StreamEB: Stream Edge Bundling*

Quan Nguyen**, Peter Eades, and Seok-Hee Hong

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

Abstract. *Graph streams* have been studied extensively, such as for data mining, while fairly limitedly for visualizations. Recently, *edge bundling* promises to reduce visual clutter in large graph visualizations, though mainly focusing on *static* graphs.

This paper presents a new framework, namely **StreamEB**, for edge bundling of graph streams, which integrates temporal, neighbourhood, data-driven and spatial compatibility for edges. Amongst these metrics, *temporal* and *neighbourhood compatibility* are introduced for the first time. We then present force-directed and tree-based methods for stream edge bundling. The effectiveness of our framework is then demonstrated 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 structure data produced from modern applications, such as, social communication networks like Facebook and Twitter, airline monitoring and stock market surveillance systems. 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, ranging from social network metrics, to node clustering of multi-dimensional data [5, 6, 26].

However, a comparatively tiny number of works have focused on visualization of graph streams [4, 9, 12, 14, 30]. *Stream graph visualization* differs from traditional *dynamic graph visualization* in that: (a) the size of graph streams can be very huge, which typically involve a long *time* dimension; (b) the data comes in incrementally as a stream and possibly in considerably high speed; (c) the entire graph is not always available in streaming context; and (d) the response time may be a relevant factor to design visualization methods.

Edge bundling becomes popular in Graph Drawing and Information Visualization communities to reduce visual clutter for high-level edge patterns [13, 18–20, 20–22]. 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. Our research is based on a premise that the edge connection patterns may change fairly fast, but “edge bundles”

* See the full version of the paper [25].

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

should change much slower. The main goal is an exploration of how edge bundling can help reduce visual clutter in graph stream visualizations, to assist the key tasks of community detection and change detection in the underlying network.

We present a new framework, namely **StreamEB**, which is the first work that addresses edge bundling for graph streams. Our research also addresses *flexibility* (e.g., adaptive to user inputs at *runtime*) of methods to extract high-level patterns in graph streams. Our framework integrates several compatibility measures such as *temporal*, *neighborhood*, *data-driven* and *spatial* compatibility for stream bundling. For these metrics, *temporal* compatibility and *neighborhood* compatibility are introduced for the first time, whereas the other compatibility metrics are carefully adapted for graph streams. We then 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 stock 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.

2 Related Work

2.1 Stream Algorithms

Streaming research has extensively studied for graph problems [16], such as distance and connectivity estimation. Node clustering determines groups of nodes from the link density including graph-partitioning, minimum-cut and dense subgraph determination.

Window-processing [8] is one of the most common techniques, which extracts windows from the stream the last data stream elements. Such extraction can be *sequence-based* (i.e., a given number of tuples) or *time-based* (i.e., all the tuples which occur during a given time interval).

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 [4, 9, 12, 14, 30]. Visualization of graph streams is harder due to the dynamics of the streams.

For dynamic graphs, it is critical to be able to see the statistical trends and changes over time, while preserving user’s mental map [23]. The most common techniques for representing temporal data are via *animation* and the “*small multiples*” display.

The broad challenges in dynamic graph visualization are often involved in the control between *readability* and *stability* (see a recent survey [11]), and the visual clutter often incurred in the visualizations.

2.3 Edge Bundling

Edge bundling has been quite successful in reducing clutter and has been extensively studied including hierarchical edge bundling [19], geometry-based edge clustering [13, 19, 22, 31], force-directed edge bundling [20, 21, 24, 28] and multi-level agglomerative edge bundling [17]. However, most existing bundling techniques are not concerned with dynamic graphs “directly”. There is an isolated study in [10] applying force-directed edge bundling method [20] to yearly migration graphs and then displaying the bundled results in small multiples for analysis of the US migration flows over year. Another work proposes the use of edge bundling in a so-called 1.5D visualization [29]; a limited number of time-data points are centered of the proposed metaphor and edges of the same time are routed via the time nodes.

