in V$ do
8          $sum \leftarrow 0$
9          forall $n \in N(v)$ do
10             $sum += w_{nv} sin(\theta_u + \frac{h}{2}K1_v - \theta_n - \frac{h}{2}K1_n)$
11         $K2_v \leftarrow \Omega_v + sum$
12     forall $v \in V$ do
13         $sum \leftarrow 0$
14         forall $n \in N(v)$ do
15             $sum += w_{nv} sin(\theta_u + \frac{h}{2}K2_v - \theta_n - \frac{h}{2}K2_n)$
16         $K3_v \leftarrow \Omega_v + sum$
17     forall $v \in V$ do
18         $sum \leftarrow 0$
19         forall $n \in N(v)$ do
20             $sum += w_{nv} sin(\theta_u + hK3_v - \theta_n - hK3_n)$
21         $K4_v \leftarrow \Omega_v + sum$
22     forall $v \in V$ do
23         $\theta'_v \leftarrow \theta_v + \frac{h}{6}(K1_v + 2K2_v + 2K3_v + K4_v)$
24         $\Theta'.add(\theta'_v)$
25     return $\Theta'$
```

ters of *sGrow* that contribute the most to the emergence of butterflies and caterpillars: $[L_{min}, L_{max}]$ (range of preferential random walk dynamic lengths), and ρ (burst connection

Algorithm 15: Drift Detection V1

```
1  Function DetectDriftv1($O_1, O_2, d, t, winNum$)
       Input: $O_1$, sequence of order parameter values for observed phases
       $O_2$, sequence of order parameter values for estimated phases
       $d$, number of detections
       $t$, current timestep, index of the last ingested sgr
       $winNum$, current window number
       Output: Drift event with description
2      $S \leftarrow 500(d+2)/(d+1)$
3      $S' \leftarrow (1-0.05d)S$
4      $\mu_1 \leftarrow$ mean of last $S$ values in $O_1$
5      $Nmore \leftarrow$ number of elements among the last $S$ elements of $O_2$ which are
        greater than $O_2[t]$
6      $Nless \leftarrow$ number of elements among the last $S$ elements of $O_2$ which are less
        than $O_2[t]$
7      $\alpha \leftarrow d+2$
8      if $(|[(10^{\alpha}\mu_1 - 10^{\alpha}O_1[t])]|/10^{\alpha}) < 10^{-\alpha})$ & $(Nmore \geq S'$ or $Nless \geq S')$ then
9          Signal a drift at sgr index $\leq t$ and window $winNum$
10         $d{+}{+}$
11         return $d$
```

probability). Two other parameters of the model (window parameter β and batch size M) are fixed. Parameters are set as follows:

- M and β can be set to any user-specified value without affecting the patterns of generated stream. Therefore, the default values $\beta = 5$ and $M = 10$ are used in the experiments.
- The default value for ρ is 0.3 and for $[L_{min}, L_{max}]$ is $[1, 2]$. Increasing ρ to values less than 0.7 and expanding the range of $[L_{min}, L_{max}]$ ensure preserving butterfly emergence patterns, while decreasing the generation time and increasing burst size. Therefore, $\rho = 0.4$ and $[L_{min}, L_{max}] = [1, 4]$ are used as initial values to reduce the generation time while preserving realistic patterns and leaving room for increment during drift simulation.

For gradual CD (Figure 6.6(a)), parameters switch as follows:

Algorithm 16: Drift Detection V2

```
1  Function DetectDriftv2($\bar{B}, maxB, O_1, O_2, d, t, winNum$)
       Input: $\bar{B}$, average burst size
       $maxB$, Maximum seen burst size
       $O_1$, sequence of order parameter values for observed phases
       $O_2$, sequence of order parameter values for estimated phases
       $d$, number of detections
       $t$, current timestep, index of the last ingested sgr
       $winNum$, current window number
       Output: Drift event with description
2      $S \leftarrow 1000(\frac{\lfloor log_{10}(Max(maxB,100)) \rfloor}{\lfloor log_{10}(Max(\bar{B},10)) \rfloor})^{(-1)^d}$
3      $S' \leftarrow (1 - 0.05d)S$
4      $\mu_1 \leftarrow$ mean of last $S'$ values in $O_1$
5      $Nmore \leftarrow$ number of elements among the last $S$ elements of $O_2$ which are
        greater than $O_2[t]$
6      $Nless \leftarrow$ number of elements among the last $S$ elements of $O_2$ which are less
        than $O_2[t]$
7      $\alpha \leftarrow d + 2$
8      if $(|[(10^{\alpha}\mu_1 - 10^{\alpha}O_1[t])]|/10^{\alpha}) < 5 \times 10^{-\alpha})$ & $(Nmore \geq S'$ or $Nless \geq S')$
        then
9          Signal a drift at sgr index $\leq t$ and window $winNum$
10         $d{+}{+}$
11         return $d$
```

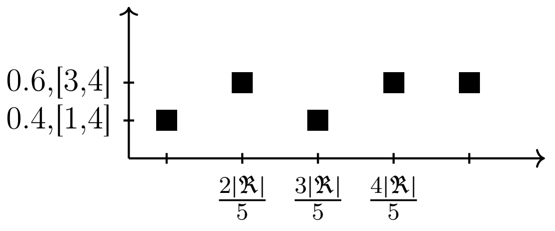
- $[L_{min}, L_{max}] = [1, 4]$ and $\rho = 0.3$ increase to $[3, 4]$ and 0.6 when $2/5|\mathfrak{R}|$ or $4/5|\mathfrak{R}|$ sgrs are generated.
- $[L_{min}, L_{max}] = [3, 4]$ and $\rho = 0.6$ decrease to $[1, 4]$ and 0.3 when $3/5|\mathfrak{R}|$ sgrs are generated.

For recurring CD (Figure 6.6(b)), parameters switch as follows:

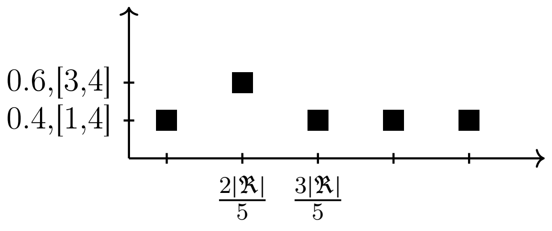
- $[L_{min}, L_{max}] = [1, 4]$ and $\rho = 0.3$ increase to $[3, 4]$ and 0.6 when $2/5|\mathfrak{R}|$ sgrs are generated.
- $[L_{min}, L_{max}] = [3, 4]$ and $\rho = 0.6$ decrease to $[1, 4]$ and 0.3 when $3/5|\mathfrak{R}|$ sgrs are generated.

Algorithm 17: Slide

```
1 Function slide(UWGO)
      Input:
      $UWGO = (V, E)$, the graph structure to maintain the system state
      $k$, current window number
      Output:
      $\langle UWGO,k\rangle$, updated system state
2     if $isBursty$ & $UWGO \neq \emptyset$ then
3         Randomly remove $0.1log_2maxB$ of vertices in $V$
4     $winNum++$
5 return $\langle UWGO,k\rangle$
```



(a) Gradual CD



(b) Reoccurring CD

Figure 6.6: Evolution of *sGrow*'s parameters over the timeline of sgr generation.

Drift intervals of 1×10^5 and 2×10^5 sgrs are used. The timestamp at which the drift is introduced is recorded for the evaluations. Five stream instances are generated per pattern per drift interval for a total of 20 streams. The streams are denoted as R_{ab} and G_{ab}, where

- R refers to recurring drifts.
- G refers to gradual drifts.
- $a = 1$ refers to a stream with drift interval of 1×10^5 sgrs (called close-drift stream).
- $a = 2$ refers to a stream with drift interval of 2×10^5 sgrs (called far-drift stream).
- $b \in \{1, 2, 3, 4, 5\}$ refers to the stream instance number.

Butterflies are connected through shared j-vertices. $N_b = 1$ is fixed in all experiments since the data streams are bursty.

Metrics. Two variants of V2 are considered, one with decreasing rate of 0.05 for S′ (V2 0.05) and the other with 0.1 (V2 0.1) to evaluate the impact of this value.

The effectiveness of Algorithms V1 and V2 is evaluated with respect to accuracy. The accuracy is evaluated using three rates computed over 50 executions of *sGradd* over each data stream:

- True detection rate (T): the fraction of detections that are true (precede a drift within less than 30 seconds or the first detection following a drift).
- False detection rate (F): the fraction of detections which duplicate a true detection (F^{dup}) or fraction of detections which occur early in the stream before any drift happens (F^{early}).
- Miss detection rate (M): the fraction of drifts that are not detected.

$T_i/F_i^{dup}/M_i$ denote the true/false/miss detection rates of the i-th drift, that is the fraction is calculated over the detections corresponding to i-th drift. $T/F/M$ denote the overall rates, that is the fraction is calculated over all detections. Therefore, the following hold:

$$\begin{aligned} T_i + F^{early} + F_i^{dup} &= 1, i = 1 \\ T_i + F_i^{dup} &= 1, i \neq 1 \\ T + F &= 1 \end{aligned}$$

The efficiency of V1 and V2 is evaluated by measuring the detection delay, which is defined as the elapsed time (in milliseconds) between a true detection and a drift. Average detection delays are calculated over 50 execution instances of algorithms. Delay-i denotes the average detection delay for i-th drift. The reported numbers for latency and accuracy correspond to the same execution of algorithms.

When the drifts are closer to each other (i.e. in $R_{11} - R_{15}$ and $G_{11} - G_{15}$), there is a concern about the evaluation of both accuracy and latency since the detection can be delayed to a time point after the next drifts [173]. In such situations, we cannot be certain whether a detection is a duplicate false detection or it is a delayed detection corresponding to previous drifts. In another view, we cannot be certain whether a detection is missed or delayed to a time after subsequent drifts. To address this concern, the accuracy rates and latency of the sequential drifts are evaluated simultaneously for $R_{11} - R_{15}$ and $G_{11} - G_{15}$ and individually for $R_{21} - R_{25}$ and $G_{21} - G_{25}$.

Computing setup. Experiments are conducted on a machine with 15.6 GB native memory and Intel Core $i7 - 6770HQCPU@2.60GHz * 8$ processor. All algorithms are implemented in Java (OpenJDK 17.0.5).

6.4.1 Accuracy over far-drift streams

Table 6.2: True detection rates of V1 and V2.
Bold numbers in each row denote the top values of T_i/T among three versions.

	V1				V2 0.05				V2 0.1			
	T_1	T_2	T_3	T	T_1	T_2	T_3	T	T_1	T_2	T_3	T
R_{11}	**0.94**	0.26		0.32	**1**	0.36		0.36	0.55	**0.48**		**0.49**
R_{12}	0.5	0.26		0.27	**0.56**	0.65		**0.6**	0.4	**0.87**		0.48
R_{13}	**0.48**	0.4		0.43	0.42	0.77		**0.53**	0.37	**0.96**		0.46
R_{14}	**0.57**	0.3		0.36	0.43	0.64		0.52	0.41	**0.88**		**0.59**
R_{15}	0.53	0.01		0.23	0.54	**0.64**		0.59	**0.6**	0.62		**0.61**
R_{21}	**0.61**	**0.88**		**0.7**	0.39	0.72		0.49	0.33	0.25		0.38
R_{22}	**0.8**	0.75		**0.77**	0.46	0.62		0.53	0.37	**0.76**		0.44
R_{23}	**0.82**	0.32		0.38	0.5	0.68		**0.57**	0.35	**0.81**		0.43
R_{24}	**0.79**	0.39		**0.5**	0.47	0.28		0.37	0.33	**0.96**		0.42
R_{25}	**0.55**	0.49		0.51	0.46	0.61		**0.52**	0.38	**0.95**		0.46
G_{11}	**0.91**	**1**	**0.91**	**0.91**	**0.92**	0	0.82	0.86	0.85	0	0.53	0.61
G_{12}	0.63	0.8	0.79	0.72	**0.67**	**1**	**0.87**	**0.8**	0.59	**1**	**0.88**	0.75
G_{13}	**0.63**	0.69	0.85	0.69	0.56	**1**	**0.93**	**0.72**	0.58	**1**	0.67	0.68
G_{14}	0.68	0	0.7	0.78	**0.88**	**1**	**0.74**	**0.84**	0.68	0.93	0.58	0.78
G_{15}	0.79	0.8	**1**	0.83	**0.9**	**1**	**1**	**0.95**	0.77	**1**	0.87	0.85
G_{21}	**0.64**	0.64	0.88	**0.69**	0.46	0.75	0.9	0.61	0.35	**0.76**	**0.91**	0.48
G_{22}	**0.89**	0.45	0.75	0.54	0.54	0.71	0.8	**0.65**	0.42	**0.75**	**0.84**	0.55
G_{23}	**0.69**	0.44	**1**	0.51	0.6	**0.91**	0.9	**0.8**	0.59	0.88	0.71	0.67
G_{24}	0.71	0.56	**0.9**	0.61	**0.78**	0.6	0.83	**0.7**	0.54	**0.7**	**0.96**	0.66
G_{25}	**1**	0.5	0.62	0.58	0.83	**0.88**	**0.95**	**0.87**	0.6	0.76	**1**	0.67

Table 6.3: False detection rates of V1 and V2.
Bold numbers in each row denote the lowest values of F_i/F among three versions.

