THE UNIVERSITY OF SYDNEY

SCHOOL OF INFORMATION TECHNOLOGIES

STREAM EB: STREAM EDGE BUNDLING

TECHNICAL REPORT 689

QUAN NGUYEN, PETER EADES, SEOK-HEE HONG

JULY 2012

StreamEB: Stream Edge Bundling

Quan Nguyen*, Peter Eades, Seok-Hee Hong

School of Information Technologies, University of Sydney, Australia
{qnguyen,peter,shhong}@it.usyd.edu.au

Abstract. *Graph streams* have been extensively studied, for instance, for data mining and for stream reasoning, while a fairly limited number of studies have been conducted on visualizations. Recently, *edge bundling* methods has been extensively investigated to reduce visual clutter in large graph visualizations, however, have focused on *static* graphs.

This paper presents a new framework, namely StreamEB[1], for edge bundling of graph streams, which integrates temporal, neighborhood, data-driven and spatial compatibility for edges. Amongst these metrics, *temporal compatibility* and *neighborhood compatibility* are introduced for the first time, whereas the other metrics have not yet been studied for graph streams.

Based on the framework, we present two bundling methods, called FStreamEB (Force-directed Stream Edge Bundling) and TStreamEB (Tree-based Stream Edge Bundling). We demonstrate effectiveness of our framework using US flights data and Thompson-Reuters stock data.

1 Introduction

Data streaming has become ubiquitous and graph streaming is increasingly popular. The streaming model permits processing large graph structures produced from modern applications. In *graph streams*, individual edges of the underlying graphs arrive sequentially in a stream and thus any computation on the stream consumes a relatively small amount of memory. A vast number of graph stream algorithms exist; for example, for computing social network metrics over graphs [7, 19, 23], for clustering algorithms for graph streams, such as, for node clustering via multi-dimensional data [2, 3, 32], and for stream reasoning [8].

Existing stream algorithms share common principles: processing data streams on the fly; exploiting the temporal order of the data stream to optimize the computation; using one pass (or a small number of passes) on the data; and requiring a workspace much smaller than the size of the data.

However, a comparatively tiny number of works have focused on visualization of the rapid changes such as in the context of graph streams [1, 9, 13, 15, 35].

Stream graph visualization differs from traditional *dynamic graph visualization* in that: (a) the size of graph stream can be very huge; (b) the data changes very quickly and it comes in incrementally as a stream; (c) the whole graph in streaming context is

* The first author was partially supported by Capital Markets CRC Ltd., Australia

[1] See pictures and movies at http://www.it.usyd.edu.au/∼qnguyen/streameb/

not always available; and (d) the response time may become a critical factor to design visualization methods to show the evolution of graph streams.

Figure 1 shows different types of dynamic graphs. *Relational time series* are dynamic graphs of time-stamped edges. *Graph streams*, on the other hand, are updated very frequently, and edges arrive in a flow with or without a timestamp. A stricter class of dynamic graphs is the *Streaming Relational Time Series* containing graph streams of time-stamped tuples.

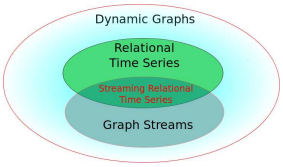


Fig. 1: The classes of dynamic graphs

Edge bundling becomes popular in Graph Drawing and Information Visualization communities to reduce visual clutter and show high-level edge patterns [14, 17, 21, 24, 25, 25, 27, 28, 31, 34]. Most edge bundling methods, however, have focused on *static* graphs.

In this paper, we initiate the study of *stream edge bundling*, to support visual temporal analysis for the changes in topology of the graphs over time. More specifically, we aim to show how edge bundling can reduce visual cluttering in the visualizations of graph streams to assist key mining tasks such as: *(1) Community discovery:* how edge bundles may help identifying groups or communities of nodes/edges that are associated with each other? *(2) Change detection:* how edge bundles may help detecting the newly forming or decaying communities in the underlying network?

Our research is based on a premise that the edge connection patterns may change rapidly, but "edge bundles" should change much slower. The term *edge bundle* in both static and dynamic context may refer to a group of edges that are drawn closely together. Edge bundled layouts show high-level structure of the network with a collection of edge bundles. Edge bundling methods may help unveil the extent to which communities have shrunk, split or emerged over time.

Our contribution: We present a new framework, namely StreamEB, which is the first work that addresses edge bundling for graph streams. Our main research addresses *flexibility* (e.g., adaptive to user inputs at *runtime*) of methods to extract high-level patterns in graph streams.

Our StreamEB framework integrates several compatibility measures such as *temporal*, *neighborhood*, *data-driven* and *spatial* compatibility for stream bundling. Amongst these metrics, *temporal compatibility* and *neighborhood compatibility* are introduced for the first time, whereas the other compatibility metrics are carefully adapted for fast graph streams. Based on the framework, we present two bundling methods, called FStreamEB (Force-directed Stream Edge Bundling) and TStreamEB (Tree-based Stream Edge Bundling).

We evaluate our framework using US flights data and Reuters stocks data to show effectiveness to support various stream mining tasks. The main aim is the comprehension of the massive relational time series, dynamically generated from financial activities

and flight monitoring activities. More specifically, we show how our framework can be used in the following scenarios:

- identifying important actors and groups of closely related actors, and
- locating time-varying (abnormal) patterns.

The rest of this paper is organized as follows. After surveying the related work on stream mining/reasoning, dynamic graph visualization and edge bundling, we show our motivating examples in Section 3. Then we present the details of our StreamEB framework in Section 4 and our bundling algorithms in Section 5. We then discuss implementation in Section 6. Our case studies of our stream bundling methods to a collection of real-world datasets are presented in Section 7 with experimental results are given in Section 8. Section 9 gives some discussions and future work and Section 10 concludes.

2 Related work

2.1 Stream Algorithms

Streaming research has extensively studied graph problems [19], such as counting triangles, distance estimation, connectivity and graph matching. Node clustering determines groups of nodes based on the density of linkage behavior; for example, graph-partitioning, minimum-cut and dense subgraph determination. Recently, stream reasoning has become popular for stream research [8].

Typically, a stream reasoner/miner selects the relevant data in the input stream by exploiting the *window-processing* [6] and *load-shedding* techniques. In window-processing [6], a window extracts from the stream the last data stream elements, which are considered by the query. Such extraction can be *sequence-based* (a given number of tuples) or *time-based* (all the tuples which occur during a given time interval, the number of which varies over time). Less common methods include the load-shedding techniques, which probabilistically drop stream data elements based on a set of sampling policies.

2.2 Dynamic Graph Visualization

The previous research in Graph Visualization has fairly focused on a (a set of) static graph(s) of relatively modest size. A comparatively small number of works have focused on visualization of graph streams [1, 9, 13, 15, 35]. Visualization of graph streams is harder due to the dynamics of the streams.

When analyzing dynamic graphs, it is critical to be able to see the statistical trends and changes over time, while preserving user's mental map [29].

The most common techniques for representing temporal data are via *animation* and *the "small multiples" display*. The animation approach shows visualizations of the sequence of graphs displayed in consecutive frames. The small multiples display uses multiple charts laid side-by-side and corresponding to consecutive time periods or moments in time [5].

The challenges of dynamic graph visualization include:

Quality and stability The first challenge is the control between readability and stability when visualizing graph changes. A common technique for drawing graphs is stress-majorization [22], or more generally, multi-dimensional scaling. The most common approach is to set the initial layout for each graph with the preceding layout [26, 30]; yet layout readability may degrade over the sequence of graphs. Other approaches address stability by either placing vertices of fixed vertex locations from the layout of an aggregate of all graphs [30]; or anchoring vertices to reference positions [12]; or linking vertices to instances of themselves that are close in the sequence [18]. A comprehensive study of existing models and the trade-offs between stability and readability is given in Brandes et al. [11].

Visual clutter The second and also the most challenging is that visualizations of large graphs often suffer from visual clutter. Edge bundling [14, 24, 27, 28, 31, 34, 37] is one of the most popular approaches to reduce visual clutter and to detect clusters in static graph visualizations.

In general, the challenges of dynamic graph visualization include the control between *readability* and *stability* (see Brandes et al. [11]), and the visual clutter incurred in visualizations of large graphs. The visualization of graph streams is harder since the dynamics of the graphs occur more frequently.