Hierarchical edge bundling (HEB) [19] draws edges along the associated paths in a hierarchy. Intuitively, HEB uses a “single step” interpolation of individual edges w.r.t. the “backbone” tree as a compatibility between pairs of edges. This single interpolation can be expressed as $inter(ctrl(e_i), e_i, \beta)$, 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 $\beta \in [0, 1]$.

Force-directed Edge Bundling (FDEB) [20] uses a force-directed method to position the control points on each edge. 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 :

$$F_{e_i^{(l)}} = k_e(|p_{e_i^{(l-1)}} - p_{e_i^{(l)}}| + |p_{e_i^{(l)}} - p_{e_i^{(l+1)}}|) + F_s \quad (1)$$

$$F_s = \sum_{e_j \in C(e_i)} S(e_i, e_j) * |p_{e_i^{(l)}} - p_{e_j^{(l)}}|^{-d}, \quad (2)$$

where k_e is the stiffness of the edges, $C(e_i)$ 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 for different bundling criteria in various application domains [21, 24, 28].

3 StreamEB Framework

3.1 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 $(e_0, e_1, \dots, e_i, \dots)$. Each element e_i has a general form of $(x_i, y_i, t_i, [d_i], a_i^1, \dots, a_i^m)$, where:

- (x_i, y_i) is a directed edge encountered at the time-stamp t_i , where $x_i, y_i \in V$.
- d_i is the *duration* associated with the edge (x_i, y_i) at time t_i . The edge e_i exists 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.

Fig. 1. Stream Bundling Pipeline

- each incoming streamed edge has multiple attributes $a_i^1, \dots, a_i^m$, 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* $G_W^{(T)}(t_i)$ 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 W_i to denote the current window of some size W .

3.2 General Model for Stream Bundling

Fig. 1 depicts the general stream bundling pipeline and our StreamEB framework.

- *Data Input* includes the main input source (Stream Data) of a sequence of tuples $(x_i, y_i, t_i, d_i, a_i^1, \dots, a_i^m)$, 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*.

3.3 Aggregating Compatibility

The aggregate compatibility value is typically addressed in a so-called *multiplicative* model: $\mathcal{C} = \prod_{1..k} C_i$ in a number of works [20, 21, 24, 28]. However, for stream bundling we propose several alternatives for aggregating compatibilities:

- *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,

where the parameters α_i can be adjusted at runtime to scale C_i 's for analysis needs. One can use (generalised) multiplicative model for *closely-related* metrics (e.g., the spatial compatibilities Angle-Scale-Position-Visibility [20]), whilst linear model is for *independent* metrics.

3.4 Compatibility Metrics

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

Temporal Compatibility. We introduce a new measure, namely “*temporal compatibility*” for stream elements. Temporal compatibility $T(e_i, e_j)$ 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. Timestamp compatibility $T_T(e_i, e_j)$ aims to avoid bundling edges that have huge difference in timestamps. Often one prefers analyzing 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; 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 end-time compatibility $T_E(e_i, e_j)$ and time-overlapping compatibility $T_O(e_i, e_j)$ can be defined as:

$$T_T(e_i, e_j) = f(|t_i - t_j|) \quad (3)$$

$$T_D(e_i, e_j) = f(|d_i - d_j|) \quad (4)$$

$$T_E(e_i, e_j) = f(|t_i + d_i - t_j - d_j|) \quad (5)$$

$$T_O(e_i, e_j) = f(o(e_i, e_j)), \quad (6)$$

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

The final temporal compatibility $T(e_i, e_j)$ is defined in a general form as:

$$T(e_i, e_j) = T_T^{\alpha_1}(e_i, e_j) \cdot T_D^{\alpha_2}(e_i, e_j) \cdot T_E^{\alpha_3}(e_i, e_j) \cdot T_O^{\alpha_4}(e_i, e_j), \quad (7)$$

for some parameters $\alpha_1, \alpha_2, \alpha_3$ 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 (i.e., trades are never expired generally, and only a few cases trades can get cancelled). 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.