	V1					V2 0.05					V2 0.1				
	F^{early}	F_1^{dup}	F_2^{dup}	F_3^{dup}	F	F^{early}	F_1^{dup}	F_2^{dup}	F_3^{dup}	F	F^{early}	F_1^{dup}	F_2^{dup}	F_3^{dup}	F
R_{11}	**0**	0.06	0.74		0.68	**0**	**0**	0.64		0.64	**0**	0.44	**0.52**		**0.51**
R_{12}	**0**	0.5	0.74		0.73	**0**	**0.44**	0.35		**0.4**	**0**	0.6	**0.13**		0.52
R_{13}	0.15	**0.37**	0.6		0.57	**0**	0.58	0.23		**0.47**	**0**	0.63	**0.04**		0.54
R_{14}	0.31	**0.12**	0.54		0.64	**0.18**	0.36	0.48		**0.08**	0.32	0.27	**0.12**		0.23
R_{15}	0.16	**0.31**	0.99		0.77	**0**	0.46	**0.36**		0.41	**0**	0.4	0.38		**0.39**
R_{21}	0.13	**0.39**	0.12		0.3	**0**	0.61	0.28		0.51	**0**	0.67	**0.07**		0.62
R_{22}	0.08	**0.12**	0.25		**0.23**	**0.02**	0.52	0.38		0.37	0.05	0.58	**0.24**		0.56
R_{23}	0.18	0	0.69		0.62	**0.01**	0.49	0.32		0.43	**0.03**	0.62	0.19		0.57
R_{24}	**0.04**	**0.17**	0.61		**0.5**	**0**	0.52	0.71		0.63	**0.01**	0.66	**0.04**		0.58
R_{25}	0.08	**0.37**	0.51		0.49	**0**	0.54	0.39		**0.48**	**0**	0.62	**0.05**		0.54
G_{11}	**0**	**0.09**	**0**	0.09	0.09	**0**	**0.08**	**0**	**0.18**	0.14	**0**	0.15	**0**	0.47	0.39
G_{12}	**0**	0.37	0.2	0.21	0.28	**0**	**0.33**	**0**	**0.12**	0.2	**0**	0.4	**0**	**0.12**	0.25
G_{13}	0.08	**0.29**	0.31	0.15	0.31	**0.05**	0.39	**0**	**0.06**	0.28	0.11	0.31	**0**	0.33	0.32
G_{14}	0.2	**0.14**	**0**	0.3	0.22	**0**	**0.12**	**0**	**0.26**	**0.16**	**0**	0.32	0.07	0.42	0.32
G_{15}	0.14	**0.07**	0.2	**0**	0.17	**0**	0.1	**0**	**0**	**0.05**	**0**	0.23	**0**	0.12	0.15
G_{21}	0.26	**0.1**	0.36	**0.12**	**0.31**	**0.04**	0.5	**0.25**	**0.1**	0.39	0.12	0.53	**0.24**	**0.09**	0.52
G_{22}	0.11	**0**	0.55	0.25	0.46	**0**	0.46	0.29	0.2	**0.35**	**0**	0.58	**0.25**	**0.16**	0.45
G_{23}	0.25	**0.06**	0.56	0	0.49	**0**	0.4	**0.09**	0.1	0.2	**0**	0.41	**0.12**	0.29	0.33
G_{24}	0.24	**0.05**	0.44	0.1	0.39	**0**	0.22	0.4	0.17	**0.3**	**0**	0.46	**0.3**	**0.04**	0.34
G_{25}	**0**	0.17	**0.12**	**0.05**	**0.13**	**0**	**0**	0.5	0.38	0.42	**0**	0.4	0.24	**0**	0.33

According to Tables 6.2, 6.3, and 6.4, V1 has better accuracy in detecting the first drift, while V2 outperforms V1 in detecting the next drifts.

- True/False detection rate. V1 has a higher true detection rate for the first drift due to lower duplicate false detections (higher T_1 and lower F_1^{dup} on average and in all data streams except for G_{25} - higher T on average and in all data streams except for R_{23} and R_{25}). In detecting the next drifts, V2 displays a better true detection rate especially when S' decreases with a faster rate of 0.1 (higher T_2 on average and in all data streams except for R_{21}, higher T_3 on average and in all data streams with gradual CD except for G_{23}). V2 has almost zero early false detection, while V1 has a 10% rate.

- Miss detection rate. V2, especially with decreasing rate of 0.05 for S', has better miss detection rate for each drift (lower M_1, M_2, and M_3) and overall (lower M) in

Table 6.4: Miss detection rates of V1 and V2.
Bold numbers in each row denote the lowest value of M_i/M among three versions.

	V1				V2 0.05				V2 0.1			
	M_1	M_2	M_3	M	M_1	M_2	M_3	M	M_1	M_2	M_3	M
R_{11}	**0.68**	**0**		**0.34**	0.98	**0**		0.49	0.9	**0.02**		0.46
R_{12}	0.9	**0**		0.45	**0.02**	0.12		**0.07**	**0**	0.6		0.3
R_{13}	0.2	0.02		0.11	**0**	0.08		0.04	**0**	0.52		0.26
R_{14}	0.46	**0**		0.23	0.08	0.06		**0.08**	**0.04**	0.42		0.23
R_{15}	0.38	**0.02**		0.18	**0.08**	**0**		0.04	0.12	0.08		0.1
R_{21}	0.08	0.3		0.19	**0**	**0.22**		**0.11**	**0**	0.74		0.37
R_{22}	**0.04**	0.16		0.1	0.1	**0.08**		0.09	**0.04**	0.56		0.3
R_{23}	0.34	**0**		0.32	**0.04**	0.08		0.06	**0.02**	0.58		0.3
R_{24}	0.24	**0.02**		0.13	**0**	0.34		0.17	**0**	0.5		0.25
R_{25}	0.34	0.16		0.25	**0**	**0.12**		0.06	**0**	0.6		0.3
G_{11}	0.58	**0.98**	0.8	0.79	**0.52**	1	0.36	0.63	0.56	1	**0.2**	**0.59**
G_{12}	0.62	0.92	0.48	0.67	**0.02**	**0.32**	0.44	0.26	**0**	0.46	**0.12**	**0.19**
G_{13}	0.52	0.82	0.78	0.71	0.12	0.62	0.42	0.39	**0.08**	**0.52**	**0.22**	**0.27**
G_{14}	0.4	1	0.54	0.65	**0**	0.58	0.2	0.26	**0**	**0.44**	**0.06**	**0.17**
G_{15}	0.78	0.92	0.9	0.87	**0.1**	**0.22**	0.94	**0.42**	0.18	0.46	**0.86**	0.5
G_{21}	0.5	0.36	0.56	0.47	**0.06**	**0.22**	**0.48**	**0.25**	0.1	0.56	0.62	0.43
G_{22}	0.84	**0.1**	0.64	0.53	**0.1**	0.3	**0.44**	**0.28**	**0.1**	0.58	0.48	0.39
G_{23}	0.78	**0.06**	0.84	0.56	**0.08**	0.42	**0.6**	**0.37**	0.1	0.42	0.5	0.39
G_{24}	0.7	0.22	0.64	0.52	0.28	**0.12**	0.62	0.34	**0.02**	0.22	**0.46**	**0.23**
G_{25}	0.68	**0.06**	0.74	0.49	**0.02**	0.14	**0.62**	**0.26**	**0.02**	0.3	0.88	0.4

all data streams (except for M_1 in R_{22} and M_2 in R_{23}, R_{24}, G_{22}, and G_{23}) and on average.

6.4.2 Detection Latency over far-drift streams

According to Table 6.5, average detection delay has the lowest value for the first drift and the highest values for the second drift in all data streams with either pattern. This is due to the change of burstiness profile in data stream as the bursts sizes get high with the increase of generator parameters in the interval between the first and second drift.

According to Figures 6.7, 6.8, 6.9, and 6.10, V2 has lower delay-1 and delay-2 and higher delay-3 compared to V1. Unlike V1, V2 has a consistent performance across the execution instances and across the sequential drifts since V2 is more robust to the random removal of vertices from the W_{UWGO}.

- Delay-1. For recurring CD, V1 has a cluster of delay-1 values between 0 to 4 (s) and a rest of values between 15 to 160 (s), while V2 clusters the values between 0 to 2 (s) and a rest of values around 16 (s). For Gradual CD, both V1 and V2 have cluster of delay-1 values between 1 to 5 (s). However V2 has a denser cluster due to lower miss rate for the first drift , thus the average delay-1 over the execution instances of V1 is up to 2.8× higher (e.g. see delay-1 for G_{11} and G_{13} in Table 6.5).
- Delay-2. For recurring CD, similar observations can be made as delay-1 stands. For instance, executing V1 over R_{25} incurs delay-2 of 0 to 400 (s), whereas V2 incurs latency below 100 (s) for this stream. For gradual CD, V1 has a cluster of delay-2 values between 0 to 200 (s) and values centered around 500 and 1400 (s) for G_{23} and G_{25}, while V2 has a cluster compacted around 0 to 100 (s) and values between 100 to 400 for G_{13} and 700 to 900 for G_{25}. The average of delay-2 values over the execution instances of V1 is up to 1.8× higher (e.g. see delay-2 for G_{25} in Table 6.5).
- Delay-3. For gradual CD, V1 displays 0 to 100 (s) and outliers up to 380 (s), while V2 has delay-3 values between 0 to 200 (s) and outliers up to 800 (s).

6.4.3 Accuracy and detection latency over close-drift streams

According to Tables 6.2, 6.3, 6.4, and 6.5, for recurring CD, similar to far-drift streams, V1 has higher latency compared to V2s and as a result the detections for the first and second drifts are closer and more difficult to evaluate. The (average) true detection rates are higher than those of running algorithms over $R_{21} - R_{25}$. For gradual CD, the true/miss detection rates and detection delay of both algorithms are higher than those of running algorithms over $G_{21} - G_{25}$.

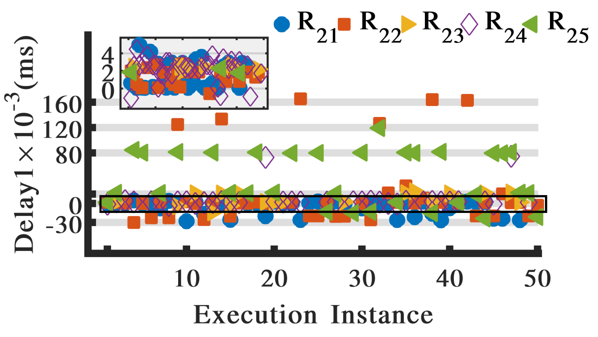
- First drift. For recurring CD, V1 has a higher true detection rate for the first drift compared to that of the second drift, however the rates are not very high ($T_1 > T > T_2$ and average T_1 is 0.6). The first drift is either never detected ($T_2 < T$, $F^{early} + F_1^{dup} < F$, and $M_1 > M$ in R_{11} and R_{12}) or detected with a delay higher than the drift interval ($M_2 \approx 0$, $M_1 > M$ in R_{13}, R_{14}, and R_{15}) with a low probability of rapid detection (delay-1 lower than average delay). V2 0.05 and V2 0.1 detect the first drift mostly with low miss rate ($M_1 \leq M$ in R_{12}, R_{13}, R_{14}, R_{15} – $T_2 < T$, $F^{early} + F_1^{dup} < F$, and $M_1 > M$ V2 0.05 and V2 0.1 over R_{11}) with high probability of rapid detection (delay-1 lower than average). For gradual CD, both V1 and V2 (especially V2 0.05) have high true detection rate and low false duplicate detection rate for the first drift (T_1 up to 92%). T_1 is high regardless of M_1 being high or low, which means T_1 is certainly high.

- Second drift. For recurring CD, while the second drift is rarely missed, it has a low true detection rate ($M_2 \approx 0$ and $T_2 < T$), since its detection is dominated by the false duplicate detections due to delayed detections of the previous drift. V2 0.05 and V2 0.1 detect the second drift with higher miss detection rate compared to that of the first drift (average $M_2 >$ average M_1), however the true detection rate is higher ($T_2 > T_1$ in all streams except for R_{11}). For gradual drift, both V1 and V2 (especially V2 0.05) extremely high/low true detection rate for the second drift (T_2 either zero or 100%). $T_2 > T$ accompanied with $F_3^{dup} < F$, $M_2 > M$, and delay-2 lower than the average delay suggests that whenever the second drift is detected with a delay lower than the time interval between the second and third drifts, the detection is certainly a true detection (V2's detections of the second drift in G_{12}, G_{13}, and G_{14} and V1's detection of the second drift in all streams). While there are cases that the detections are not delayed ($M_3 > M$, while $M_1/M_2 < M$, e.g. V2 0.05 and V2 0.1 over G_{15}), the detection of the second drift is sometimes delayed to a time after the occurrence of the next drift since $M_2 > M$ (M_2 is high), $M_3 < M$ (M_3 is low).

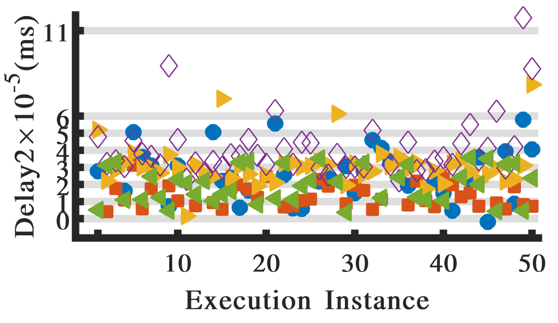
- Third drift. for gradual CD, both V1 and V2 (especially V2 0.05) have high true detection rate and low false duplicate detection rate for the third drift (T_3 up to 100%). Average delay of detecting the third drift is relatively low considering the burstiness of the stream and average delay over different streams (V2 0.05 and V2 0.1 over G_{13}). On the other hand, the miss detection rate in most streams follows the order of $M_1 < M_3 < M_2$ and V2 has better miss detection rates than V1.