2.3 Edge bundling

Edge bundling has been quite successful in reducing clutter and has been extensively studied including hierarchical edge bundling [24], geometry-based edge clustering [14, 24, 28, 37], force-directed edge bundling [25, 27, 31, 34] and multi-level agglomative edge bundling [20]. However, most existing bundling techniques are not concerned with dynamic graphs "directly". There is an isolated study in [10] applying force-directed edge bundling [25] to yearly migration graphs and then displaying the bundled results in small multiples for analysis of the US migration flows over year.

Hierarchical edge bundling(HEB) [24] draws edges along the associated paths in a hierarchy. HEB applies tree algorithms for the input hierarchy and then bundles adjacency edges along the tree branches. Intuitively, HEB uses a "single step" interpolation the geometry of individual edges with respect to the "backbone" tree as a compatibility between pairs of edges. This single interpolation can be expressed as $inter(ctrl(e_i), e_i, \beta)$ for some bundling strength $\beta \in [0, 1]$, where $ctrl(e_i)$ is the set of control points of an edge e_i, and $inter(S, e_i, \beta)$ denotes the set of interpolation points of a point set S with respect to e_i and a bundling strength factor $\beta \in [0, 1]$. An example of interpolation $inter$ proposed in HEB is given as follows:

$$p'_k = \beta.p_k + (1-\beta)(\mathsf{p}_{e_i^{start}} + \frac{k+1}{P+1}(\mathsf{p}_{e_i^{end}} - \mathsf{p}_{e_i^{start}})), \tag{1}$$

where P is the number of control points; p_k is the control point at index $k \in 0, ..., P-1$; p'_k is the interpolated point of p_k; p_x is the location of x; e_i^{start} and e_i^{end} is the start and end of e_i, respectively.

Holten and van Wijk [25] introduces *Force-directed Edge Bundling (FDEB)* method, which models edges intuitively as flexible springs that can attract each other. The FDEB algorithm first inserts control points in each edge, and then uses a force-directed method

to compute the position of the control points. The attractive forces depend on the so-called "spatial compatibility" $S(e_i, e_j)$. For a subdivision point $e_i^{(l)}$ on edge e_i, the total force $F_{e_i^{(l)}}$ exerted on $e_i^{(l)}$ is a sum of the two spring forces exerted by two neighbors $e_i^{(l-1)}$ and $e_i^{(l+1)}$, and the total of electrostatic forces F_s:

$$F_{e_i^{(l)}} = k_e(|\mathsf{p}_{e_i^{(l-1)}} - \mathsf{p}_{e_i^{(l)}}| + |\mathsf{p}_{e_i^{(l)}} - \mathsf{p}_{e_i^{(l+1)}}|) + F_s, \tag{2}$$

where k_e is the stiffness of the edges. The electrostatic force model in FDEB is

$$F_s = \sum_{e_j \in C(e_i)} S(e_i, e_j) * |\mathsf{p}_{e_i^{(l)}} - \mathsf{p}_{e_j^{(l)}}|^{-d}, \tag{3}$$

where $C(e_i)$ is the set of compatible edges of e_i; and $S(e_i, e_j)$ is the spatial compatibility for a pair of edges e_i and e_j.

Several works adapted FDEB to cope with different bundling criteria in various application domains [27, 31, 34]; for example, for analyzing semantic graphs [27], for analyzing important connections and cluster-graphs [31], for analysis of connectivity and edge directions for edge bundling [34].

3 Motivating example

This section compares the visualizations using straight-line drawing, FDEB, and our methods.

Figure 2(a) shows an example visualization of stock trading data from TSX Venture-Canada equities exchange from Reuters Thompson. Traders are nodes that are colored based on the total trades varying from blue (small amount) to red (large amount). Edges represent trades and are encoded a gradient color between the end nodes. Node placement is computed from a multidimensional scaling method on the performance summaries of individual traders. The resulting visual clutter hinders most analytic tasks.

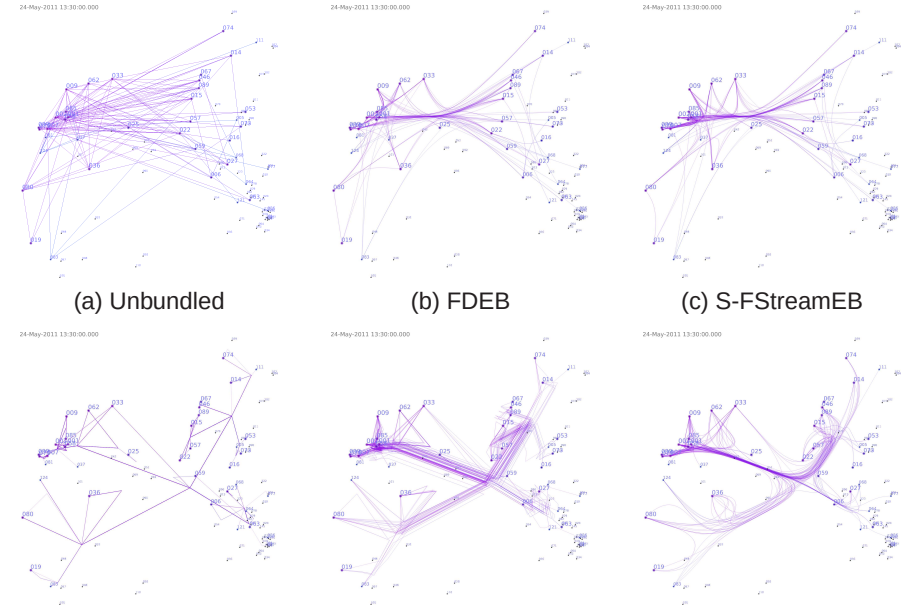
Figure 2(b) applies Holten and van Wijk's force-directed edge bundling [25], showing high-level edge patterns hidden in unbundled graph.

Figure 2(c) applies FStreamEB to the stock trade network. Figure 2(d) shows a bundled layout in which edges are bundled along a quad tree computed from node placement. Figures 2(e) and 2(f) show the visualizations of TStreamEB to the stock trade network before and after edge smoothing. In Figures 2(c), 2(e) and 2(f), the visualizations use a combination of spatial, temporal, data-driven and neighborhood compatibility (the ratios are 0.7, 0.1, 0.1, and 0.1, respectively). Our methods allow varying ratios of compatibility measures at runtime; such flexibility is absent from existing edge bundling methods.

4 StreamEB Framework

4.1 Problem Definition and Notation

A graph stream has an underlying graph G=(V,E), where V is the vertex set and E is the edge set of G. The graph stream is a sequence of elements e_0, e_1, ..., e_i, ... Each element e_i has a general form of $(x_i, y_i, t_i, [d_i], a_i^1, \ldots, a_i^m)$, where:



(a) Unbundled (b) FDEB (c) S-FStreamEB

(d) Routing along QuadTree (e) TStreamEB w/o smoothing (f) TStreamEB after smoothing

Fig. 2: Visualizations of the stock trade data from TSX Venture-Canada equity exchange: sequence-based window of size W=160. Node placement is computed with multi-dimensional scaling from the trader performance. The visualizations (c), (e) and (f) use a combination of spatial, temporal, data-driven and neighborhood compatibility with the ratios of 0.7, 0.1, 0.1, and 0.1, respectively.

- (x_i, y_i) is a directed edge encountered at the time-stamp t_i in which x_i and y_i are the nodes in V.
- d_i is the *duration* associated with the edge (x_i, y_i) at time t_i. The e_i exist from time t_i to time $t_i + d_i$ unless a new edge of the same nodes (x_i, y_i) arrives at time t_j such that $t_j < t_i + d_i$. In some applications d_i is not well-defined.
- each incoming streamed edge has multiple attributes $a_i^1, \ldots, a_i^m$, where a_i^j denotes the j-th attribute associated with the edge (x_i, y_i) at time t_i.

An edge (x_i, y_i) may appear multiple times at different timestamps, but no edge occurs multiple times at any one timestamp. There is no self-loop edge. At a single timestamp, multiple edges may occur.

To process a long stream, the sliding window is commonly used to select more recent elements. Two types of sliding windows are:

- A *sequence-based window* $G_W^{(S)}(n)$ consists of W most recent data elements from max(0,n-W+1)-th to the n-th data elements.
- A *time-based window* $G_W^{(T)}(t_i)$ at time t_i consists of data elements arriving within the last W time units, with timestamps from t= max(0, t_i-W+1) to t_i.