Neighbourhood Compatibility is a measure of 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 the other stock trades. The neighbourhood proximity $N(e_i, e_j)$ 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 $e_i = (x_i, x_j)$ 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 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 $N_T(e_i, e_j)$ 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 [28], which tries 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: $[0, +\infty) \rightarrow [0, 1]$.

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

$$N(e_i, e_j) = N_E^{\alpha_1}(e_i, e_j) \cdot N_T^{\alpha_2}(e_i, e_j), \quad (8)$$

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

Spatial Compatibility aims to avoid bundling edges that are very diverse in length, distance, visibility and crossing angles [20]. Spatial compatibility has still been intensively studied but mainly for edge bundling of static graphs [21,24,28]. Here we extend the notion to denote the spatial compatibility $S(e_i, e_j)$ of a pair of stream elements.

Data-Driven Compatibility. We extend the *semantic compatibility* [21] which was proposed for static graphs, to apply for graph streams. Let $\langle a_i^0, \dots, a_i^m \rangle$ and $\langle a_j^0, \dots, a_j^m \rangle$ denote the two data vectors of the edges e_i and e_j . This data-driven metric $D(e_i, e_j)$ determines the similarity of a pair of multi-attributed streamed edges as the similarity/dissimilarity between the two vectors.

3.5 Aggregate Compatibility

The final compatibility $\mathcal{C}(e_i, e_j)$ between two edges is the aggregate value of temporal, neighbourhood, spatial and data-driven compatibility metrics:

$$\mathcal{C}(e_i, e_j) = \eta_S \cdot S(e_i, e_j) + \eta_T \cdot T(e_i, e_j) + \eta_D \cdot D(e_i, e_j) + \eta_N \cdot N(e_i, e_j), \quad (9)$$

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

4 Stream Bundling Algorithms

There are two approaches to compute bundled layout of the current window W_i (see notations in Section 3.1). The first approach does local updates on the bundled result of the previous window with respect to the recent changes. The second approach considers W_i globally rather than just the recent changes. The former is faster but could lead to results in which edges have swinging effects. Thus, we focus on the latter approach.

Our StreamEB framework bundles graph streams by integration of the compatibility measures introduced in previous section. We introduce specific methods that realizes the StreamEB framework: FStreamEB (Force-directed stream bundling) and TStreamEB (Tree-based stream bundling).

4.1 FStreamEB: Force-Directed Stream Bundling

Our 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 [20] (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).

S-FStreamEB (*Simple version of FStreamEB*) extends FDEB [20] 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

$$F_s = \sum_{e_j \in C(e_i)} \mathcal{C}(e_i, e_j) * g(|\mathbf{p}_{e_i^{(l)}} - \mathbf{p}_{e_j^{(l)}}|), \quad (10)$$

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

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

The general force model of our F-FStreamEB extends the classical force-directed methods [15]. The force on a node x is a combination of forces:

$$F = F_{spring} + F_{repulsion} + F_{external} + F_{bundle}, \quad (11)$$

where:

- F_{spring} is the spring force between each vertex and its neighbors
- $F_{repulsion}$ is the repulsion force between nodes, and
- $F_{external}$ is the force from other sources. In our model, $F_{external}$ includes the gravity force, the magnetic force and the anchoring force. Specifically, the anchoring force for a node x is defined as: $F_{anchor}(x) = \sum_{x \in |V|} k_a |q_x - \mathbf{p}_x|$, where q_x is the preferred location of node x , and k_a is the anchoring force strength.
- F_{bundle} is a new type of force exerted by the related bundles. This *bundle-aware* force F_{bundle} provides feedbacks of bundling results to the node placement (depicted in Figure 1). The F_{bundle} of a node x can be defined as:

$$F_{bundle}(x) = \sum_{e_i \in N_{out}(x)} k_e |\mathbf{p}_{e_i^{(0)}} - \mathbf{p}_x| + \sum_{e_i \in N_{in}(x)} k_e |\mathbf{p}_x - \mathbf{p}_{e_i^{(P-1)}}|, \quad (12)$$

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.