6.4.4 Discussion

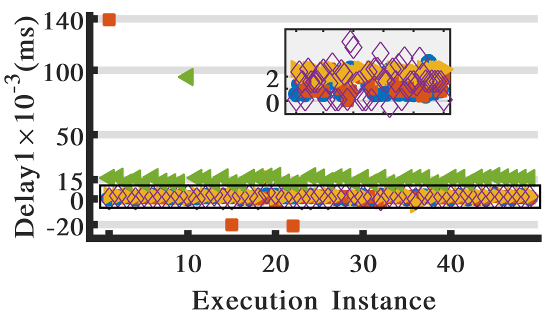
The high values of T_1 for V1's detections, as well as significantly increasing trend of T_1 for V2's detections, indicate that V1 has better true detection rate for the first drift, whereas



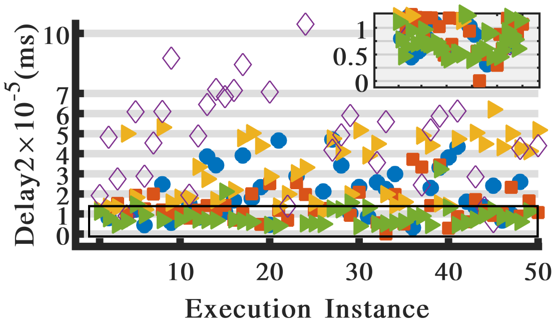
(a) Delay-1, V1



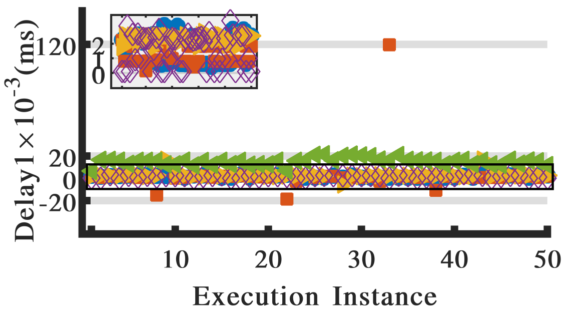
(b) Delay-2, V1



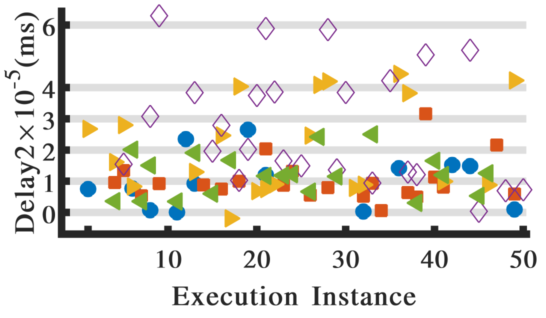
(c) Delay-1, V2 0.05



(d) Delay-2, V2 0.05

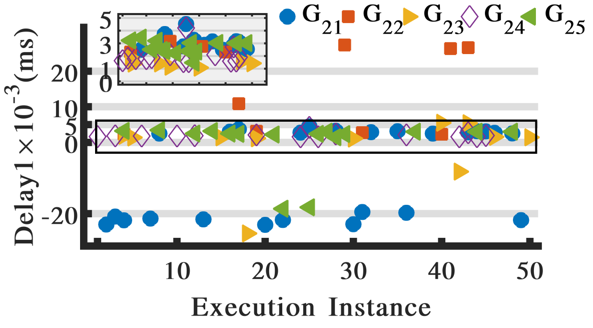


(e) Delay-1, V2 0.1

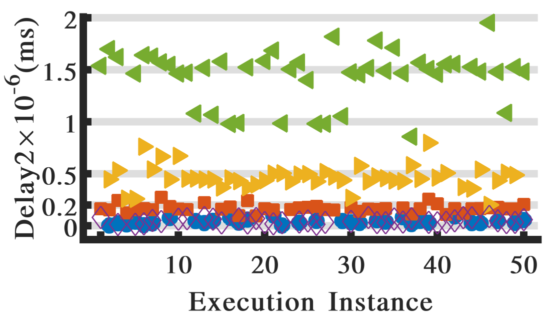


(f) Delay-2, V2 0.1

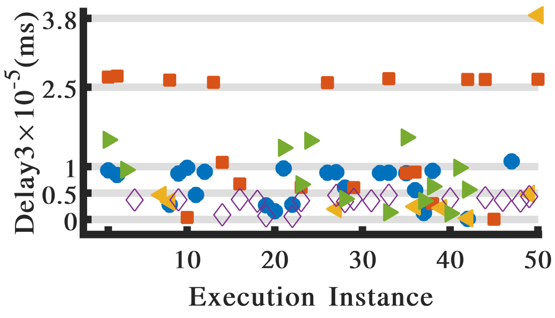
Figure 6.7: Delay of detecting the first and second drifts for 50 runs of V1, V2 0.05, and V2 0.1 over $R_{21} - R_{25}$.



(a) Delay-1

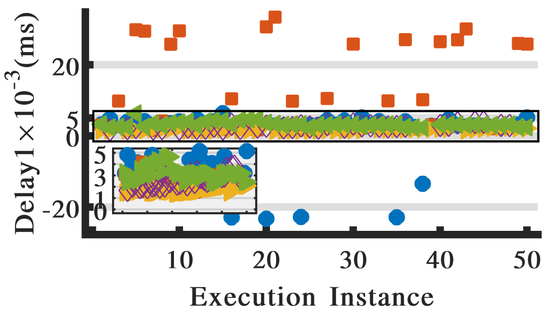


(b) Delay-2

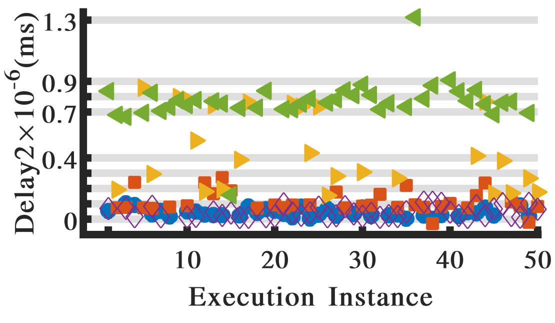


(c) Delay-3

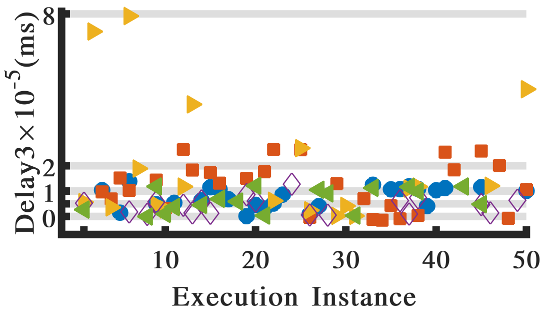
Figure 6.8: Delay of detecting the first, second, and third drifts for 50 runs of V1 over $G_{21} - G_{25}$.



(a) Delay-1

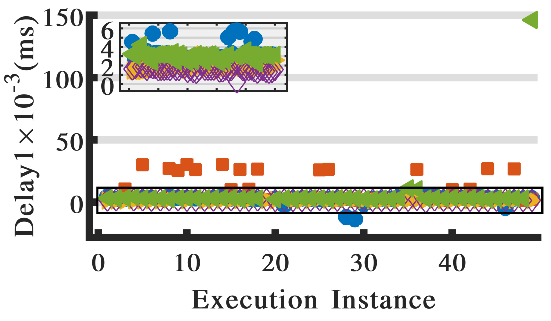


(b) Delay-2

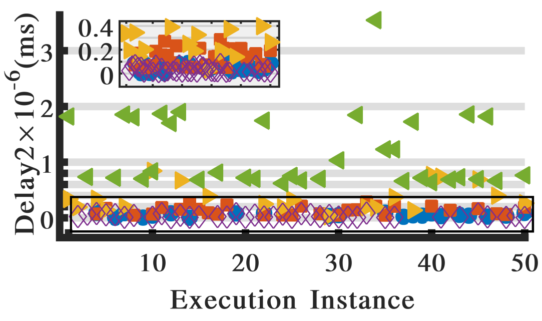


(c) Delay-3

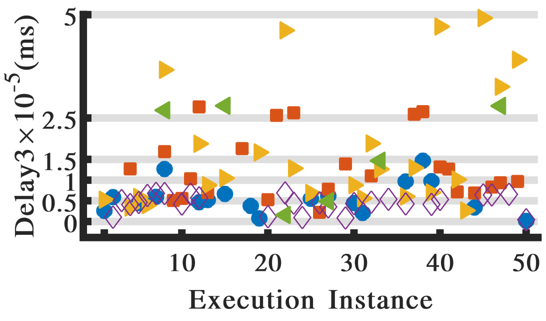
Figure 6.9: Delay of detecting the first, second, and third drifts for 50 runs of V2 0.05 over $G_{21} - G_{25}$.



(a) Delay-1



(b) Delay-2



(c) Delay-3

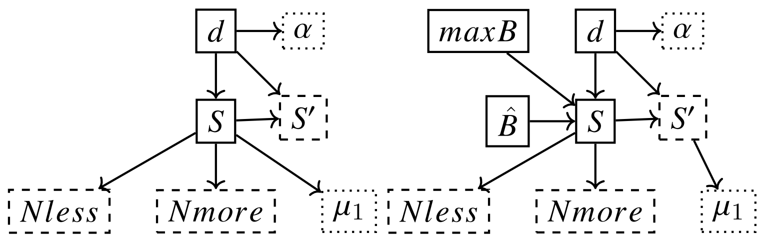
Figure 6.10: Delay of detecting the first, second, and third drifts for 50 runs of V2 0.1 over $G_{21} - G_{25}$.

Table 6.5: Detection latency of V1 and V2.

	V1			V2 0.05			V2 0.1		
	Delay-1	Delay-2	Delay-3	Delay-1	Delay-2	Delay-3	Delay-1	Delay-2	Delay-3
R_{11}	22253.94	540702.6		65451	624864.28		61605.4	611761.71	
R_{12}	12885.4	230963.56		2691.24	153758.73		2829.88	144740.85	
R_{13}	6442.05	387632.51		2672.9	244856.89		3119.18	124455.04	
R_{14}	6073.93	345451.08		4338.02	390906.19		7616.46	257873.07	
R_{15}	32094.67	291978.9		49691.87	186069.56		49207.61	130548.09	
AVG	15950	359345.73		24969	320091.13		24875.71	253875.75	
R_{21}	7904.56	263194.57		1595.76	213102.95		1646.28	102230.31	
R_{22}	23296.79	144621.17		47818.8	138562.91		4530.041	102385.82	
R_{23}	9527.39	301745.36		2667.17	298226.19		2823.408	214460.95	
R_{24}	6108.92	400390.28		1710.5	478270.94		1624.56	240721.36	
R_{25}	45147.09	171777.07		16367.92	87739.32		14983.04	120420.85	
AVG	18396.95	256345.69		14032.03	243180.36		5121.465	156043.86	
G_{11}	3065.39	254034	74091.64	6273.62	N/A	185651.43	6413.59	N/A	169774.77
G_{12}	36950.37	1525	113011.57	23050.69	28158.76	158805.28	18692.88	20054.96	138997.42
G_{13}	11139.46	77988.33	39305.18	5961.59	20913.79	80479.9	7109.22	13042.25	88383.59
G_{14}	12866.43	N/A	82805.56	12950.94	29216.95	90990.2	12289.02	21878.64	83301.49
G_{15}	5925.91	162395.5	78400.2	2382.42	142926.92	93514.67	2361.71	131756.78	115332.71
AVG	13900.4	117986.6	142771.42	50619.26	55304.1	121888.3	9102.36	37346.53	119158
G_{21}	11218.52	40456.31	66766.95	5292.89	42777.05	77775.42	3899.82	51292.14	57129.53
G_{22}	12731.37	169076.82	160692.11	10764.84	115692.66	122396.76	10842.87	186502.62	121164.73
G_{23}	4966	439955.25	70503.12	1933.37	481555.79	190583.1	1744.58	347943.43	166380.04
G_{24}	2041	46369.79	34844.17	2592.05	44344.66	43166.95	1527.61	29315.36	43271.15
G_{25}	4670.25	1445005.02	82159.38	3416.57	795168.6	71119.58	5959.61	939723.46	173239.5
AVG	7125.43	428172.64	82993.15	4799.94	295907.75	102135.55	4794.9	310955.4	112236.99

V2, particularly with decreasing rate of 0.1 for S', has improving true detection rate over time. The almost zero values of F^{early} as well as significantly decreasing trend of F_1^{dup} for both V1 and V2 indicate that V1 and V2 have very low false detection rates particularly false positives. The low range of M_1 for V2s' detections and significant negative magnitude of $M_2 - M_1$ for V1's detections indicate that that V1 has improving miss detection rate and V2s have better initial miss detection rates.

To explain the performance of V1 and V2, the impact of variables/parameters are discussed in the following (Figure 6.11). V1 adapts to the number of detections (through parameter d), while V2 adapts to the streaming rate (through parameters $maxB$ and $\hat{B}$) as well as the number of detections. Both algorithms signal a drift when two conditions are satisfied: C1 (including variable α and μ_1, dotted nodes in Figure 6.11) and C2 (including



(a) V1 (b) V2

Figure 6.11: Impact of algorithm parameters on drift detection.

variables $Nmore$, $Nless$, and S', dashed nodes in Figure 6.11).

As d increases over time, V1 signals more detections for each drift. This increases F_i^{dup} and lowers M_i and T_i. The reason is that C1 is the bottleneck performance of V1, which becomes more difficult over time as the followings happen.

- α increases and the upper bound of C1 is decreased.
- S and S' decrease in V1 and fluctuate in V2. Consequently, $Nmore$, $Nless$, and μ_1 are computed over smaller O_2 suffix sizes in V1 and smaller/longer suffixes in V2; Moreover, the fluctuations in O_2 values, which occur due to the drift, are not examined in V1.

As $maxB$ and $\bar{B}$ increase over time, V2 adapts to the streaming rate and signals fewer duplicate detections. This lowers F_i^{dup} and increases T_i. The reason is that the performance of V2 changes according to both C1 and C2 and C2 becomes more difficult/easier in V2 due to the followings.

- S and s' increase/decrease when d is even/odd.
- $Nmore$, $Nless$, and μ_1 are computed over smaller/longer O_2 suffixes.