Our methods are insensitive to whether the sliding windows are sequence-based or time-based. We simply use $G_i = \{V, E_i\}$ to denote the current window of some size W, unless it is necessary to distinguish.

4.2 General Model for Stream Bundling

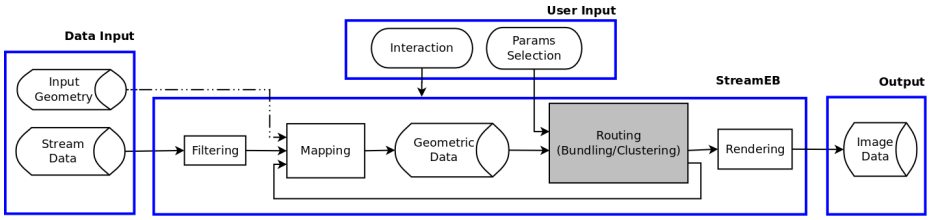


Fig. 3: Stream Bundling Pipeline

The general pipeline for stream bundling and our StreamEB framework are depicted in Figure 3.

- *Data Input* includes the main input source (Stream Data) of a sequence of tuples $(x_i, y_i, t_i, d_i, a_i^1, \ldots, a_i^m)$, and an optional input (Input Geometry) giving (reference) locations for the graph vertices.
- *StreamEB* framework consists of several modules. The *Mapping* module may take optional Input Geometry and/or use a graph layout algorithm to compute node placement. Central is the Routing module, which takes the Geometric Data and performs Edge Bundling for streams. The bundling process may provide feedback for the node placement to better cope with changes in graph streams and user inputs.
- *User Input* includes user interactions to support analytic tasks. The interactions include, for example, selecting sliding window types, selecting layout algorithms, tuning parameters for bundling, zooming the bundled visualizations, and highlighting stream elements.
- *Output* contains final bundled images generated by the Rendering.

4.3 Criteria for Stream Analytics

There are several criteria for enabling stream analytics.

- *Aesthetics* aim for nice bundling results, which have smooth curves and clear bundles. The well formed clusters of edges help analysts to identify both global and local structural changes.
- *Mental map* preservation is concerned about the relative positions and directions of the resulting bundles to each other over time.

4.4 Compatibility metrics

This section defines several compatibility measures for a pair of stream elements e_i and e_j in the current active window.

Temporal compatibility We introduce a new measure, namely "*temporal compatibility*", $T(e_i, e_j)$, for stream elements. Temporal compatibility is defined from the temporal information of the edges and thus is independent of spatial compatibility. The following measures of temporal compatibility are defined based on the timestamps t_i and durations d_i of the streamed edges.

- *Timestamp compatibility* takes the timestamps of edges into account. This timestamp compatibility $T_T(e_i, e_j)$ aims to avoid bundling edges that have huge difference in timestamps. Often one prefers to analyze edges of similar ages.
- *Duration compatibility* is another new measure concerning the time interval a element may last. The duration compatibility $T_D(e_i, e_j)$ avoids bundling a long-lasting element with a short-lasting element. The intuition is that short-lasting stream may be expired in a couple of windows; and thus bundling a long-lasting one with a short-lasting one may not preserve the mental map.
- *Endtime compatibility* The dual of a start time (given by the timestamp) is the end time of a stream element. Sometimes one is more interested in how close when stream elements end rather than when they start or last. For instance, one analytic task may need to cluster the flights of similar arrival times to arrange facilities and public transports.
- *Time-overlapping compatibility* In some cases, one may be more interested in the amount of time overlap between two edges. For instance, one can determine the crash-potential of flights from the amount of time overlap between the flights. As such, edge bundling prefers to bundle edges if their time-intervals $[t_i, t_i + d_i]$ and $[t_j, t_j + d_j]$ have a large overlap and not to bundle otherwise. Let $o(e_i, e_j)$ denote the overlap between two edges.

The timestamp compatibility $T_T(e_i, e_j)$, the duration compatibility $T_D(e_i, e_j)$, the endtime compatibility $T_E(e_i, e_j)$ and time-overlapping compatibility $T_O(e_i, e_j)$ can be defined as:

$$T_T(e_i, e_j) = f(|t_i - t_j|) \tag{4}$$
$$T_D(e_i, e_j) = f(|d_i - d_j|) \tag{5}$$
$$T_E(e_i, e_j) = f(|t_i + d_i - t_j - d_j|) \tag{6}$$
$$T_O(e_i, e_j) = f(o(e_i, e_j)), \tag{7}$$

where $f(x)$ is a continuous and decreasing function: [0,+∞) $\rightarrow$ [0,1]; for example, $(1 + x)^{-1}$ or $(1 + log(1 + x))^{-1}$.

Since the newly defined compatibility metrics are closely-related (e.g., temporal compability), the final temporal compatibility $T(e_i, e_j)$ is defined in a general form, using the Generalised Multiplicative model in Section 4.7, as:

$$T(e_i, e_j) = T_T^{\alpha_1}(e_i, e_j).T_D^{\alpha_2}(e_i, e_j).T_E^{\alpha_3}(e_i, e_j).T_O^{\alpha_4}(e_i, e_j), \tag{8}$$

for some parameters α_1, α_2, α_3 and α_4. The durations are quite important in some applications such as flight scheduling, while are less so in other applications such as stock trading. In the absence of durations d_i, temporal compatibility $T(e_i, e_j)$ can be simply expressed in terms of timestamp compatibility $T_T(e_i, e_j)$ only.

Neighborhood compatibility The basic *neighborhood compatibility* $N(e_i, e_j)$ to determine the *causal relation* between neighboring edges. For instance, one delayed flight may (directly) result in other delayed flight(s); or one stock trade may affect on other stock trades. The neighborhood promixity is defined by some path-related distance function $d(e_i, e_j)$ between e_i and e_j. Often, one is interested in bundling edges when such a distance is small.

- *Ego-centric compatibility* An ego-network $ego(x_i)$ of a node x_i contains x_i, x_i's neighbors (so-called "alters"), and the induced edges. We define the ego-network of an edge (x_i, x_j) as the union of $ego(x_i)$ and $ego(x_j)$.
  Our ego-centric compatibility $E(e_i, e_j)$ avoids bundling edges e_i and e_j that belong to different local communities or have few common neighbors. The ego-centric compatibility can be defined as: $N_E(e_i, e_j) = f(d_E(e_i, e_j))$, where $d_E(e_i, e_j)$ is on the intersection of the ego-networks of each edge, and $f(x)$ is a continuous and decreasing function: $f(x) \in$ [0,1], for all $x \geq 0$.
- *Trace compatibility* is a measure to avoid bundling edges that are very far to reach each other. The trace compatibility $T(e_i, e_j)$ is based on $d_T(e_i, e_j)$ - the smallest value of graph theoretic distances between one end of e_i to one end of e_j. This metric is similar to *connectivity compatibility* proposed by Selassie et. al [34], which is proposed to avoid bundling edges in different disconnected components in static graphs.
  For tracing, one is often interested in cases with a small value of $d_T(e_i, e_j)$ (≤ 2). An example formula for trace compatibility is defined as: $N_T(e_i, e_j) = f(d_T(e_i, e_j))$, where $f(x)$ is a continuous and decreasing function: [0,+∞) $\rightarrow$ [0,1].

Thus, the neighborhood compatibility can be defined from the ego-compatibility and the trace compatibility.

$$N(e_i, e_j) = N_E^{\alpha_1}(e_i, e_j).N_T^{\alpha_2}(e_i, e_j), \tag{9}$$

for some α_1 and α_2. A common usecase may involve either the ego-compatibility or the trace compatibility.

Spatial compatibility Spatial compatibility $S(e_i, e_j)$ is to avoid bundling edges that are very diverse in length, distance, visibility and crossing angles [25]. Though intensively studied in recent work, spatial compatibility has still been mainly applied for edge bundling of static graphs [27, 31, 34]. Here the use of spatial compatibility is extended for stream bundling in our framework to denote the spatial compatibility of a pair of stream elements.