G-FStreamEB (*FStreamEB with Static Geometry*) is the third variant that aims to reduce computational costs for stream bundling of the domains in which spatial information of nodes are unchanged. This approach precomputes all compatible edges for each edge and compatibility values for every pair of edges in an “offline” step, which is taken once prior to the streaming. The computed offline information helps avoid redundant computations (spatial compatibility and compatible edge sets) in “online” step which processes data streams on-the-fly.

4.2 TStreamEB: Tree-Based Stream Bundling

We also introduce a new stream bundling method that extends HEB [19]. 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 $\beta_{ij} = \gamma \cdot \beta \cdot \mathcal{C}(e_i, e_j)$, and $\gamma \in [0, 1]$ is the second bundling strength.

Our TStreamEB is not restricted to any specific type of tree structure, which may be part of the input given from the application domain, or may be from hierarchical clustering algorithms.

4.3 Time Complexity

The computation of the compatibility between all pairs of edges requires $\Omega(M^2)$ time, where $M=|E_i|$ is the number of edges in the current graph W_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. For F-FStreamEB, the anchor forces need $O(N)$ time and the bundle-aware forces at the last iteration require $O(M^2)$ time. For TStreamEB, the double-interpolation takes $O(M^2P)$ time, where P is the average number of control points per edge; the hierarchy requires $O(N)$ time, where N is the number of nodes.

5 Experimental Results

We have implemented a prototype of our StreamEB framework based on GEOMI visualization framework [7] and Neo4j graph database [1]. In most experiments, we set the default values of compatibility parameters to $\eta_S=0.7$, $\eta_T=0.1$, $\eta_N=0.1$, $\eta_D=0.1$. For TStreamEB, we simply use a geometric clustering – a quadtree implementation in FADE [27], as our studies are not involved in hierarchical data.

5.1 Visual Analysis on US Flights Data

We used US flights data, which contains the schedules and the actual delays of US flights in years 1987-2008 [3]. For each year, the total number of edges range from 3149 to 5200, and the number of flights range from one to over seven millions.

The impact of the terrorist attacks on 11-Sep-2001 was significant and led to the cancellations of most flights. For instance, all flights were cancelled after 9:20am for half day and there was a single flight on the next day.

Analysis of Flight Delays. Figure 2a applies S-FStreamEB on the sliding windows taken before and after the terrorist attacks on 11-Sep-2001. The figure shows a significant drop of flights from 8am to 8:30am, and from 9am to 9:20am. The number of edges in each window is 741, 549, 392 and 255, respectively. The figure also shows that many flights actually departed earlier (red) than scheduled in the 8am window. Then many flights got delayed (green) around 9am to 9:20am - the period after the final attack and before the shut-down period.

Figure 2b depicts the visualization of flights using our TStreamEB. Most flights from 8am 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 2a and 2b show that most flights from 8am to 9:20am were within the Eastern US region. The figures also show the distribution of

Fig. 2. US flight delays before and after the 11-Sep-2001 attacks

the flight delays 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, they also show the evolution of connecting flights, and flight delays among a group of airports. For instance, many flights in a small group of airports in the Middle-South region departed earlier (than the schedule) at 8am, but then many of them got delayed from 8:30am onward.

5.2 Visual Analysis on Trading Data

We use the trading data from Thompson Reuters [2], which consists of time series feeds of stock trades. 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).

Our analysis gives the overview of the performance of traders and groups of traders over time, and the evolution of the money flows.