6.4.5 Computational Complexity

THEOREM 4 *At each time step,* sGradd *has either of following computational complexities depending on the drift detection module used.*

$$O(\frac{K_{i,W_k}(K_{i,W_k}-1)}{2}K_{j,W_k}\mathcal{R}N_t^{W_k}) + O(d^2)$$

$$O(\frac{K_{i,W_k}(K_{i,W_k}-1)}{2}K_{j,W_k}\mathcal{R}N_t^{W_k}) + O((d+1)(maxB/\bar{B})^{(-1)^d})$$

*where $\mathcal{R}$ is the average stream rate, K_{i,W_k} (K_{j,W_k}) is the lower bound of degree of i(j)-vertices in W_k, **maxB** is the maximum seen burst size, $\bar{b}$ is the average burst size, and **d** is the number of drift detections.*

PROOF 5 *The computations of* sGradd *at each time step are dominated by the windowed analytics since data ingestion and window sliding have constant computational cost. The windowed analytics include projection to create/update UWGO, computing the drift criteria, and evaluating the drift criteria as described below.*

- *The function project(.) identifies the butterflies, maps them to connected phase oscillators (i.e. creates the structure of UWGO), and next assigns phases and frequencies to oscillators. Therefore, it takes $O(ButterflyListing)O(Mapping)+O(Assignments)$. To identify the butterflies, the exact listing algorithm in* sGrapp *is used, which is $O(\frac{K_{i,W_k}(K_{i,W_k}-1)}{2}K_{j,W_k}\mathcal{R}N_t^{W_k})$, where $\mathcal{R}$ is the average stream rate and K_{i,W_k} (K_{j,W_k}) is the lower bound of degree of i(j)-vertices in W_k (Theorem 2). To map the butterflies to unipartite vertices and connect them to create the UWGO's structure, the function updateUWGOstructure(.) is executed which takes constant units of computation. The final phase of attribute assignment also runs in constant units of computations as it happens in a parallel loop. Consequently, the projection occurs in $O(\frac{K_{i,W_k}(K_{i,W_k}-1)}{2}K_{j,W_k}\mathcal{R}N_t^{W_k})$.*

- *The function DriftCriteria(.) computes the order parameter twice: once using the current phases, and once using the Kuramoto phases computed via Runge Kutta method. Therefore, it takes $O(OrderParameter) + O(Kuramoto)$. The function computeOrderParameter(.) takes constant units of computations. The function Kuramoto(.) has five sequential parallel loops, with the first four loops including an inner parallel loop; Therefore, this function takes constant units of computation as well. Consequently, computing the drift criteria occurs in $O(c)$.*

- *Either of the functions DriftDetectv1(.) or DriftDetectv2(.) is used to evaluate the drift criteria and signal a drift. DriftDetectv1(.) takes $O(S)$ units of computation, where S, the suffix size for drift detection, is determined based on a function of d^2 (d is the number of detections). Therefore, it takes $O(d^2)$. DriftDetectv2(.) takes $O(S + S')$, where $S' \in O(S \times d)$ and $S \in O((maxB/\bar{B})^{(-1)^d})$; hence it takes $O((d+1)(maxB/\bar{B})^{(-1)^d})$. Consequently, drift detection occurs in $O(d^2)$ or $O((d+1)(maxB/\bar{B})^{(-1)^d})$.*

6.5 Summary

Essence and features. CD is a natural phenomenon in streaming data. Understanding, detection, and adapting to CD in streaming data is vital for effective and efficient analysis/analytics since reliable outputs are generated based on adaptation to fresh inputs. The problem is challenging due to the pressing issue of drift label acquisition and the fact that CD detection intrinsically requires stateful yet efficient operations as well as effective mechanisms for CD evaluations. Finally being able to detect and adapt to CD is impactful since a variety of practical use-case scenarios reside in streaming settings that experience CD. The research on this topic is mostly focused on (i) black-box drift adaptation, (ii) drift detection over independent data instances or sequences of graph snapshots, and/or (iii) drift detection with respect to a supervised downstream task which determines the target concept. These assumptions do not always hold.

To address the aforementioned issues, a definition is introduced for CD in the case of transient, interconnected, and sequential data instances forming a streaming graph which serves as the input to any online adaptive analytic task (in both supervised and unsupervised mode). Powered by this definition, a modular framework, *sGradd*, is proposed for understanding and unsupervised detection of transient concept drifts. The components of *sGradd* are designed as a collection of explainable, unsupervised, and adaptive techniques for understanding and detecting the drifts in hidden contexts (generative source) which are reflected as drifts in target transient concepts (inter-connectivity patterns). Specifically, *sGradd* has a data management component with two sub-components for window management and system state management. For window management, two different windowing methods are used, which are adaptive to the streaming rate. For system state management, a graph structure is used to retain the system state incrementally. For the drift detection component, two versions (V1 and V2) of an algorithm are introduced each incorporating a different adaptation technique for evaluating the occurrence of drifts. V1 and V2 are parameterized as described below.

Evaluations. *sGradd* is examined in different scenarios where the drift pattern, drift interval, and stream burstiness vary. Experiments show that while V1's drift detection strategies, which adapt to just the number of detections, display higher initial true detection rate and improvement (decreasing trend) of miss detection rate over time, V2's adaptation to the streaming rate (burstiness), as well as the number of detections, display improving true detection rate, better initial miss detection rate, and lower detection latency. Both strategies result in almost zero false positives in streams with either gradual or reoccurring drift patterns and are able to effectively lower the duplicate false detections down to almost zero. Detection latency in both algorithms increases with the increase of burstiness regardless of the drift detection's adaptation mechanism and the drift pattern. However, as the detection algorithm adapts to the streaming rate, the delay is lowered and the latency performance becomes more consistent. This occurs in spite of sliding window mechanisms that randomly remove vertices from the window while adapting the window to the streaming rate.

Chapter 7

Conclusions and Future Works

7.1 Conclusion

A wide variety of real-world datasets include inter-connected entities best modeled as a graph. The graph model treats both the entities (vertices) and relationships (edges) as first-class objects. The model can also describe several types of inter-dependencies among data records as well as their compositions. Management and processing of graphs have always been driven by the characteristics of the data and/or workloads (often specified by the applications). In most modern applications (e.g. user-item interactions), graphs are not static, but change over time. A particular type that is of interest is *streaming graphs*, an unbounded sequence of arriving graph data records (a timestamp, and a payload indicating the vertex/edge). The streaming graph model assumes that the graph emerges incrementally as data records arrive. The main characteristics are unknown stream length, unavailable full structure, non-stationary distributions of the underlying data snapshot, non-global updates, out-of-order arrivals, non-uniform inter-arrival times, and burstiness.

The focus of this thesis is data-driven algorithm design for explainable and interpretable analytics over streaming graphs with respect to (2,2)-bicliques (known as butterflies) in bipartite structures (Table 7.1). This is done in three steps (Table 7.1).

Step 1: Given the real-world streaming graphs as input, exploratory analysis is performed for in-depth understanding and discovery of the emergence principles in data.

Step 2: Real-world data and the discovered patterns are utilised to model and explain the growth patterns in streaming graphs as *sGrow* model. These two steps provide the requirements for ultimately designing effective and efficient processing algorithms in the next step (i.e. accomplish the data-driven approach).

Step 3: The mined patterns and *sGrow* model as well as the data are utilised to architect two frameworks *sGrapp* and *sGradd*.

These frameworks are designed to perform iterative and stateful tasks (butterfly counting and concept drift detection) over streaming graphs. The reason behind design choices of algorithms are explained and the outputs provide actionable descriptions. For instance, *sGrow*'s choices of target vertices for establishing new connections are explained via the introduced local rules; And *sGradd*'s alerts include details about time/location of drifts that can be used to improve the performance of a down-stream online algorithm.

	Input	Component	Output
Step 1	data	analysis	streaming graph emergence patterns
Step 2	data, patterns	modelling	*sGrow*: a streaming growth model
Step 3	data, patterns, *sGrow*	analytics	*sGrapp*: a framework for streaming graph butterfly-count approximation, *sGradd*: a framework for streaming graph drift detection

Table 7.1: Research components.

7.2 Future Work

Future directions of the research in this thesis can be either application-driven data mining tasks, or designing data-driven algorithms and frameworks based on updates to the data, to the methodology, or to the computational approaches considered in this thesis.

Application-driven data mining. In many real-world scenarios, a scheme such as the followings can be used to represent the snapshots of streaming data by a bipartite streaming graphs.

- When the payload in streaming records denotes interactions between two disjoint sets, a batch of streaming records form a naturally occurring bipartite streaming graph.

- When the payload in streaming records denotes interactions between two or more vertices of the same mode, a batch of streaming records form a unipartite or hyper-graph that can be projected to a bipartite structure. For instance, social networks inherit a backbone bipartite structure which can be extracted based on some heuristic data modeling approach; or the vertices and edges in hyper-graphs can be conceived as two connected disjoint sets; or property graphs include vertices/edges attributed with numeric/vectorized properties, and the property values and the corresponding vertices/edges can form two connected disjoint sets.
- When the payload in streaming records denotes a non-graph element such as items in spatial/sensory time series with an entity ID as well as a variable such as geographic location, the IDs and instances of variable can be represented as two connected disjoint sets.

Once the data is represented as a bipartite streaming graph, it can be analyzed to discover knowledge for practical purposes in the corresponding domain-specific context. In this stage, the butterfly-based analytics approaches proposed in this thesis can be applied and further extended. For instance, when there is a data mining task based on data cohesion, *sGrapp* can be utilised to effectively and efficiently estimate the clustering coefficient; or when there is an online adaptive data mining task, *sGradd* can enhance a down-stream adaptation algorithm.

Data-driven algorithm design. This thesis followed a data-driven approach for devising techniques and algorithms for graph modelling and graph analytics. This approach can be modified in the following directions to cover other use cases.

- The followed methodology includes exploratory data analysis to inform the design of pattern-driven data modelling/analytics. The exploratory phase can be performed to discover other patterns such as the following examples to introduce generative models and analytics tasks.
  - Temporal patterns for the generative mechanisms of timestamps can be studied by considering bipartite graphs of edge-timestamps.
  - Temporal-structural patterns for other types of subgraphs/connectivities can be studied by defining other assortativity measures. For example, a (timestamped) subgraph assortativity measure can be defined as the tendency of (close-timestamped) vertices to form a certain subgraph.

  - The connection between the probability of butterflies formed by hubs and the magnitude of the power law exponent can be investigated to further optimise *sGrapp*. Since this probability changes over windows, the exponent will dynamically change based on the hub connectivity patterns. As mentioned in [297], butterfly approximation can be highly accurate when the exponent equals to the probability of having at least one i-hub plus the probability of having at least one j-hub in the butterflies at a time when the number of hubs is stabilized in the graph. This tipping point can be studied to better understand the best way to initialize the approximation exponent.

- The computational approach can be modified based on other processing techniques (e.g. parallel/distributed settings, or windowing techniques) and/or storage setups (e.g. applying the algorithms to the data stored in a database, or optimising low-level main-memory data structures).

References

[1] Margareta Ackerman and Sanjoy Dasgupta. Incremental clustering: The case for extra clusters. In *Proc. 27th Annual Conf. on Neural Information Processing Systems*, pages 307–315, 2014.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. and Data Eng.*, 6:734–749, 2005.

[3] Charu C Aggarwal and Haixun Wang. Graph data management and mining: A survey of algorithms and applications. In *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 13–68. Springer, 2010.

[4] Supriya Agrahari and Anil Kumar Singh. Concept drift detection in data stream mining: A literature review. *J. King Saud Univ. Comput. Inf. Sci.*, 34(Issue 10, Part B):9523–9540, 2021.

[5] Yousuf Ahmad, Omar Khattab, Arsal Malik, Ahmad Musleh, Mohammad Hammoud, Mucahid Kutlu, Mostafa Shehata, and Tamer Elsayed. La3: a scalable link-and locality-aware linear algebra-based graph analytics system. *Proc. VLDB Endowment*, 11(8):920–933, 2018.

[6] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proc. 22nd Int. World Wide Web Conf.*, pages 37–48, 2013.

[7] Nesreen K Ahmed, Nick Duffield, Jennifer Neville, and Ramana Kompella. Graph sample and hold: A framework for big-graph analytics. In *Proc. 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 1446–1455, 2014.

[8] Rezwan Ahmed and George Karypis. Algorithms for mining the coevolving relational motifs in dynamic networks. *ACM Trans. Knowl. Discov. Data*, 10(1):1–31, 2015.

[9] Leman Akoglu and Christos Faloutsos. Rtg: A recursive realistic graph generator using random typing. In *Proc. European Conf. on Machine Learning and Knowledge Discovery in Databases*, pages 13–28, 2009.

[10] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Rtm: Laws and a recursive generator for weighted time-evolving graphs. In *Proc. 8th IEEE Int. Conf. on Data Mining*, pages 701–706, 2008.

[11] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Proc. Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 410–421, 2010.

[12] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*, 29(3):626–688, 2015.

[13] Sinan G Aksoy, Cliff Joslyn, Carlos Ortiz Marrero, Brenda Praggastis, and Emilie Purvine. Hypernetwork science via high-order hypergraph walks. *EPJ Data Science*, 9(1):16, 2020.

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

[15] Marie Al-Ghossein, Talel Abdessalem, and Anthony BARRÉ. A survey on stream-based recommender systems. *ACM Computing Surveys*, 54(5):1–36, 2021.

[16] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47, 2002.

[17] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

[18] Barabasi Albert-Laszlo. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207–211, 2005.

[19] Irfan Ali and Sang-Wook Kim. Group recommendations: Approaches and evaluation. In *Proc. 9th Int. Conf. Ubiquitous Information Management and Communication*, page 105, 2015.

[20] Sadegh Aliakbary, Jafar Habibi, and Ali Movaghar. Quantification and comparison of degree distributions in complex networks. In *Proc. 7th Int. Symp. on Telecommunications*, pages 464–469, 2014.

[21] Yasser Altowim, Dmitri V. Kalashnikov, and Sharad Mehrotra. Progressive approach to relational entity resolution. *Proc. VLDB Endowment*, 7:999–1010, 2014.

[22] Güneş Aluç, Olaf Hartig, M Tamer Özsu, and Khuzaima Daudjee. Diversified stress testing of rdf data management systems. In *Proc. 13th Int. Semantic Web Conf.*, pages 197–212, 2014.

[23] Sihem Amer-Yahia, Volker Markl, Alon Halevy, AnHai Doan, Gustavo Alonso, Donald Kossmann, and Gerhard Weikum. Databases and web 2.0 panel at vldb 2007. *ACM SIGMOD Rec.*, 37:49–52, 2008.

[24] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawlat, Gautam Das, and Cong Yu. Group recommendation: Semantics and efficiency. *Proc. VLDB Endowment*, 2:754–765, 2009.

[25] Khaled Ammar and M Tamer Özsu. Wgb: Towards a universal graph benchmark. In *Advancing Big Data Benchmarks*, pages 58–72. Springer, 2013.

[26] Arvind Arasu, Brian Babcock, Shivnath Babu, Jon McAlister, and Jennifer Widom. Characterizing memory requirements for queries over continuous data streams. *ACM Trans. Database Syst.*, 29(1):162–194, 2004.

[27] Shaikh Arifuzzaman, Maleq Khan, and Madhav Marathe. Patric: A parallel algorithm for counting triangles in massive networks. In *Proc. 22nd ACM Int. Conf. on Information and Knowledge Management*, pages 529–538, 2013.

[28] Shaikh Arifuzzaman, Maleq Khan, and Madhav Marathe. Fast parallel algorithms for counting and listing triangles in big graphs. *ACM Trans. Knowl. Discov. Data*, 14(1):1–34, 2019.

[29] Naomi A Arnold, Raul J Mondragón, and Richard G Clegg. Likelihood-based approach to discriminate mixtures of network models that vary in time. *Scientific reports*, 11(1):1–13, 2021.

[30] Kenneth J. Arrow. A difficulty in the concept of social welfare. *Journal of political economy*, 58:328–346, 1985.