Data-driven compatibility We extend the *semantic compatibility* [27] which was proposed for static graphs, to apply for graph streams. This data-driven metric determines the similarity of a pair of multi-attributed streamed edges. Let $\langle a_i^0, \ldots, a_i^m \rangle$ and $\langle a_j^0, \ldots, a_j^m \rangle$ denote the two data vectors of the edges e_i and e_j. The data-driven compatibility $D(e_i, e_j)$ can be defined as the similarity/dissimilarity between the two vectors.

4.5 Aggregate compatibility

The compatibility $\mathcal{C}(e_i, e_j)$ between two edges is the aggregate value of the temporal compatibility $T(e_i, e_j)$, neighborhood compatibility $N(e_i, e_j)$, spatial compatibility $S(e_i, e_j)$ and data-driven compatibility $D(e_i, e_j)$. For instance, the $\mathcal{C}(e_i, e_j)$ can be defined using the Linear model in Section 4.7 as

$$\mathcal{C}(e_i, e_j) = \eta_S.S(e_i, e_j) + \eta_T.T(e_i, e_j) + \eta_D.D(e_i, e_j) + \eta_N.N(e_i, e_j), \quad (10)$$

for some parameters η_S, η_T, η_D, η_N in $[0, 1]$ and $\eta_S + \eta_T + \eta_D + \eta_N$=1.

4.6 High-level Stream Bundling Methods

Offline methods process the whole graphs in an offline fashion and are ideal for historical data provided that all graphs are known in advance. At every sliding window, all control points for each edge can be precomputed. Thus, during streaming, the only task is to render the splines or polylines for edges. Considerate efforts are to make smooth transitions of one bundled image to the next bundle image.
Online methods process graphs as if there were no information available per priori. The approach computes control points for all edges at every sliding window. This approach works for real-time scenario in which all past information is transparent. It is the hardest case since both fast recomputation of bundles for each window and smooth transitions between frames are required.
Hybrid methods, on the other hand, may assume on history to achieve mental map preservation. The global information is assumed to help the Mapping and Routing process at each timeframe. The bundling is still performed on the fly but the transitions between frames are less challenging than the totally online approach. This approach is potentially adaptable to both the historical data and the real-time scenarios.

4.7 Aggregating Compatibility

Edge bundling performs bundling with several compatibility metrics; for example, Angle-Scale-Position-Visibility [25], Topology-Geometry-Importance in [31], Spatial-Direction-Connectivity in [34], and Spatial-Semantic in [27]. These works defines the aggregate compability value in a so-called *multiplicative model*: $\mathcal{C} = \prod_{1..k} C_i$. However, this model may not be flexible enough, to be adjustable at runtime. Analytical tasks may often need to scale the concerns differently *at runtime*.

For runtime supports, we propose several ways to compute the aggregate value $\mathcal{C}(C_1, \ldots, C_k)$ for stream bundling:

- *Generalised multiplicative model* defines $\mathcal{C} = \prod_{1..k} C_i^{\alpha_i}$,
- *Linear model* defines $\mathcal{C} = \sum_{1..k} \alpha_i C_i$, and
- *Hybrid model* combines the above models

The new models permit the parameters α_i to be adjusted at runtime for analytics with different scales of the concerns C_i's. Typically, one can use (generalised) multiplicative model for a set of *closely-related concerns*, e.g., the spatial compatibilities. The linear model is used for a set of *independent* concerns.

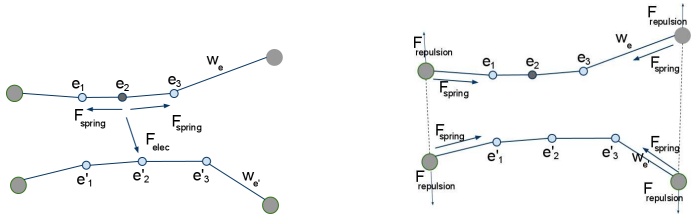
5 Stream Bundling Algorithms

Our StreamEB framework bundles graph streams by integration of the compatibility measures introduced in Section 4.4. Specifically, we introduce two specific approaches for stream bundling that realizes the StreamEB framework: Force-directed stream bundling (FStreamEB) and Tree-based stream bundling (TStreamEB).

5.1 FStreamEB: Force-directed stream bundling

Our Force-Directed stream bundling (FStreamEB) inserts control points into stream edges and applies a spring algorithm for those control points with respect to the aggregate compatibility between pair of stream edges. Three variations of the force-directed edge bundling method are proposed: a simple extension of FDEB [25] (S-StreamEB), an integration of *dynamic mapping* (e.g., Force Layout) with stream bundling (F-FStreamEB), and a version optimized for static geometry (G-FStreamEB).

Figure 4 shows the force models our FStreamEB algorithms.



(a) S-FStreamEB (b) F-FStreamEB

Fig. 4: Force models: (a) S-FStreamEB: forces applied on control points only; (b) F-FStreamEB: forces applied on nodes + control points

S-FStreamEB A simple version of our FStreamEB model extends FDEB [25] by integrating new stream compatibility measures. For a subdivision point $e_i^{(l)}$ on edge e_i, the electrostatic force model of our model is

$$F_s = \sum_{e_j \in C(e_i)} \mathcal{C}(e_i, e_j) * g(|\mathsf{p}_{e_i^{(l)}} - \mathsf{p}_{e_j^{(l)}}|, \tag{11}$$

where $\mathcal{C}(e_i, e_j)$ is the aggregate of spatial, temporal, data-driven and neighborhood compatibility measures; $C(e)$ is the set of compatible edges of e; and g is a function of $|\mathsf{p}_{e_i^{(l)}} - |\mathsf{p}_{e_j^{(l)}}|$; for example, g=$|\mathsf{p}_{e_i^{(l)}} - |\mathsf{p}_{e_j^{(l)}}|^{-d}$ of a constant d.

This method applies to the current graph G_i at every timeframe.

Figure 1 outlines our S-FStreamEB algorithm, which is adapted from FDEB algorithm.

F-FStreamEB (FStreamEB with Dynamic layout) A more general model of FStreamEB integrates a dynamic layout (Mapping) with force-directed bundling (Routing). In particular, we integrate force-directed algorithm for dynamic layout.

In the classical force-directed methods [16], the force on a node x is a combination of forces $F = F_{spring} + F_{repusion} + F_{external}$, where F_{spring} is the spring force for

input : E the edge set of current graph
$t \longleftarrow constant$;
foreach *edge* e_i *of* E **do**
  $C(e_i) \longleftarrow \{e_j | \, e_j \in \{E - e_i\} \ \wedge \ S(e_i, e_j) \geq t\}$
end
foreach *step* $s \in [0..I]$ **do**
  foreach *edge* e_i *of* E **do**
    $pnts_i \longleftarrow$ points of e_i;
    foreach *edge* e_j *of* $C(e_i)$ **do**
      $pnts_j \longleftarrow$ points of e_j;
      `force(`$pnts_i, pnts_j$`)`
    end
  end
end

Algorithm 1: S-FStreamEB

each vertex exerted by its neighbors, $F_{repusion}$ is the repulsion force between nodes, and $F_{external}$ is the force from other sources, such as a magnetic field or an anchoring force.

The general force model of our F-FStreamEB includes a new force F_{bundle}, which is the force exerted by the related bundles. This *bundle-aware* force provides feedbacks of bundling results to the node placement, as depicted in Figure 3. Figure **??** depicts the force models of S-FStreamEB and F-FStreamEB. The force on a node is therefore:

$$F = F_{spring} + F_{repusion} + F_{external} + F_{bundle} \tag{12}$$

The *bundle-aware* force F_{bundle} of a node x can be defined as:

$$F_{bundle}(x) = \sum_{e_i \in N_{out}(u)} k_e |\mathsf{p}_{e_i^{(0)}} - \mathsf{p}_x|) + \sum_{e_i \in N_{in}(x)} k_e |\mathsf{p}_x - \mathsf{p}_{e_i^{(P-1)}}|, \tag{13}$$

where $N_{in}(x)$ and $N_{out}(x)$ are the in-neighbors and out-neighbors of x, and k_e is the stiffness of the edges.

In our model, $F_{external}$ includes the gravity force, the magnetic force and the anchoring force. Specifically, the anchoring force F_{anchor} for a node x is defined as: $F_{anchor}(x) = \sum_{x \in |\mathsf{V}|} k_a |\mathsf{q}_x - \mathsf{p}_x|$, where q_x is the preferred location of node x, and k_a is the anchoring force strength.