Analysis of Trader Performance. Figures 3a and 3b show the activeness of the traders. Figure 3a applies S-FStreamEB and MDS layout on the sequence-based sliding windows $W=160$. Figure 3b applies TStream with $\beta=1$ and $\gamma=0.2$, and MDS layout.

Fig. 3. Visualization of the dataset V on sequence-based sliding windows $W=160$

Fig. 4. Visualization of dataset V : (a) TStreamEB with circular layout of the first few seconds after market opening; (b) F-FStreamEB with anchored positions from a circular layout of sequence-based sliding window $W=160$

The overall trade patterns in the dataset V at market open time are depicted in Figure 4(a). The figure applies circular layout and TStreamEB method with $\beta=1$ and $\gamma=0.2$. For similar analysis, Figure 4(b) applies F-FStreamEB on sliding windows of $W=160$ with anchoring forces that anchor vertices to 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 at moderate amounts during the first few minutes, and then increased their trading amounts over time.

Analysis of Trades between Groups. Figure 3a shows four groups of traders. The groups, ranked by trading activeness, are located at top-left, top-right, bottom-left and bottom-right corners, respectively.

Better pictures of trading groups and their trading patterns are depicted in Figure 3b. The figure 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.

Analysis of Market Manipulation. Figure 3a and Figure 3b show that a number of active traders (top-left) are “frequent” traders, though their total trade volumes are fairly small. For instance, traders 088 and 124 are fairly blue at the first 30 minutes. Whilst some other traders in the least active group, such as traders 027, 066 and 068, initially traded at high volumes.

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.

5.3 Performance Comparison

Figure 5 compares the performance of three force-directed variations S-, G- and F-, and the tree-based TStreamEB. A thousand graph instances of at most nine hundred edges were randomly extracted from graph streams of the flight dataset.

Fig. 5. Runtime comparisons of stream bundling methods

The figure shows that TStreamEB, which requires a single iteration, is the fastest. For large graphs, the G- variant outperforms all the other variants of FStreamEB. The bundle-aware forces add little overheads to F-FStreamEB.

6 Discussions

The bundling parameters provide flexibility to adapt to analytic needs at runtime; however, they also require user’s adjustment from the large number of parameters to match with the analysis.

In current implementation, TStreamEB uses a synthetic quadtree and bundled results may have artifacts that could cause misleading interpretations.