[31] Arasu Arvind, Shivnath Babu, and Jennifer Widom. The cql continuous query language: Semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.

[32] Ion Madrazo Azpiazu, Michael Green, Oghenemaro Anuyah, and Maria Soledad Pera. Can we leverage rating patterns from traditional users to enhance recommendations for children? *CoRR*, abs/1808.08274, 2018.

[33] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, page 1–16, 2002.

[34] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proc. 21st ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems*, pages 1–16, 2002.

[35] Stephen H Bach and Marcus A Maloof. Paired learners for concept drift. In *Proc. 8th IEEE Int. Conf. on Data Mining*, pages 23–32, 2008.

[36] Linas Baltrunas, Tadas Makcinskas, and Francesco Ricci. Group recommendations with rank aggregation and collaborative filtering. In *Proc. 4th ACM Conf. on Recommender Systems*, pages 119–126, 2010.

[37] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 623–632, 2002.

[38] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[39] Michael J Barber. Modularity and community detection in bipartite networks. *Physical Review E*, 76(6):066102, 2007.

[40] Alain Barrat, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. The architecture of complex weighted networks. *Proc. National Academy of Sciences*, 101(11):3747–3752, 2004.

[41] Alain Barrat, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. Weighted evolving networks: coupling topology and weight dynamics. *Physical Review Letters*, 92(22):228701–228704, 2004.

[42] Alain Barrat, Marc Barthélemy, and Alessandro Vespignani. Modeling the evolution of weighted networks. *Physical Review E*, 70(6):066149, 2004.

[43] Peter L Bartlett, Shai Ben-David, and Sanjeev R Kulkarni. Learning changing concepts by exploiting the structure of change. *Machine Learning*, 41(2):153–174, 2000.

[44] Senjuti Basu Roy, Laks VS Lakshmanan, and Rui Liu. From group recommendations to group formation. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1603–1616, 2015.

[45] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 16–24, 2008.

[46] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3):1–28, 2010.

[47] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proc. 15th Annual Conf. on Neural Information Processing Systems*, pages 585–591, 2002.

[48] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1):151–175, 2010.

[49] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Proc. 19th Annual Conf. on Neural Information Processing Systems*, pages 137–144, 2006.

[50] Shai Ben-David and Ruth Urner. On the hardness of domain adaptation and the utility of unlabeled target samples. In *Proc. Int. Conf. on Algorithmic Learning Theory*, pages 139–153, 2012.

[51] Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

[52] Austin R Benson, Paul Liu, and Hao Yin. A simple bipartite graph projection model for clustering in networks. *arXiv preprint arXiv:2007.00761*, 2020.

[53] Suman K Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *Proc. 34th Symp. on Theoretical Aspects of Computer Science*, volume 66 of *LIPIcs*, pages 11:1–11:14, 2017.

[54] Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph convolutional matrix completion. *CoRR*, abs/1706.02263, 2017.

[55] Shlomo Berkovsky and Jill Freyne. Group-based recipe recommendations: analysis of data aggregation strategies. In *Proc. 4th ACM Conf. on Recommender Systems*, pages 111–118, 2010.

[56] Cesare Bernardis, Maurizio Ferrari Dacrema, and Paolo Cremonesi. A novel graph-based model for hybrid recommendations in cold-start scenarios. *CoRR*, abs/1808.10664, 2018.

[57] Massimo Bernaschi, Alessandro Celestini, Stefano Guarino, Flavio Lombardi, and Enrico Mastrostefano. Spiders like onions: on the network of tor hidden services. In *Proc. 28th Int. World Wide Web Conf.*, pages 105–115, 2019.

[58] Kanagal Bhargav, Amr Ahmed, Sandeep Pandey, Vanja Josifovski, Jeff Yuan, and Lluis Garcia-Pueyo. Supercharging recommender systems using taxonomies for learning user purchase behavior. *Proc. VLDB Endowment*, 10:956–967, 2012.

[59] Ginestra Bianconi and A-L Barabási. Competition and multiscaling m evolving networks. In *The Structure and Dynamics of Networks*, pages 361–367. IOP Publishing, 2011.

[60] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *Proc. 2007 SIAM Int. Conf. on Data Mining*, pages 443–448, 2007.

[61] Marcel Blattner, Yi-Cheng Zhang, and Sergei Maslov. Exploring an opinion network for taste prediction: An empirical study. *Physica A: Statistical Mechanics and its Applications*, 373:753–758, 2007.

[62] Stefano Boccaletti, Vito Latora, Yamir Moreno, Martin Chavez, and D-U Hwang. Complex networks: Structure and dynamics. *Physics reports*, 424(4-5):175–308, 2006.

[63] Angela Bonifati, George Fletcher, Jan Hidders, and Alexandru Iosup. A survey of benchmarks for graph-processing systems. In *Graph Data Management*, pages 163–186. Springer, 2018.

[64] Angela Bonifati, Irena Holubová, Arnau Prat-Pérez, and Sherif Sakr. Graph generators: State of the art and open challenges. *ACM Computing Surveys*, 53(2):1–30, 2020.

[65] Ludovico Boratto and Salvatore Carta. Modeling the preferences of a group of users detected by clustering: a group recommendation case-study. In *Proc. 4th Int. Conf. on Web Intelligence, Mining and Semantics*, page 16, 2014.

[66] Ludovico Boratto, Salvatore Carta, and Michele Satta. Groups identification and individual recommendations in group recommendation algorithms. In *Proc. the Workshop on the Practical Use of Recommender Systems, Algorithms and Technologies*, pages 27–34, 2010.

[67] Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Proc. 40th Int. Colloquium on Automata, Languages, and Programming*, pages 244–254, 2013.

[68] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. Motif counting beyond five nodes. *ACM Trans. Knowl. Discov. Data*, 12(4):1–25, 2018.

[69] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *Proc. 2nd Int. Conf. on Learning Representations*, 2014.

[70] Peter Brusilovsky and eds Daqing He, editors. *Social Information Access - Systems and Technologies*, volume 10100 of *Lecture Notes in Computer Science*. Springer, 2018.

[71] Yingyi Bu, Vinayak Borkar, Jianfeng Jia, Michael J Carey, and Tyson Condie. Pregelix: Big (ger) graph analytics on a dataflow engine. *Proc. VLDB Endowment*, 8(2):161–172, 2014.

[72] Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. Triangle counting in dynamic graph streams. *Algorithmica*, 76(1):259–278, 2016.

[73] Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 253–262, 2006.

[74] Luciana S Buriol, Gereon Frahling, Stefano Leonardi, and Christian Sohler. Estimating clustering indexes in data streams. In *Proc. European Symposium on Algorithms*, pages 618–632, 2007.

[75] Francis A Buttle. Word of mouth: understanding and managing referral marketing. *Journal of Strategic Marketing*, 6(3):241–254, 1998.

[76] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Trans. Knowl. and Data Eng.*, 30(9):1616–1637, 2018.

[77] Zhuhua Cai, Dionysios Logothetis, and Georgos Siganos. Facilitating real-time graph mining. In *Proc. 4th Int. Workshop on Cloud Data Management*, pages 1–8, 2012.

[78] Guido Caldarelli, Romualdo Pastor-Satorras, and Alessandro Vespignani. Structure of cycles and local ordering in complex networks. *The European Physical Journal B*, 38(2):183–186, 2004.

[79] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30(4):891–927, 2016.

[80] John Canny. Collaborative filtering with privacy via factor analysis. In *Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 238–245, 2002.

[81] Da Cao, Xiangnan He, Lianhai Miao, Yahui An, Chao Yang, and Richang Hong. Attentive group recommendation. In *Proc. 41th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 645–654, 2018.

[82] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proc. 24th ACM Int. Conf. on Information and Knowledge Management*, pages 891–900, 2015.

[83] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1):2–es, 2006.

[84] Badrish Chandramouli, Justin J. Levandoski, Ahmed Eldawy, and Mohamed F. Mokbel. Streamrec: a real-time recommender system. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1243–1246, 2011.

[85] Lijun Chang, Jeffrey Xu Yu, Lu Qin, Hong Cheng, and Miao Qiao. The exact distance to destination in undirected world. *VLDB J.*, 21(6):869–888, 2012.

[86] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A. Hasegawa-Johnson, and Thomas S. Huang. Streaming recommender systems. In *Proc. 26th Int. World Wide Web Conf.*, pages 381–389, 2017.

[87] Chen Chen, Hongzhi Yin, Junjie Yao, and Bin Cui. Terec: A temporal recommender system over tweet stream. *Proc. VLDB Endowment*, 6:1254–1257, 2013.

[88] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *Proc. 6th Int. Conf. on Learning Representations*, 2018.

[89] Raymond Cheng, Ji Hong, Aapo Kyrola, Youshan Miao, Xuetian Weng, Ming Wu, Fan Yang, Lidong Zhou, Feng Zhao, and Enhong Chen. Kineograph: taking the pulse of a fast-changing and connected world. In *Proc. 7th ACM European Conf. on Computer Systems*, pages 85–98, 2012.

[90] Zhiyong Cheng, Ying Ding, Xiangnan He, Lei Zhu, Xuemeng Song, and Mohan S. Kankanhalli. A3ncf: An adaptive aspect attention model for rating prediction. In *Proc. 27th Int. Joint Conf. on AI*, pages 3748–3754, 2018.

[91] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[92] Shumo Chu and James Cheng. Triangle listing in massive networks and its applications. In *Proc. 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 672–680, 2011.

[93] Shumo Chu and James Cheng. Triangle listing in massive networks. *ACM Trans. Knowl. Discov. Data*, 6(4):1–32, 2012.

[94] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning. *IEEE Trans. Knowl. and Data Eng.*, 13(1):64–78, 2001.

[95] Vittoria Colizza, Alessandro Flammini, M Angeles Serrano, and Alessandro Vespignani. Detecting rich-club ordering in complex networks. *Nature Physics*, 2(2):110–115, 2006.

[96] Jack Collins, Kiel Howe, and Benjamin Nachman. Anomaly detection for resonant new physics with machine learning. *Physical Review Letters*, 121(24):241803, 2018.

[97] Colin Cooper and Alan Frieze. A general model of web graphs. *Random Structures & Algorithms*, 22(3):311–335, 2003.

[98] Dan Cosley, Steve Lawrence, and David M. Pennock. Referee: An open framework for practical testing of recommender systems using researchindex. In *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 35–46, January 2002.

[99] Peter Csermely, András London, Ling-Yun Wu, and Brian Uzzi. Structure and dynamics of core/periphery networks. *Journal of Complex Networks*, 1(2):93–123, 2013.

[100] Gregorio D'Agostino, Antonio Scala, Vinko Zlatić, and Guido Caldarelli. Robustness and assortativity for diffusion-like processes in scale-free networks. *EPL (Europhysics Letters)*, 97(6):68006, 2012.

[101] Shai Ben David, Tyler Lu, Teresa Luu, and Dávid Pál. Impossibility theorems for domain adaptation. In *Proc. 13th Int. Conf. on Artificial Intelligence and Statistics*, pages 129–136, 2010.

[102] Tamas David-Barrett. Herding friends in similarity-based architecture of social networks. *Scientific Reports*, 10(1):1–6, 2020.

[103] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proc. 29th Annual Conf. on Neural Information Processing Systems*, pages 3844–3852, 2016.

[104] Laxman Dhulipala, Changwan Hong, and Julian Shun. Connectit: a framework for static and incremental parallel graph connectivity algorithms. *Proc. VLDB Endowment*, 14:653–667, 2021.

[105] Manh Tuan Do, Se-eun Yoon, Bryan Hooi, and Kijung Shin. Structural patterns and generative models of real-world hypergraphs. In *Proc. 26th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 176–186, 2020.

[106] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. Butterfly-core community search over labeled graphs. *Proc. VLDB Endowment*, 14:2006–2018, 2021.

[107] Sergey N Dorogovtsev and José FF Mendes. Scaling properties of scale-free evolving networks: Continuous approach. *Physical Review E*, 63(5):056125, 2001.

[108] Mikhail Drobyshevskiy and Denis Turdakov. Random graph modeling: A survey of the concepts. *ACM Computing Surveys*, 52(6):1–36, 2019.

[109] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The yahoo! music dataset and kdd-cup'11. In *Proc. KDD Cup 2011*, pages 3–18, 2012.

[110] Doina Alexandra Dumitrescu and Simone Santini. Improving novelty in streaming recommendation using a context model. In *ACM RecSys Workshop on Context-Aware Recommender Systems*, CEUR Workshop Proceedings, 2012.

[111] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[112] Simen Eide and Ning Zhou. Deep neural network marketplace recommenders in online experiments. In *Proc. 12th ACM Conf. on Recommender Systems*, pages 387–391, 2018.

[113] Ryan Elwell and Robi Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Trans. Neural Networks*, 22(10):1517–1531, 2011.

[114] Andrew F Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng-Keen Wong. Systematic construction of anomaly detection benchmarks from real data. In *Proc. ACM SIGKDD Workshop on Outlier Detection and Description*, pages 16–21, 2013.

[115] Paul Erdös, Alfréd Rényi, et al. On random graphs. *Publicationes Mathematicae*, 6(26):290–297, 1959.

[116] Wenfei Fan, Chunming Hu, and Chao Tian. Incremental graph computations: Doable and undoable. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, page 155–169, 2017.

[117] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. Cohesive subgraph search over big heterogeneous information networks: Applications, challenges, and solutions. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 2829–2838, 2021.

[118] Alexander Felfernig, Ludovico Boratto, Martin Stettinger, and Marko Tkalčič. *Group Recommender Systems: An Introduction.* Springer, 2018.

[119] Frank Fischer and Christoph Helmberg. Dynamic graph generation for the shortest path problem in time expanded networks. *Mathematical Programming*, 143(1):257–297, 2014.

[120] Bailey K Fosdick, Daniel B Larremore, Joel Nishimura, and Johan Ugander. Configuring random graph models with fixed degree sequences. *SIAM Review*, 60(2):315–355, 2018.

[121] Dan Frankowski, Shyong K Lam, Shilad Sen, F Maxwell Harper, Scott Yilek, Michael Cassano, and John Riedl. Recommenders everywhere: the wikilens community-maintained recommender system. In *Proc. Int. Symp. on Wikis*, pages 47–60, 2007.

[122] Isvani Frias-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yaile Caballero-Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Trans. Knowl. and Data Eng.*, 27(3):810–823, 2014.