G-FStreamEB (FStreamEB with Static Geometry) We introduce another variant, G-FStreamEB, that reduces computational costs for stream bundling when spatial information of nodes remain unchanged. This approach precomputes all compatible edges for each edge and compatibility values for every pair of edges in an "offline" step. Then during streaming, the offline information helps to avoid redundant computations (spatial compatibility and compatible edge sets) in "online" step.

Let G_i=$\{V, E_i\}$ and G=$\{V, E\}$ be the graph in current window and the underlying graph of the stream, respectively. Let $S^G(e_i, e_j)$ and $S(e_i, e_j)$ denote the spatial compatibility for edges e_i and e_j in G and G_i, respectively. With fixed geometry, $S(e_i, e_j)$ has the same value of $S^G(e_i, e_j)$. Let $C^G(e_i)$ and $C(e_i)$ denote the set of compatible edges for edge e_i in G and G_i, respectively.

– The *offline step* computes the spatial compatibility $S^G(e_i, e_j)$, and compatible edge set $C^G(e_i)$ for each edge e_i in G. The set $C^G(e_i)$ contains only edges e_j, each of which has $S^G(e_i, e_j)$ greater than a threshold $t_{off} \in [0, 1]$.
– The *online step* computes the compatible edge set $C(e_i)$ for each edge $e_i \in E_i$ by taking all the vertices in $C^G(e_i) \cap E_i$ such that the aggregate compatibility $\mathcal{C}(e_i, e_j)$ is greater than a threshold $t_{on} \in [0, 1]$. The iterations are then progressed in similar manner of the S-FStreamEB.

Figures 2 and 3 show the algorithms for the offline and online steps of our G-StreamEB.

input : **E** the edge set of the union of graphs
$t_{off} \longleftarrow constant$;
foreach *edge* e_i *of* **E** **do**
  $C^G(e_i) \longleftarrow \{e_j | \, e_j \in \{E - e_i\} \ \wedge \ S^G(e_i, e_j) \geq t_{off}\}$
end

Algorithm 2: Offline step of G-FStreamEB

input : E the edge set of current graph
$t_{on} \longleftarrow constant$;
foreach *edge* e_i *of* E **do**
  $C(e_i) \longleftarrow \{e_j | \, e_j \in C^G(e_i) \cap \{E - e_j\} \ \wedge \ \mathcal{C}(e_i, e_j) \geq t_{on}\}$;
end
foreach *step* $s \in [0..I]$ **do**
  foreach *edge* e_i *of* E **do**
    $pnts_i \longleftarrow$ points of e_i;
    foreach *edge* e_j *of* $C(e_i)$ **do**
      $pnts_j \longleftarrow$ points of e_j;
      `force(`$pnts_i, pnts_j$`)`
    end
  end
end

Algorithm 3: Online step of G-FStreamEB

5.2 TStreamEB: Tree-based stream bundling

We also introduce a new stream bundling method that extends HEB [24].

The key idea of our TStreamEB is a *double-interpolation* process which takes the aggregate compatibility $\mathcal{C}(e_i, e_j)$ between a pair of edges into account. The first interpolation is the same as the interpolation in HEB. The second interpolation $inter(P, e_j, \beta_{ij})$ where $inter$ can be defined in Section 2.3, where $\beta_{ij} = \gamma.\beta.\mathcal{C}(e_i, e_j)$, and $\gamma \in [0, 1]$ is the second bundling strength.

Compared to iterative force calculations in FStreamEB, TStreamEB is faster since only a single iteration of interpolations is performed. Our TStreamEB is not restricted to any specific type of tree structure. In fact, the tree structure may be part of the input given from the application domain, or may be from hierarchical clustering algorithms.

Figure 4 shows the algorithm of our TStreamEB bundling method.

Figure 5 depicts the interpolations in our TStreamEB.

input : E the edge set of current graph; H the hierarchy
foreach *edge* e_i *of* E **do**
  $ctrl(e_i) \longleftarrow$ points along H from e_i^{start} to e_i^{end};
  $pnts_i \longleftarrow$ `inter(`$ctrl(e_i), e_i, \beta$`);`
  foreach *edge* e_j *of* $C(e_i)$ **do**
    $pnts_i \longleftarrow$ `inter(`$pnts_i, e_j, \beta_{ij}$`);`
  end
end

Algorithm 4: TStreamEB(E,H)

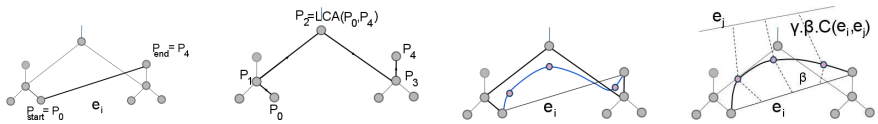


Fig. 5: TStreamEB: hierarchy and double-interpolation

5.3 Time Complexity

The most expensive step for stream bundling methods is the computation of the compabitility between all pairs of edges and thus require $\Omega(M^2)$ time, where M is the number of edges $|E_i|$ in the current graph G_i.

- S-FStreamEB takes $O(IM^2P)$ time at every timestep, where I is the number of iterations, and P is the number of control points per edge. The offline step of G-FStreamEB requires $O(|E|^2)$ time to compute the compatibility between all pairs of edges, where $|E|$ is the number of edges in the union graph G. For F-FStreamEB, the anchor forces need $O(N)$ time and the bundle-aware forces at the last iteration require $O(M^2)$ time.
- TStreamEB uses a double-interpolation of $O(M^2P)$ time, where P is the average number of control points per edge. The edge smoothing in the post-processing step takes $O(MPI_{smooth})$ where I_{smooth} is the number of iterations in the smoothing process. The hierarchy requires $O(N)$ time, where N is the number of nodes. When such hierarchy is fixed, it does not require recalculations.

6 Implementation

We have implemented a prototype of our StreamEB framework based on GEOMI visualization framework [4]. Particularly, we have implemented the FStreamEB (with its three variations S-, G- and F-) and TStreamEB methods that integrate the stream compatibility measures described in Section 4.4. In most experiments, we set the default values of compatibility parameters to η_S= 0.7, η_T= 0.1, η_N= 0.1, η_D= 0.1.

Since our studies are not involved in hierarchical data, we simply use a geometric clustering – a Quad tree decomposition implementation in FADE [33], for our TStreamEB bundling method.

All figures in this paper are produced by our prototype. All time measurements were conducted on a 1.73GHz Intel Quad Core i7-740QM laptop with 6GB L3 cache, ATI

Radeon HD5000 GPU, Ubuntu 10.04 OS and OpenGL enabled. The experiments ran on jdk-1.7 and j3d-1.5.2.
Animation We have also provided some animations[2] for the case studies.

6.1 Rendering

Holten and van Wijk [25] suggested a continuous interpolation *inter* between straight edges and bundled edges for smoothing curves and for users to understand the bundle structure. We adopt that approach for our FStreamEB and TStreamEB methods. Edges are then rendered using Java3D.

6.2 Bundling parameters

Our framework provides UI for users to adjust bundling parameters for analytic tasks. For example, users can select the bundling parameters for temporal, neighborhood, spatial and data-driven compatibility at runtime for both F-FStreamEB and TStreamEB. Users can also select parameters, such as $F_{repulsion}$, F_{spring}, F_{bundle} and F_{anchor} of the Force-layout in F-FStreamEB.

To illustrate this, Figure 7 shows different bundled results of a random graph using S-FStreamEB. The random graph has ten nodes and edges are randomly connected. Each edge has a random timestamp and a data vector of three random integers. For TStreamEB, users may also alter the bundling parameters β, γ to achieve different bundling results. Figure 8 depicts different bundled results of the same random graph using force-directed edge bundling and TStreamEB.

6.3 User Interactions

Our StreamEB framework provides users with geometric zooms, high-resolution screen capturing and movie recording.

6.4 Sliding windows

Two types of sliding windows, sequence-based and time-based, have been implemented. Users can switch between one type to the other at runtime. The GUI allows users to change the window size by varying a slider.

6.5 Interpolation

We have also implemented a simple interpolation scheme for showing transitions between consecutive bundled layouts. After computing the next bundled layout, we interpolate all control points of an edge e_i in previous layout to newly computed locations. The interpolation helps tracking changes of bundles in both replay and online scenarios.