References

1. Neo4j, the graph database (2011), <http://neo4j.org/>
2. Thomson Reuters (2011), <http://thomsonreuters.com/>
3. US Flights (2011),
<http://stat-computing.org/dataexpo/2009/the-data.html>
4. Abello, J., Finocchi, I., Korn, J.L.: Visualization of state transition graphs graph sketches. In: *InfoVis*, p. 67 (2001)
5. Aggarwal, C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: *VLDB*, vol. 29, pp. 81–92. VLDB Endowment (2003)
6. Aggarwal, C., Wang, H.: *Managing and Mining Graph Data*, vol. 40. Springer (2010)
7. Ahmed, A., Dwyer, T., Forster, M., Fu, X., Ho, J., Hong, S.-H., Koschützki, D., Murray, C., Nikolov, N.S., Taib, R., Tarassov, A., Xu, K.: GEOMI: GEOMetry for Maximum Insight. In: Healy, P., Nikolov, N.S. (eds.) *GD 2005. LNCS*, vol. 3843, pp. 468–479. Springer, Heidelberg (2006)
8. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. *VLDB* 15(2), 121–142 (2006)
9. Binucci, C., Brandes, U., Di Battista, G., Didimo, W., Gaertler, M., Palladino, P., Patrignani, M., Symvonis, A., Zweig, K.: Drawing Trees in a Streaming Model. In: Eppstein, D., Gansner, E.R. (eds.) *GD 2009. LNCS*, vol. 5849, pp. 292–303. Springer, Heidelberg (2010)
10. Boyandin, I., Bertini, E., Lalanne, D.: Using flow maps to explore migrations over time. In: *Geospatial Visual Analytics Workshop*, vol. 2 (2010)
11. Brandes, U., Mader, M.: A Quantitative Comparison of Stress-Minimization Approaches for Offline Dynamic Graph Drawing. In: van Kreveld, M., Speckmann, B. (eds.) *GD 2011. LNCS*, vol. 7034, pp. 99–110. Springer, Heidelberg (2012)
12. Chin, G., Singhal, M., Nakamura, G., Gurumoorathi, V., Freeman-Cadoret, N.: Visual analysis of dynamic data streams. *InfoVis* 8(3), 212 (2009)
13. Cui, W., Zhou, H., Qu, H., Wong, P., Li, X.: Geometry-based edge clustering for graph visualization. *TVCG*, 1277–1284 (2008)
14. De Pauw, W., Andrade, H.: Visualizing large-scale streaming applications. *InfoVis* 8(2), 87 (2009)
15. Eades, P.A.: A heuristic for graph drawing. *Congressus Numerantium* 42, 149–160 (1984)
16. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: Graph distances in the streaming model: the value of space. In: *SODA*, pp. 745–754 (2005)
17. Gansner, E., Hu, Y., North, S., Scheidegger, C.: Multilevel agglomerative edge bundling for visualizing large graphs. In: *IEEE PacificVis*, pp. 187–194 (2011)
18. Gansner, E.R., Koren, Y.: Improved Circular Layouts. In: Kaufmann, M., Wagner, D. (eds.) *GD 2006. LNCS*, vol. 4372, pp. 386–398. Springer, Heidelberg (2007)
19. Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *TVCG*, 741–748 (2006)
20. Holten, D., van Wijk, J.J.: Force-directed edge bundling for graph visualization. *Computer Graphics Forum* 28(3), 983–990 (2009)
21. Kienreich, W., Seifert, C.: An application of edge bundling techniques to the visualization of media analysis results. In: *InfoVis*, pp. 375–380. IEEE (2010)
22. Lambert, A., Bourqui, R., Auber, D.: Winding Roads: Routing edges into bundles. *Computer Graphics Forum* 29, 853–862 (2010)
23. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. *Journal of Visual Languages and Computing* 6(2), 183–210 (1995)
24. Nguyen, Q., Hong, S.-H., Eades, P.: TGI-EB: A New Framework for Edge Bundling Integrating Topology, Geometry and Importance. In: van Kreveld, M., Speckmann, B. (eds.) *GD 2011. LNCS*, vol. 7034, pp. 123–135. Springer, Heidelberg (2012)

25. Nguyen, Q., Eades, P., Hong, S.H.: StreamEB: Stream Edge Bundling. Tech. Rep. TR-689, University of Sydney (July 2012)
26. O’Callaghan, L., Mishra, N., Meyerson, A., Guha, S., Motwani, R.: Streaming-data algorithms for high-quality clustering. In: ICDE, pp. 685–694. IEEE (2002)
27. Quigley, A., Eades, P.: FADE: Graph Drawing, Clustering, and Visual Abstraction. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 197–210. Springer, Heidelberg (2001)
28. Selassie, D., Heller, B., Heer, J.: Divided edge bundling for directional network data. TVCG 17(12), 2354–2363 (2011)
29. Shi, L., Wang, C., Wen, Z.: Dynamic network visualization in 1.5D. In: PacificVis, pp. 179–186 (2011)
30. Sun, J., Faloutsos, C., Papadimitriou, S., Yu, P.: Graphscope: parameter-free mining of large time-evolving graphs. In: SIGKDD, pp. 687–696. ACM (2007)
31. Zhou, H., Yuan, X., Cui, W., Qu, H., Chen, B.: Energy-based hierarchical edge clustering of graphs. In: IEEE PacificVis, pp. 55–61. IEEE (2008)