[123] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. In *Machine Learnig*, volume 29, pages 131–163, 1997.

[124] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Proc. Brazilian Symp. on Artificial Intelligence*, pages 286–295, 2004.

[125] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, 2014.

[126] Monali Gandhi and Sonali Gandhi. An enhanced approach for tourism recommendation system using hybrid filtering and association rule mining. *Asian Journal for Convergence in Technology*, 2019.

[127] Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *Proc. VLDB Endowment*, 5:2018–2019, 2012.

[128] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proc. National Academy of Sciences*, 99(12):7821–7826, 2002.

[129] Lukasz Golab and M. Tamer Özsu. Update-pattern-aware modeling and processing of continuous queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 658–669, 2005.

[130] Lukasz Golab and M Tamer Özsu. *Data stream management*, volume 2. Morgan & Claypool, 2010.

[131] Lukasz Golab and M. Tamer Özsu. *Data Stream Systems.* Morgan & Claypool, 2010.

[132] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495, 2017.

[133] Robert Görke, Roland Kluge, Andrea Schumm, Christian Staudt, and Dorothea Wagner. An efficient generator for clustered dynamic random networks. In *Proc. Mediterranean Conf. on Algorithms*, pages 219–233, 2012.

[134] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowl. Based Syst.*, 187, 2020.

[135] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*, 2018.

[136] Derek Greene and Pádraig Cunningham. Producing a unified graph representation from multiple social network views. In *Proc. 5th Annual ACM Web Science conf.*, pages 118–121, 2013.

[137] R. Greinemr, X. Su, B. Shen, and W. Zhou. Structural extension to logistic regression: discriminative parameter learning of belief net classifiers. *Machine Learning*, 3:297–322, 2005.

[138] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 855–864, 2016.

[139] Jelena Grujic, Marija Mitrovic, and Bosiljka Tadic. Mixing patterns and communities on bipartite graphs on web-based social interactions. In *Proc. 16th Int. Conf. on Digital Signal Processing*, pages 1–8, 2009.

[140] Xianbin Gu, Jeremiah D Deng, and Martin K Purvis. Superpixel-based segmentation using multi-layer bipartite graphs and grassmann manifolds. In *Proc. 29th Int. Conf. on Image and Vision Computing*, pages 119–123, 2014.

[141] Jean-Loup Guillaume and Matthieu Latapy. Bipartite structure of all complex networks. *Information Processing Letters*, 90(5):215–221, 2004.

[142] Jean-Loup Guillaume and Matthieu Latapy. Bipartite graphs as models of complex networks. *Physica A: Statistical Mechanics and its Applications*, 371(2):795–813, 2006.

[143] Roger Guimerà, Marta Sales-Pardo, and Luís A Nunes Amaral. Module identification in bipartite and directed networks. *Physical Review E*, 76(3):036102, 2007.

[144] Guibing Guo, Jie Zhang, Daniel Thalmann, and Neil Yorke-Smith. Etaf: An extended trust antecedents framework for trust prediction. In *Proc. 2014 IEEE/ACM Int. Conf. on Advances in Social Networks Analysis and Mining*, pages 540–547, 2014.

[145] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *Proc. 30th Int. Conf. on Very Large Data Bases*, pages 576–587, 2004.

[146] Guy Hadash, Oren Sar Shalom, and Rita Osadchy. Rank and rate: multi-task learning for recommender systems. In *Proc. 12th ACM Conf. on Recommender Systems*, 2018.

[147] Ali Hadian, Sadegh Nobari, Behrooz Minaei-Bidgoli, and Qiang Qu. Roll: Fast in-memory generation of gigantic scale-free networks. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1829–1842, 2016.

[148] Wentao Han, Youshan Miao, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Wenguang Chen, and Enhong Chen. Chronos: a graph engine for temporal graph analysis. In *Proc. 9th European Conf. on Computer Systems*, page 1, 2014.

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

[150] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Trans. Interactive Intell. Syst.*, 5(4):1–19, 2015.

[151] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. Exploring cohesive subgraphs with vertex engagement and tie strength in bipartite graphs. *Information Sciences*, 572:277–296, 2021.

[152] Jelle Hellings, George H.L. Fletcher, and Herman Haverkort. Efficient external-memory bisimulation on dags. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 553–564, 2012.

[153] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. *ACM SIGIR Forum*, 51:227–234, 2017.

[154] Petter Holme. Core-periphery organization of complex networks. *Physical Review E*, 72(4):046111, 2005.

[155] Vagelis Hristidis and Yannis Papakonstantinou. Discover: Keyword search in relational databases. In *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 670–681, 2002.

[156] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian YH Lam. Discovering maximal motif cliques in large heterogeneous information networks. In *Proc. 35th Int. Conf. on Data Engineering*, pages 746–757. IEEE, 2019.

[157] Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. Massive graph triangulation. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 325–336, 2013.

[158] Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. I/o-efficient algorithms on triangle listing and counting. *ACM Trans. Database Syst.*, 39(4):1–30, 2014.

[159] Jiewen Huang and Daniel J Abadi. Leopard: Lightweight edge-oriented partitioning and replication for dynamic graphs. *Proc. VLDB Endowment*, 9(7):540–551, 2016.

[160] Zan Huang. Link prediction based on graph topology: The predictive value of generalized clustering coefficient. *Available at SSRN: https://ssrn.com/abstract=1634014 or http://dx.doi.org/10.2139/ssrn.1634014*, 2010.

[161] Shuta Ito and Takayasu Fushimi. Fast clustering of hypergraphs based on bipartite-edge restoration and node reachability. In *Proc. 22nd Int. Conf. on Information Integration and Web-based Applications & Services*, pages 115–124, 2020.

[162] Anand Padmanabha Iyer, Li Erran Li, Tathagata Das, and Ion Stoica. Time-evolving graph processing at scale. In *Proc. 4th Int. Workshop on Graph Data Management Experiences and Systems*, page 5, 2016.

[163] Mohsen Jamali, Gholamreza Haffari, and Martin Ester. Modeling the temporal dynamics of social rating networks using bidirectional effects of social relations and rating patterns. In *Proc. 20th Int. World Wide Web Conf.*, pages 527–536, 2011.

[164] Anthony Jameson and Barry Smyth. Recommendation to groups. In *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, pages 596–627. Springer, 2007.

[165] Jun Ji, Aifen Fang, Chenlu Qiu, and Lei Zhao. Detection of abnormal database queries in weighted bipartite graph. In *Proc. Int. Conf. on Big Data Engineering and Technology*, pages 7–11, 2018.

[166] Shuo Ji and Yinliang Zhao. A local approximation approach for processing time-evolving graphs. *Symmetry*, 10(7):247, 2018.

[167] Shuo Ji, Yinliang Zhao, and Xiaomei Zhao. A low-latency computing framework for time-evolving graphs. *The Journal of Supercomputing*, 75(7):3673–3692, 2019.

[168] Ruoming Jin, Hui Hong, Haixun Wang, Ning Ruan, and Yang Xiang. Computing label-constraint reachability in graph databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 123–134, 2010.

[169] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *Proc. Int. Conf. Web Search and Web Data Mining*, pages 219–230, 2008.

[170] Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *Int. Computing and Combinatorics Conf.*, pages 710–716, 2005.

[171] Martin Junghanns, Max Kießling, Niklas Teichmann, Kevin Gómez, André Petermann, and Erhard Rahm. Declarative and distributed graph analytics with gradoop. *Proc. VLDB Endowment*, 11(12):2006–2009, 2018.

[172] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Relational representation learning for dynamic (knowledge) graphs: A survey. *arXiv preprint arXiv:1905.11485*, 2019.

[173] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proc. 30th Int. Conf. on Very Large Data Bases*, volume 4, pages 180–191, 2004.

[174] Byeong Man Kim, Qing Li, Jong-Wan Kim, and Jinsoo Kim. A new collaborative recommender system addressing three problems. In *Proc. Pacific Rim Int. Conf. Artificial Intelligence*, pages 495–504. Springer, 2004.

[175] Hyun-Joo Kim and Jin Min Kim. Cyclic topology in complex networks. *Physical Review E*, 72:036109, 2005.

[176] Jinha Kim, Wook-Shin Han, Sangyeon Lee, Kyungyeol Park, and Hwanjo Yu. Opt: a new framework for overlapped and parallel triangulation in large-scale graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 637–648, 2014.

[177] Myunghwan Kim and Jure Leskovec. Modeling social networks with node attributes using the multiplicative attribute graph model. *arXiv preprint arXiv:1106.5053*, 2011.

[178] Myunghwan Kim and Jure Leskovec. Latent multi-group membership graph model. *arXiv preprint arXiv:1205.4546*, 2012.

[179] Myunghwan Kim and Jure Leskovec. Multiplicative attribute graph model of real-world networks. *Internet Mathematics*, 8(1-2):113–160, 2012.

[180] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

[181] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[182] Jon M Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S Tomkins. The web as a graph: Measurements, models, and methods. In *Proc. Int. Computing and Combinatorics Conf.*, pages 1–17, 1999.

[183] Seongyun Ko and Wook-Shin Han. Turbograph++: A scalable and fast graph analytics system. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 395–410, 2018.

[184] Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(11):P11005, 2011.

[185] Pavel L Krapivsky and Sidney Redner. Network growth by copying. *Physical Review E*, 71(3):036118, 2005.

[186] Pavel L Krapivsky, Geoff J Rodgers, and Sidney Redner. Degree distributions of growing networks. *Physical Review Letters*, 86(23):5401, 2001.

[187] Valdis E Krebs. Mapping networks of terrorist cells. *Connections*, 24:43–52, 2002.

[188] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D Sivakumar, Andrew Tomkins, and Eli Upfal. Stochastic models for the web graph. In *Proc. 41st Annual Symp. on Foundations of Computer Science*, pages 57–65, 2000.

[189] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proc. 25th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 1269–1278, 2019.

[190] Ludmila I Kuncheva and Indrė Žliobaitė. On the window size for classification in changing environments. *Intelligent Data Analysis*, 13(6):861–872, 2009.

[191] Jérôme Kunegis. Konect: the koblenz network collection. In *Proc. 22nd Int. World Wide Web Conf.*, pages 1343–1350, 2013.

[192] Marialena Kyriakidi, Kostas Stefanidis, and Yannis Ioannidis. On achieving diversity in recommender systems. In *Proc. Int. Workshop on Exploratory Search in Databases and the Web*, page 4, 2017.

[193] Hamilton William L., Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proc. 30th Annual Conf. on Neural Information Processing Systems*, pages 1024–1034, 2017.

[194] Matthieu Latapy, Clemence Magnien, and Nathalie Del Vecchio. Basic notions for the analysis of large two-mode networks. *Social Networks*, 30(1):31–48, 2008.

[195] Matthieu Latapy, Clémence Magnien, and Nathalie Del Vecchio. Basic notions for the analysis of large two-mode networks. *Social Networks*, 30(1):31–48, 2008.

[196] Dongjin Lee, Kijung Shin, and Christos Faloutsos. Temporal locality-aware sampling for accurate triangle counting in real graph streams. *VLDB J.*, 29(6):1501–1525, 2020.

[197] Xi Tong Lee, Arijit Khan, Sourav Sen Gupta, Yu Hann Ong, and Xuan Liu. Measurements, analyses, and insights on the entire ethereum blockchain network. In *Proc. The Web Conf.*, pages 155–166, 2020.

[198] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287*, 2019.

[199] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 462–470, 2008.

[200] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 631–636, 2006.

[201] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 177–187, 2005.

[202] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1):2–es, 2007.

[203] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[204] CC Leung and HF Chau. Weighted assortative and disassortative networks model. *Physica A: Statistical Mechanics and its Applications*, 378(2):591–602, 2007.

[205] Jin Li, Kristin Tufte, Vladislav Shkapenyuk, Vassilis Papadimos, Theodore Johnson, and David Maier. Out-of-order processing: a new architecture for high-performance stream systems. *Proc. VLDB Endowment*, 1(1):274–288, 2008.

[206] Rundong Li, Pinghui Wang, Peng Jia, Xiangliang Zhang, Junzhou Zhao, Jing Tao, Ye Yuan, and Xiaohong Guan. Approximately counting butterflies in large bipartite graph streams. *IEEE Trans. Knowl. and Data Eng.*, 2021.

[207] Xiaodong Li, Reynold Cheng, Kevin Chen-Chuan Chang, Caihua Shan, Chenhao Ma, and Hongtai Cao. On analyzing graphs with motif-paths. *Proc. VLDB Endowment*, 14:1111–1123, 2021.

[208] Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, and Hady Wirawan Lauw. Detecting product review spammers using rating behaviors. In *Proc. 19th ACM Int. Conf. on Information and Knowledge Management*, pages 939–948, 2010.

[209] Yongsub Lim, Minsoo Jung, and U Kang. Memory-efficient and accurate sampling for counting local triangles in graph streams: from simple to multigraphs. *ACM Trans. Knowl. Discov. Data*, 12(1):1–28, 2018.

[210] Pedro G. Lind, Marta C. Gonzalez, and Hans J. Herrmann. Cycles and clustering in bipartite networks. *Physical Review E*, 72:056127, 2005.

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

[212] Anjin Liu, Guangquan Zhang, and Jie Lu. Fuzzy time windowing for gradual concept drift adaptation. In *Proc. 2017 IEEE Int. Conf. on Fuzzy Systems*, pages 1–6, 2017.

[213] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 93–106, 2008.

[214] Ling Liu and M Tamer Özsu. *Encyclopedia of Database Systems*, volume 6. Springer, 2009.

[215] Paul Liu, Austin R Benson, and Moses Charikar. Sampling methods for counting temporal motifs. In *Proc. 12th ACM Int. Conf. Web Search and Data Mining*, pages 294–302, 2019.

[216] Weide Liu, Chi Zhang, Guosheng Lin, Tzu-Yi Hung, and Chunyan Miao. Weakly supervised segmentation with maximum bipartite graph matching. In *Proc. 28th ACM Int. Conf. on Multimedia*, pages 2085–2094, 2020.

[217] Pasquale Lops, Dietmar Jannach, Cataldo Musto, Toine Bogers, and Marijn Koolen. Trends in content-based recommendation - preface to the special issue on recommender systems based on rich item descriptions. *User Model. User Adapt. Interact.*, 29:1–11, 2019.