[2] See pictures and movies at http://www.it.usyd.edu.au/∼qnguyen/streameb/

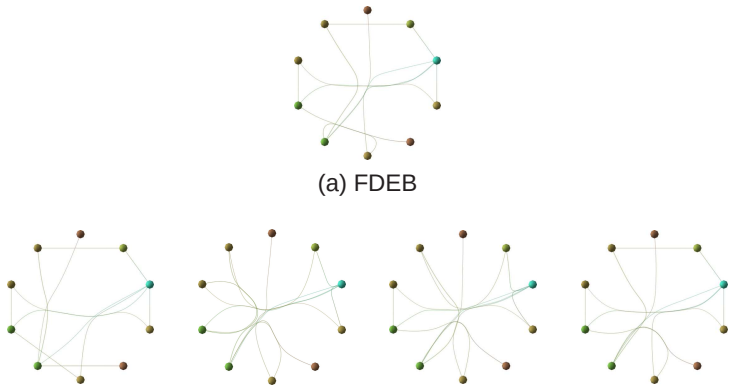


Fig. 6: Visualizations of a random graph using (a) FDEB and (b) S-FStreamEB: with parameters η_S,η_T,η_D,η_N) of (0.7,0.3,0,0), (0.7,0,0.3,0), (0.7,0,0,3); and (e) and (0.7,0.1,0.1,0.1), respectively, from left to right.

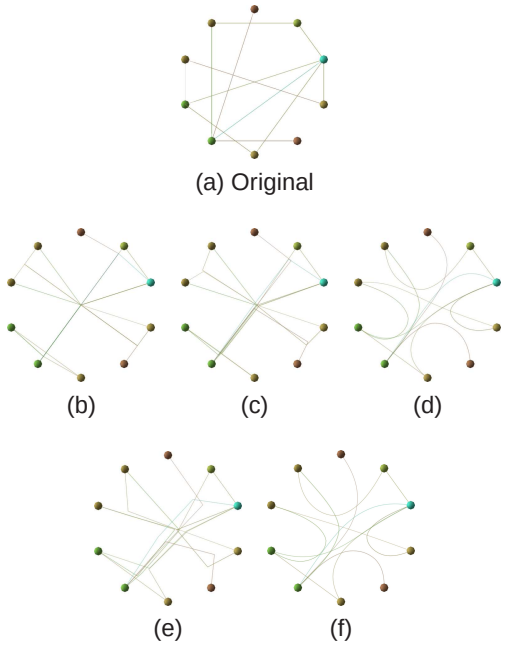


Fig. 7: Visualizations of TStreamEB: (b) (β,γ)=(1,0);(c) (β,γ)=(1,0.2) and no smoothing; (d) (β,γ)=(1,0.2) with smoothing; (e) (β,γ)=(1,0.8) w/o smoothing; (f) (β,γ)=(1,0.8) with smoothing

7 Data sets

7.1 Flight Data

The US flight dataset contains the schedules and the actual delays of US flights from year 1987 to year 2008. The number of flights varies from over one million to over seven millions per year. In our studies, we only keep all flights that actually departed and landed.

The most relevant attributes from the total of 29 attributes of each flight record include origin, destination, date-time, actual duration, scheduled duration, departure delay, arrival delay and distance. Our study considers only flights that connect between US mainland airports.

Table 1 depicts general statistics of the dataset for each year. Over the years, the total number of edges $|\mathbf{E}|$ range from 3149 to 5200; and the total number of flights range from around one million to over seven millions.

Year	$\|\mathbf{E}\|$	$\|Stream\|$	Year	$\|\mathbf{E}\|$	$\|Stream\|$	Year	$\|\mathbf{E}\|$	$\|Stream\|$
1987	3440	1287334	1995	3351	5219141	2003	4320	6375690
1988	3804	5126499	1996	3183	5209327	2004	4409	6987730
1989	3647	4925483	1997	3153	5302000	2005	4487	6992839
1990	3684	5110528	1998	3149	5227052	2006	4655	7003803
1991	3694	4995006	1999	3242	5360019	2007	5032	7275289
1992	3572	5020652	2000	3342	5481304	2008	5200	6855030
1993	3419	4993588	2001	3530	5723674			
1994	3370	5078412	2002	3213	5197861			

Table 1: US Flight dataset

Color encodings For each flight, a positive delay indicates an actual delay, while a negative delay means earlier-than-scheduled. To further separate departure-arrival delays, we use our proposed *double-circle* metaphors for nodes. Each node is represented by two concentric circles: the inner/outer circle represents aggregate value for incoming/outcoming edges. For both sides, the aggregate values are scaled and assigned from red to green gradient colors. Node radius (outer circle) is based on the out-value; the radius of innermost circle is halved. Edges have gradient colors of the out-colors of their end nodes. We use the coloring scheme for individual flight; that is, the color of end nodes are updated after each flight.

7.2 Financial Data

We use the trading data from Thompson Reuters, which consists of time series feeds of stock trades. Our case study aims to identify important traders, groups of traders and the dynamic money flows among them. Four datasets were downloaded from following equity exchanges: TSX Venture - Canada (V), Pure Trading-Canada (GO), OMX Nordic Exchange - Denmark (CO) and Stockholm SE - Sweden (ST). Each dataset contains transactions in the last week of May 2011 from each exchange. The datasets are named by the corresponding stock exchanges, e.g., V, GO, CO and ST.

The most relevant attributes from the 44 attributes in the data include buyer, seller, data-time, price and volume. The total number of traders in the datasets are 93 (V), 85 (GO), 71 (CO) and 89 (ST), , respectively. The total number of traded securities (stocks) are 1956 (V), 1315 (GO), 217 (CO) and 240 (ST), respectively.

The total number of trading connections between traders over the entire period are 3149, 1107, 6336 and 2955, respectively; and the number of transactions are 249800, 196662, 18545932 and 188488 for datasets CO, GO, ST and V respectively.

Color encodings For analysis of the trading volumes between traders, two following color encodings are used. *Total-based coloring* colors traders based on the traders' total traded volumes (both buys and sells), using blue to red gradient colors. Node size is proportional to the trader's performance. Edges are colored based on the interpolated colors of their end nodes. On the other hand, *buy-sell coloring* distinguishes active buyers from active sellers by using the double-circle metaphor for nodes. Each node is represented by two concentric circles: the inner/outer circle represents aggregate buys/sells value.

Ex.	$\|\mathbf{E}\|$	$\|Stream\|$	Ex.	$\|\mathbf{E}\|$	$\|Stream\|$
CO	3149	249800	GO	1107	196662
ST	6336	18545932	V	2955	188488

Table 2: Stock Trading data

The total number of securities (stocks) that were traded between the traders are 217 (CO), 1315 (GO), 240 (ST), 1956 (V), respectively. The total number of trading connections between traders over the entire period are 3149, 1107, 6336, 2955; and the number of transactions are 249800, 196662, 18545932, 188488 for datasets CO, GO, ST and V respectively.

8 Experimental Results

8.1 Visual analysis on US flights data

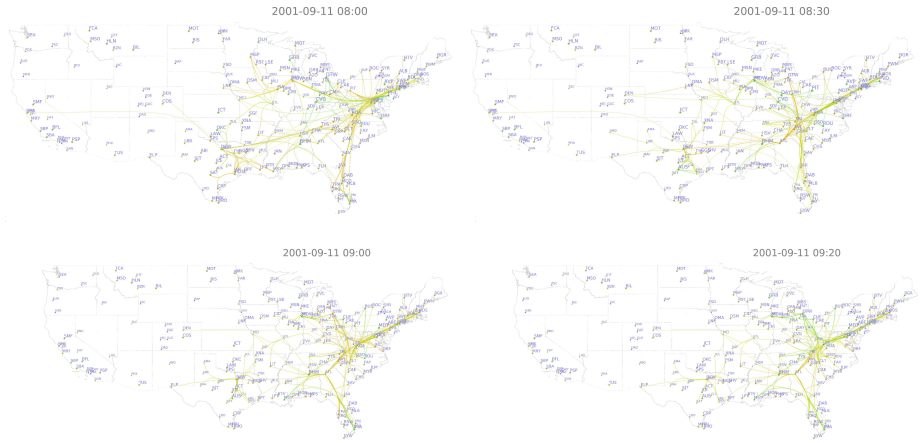
We used US flights data[3] to show the effect of the terrorist attacks on 11-Sep-2001, since most of flights had been canceled due to the event.

The effect of the attacks was significant, for instance, all flights were canceled after 9:20am for half a day and there was a single flight on the next day.

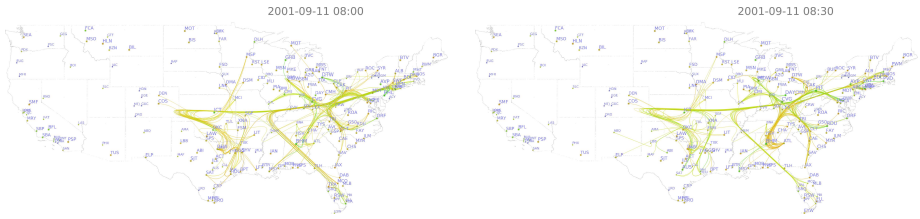
Common analytic tasks include identifying the airports that have suffered from severe delays, pairs of airports that often had delays, US regions where most flight delays occurred, and correlations between geographic distances and delays.

Analysis of flight delays Figure 6a applies S-FStreamEB on the windows taken from 8am to 9:20am, before and after the terrorist attacks on 11-Sep-2001. The figure shows a significant drop of flights from 8am window to 8:30am window and from 9:00am window to 9:20am window. It also shows that many flights actually departed earlier (red) than scheduled in the 8am window. However, many flights were delayed (green) around 9am to 9:20am - after the final attack and before the shutdown period.

[3] US Flights http://stat-computing.org/dataexpo/2009/the-data.html



(a) S-FStreamEB



(b) TStreamEB

Fig. 8: US Flight delays before and after the 11-Sep-2001 attacks. Visualization of W=160 sequence-based windows using S-FStreamEB and TStreamEB

Figure 6b depicts the visualization of flights using our TStreamEB. Most flights from 8:00am to 8:30am were within the Eastern US region. A few flights were delayed (green-yellow), while many of them departed earlier (red-yellow).

Geographic comparison Figures 6a and 6b show that most flights from 8am to 9:20am were within the Eastern US region.

They also show the delay distribution of the flights across US cities over time. The figures, thus, may help with the common tasks in flight scheduling/monitoring.

Group of flights The figures show the bundles of flights that are closely related. Furthermore, it also show the evolution of connecting flights and flight delays among a group of flights. For instance, a small group of airports in the Middle-South region of US, which had many flights earlier than the schedule at 8am, got delayed from 8:30am onward.

8.2 Visual analysis on trading data

The most common tasks include, for example, determining frequent traders, frequent trading pairs, and groups of traders that have similar trading behaviours. By examining those aspects, it helps to answer sophisticated questions in stock market surveillance, such as detecting market manipulation.

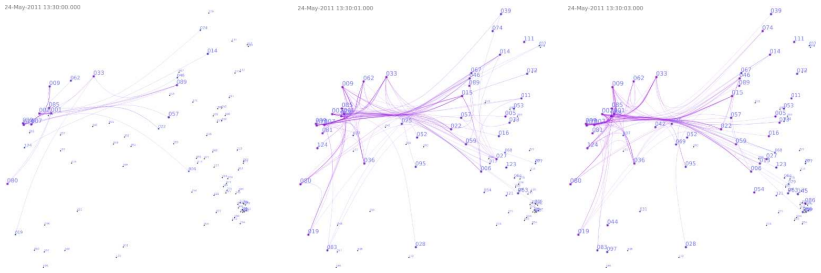


Fig. 9: Visualization of the dataset V using MDS and S-FStreamEB: sequence-based sliding window W=160

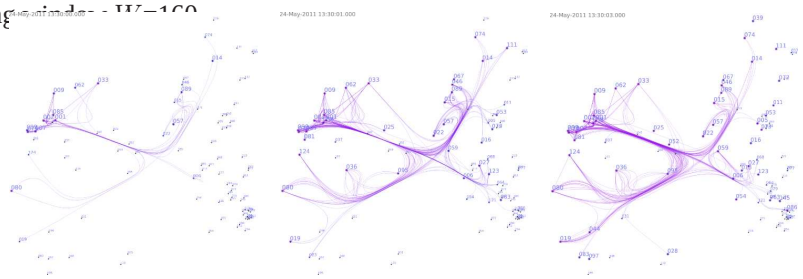


Fig. 10: Visualization of the dataset V using MDS and TStreamEB. Sequence-based sliding window W=160

We use the trading data from Thompson Reuters[4], which consists of time series feeds of stock trades. Our analysis includes overviews of the performance of traders and groups of traders over time; and showing evolution of the money flows.

Analysis of Trader Performance Figures 9 and 10 show how active the traders are in terms of buying and selling. Figure 9 applies S-FStreamEB and MDS layout on the sequence-based sliding windows W= 160. Figure 10 applies TStream with β= 1 and γ= 0.2, and MDS layout.

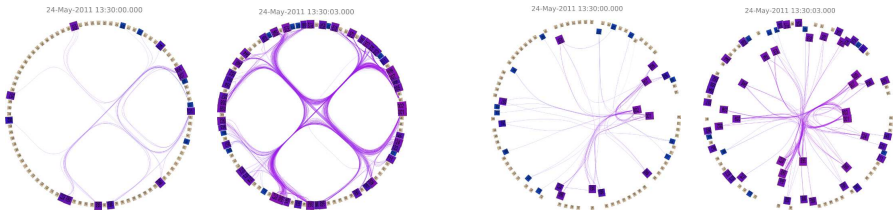
The total trade patterns in the dataset V at market open time is depicted in Figure 11(a). The figure applies circular layout and TStreamEB method with β= 1 and γ= 0.2. For similar analysis, Figure 11(b) applies F-FStreamEB on sliding windows W= 160 with anchoring forces that anchor vertices to a locations from a circular layout.

From the figures, the most active traders are 001, 002, 007 and 009; while most other traders still remain inactive at the time. Some traders such as 005 and 019, traded a moderate amount at first (few minutes), but then they traded more over time.

Analysis of Trades between Groups Figures 9 shows four groups of traders located at the corners. The most active traders are located at top-left corner. Then second-most, third-most and the least active traders are located at top-right corner, bottom-left corner and bottom-right corner, respectively.

Better pictures of trading groups and their trading patterns are depicted in Figure 10. The figure also shows that the most active group traded most frequently with the second, the third and the least active groups in that order. Interestingly, there were less trades between the second and the third compared to the number of trades between the second and the least.

[4] Thomson Reuters http://thomsonreuters.com/



(a) (b)

Fig. 11: Visualization of the dataset V (a) the first few seconds of market opening, 500 seconds windows, using Circular layout and TStreamEB; (b) using F-FStreamEB: Anchoring positions from Circular Layout. Sequence-based sliding window W=160

Analysis of Market Manipulation Figure 9 and Figure 10 show that a number of active traders (top-left) are "frequent" traders whose the total trade volumes are fairly small. For instance, traders 088 and 124 are fairly blue at the first 30 minutes. Whilst some traders in the least active group had traded a large volume at the very first moment; for instance, traders 027, 066 and 068.

These abnormal trade patterns are useful for detecting price/volume manipulations. When combined with more specialized tools for stock analysis, these may be very useful for replaying and confirming market manipulation cases.

8.3 Performance

We compared the performance of our three force-directed variations (the S-FStreamEB, G-FStreamEB and F-FStreamEB) and our tree-based bundling (TStreamEB). The number of iterations I in all S-, G- and F- variants was set to 20; while the number of smoothing iterations I_{smooth} in TStreamEB was set to 10.

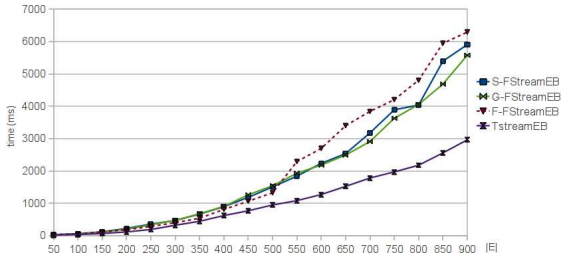


Fig. 12: Runtime comparisons of stream bundling methods.