[218] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Trans. Knowl. and Data Eng.*, 31(12):2346–2363, 2019.

[219] Ning Lu, Guangquan Zhang, and Jie Lu. Concept drift detection via competence models. *Artificial Intelligence*, 209:11–28, 2014.

[220] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. Maximum biclique search at billion scale. *Proc. VLDB Endowment*, 13(9):1359–1372, 2020.

[221] Chenhao Ma, Reynold Cheng, Laks VS Lakshmanan, Tobias Grubenmann, Yixiang Fang, and Xiaodong Li. Linc: a motif counting algorithm for uncertain graphs. *Proc. VLDB Endowment*, 13(2):155–168, 2019.

[222] Antonio Maccioni and Daniel J Abadi. Scalable pattern matching over compressed graphs via dedensification. In *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 1755–1764, 2016.

[223] Hamidreza Mahyar, Elahe Ghalebi K, S. Mojde Morshedi, Saina Khalili, Radu Grosu, and Ali Movaghar. Centrality-based group formation in group recommender systems. In *Proc. 26th Int. Conf. World Wide Web Companion*, pages 1187–1196, 2017.

[224] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM Computing Surveys*, 49:1–33, 2016.

[225] Sergei Maslov and Yi-Cheng Zhang. Extracting hidden information from knowledge networks. *Physical Review Letters*, 87(24):248701, 2001.

[226] Judith Masthoff. Group modeling: Selecting a sequence of television items to suit a group of viewers. In *Personalized Digital Television: Targeting Programs to individual Viewers*, volume 6 of *Human-Computer Interaction Series*, pages 93–141. Kluwer / Springer, 2004.

[227] Naoki Masuda and Renaud Lambiotte. *A Guide to Temporal Networks.* World Scientific, 2016.

[228] Joseph F. McCarthy and Theodore Anagnost. an arbiter of group preferences for computer supported collaborative workouts. In *ACM Conf. on Computer-Supported Cooperative Work*, page 348, 1998.

[229] Mary McGlohon, Leman Akoglu, and Christos Faloutsos. Weighted graphs and disconnected components: patterns and a generator. In *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 524–532, 2008.

[230] Andrew McGregor. Graph stream algorithms: A survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

[231] Frank McSherry, Derek Gordon Murray, Rebecca Isaacs, and Michael Isard. Differential dataflow. In *Proc. 6th Biennial Conf. on Innovative Data Systems Research*, 2013.

[232] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[233] Einat Minkov, William W Cohen, and Andrew Y Ng. Contextual search and name disambiguation in email using graphs. In *Proc. 29th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 27–34, 2006.

[234] Koji Miyahara and Michael J. Pazzani. Collaborative filtering with the simple bayesian classifier. In *In Proc. Pacific Rim International conf. on Artificial Intelligence*, pages 679–689, 2000.

[235] Koji Miyahara and Michael J. Pazzani. Collaborative filtering with the simple bayesian classifier. In *In Proc. Pacific Rim International Conf. on Artificial Intelligence*, 2000.

[236] Jayanta Mondal and Amol Deshpande. Managing large dynamic graphs efficiently. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 145–156, 2012.

[237] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530, 2012.

[238] Rajeev Motwani and Ying Xu. Evolution of page popularity under random web graph models. In *Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 134–142, 2006.

[239] Arjun Mukherjee, Bing Liu, and Natalie Glance. Spotting fake reviewer groups in consumer reviews. In *Proc. 21st Int. World Wide Web Conf.*, pages 191–200, 2012.

[240] Olfa Nasraoui, Jeff Cerwinske, Carlos Rojas, and Fabio Gonzalez. Performance of recommendation systems in dynamic streaming environments. In *n Proc. 2007 SIAM Int. Conf. on Data Mining*, pages 569–574, 2007.

[241] Mark EJ Newman. Assortative mixing in networks. *Physical Review Letters*, 89(20):208701, 2002.

[242] Mark EJ Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[243] Mark EJ Newman and Juyong Park. Why social networks are different from other types of networks. *Physical Review E*, 68(3):036122, 2003.

[244] Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64(2):026118, 2001.

[245] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proc. 4th Int. Conf. on Learning Representations*, pages 2014–2023, 2016.

[246] Kyosuke Nishida and Koichiro Yamauchi. Detecting concept drift using statistical testing. In *Proc. Int. Conf. on Discovery Science*, pages 264–269, 2007.

[247] Rogier Noldus and Piet Van Mieghem. Assortativity in complex networks. *Journal of Complex Networks*, 3(4):507–542, 2015.

[248] Felix L. Opolka, Aaron Solomon, Cătălina Cangea, Petar Veličković, Pietro Liò, and R. Devon Hjelm. Spatio-temporal deep graph infomax. *arXiv preprint arXiv:1904.06316*, 19.

[249] Timur Osadchiy, Ivan Poliakov, Patrick Olivier, Maisie Rowland, and Emma Foster. Recommender system based on pairwise association rules. In *Expert Systems with Applications*, volume 115, pages 535–542, 2019.

[250] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 1105–1114, 2016.

[251] M Tamer Özsu and Valduriez Patrick. *Principles of Distributed Database Systems, 4th Edition.* Springer, 2020.

[252] Anil Pacaci, Angela Bonifati, and M Tamer Özsu. Regular path query evaluation on streaming graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1415–1430, 2020.

[253] Anil Pacaci, Angela Bonifati, and M Tamer Özsu. Evaluating complex queries on streaming graphs. In *Proc. 38th Int. Conf. on Data Engineering*, pages 272–285, 2022.

[254] Rasmus Pagh and Francesco Silvestri. The input/output complexity of triangle enumeration. In *Proc. 33rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 224–233, 2014.

[255] Kishore Papineni and Pratik Worah. A dynamical system on bipartite graphs. In *Proc. 27th ACM Int. Conf. on Information and Knowledge Management*, pages 1479–1482, 2018.

[256] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *Proc. 10th ACM Int. Conf. Web Search and Data Mining*, pages 601–610, 2017.

[257] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proc. 34th National Conf. on Artificial Intelligence*, volume 34, pages 5363–5370, 2020.

[258] Deokhwan Park, Joosoon Lee, Junseok Lee, and Kyoobin Lee. Deep learning based food instance segmentation using synthetic data. In *Proc. 18th Int. Conf. on Ubiquitous Robots*, pages 499–505, 2021.

[259] Himchan Park and Min-Soo Kim. Lineageba: A fast, exact and scalable graph generation for the Barabási-Albert model. In *Proc. 37th Int. Conf. on Data Engineering*, pages 540–551, 2021.

[260] Romualdo Pastor-Satorras, Alexei Vázquez, and Alessandro Vespignani. Dynamical and correlation properties of the internet. *Physical Review Letters*, 87(25):258701, 2001.

[261] Kostas Patroumpas and Timos Sellis. Window specification over data streams. In *Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology*, pages 445–464, 2006.

[262] Ramesh Paudel and William Eberle. An approach for concept drift detection in a graph stream using discriminative subgraphs. *ACM Trans. Knowl. Discov. Data*, 14(6):1–25, 2020.

[263] Georgina Peake and Jun Wang. Explanation mining: Post hoc interpretability of latent factor models for recommendation systems. In *Proc. 24th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 2060–2069, 2018.

[264] David M. Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: a hybrid memory- and model-based approach. In *Proc. 36th Conf. on Uncertainty in Artificial Intelligence*, 2000.

[265] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proc. 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 701–710, 2014.

[266] Veličković Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[267] Christen Peter. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Data-Centric Systems and Applications. Springer, 2012.

[268] Thomas Petermann and Paolo De Los Rios. Role of clustering and gridlike ordering in epidemic spreading. *Physical Review E*, 69, 2004.

[269] Mason A. Porter and James P. Gleeson. Dynamical systems on networks: A tutorial. *arXiv preprint arXiv:1403.7663*, 2014.

[270] Yanjun Qi, Ziv Bar-Joseph, and Judith Klein-Seetharaman. Evaluation of different biological data and computational classification methods for use in protein interaction prediction. *Structure, Function, and Bioinformatics*, 63:490–500, 2006.

[271] Huaijun Qiu and Edwin R Hancock. Image segmentation using commute times. In *Proc. British Machine Vision Conf.*, pages 929–938, 2005.

[272] Abdul Quamar, Amol Deshpande, and Jimmy Lin. Nscale: neighborhood-centric large-scale graph analytics in the cloud. *VLDB J.*, 25(2):125–150, 2016.

[273] Rajeev Rajaram and Brian Castellani. An entropy based measure for comparing distributions of complexity. *Physica A: Statistical Mechanics and its Applications*, 453:35–43, 2016.

[274] Jérémie Rappaz, Julian McAuley, and Karl Aberer. Recommendation on live-streaming platforms: Dynamic availability and repeat consumption. *Interactions*, 20:40, 2021.

[275] Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical Review E*, 67(2):026112, 2003.

[276] Xiang Ren, Jialu Liu, Xiao Yu, Urvashi Khandelwal, Quanquan Gu, Lidan Wang, and Jiawei Han. Cluscite: Effective citation recommendation by information network-based clustering. In *Proc. 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 821–830, 2014.

[277] Ying Rex, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proc. 24th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 974–983, 2018.

[278] Francesco Ricci and Quang Nhat Nguyen. Acquiring and revising preferences in a critique-based mobile recommender system. *IEEE Intelligent Systems*, 22:22–29, 2007.

[279] Burke Robin. Hybrid recommender systems: Survey and experiments. In *Proc. User Modeling and User-Adapted Interaction*, volume 12, pages 331–370, 2002.

[280] Francisco A Rodrigues, Thomas K DM Peron, Peng Ji, and Jürgen Kurths. The kuramoto model in complex networks. *Physics Reports*, 610:1–98, 2016.

[281] Ryan A Rossi, Nesreen K Ahmed, Aldo Carranza, David Arbour, Anup Rao, Sungchul Kim, and Eunyee Koh. Heterogeneous graphlets. *ACM Trans. Knowl. Discov. Data*, 15(1):1–43, 2020.

[282] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *VLDB J.*, pages 1–24, 2019.

[283] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *VLDB J.*, 29:595–618, 2020.

[284] Unicomb Samuel, Gerardo Iñiguez, James P Gleeson, and Márton Karsai. Dynamics of cascades on burstiness-controlled temporal networks. *Nature Communications*, 12(1):1–10, 2021.

[285] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. Butterfly counting in bipartite networks. In *Proc. 24th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 2150–2159, 2018.

[286] Seyed-Vahid Sanei-Mehri, Yu Zhang, Ahmet Erdem Sariyüce, and Srikanta Tirthapura. Fleet: Butterfly estimation from a bipartite graph stream. In *Proc. 28th ACM Int. Conf. on Information and Knowledge Management*, pages 1201–1210, 2019.

[287] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dynamic graph representation learning via self-attention networks. *arXiv preprint arXiv:1812.09430*, 2018.

[288] Ahmet Erdem Sarıyüce and Ali Pinar. Peeling bipartite networks for dense subgraph discovery. In *Proc. 11th ACM Int. Conf. Web Search and Data Mining*, pages 504–512, 2018.

[289] Badrul Munir Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. *World Wide Web J.*, 1:285–295, 2001.

[290] Yuya Sasaki, George H.L. Fletcher, and Makoto Onizuka. Structural indexing for conjunctive path queries. *arXiv preprint arXiv:2003.03079*, 2020.

[291] Nadathur Satish, Narayanan Sundaram, Md Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey. Navigating the maze of graph analytics frameworks using massive graph datasets. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 979–990, 2014.

[292] Junming Shao, Zahra Ahmadi, and Stefan Kramer. Prototype-based learning on concept-drifting data streams. In *Proc. 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 412–421, 2014.

[293] B. Shen, X. Su, R. Greiner, P. Musilek, and C. Cheng. Discriminative parameter learning of general bayesian network classifiers. In *Proc. 15th IEEE Int. Conf. on Tools with Artificial Intelligence*, pages 296–305, 2003.

[294] Yilin Shen, Yue Deng, Avik Ray, and Hongxia Jin. Interactive recommendation via deep neural memory augmented contextual bandits. In *Proc. 12th ACM Conf. on Recommender Systems*, pages 122–130, 2018.

[295] Aida Sheshbolouki and M. Tamer Özsu. Scale-invariant strength assortativity of streaming butterflies. *CoRR*, abs/2111.12217, 2021.

[296] Aida Sheshbolouki and M. Tamer Özsu. sGrapp: Butterfly approximation in streaming graphs. *CoRR*, abs/2101.12334, 2021.

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

[298] Aida Sheshbolouki and M. Tamer Özsu. sGrow: Explaining the scale-invariant strength assortativity of streaming butterflies. *ACM Trans. Web*, 17(3):1–46, 2023.

[299] Aida Sheshbolouki, Mina Zarei, and Hamid Sarbazi-Azad. Are feedback loops destructive to synchronization? *EPL (Europhysics Letters)*, 111(4):40010, 2015.

[300] Aida Sheshbolouki, Mina Zarei, and Hamid Sarbazi-Azad. The role of leadership in the synchronization of directed complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2015(10):P10022, 2015.

[301] Jessica Shi and Julian Shun. Parallel algorithms for butterfly computations. In *Proc. 1st Symp. on Algorithmic Principles of Computer Systems*, pages 16–30, 2020.

[302] Yue Shi, Martha Larson, and Alan Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys*, 47, 2014.

[303] Kijung Shin, Jisu Kim, Bryan Hooi, and Christos Faloutsos. Think before you discard: Accurate triangle counting in graph streams with deletions. In *Proc. European Conf. on Machine Learning and Knowledge Discovery in Databases*, pages 141–157, 2018.

[304] Kijung Shin, Sejoon Oh, Jisu Kim, Bryan Hooi, and Christos Faloutsos. Fast, accurate and provable triangle counting in fully dynamic graph streams. *ACM Trans. Knowl. Discov. Data*, 14(2):1–39, 2020.

[305] Yook Soon-Hyung, Hawoong Jeong, Albert-Laszlo Barabási, and Yuhai Tu. Weighted evolving networks. *Physical Review Letters*, 86:5835–5838, 2001.

[306] Isabelle Stanton and Ali Pinar. Constructing and sampling graphs with a prescribed joint degree distribution. *Journal of Experimental Algorithmics*, 17:3–1, 2012.