For our experiments, a thousand graph instances of at most nine hundred edges were randomly extracted from graph streams of the flight dataset. We ran each method separately and repeated experiments three times. Figure 12 shows average runtime measurements. The figure shows that TStreamEB is the fastest, as it requires only a single iteration. For large graphs, the G- variant outperformed all the other variants of FStreamEB, because G-FStreamEB avoided redundant computations of spatial compatibility and compatible edges. The bundle-aware forces added little overheads to F-FStreamEB due to the computation of force layout.

9 Discussions and Future work

Applicability Our bundled results help analytics of graph streams and particularly, reduce visual clutter in the circular layouts and the MDS layouts.
Limitations TStreamEB currently uses a synthetic QuadTree structure to guide the bundling process and may result in edge bundles that might lead to misleading interpretations. Thus, our future work is to study real hierarchical data.

The framework and methods introduced in this paper are not necessary better than existing works in terms of the quality of bundled results. Our framework is to demonstrate the usability of edge bundling of streams.
Scalability Our future work will consider Barnes-Hut algorithm to improve time complexity of FStreamEB to $O(IMlogMP)$ for each step.

Nevertheless, the estimation of compatibility values for all pairs requires $O(M^2)$ and is still dominant. To overcome this limitation, future work might investigate k-means clustering and hierarchical clustering methods to identify top k edges with highest compatility, such as, studied in Gansner et al. [20] to reduce time complexity. Such the clustering methods need also to be adapted for graph streams.
Interaction Our prototype implementation provides a few interaction techniques for user inputs and zooming. Our future work is to consider ways to interact with edge bundles more effectively. Specifically, the techniques [36] for *semantic zoom* are worth considering.
Representation of time In general, there are two ways to represent time in graph drawing: (1) animation, (2) a spatial dimension. For streamed graph, using animation is more relevant because the whole graph is not known a priori.

For animation, the mapping from data time to picture time is important. In offline representations of time, this mapping can be complex. In real time applications, it is often simple as the mapping is the identity function. That is, one may use the identity function, a linear scale function or a non-linear function for the time mapping.

Our case studies are a replay of real time scenarios; thus, the mapping from data time to picture time can become complex again. For example, in the 9/11 airline traffic data, there is a long period (in data time) that nothing happens. For picture time, this can be compressed.
Other metrics Due to the high update rates of graph streams, edge directions are not yet considered in this study. Also, the edge importance is not taken into account. Meanwhile, the clustered graph model in [31], can be easily adapted to our StreamEB framework.

10 Conclusion

This paper has presented the first study of edge bundling for graph streams. A new framework StreamEB is proposed integrating several compatibility measures, with new force-directed and tree-based bundling approaches for bundling stream elements. A prototype of the framework has been implemented in GEOMI framework including the new stream bundling methods. The experimental results on US flights datasets and financial datasets have shown that edge bundling can help to reduce visual clutter of the stream

visualizations. In fact, our stream bundling approaches are quite promising for helping stream mining/reasoning tasks.

References

1. J. Abello, I. Finocchi, and J. Korn. Graph sketches. In *InfoVis*, 2001.
2. C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for clustering evolving data streams. In *VLDB*, volume 29, pages 81–92. VLDB Endowment, 2003.
3. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*, volume 40. Springer-Verlag New York Inc, 2010.
4. A. Ahmed, T. Dwyer, M. Forster, X. Fu, J. W. K. Ho, S.-H. Hong, D. Koschtzki, C. Murray, N. S. Nikolov, R. Taib, A. Tarassov, and K. X. 0003. Geomi: Geometry for maximum insight. In P. Healy and N. S. Nikolov, editors, *Graph Drawing*, volume 3843 of *Lecture Notes in Computer Science*, pages 468–479. Springer, 2005.
5. N. Andrienko and G. Andrienko. *Exploratory analysis of spatial and temporal data: a systematic approach*. Springer Verlag, 2006.
6. A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *VLDB*, 15(2):121–142, 2006.
7. Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SCDA*, pages 623–632, 2002.
8. D. Barbieri, D. Braga, S. Ceri, E. Valle, and M. Grossniklaus. Querying rdf streams with c-sparql. *ACM SIGMOD Record*, 39(1):20–26, 2010.
9. C. Binucci, U. Brandes, G. Di Battista, W. Didimo, M. Gaertler, P. Palladino, M. Patrignani, A. Symvonis, and K. Zweig. Drawing trees in a streaming model. In *Graph Drawing*, pages 292–303. Springer, 2010.
10. I. Boyandin, E. Bertini, and D. Lalanne. Using flow maps to explore migrations over time. In *Geospatial Visual Analytics Workshop*, volume 2, 2010.
11. U. Brandes and M. Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In *Graph Drawing*, pages 99–110. Springer, 2012.
12. U. Brandes and D. Wagner. A bayesian paradigm for dynamic graph layout. In *Graph Drawing*, pages 236–247. Springer, 1997.
13. G. Chin, M. Singhal, G. Nakamura, V. Gurumoorthi, and N. Freeman-Cadoret. Visual analysis of dynamic data streams. *InfoVis*, 2009.
14. W. Cui, H. Zhou, H. Qu, P. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *TVCG*, pages 1277–1284, 2008.
15. W. De Pauw and H. Andrade. Visualizing large-scale streaming applications. *InfoVis*, 2009.
16. P. Eades. A heuristic for graph drawing, 1984.
17. O. Ersoy, C. Hurter, F. Paulovich, G. Cantareiro, and A. Telea. Skeleton-based edge bundling for graph visualization. *TVCG*, 17(12):2364–2373, 2011.
18. C. Erten, S. Kobourov, V. Le, and A. Navabi. Simultaneous graph drawing: Layout algorithms and visualization schemes. In *Graph Drawing*, pages 437–449. Springer, 2004.
19. J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the streaming model: the value of space. In *SODA*, pages 745–754, 2005.
20. E. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *IEEE PacificVis*, pages 187–194, 2011.
21. E. Gansner and Y. Koren. Improved circular layouts. In *GD*, 2006.
22. E. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *GD*, 2005.
23. S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *TODS*, 31(1):396–438, 2006.

24. D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *TVCG*, pages 741–748, 2006.
25. D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.
26. M. Huang, P. Eades, and J. Wang. On-line animated visualization of huge graphs using a modified spring algorithm. *Journal of Visual Language and Computing*, 9(6):623–645, 1998.
27. W. Kienreich and C. Seifert. An application of edge bundling techniques to the visualization of media analysis results. In *InfoVis*, pages 375–380, 2010.
28. A. Lambert, R. Bourqui, and D. Auber. Winding Roads: Routing edges into bundles. In *Computer Graphics Forum*, volume 29, pages 853–862. Wiley Online Library, 2010.
29. K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of visual languages and computing*, 6(2):183–210, 1995.
30. J. Moody, D. McFarland, and S. Bender-deMoll. Dynamic network visualization1. *American Journal of Sociology*, 110(4):1206–1241, 2005.
31. Q. Nguyen, S. Hong, and P. Eades. TGI-EB: A new framework for edge bundling integrating topology, geometry and importance. In *Graph Drawing*, pages 123–135. Springer, 2011.
32. L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *IEEE ICDE*, pages 685–694. IEEE, 2002.
33. A. Quigley and P. Eades. Fade: Graph drawing, clustering, and visual abstraction. In *Graph Drawing*, pages 197–210, London, UK, 2001. Springer-Verlag.
34. D. Selassie, B. Heller, and J. Heer. Divided edge bundling for directional network data. *TVCG*, 17(12):2354–2363, 2011.
35. J. Sun, C. Faloutsos, S. Papadimitriou, and P. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *SIG KDD*, pages 687–696, 2007.
36. N. Wong, S. Carpendale, and S. Greenberg. Edgelens: An interactive method for managing edge congestion in graphs. In *InfoVis*, 2003.
37. H. Zhou, X. Yuan, W. Cui, H. Qu, and B. Chen. Energy-based hierarchical edge clustering of graphs. In *IEEE PacificVis*, pages 55–61. IEEE, 2008.

ISBN 978-1-74210-280-1

School of Information Technologies
Faculty of Engineering & Information Technologies
Level 2, SIT Building, J12
The University of Sydney
NSW 2006 Australia

T +61 2 9351 3423
F +61 2 9351 3838
E sit.information@sydney.edu.au
sydney.edu.au/it

ABN 15 211 513 464
CRICOS 00026A