[307] X. Su and T. M. Khoshgoftaar. Collaborative filtering for multi-class data using belief nets algorithms. In *Proc. 18th IEEE Int. Conf. on Tools with Artificial Intelligence*, pages 497–504, 2006.

[308] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009, 2009.

[309] Karthik Subbian, Charu Aggarwal, and Kshiteesh Hegde. Recommendations for streaming data. In *Proc. 25th ACM Int. Conf. on Information and Knowledge Management*, pages 2204–2227, 2016.

[310] Jianing Sun, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, Xiuqiang He, Chen Ma, and Mark Coates. Neighbor interaction aware graph convolution networks for recommendation. In *Proc. 43rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, page 1289–1298, 2020.

[311] Özge Sürer, Robin Burke, and Edward C. Malthouse. Multistakeholder recommendation with provider constraints. *Proc. 12th ACM Conf. on Recommender Systems*, pages 54–62, 2018.

[312] Zeeshan Syed, Collin Stultz, Manolis Kellis, Piotr Indyk, and John Guttag. Motif discovery in physiological datasets: a methodology for inferring predictive elements. *ACM Trans. Knowl. Discov. Data*, 4(1):1–23, 2010.

[313] Partha Pratim Talukdar, Zachary G Ives, and Fernando Pereira. Automatically incorporating new sources in keyword search-based data integration. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 387–398, 2010.

[314] Jiliang Tang, Huiji Gao, Huan Liu, and Atish Das Sarma. etrust: Understanding trust evolution in an online world. In *Proc. 18th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 253–261, 2012.

[315] Carlos HC Teixeira, Arlei Silva, and Wagner Meira Jr. Min-hash fingerprints for graph kernels: A trade-off among accuracy, efficiency, and compression. *J. Inf. Data Manag.*, 3(3):227–242, 2012.

[316] Yu Ting, Cao Yan, and Mu Xiang-wei. Personalized recommendation system based on web log mining and weighted bipartite graph. In *Proc. 2013 Int. Conf. on Computational and Information Sciences*, pages 587–590, 2013.

[317] Junyu Tong, Hongyuan Ma, Wei Liu, and Bo Wang. Time and location-based hybrid recommendation system. In *Proc. IEEE 2nd International Conf. on Big Data Analysis*, pages 677–683, 2017.

[318] Stojan Trajanovski, Javier Martín-Hernández, Wynand Winterbach, and Piet Van Mieghem. Robustness envelopes of networks. *Journal of Complex Networks*, 1(1):44–62, 2013.

[319] Silke Trißl and Ulf Leser. Fast and practical indexing and querying of very large graphs. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 845–856, 2007.

[320] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *Proc. 6th Int. Conf. on Learning Representations*, 2018.

[321] Alexey Tsymbal. The problem of concept drift: definitions and related work. Technical Report TCD-CS-2004-15, The University of Dublin, Trinity College, Department of Computer Science, 2004.

[322] Katherine Van Koevering, Austin R Benson, and Jon Kleinberg. Random graphs with prescribed k-core sequences: A new null model for network analysis. *arXiv preprint arXiv:2102.12604*, 2021.

[323] Demival Vasques Filho and Dion RJ O'Neale. Degree distributions of bipartite networks and their projections. *Physical Review E*, 98(2):022307, 2018.

[324] Demival Vasques Filho and Dion RJ O'Neale. Transitivity and degree assortativity explained: The bipartite structure of social networks. *Physical Review E*, 101(5):052305, 2020.

[325] Alexei Vazquez. Knowing a network by walking on it: emergence of scaling. *arXiv preprint cond-mat/0006132*, 2000.

[326] Alexei Vázquez. Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations. *Physical Review E*, 67(5):056104, 2003.

[327] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[328] Thanh Vinh Vo and Harold Soh. Generation meets recommendation: proposing novel items for groups of users. In *Proc. 12th ACM Conf. on Recommender Systems*, pages 145–153, 2018.

[329] Andrew Z. Wang, Rex Ying, Pan Li, Nikhil Rao, Karthik Subbian, and Jure Leskovec. Bipartite dynamic representations for abuse detection. In *Proc. 27th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, page 3638–3648, 2021.

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

[331] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. Vertex priority based butterfly counting for large-scale bipartite networks. *Proc. VLDB Endowment*, 12(10):1139–1152, 2019.

[332] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. Efficient bitruss decomposition for large-scale bipartite graphs. In *Proc. 36th Int. Conf. on Data Engineering*, pages 661–672, 2020.

[333] Mengzhi Wang, Tara Madhyastha, Ngai Hang Chan, Spiros Papadimitriou, and Christos Faloutsos. Data mining meets performance evaluation: Fast algorithms

for modeling bursty traffic. In *Proc. 18th Int. Conf. on Data Engineering*, pages 507–516, 2002.

[334] Nan Wang, Jingbo Zhang, Kian-Lee Tan, and Anthony KH Tung. On triangulation-based dense neighborhood graph discovery. *Proc. VLDB Endowment*, 4(2):58–68, 2010.

[335] Pinghui Wang, Yiyan Qi, Yu Sun, Xiangliang Zhang, Jing Tao, and Xiaohong Guan. Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. *Proc. VLDB Endowment*, 11(2):162–175, 2017.

[336] Wei Wang, Furu Wei, Wenjie Li, and Sujian Li. Hypersum: hypergraph based semi-supervised sentence ranking for query-oriented summarization. In *Proc. 18th ACM Int. Conf. on Information and Knowledge Management*, pages 1855–1858, 2009.

[337] Weiqing Wang, Hongzhi Yin, Zi Huang, Qinyong Wang, Xingzhong Du, and Quoc Viet Hung Nguyen. Streaming ranking based recommender systems. In *Proc. 41th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 525–534, 2018.

[338] Yaojing Wang, Yuan Yao, Hanghang Tong, Feng Xu, and Jian Lu. A brief review of network embedding. *Big Data Mining and Analytics*, 2(1):35–47, 2018.

[339] Stanley Wasserman and Katherine Faust. Social network analysis: Methods and applications. 1994.

[340] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *Nature*, 393(6684):440, 1998.

[341] Anatol Wegner. Random graphs with motifs. *Available at https://www.mis.mpg.de/preprints/2011/preprint2011_61.pdf*, 2011.

[342] Yinwei Wei, Xiang Wang, Liqiang Nie, Xiangnan He, Richang Hong, and Tat-Seng Chua. Mmgcn: Multi-modal graph convolution network for personalized recommendation of micro-video. In *27th ACM Int. Conf. Multimedia*, pages 1437–1445, 2019.

[343] Steven Euijong Whang and Hector Garcia-Molina. Incremental entity resolution on rules and data. *VLDB J.*, 23, 2014.

[344] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[345] Baoning Wu and Kumar Chellapilla. Extracting link spam using biased random walks from spam seed sets. In *Proc. 3rd Int. Workshop on Adversarial Information Retrieval on the Web*, pages 37–44, 2007.

[346] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *Proc. 36th Int. Conf. on Machine Learning*, pages 6861–6871, 2019.

[347] Guangyu Wu, Martin Harrigan, and Pádraig Cunningham. Characterizing wikipedia pages using edit network motif profiles. In *Proc. 3rd Int. Workshop on Search and Mining User-Generated Contents*, pages 45–52, 2011.

[348] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

[349] Shuliang Xu and Junhong Wang. Dynamic extreme learning machine for data stream classification. *Neurocomputing*, 238:433–449, 2017.

[350] Xifeng Yan, Philip S Yu, and Jiawei Han. Graph indexing: a frequent structure-based approach. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 335–346, 2004.

[351] Shengqi Yang, Xifeng Yan, Bo Zong, and Arijit Khan. Towards effective partition management for large graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 517–528, 2012.

[352] Yibo Yao and Lawrence B Holder. Detecting concept drift in classification over streaming graphs. In *Proc. KDD Workshop on Mining and Learning with Graphs*, pages 2134–42, 2016.

[353] Josh Jia-Ching Ying, Ji Zhang, Che-Wei Huang, Kuan-Ta Chen, and Vincent S Tseng. Fraudetector+ an incremental graph-mining approach for efficient fraudulent phone call detection. *ACM Trans. Knowl. Discov. Data*, 12(6):1–35, 2018.

[354] Jiaxuan You, Jure Leskovec, Kaiming He, and Saining Xie. Graph structure of neural networks. In *Proc. 37th Int. Conf. on Machine Learning*, pages 10881–10891, 2020.

[355] Kai Yu, Anton Schwaighofer, Volker Tresp, Xiaowei Xu, and H-P. Kriegel. Probabilistic memory-based collaborative filtering. *IEEE Trans. Knowl. and Data Eng.*, pages 56–69, 2004.

[356] Shujian Yu, Xiaoyang Wang, and José C Príncipe. Request-and-reverify: Hierarchical hypothesis testing for concept drift detection with expensive labels. *arXiv preprint arXiv:1806.10131*, 2018.

[357] Zhiwen Yu, Xingshe Zhou, Yanbin Hao, and Jianhua Gu. Tv program recommendation for multiple viewers based on user profile merging. *User Model. User Adapt. Interact.*, 16:63–82, 2006.

[358] Daniele Zambon, Cesare Alippi, and Lorenzo Livi. Concept drift and anomaly detection in graph streams. *IEEE Trans. Neural Networks and Learning Systems*, 29(11):5592–5605, 2018.

[359] Giselle Zeno, Timothy La Fond, and Jennifer Neville. Dymond: Dynamic motif-nodes network generative model. In *Proc. The Web Conf. 2021*, pages 718–729, 2021.

[360] Hongyuan Zha. Generic summarization and keyphrase extraction using mutual reinforcement principle and sentence clustering. In *Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 113–120, 2002.

[361] Haotian Zhang, Elaheh Fata, and Shreyas Sundaram. A notion of robustness in complex networks. *IEEE Trans. on Control of Network Systems*, 2(3):310–320, 2015.

[362] Jianpeng Zhang, Kaijie Zhu, Yulong Pei, George H.L. Fletcher, and Mykola Pechenizkiy. Clustering-structure representative sampling from graph streams. In *Proc. Int. Conf. Complex Networks and their Applications*, pages 265–277, 2017.

[363] Peng Zhang, Jinliang Wang, Xiaojia Li, Menghui Li, Zengru Di, and Ying Fan. Clustering coefficient and community structure of bipartite networks. *Physica A: Statistical Mechanics and its Applications*, 387(27):6869–6875, 2008.

[364] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, 52:5:1–5:38, 2019.

[365] Yang Zhang. Language in our time: An empirical analysis of hashtags. In *Proc. 28th Int. World Wide Web Conf.*, pages 2378–2389, 2019.

[366] Yuhong Zhang, Guang Chu, Peipei Li, Xuegang Hu, and Xindong Wu. Three-layer concept drifting detection in text data streams. *Neurocomputing*, 260:393–403, 2017.

[367] Lingxiao Zhao and Leman Akoglu. On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights. *arXiv preprint arXiv:2012.12931*, 2020.

[368] Peixiang Zhao, Charu Aggarwal, and Gewen He. Link prediction in graph streams. In *Proc. 32nd Int. Conf. on Data Engineering*, pages 553–564, 2016.

[369] Peixiang Zhao, Jeffrey Xu Yu, and S Yu Philip. Graph indexing: Tree+ delta $\geq$ graph. In *Proc. 33rd Int. Conf. on Very Large Data Bases*, pages 938–949, 2007.

[370] Yi Zheng, Hongchao Qin, Jun Zheng, Fusheng Jin, and Rong-Hua Li. Butterfly-based higher-order clustering on bipartite networks. In *Proc. Int. Conf. on Knowledge Science, Engineering and Management*, pages 485–497, 2020.

[371] Zheng Zhong, Shen Yan, Zikun Li, Decheng Tan, Tong Yang, and Bin Cui. Burstsketch: Finding bursts in data streams. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 2375–2383, 2021.

[372] Dawei Zhou, Si Zhang, Mehmet Yigit Yildirim, Scott Alcorn, Hanghang Tong, Hasan Davulcu, and Jingrui He. High-order structure exploration on massive graphs: A local graph clustering perspective. *ACM Trans. Knowl. Discov. Data*, 15(2):1–26, 2021.

[373] Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. A data-driven graph generative model for temporal interaction networks. In *Proc. 26th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 401–411, 2020.

[374] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[375] Shi Zhou and Raúl J Mondragón. The rich-club phenomenon in the internet topology. *IEEE Communications Letters*, 8(3):180–182, 2004.

[376] Tao Zhou, Jie Ren, Matúš Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. *Physical Review E*, 76(4):046115, 2007.

[377] Yu Qing Zhou, Ga Wu, Scott Sanner, and Putra Manggala. Aesthetic features for personalized photo recommendation. *CoRR*, abs/1809.00060, 2018.

[378] Qiuyu Zhu, Jiahong Zheng, Hao Yang, Chen Chen, Xiaoyang Wang, and Ying Zhang. Hurricane in bipartite graphs: The lethal nodes of butterflies. In *Proc. 32nd Int. Conf. on Scientific and Statistical Database Management*, pages 1–4, 2020.

[379] Abolfazl Ziaeemehr, Mina Zarei, and Aida Sheshbolouki. Emergence of global synchronization in directed excitatory networks of type i neurons. *Scientific Reports*, 10(1):1–11, 2020.

[380] Abolfazl Ziaeemehr, Mina Zarei, Alireza Valizadeh, and Claudio R Mirasso. Frequency-dependent organization of the brain's functional network through delayed-interactions. *Neural Networks*, 132:155–165, 2020.

[381] Indrė Žliobaitė, Mykola Pechenizkiy, and Joao Gama. An overview of concept drift applications. In Nathalie Japkowicz and Jerzy Stefanowski, editors, *Big Data Analysis: New Algorithms for a New Society*, pages 91–114. Springer, 2016.

[382] Yong Zou, Reik V Donner, Norbert Marwan, Jonathan F Donges, and Jürgen Kurths. Complex network approaches to nonlinear time series analysis. *Physics Reports*, 787:1–97, 2019.

[383] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. Embedding temporal network via neighborhood formation. In *Proc. 24th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 2857–2866, 2018.

[384] Katharina Anna Zweig and Michael Kaufmann. A systematic approach to the one-mode projection of bipartite graphs. *Social Network Analysis and Mining*, 1:187–218, 2011.