VisFlow - Web-based Visualization Framework for Tabular Data with a Subset Flow Model

Bowen Yu and Cláudio T. Silva *Fellow, IEEE*

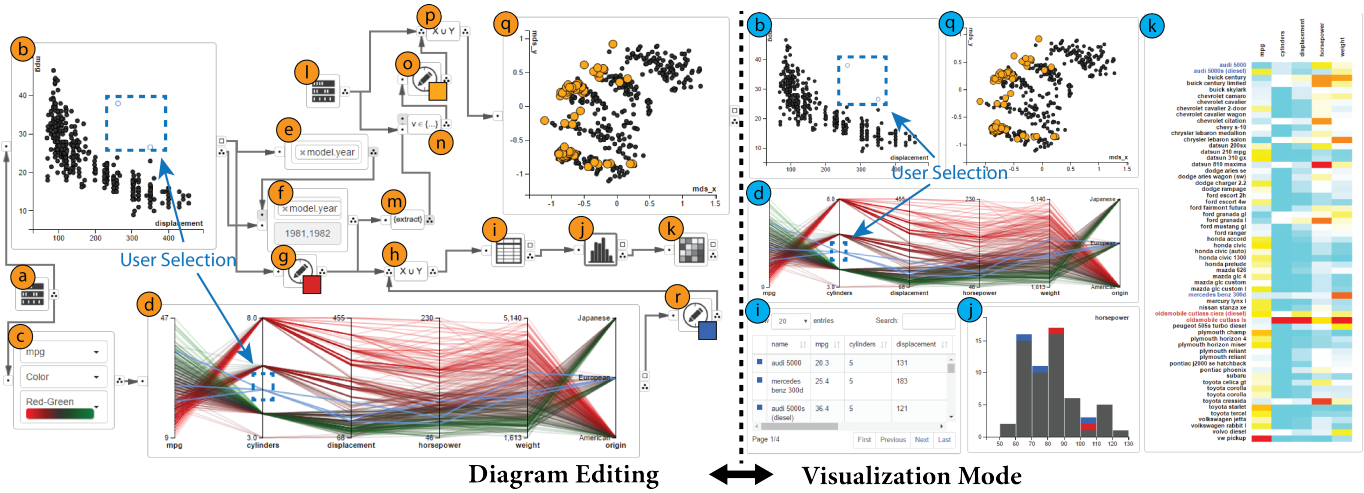

Fig. 1. An overview of VisFlow. The user edits the VisFlow data flow diagram that corresponds to an interactive visualization web application in the VisMode. The model years of the user selected outliers in the scatterplot (b) are used to find all car models designed in those years (1981, 1982), which form a subset S that is visualized in three metaphors: a table for displaying row details (i), a histogram for horsepower distribution (j) and a heatmap for multi-dimensional visualization (k). The selected outliers are highlighted in red in the downflow of (b). The user selection in the parallel coordinates are brushed in blue and unified with S to be shown in (i), (j), (k). A heterogeneous table that contains the MDS coordinates of the cars are loaded in (l) and visualized in the MDS plot (q), with S being visually linked in yellow color among the other cars.

Abstract— Data flow systems allow the user to design a flow diagram that specifies the relations between system components which process, filter or visually present the data. Visualization systems may benefit from user-defined data flows as an analysis typically consists of rendering multiple plots on demand and performing different types of interactive queries across coordinated views. In this paper, we propose VisFlow, a web-based visualization framework for tabular data that employs a specific type of data flow model called the *subset flow model*. VisFlow focuses on interactive queries within the data flow, overcoming the limitation of interactivity from past computational data flow systems. In particular, VisFlow applies embedded visualizations and supports interactive selections, brushing and linking within a visualization-oriented data flow. The model requires all data transmitted by the flow to be a data item subset (i.e. groups of table rows) of some original input table, so that rendering properties can be assigned to the subset unambiguously for tracking and comparison. VisFlow features the analysis flexibility of a flow diagram, and at the same time reduces the diagram complexity and improves usability. We demonstrate the capability of VisFlow on two case studies with domain experts on real-world datasets showing that VisFlow is capable of accomplishing a considerable set of visualization and analysis tasks. The VisFlow system is available as open source on GitHub.

Index Terms—Visualization framework, data flow, subset flow model, tabular data.

1 Introduction

Data analysis is typically an iterative process that demands both data presentation and interactive queries, as summarized by the visual information seeking mantra and task taxonomy [38]. Data analysts start from an overview of data, and dive progressively into subsets that are valuable to answering questions. Analysis workflows are often driven by data features, which require the tools applied for the analyses to be flexible. For instance, computational biologists propose hypotheses and verify the hypotheses using data collected from lab experiments. The analysis workflow of the verification is hypothesis-specific, and often involves multiple types of queries and visualizations. We seek a data flow approach that provides the user with the direct capability of manipulating the underlying analysis workflow in order to support flexible visual analytics queries and adaptive analysis. The system analysis model is specified by a *data flow diagram*. Updates on the flow diagram immediately produce a differently reacting system. The data flow approach is also very effective at capturing the workflow design decisions that an analyst assesses and reflects frequently, because of its intuitive diagram representation and supportive user interface.

Despite the flexibility, we observe that existing data flow systems typically have some drawbacks for visual data analysis:

- Many data flow systems are designed towards data processing and computations [1, 2]. In those systems flow diagrams are vi-

- *Bowen Yu and Claudio Silva are with New York University. E-mail: {bowen.yu, csilva}@nyu.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx/

sual abstractions of programming, *i.e.* the inputs and outputs of a module correspond to program method arguments. Such flow models require the user to have programming background. They have a higher learning overhead, and often produce flow diagrams that are too complicated to be easily understood.

- Analytics platforms that employ data flows [4, 5] are often not specifically designed for visualizations, in that visualizations in those data flow systems are mostly statistical summaries and do not support interactive data exploration. There are typically no outputs from the visualization components for further user manipulation. However, being able to select and filter the data interactively from visualizations is of key importance to visual analytics. Due to the complexity of operations an analytics platform performs, reactive feedback is typically not possible and explicit re-execution is required to update the visualizations.
- Data flows designed for visualization purpose [3, 11, 22, 30] mostly aim at generating rendering pipelines (*e.g.* for volume rendering). Interactivity is often limited only for navigation within rendered views. The flexibility advantage of data flow is only exploited in the application construction phase for users who are familiar with the system modules, rather than in the interactive data analysis phase for users whose goals are to visualize and analyze data.

In this paper we propose VisFlow, a web-based visualization framework for tabular data based on a particular data flow model, called the *subset flow model*. The subset flow model builds on the idea of subset manipulation in data flows [33], and extends it to interactive data exploration with information visualizations. Our flow model requires the data transmitted within the flow to be a data item subset (*i.e.* groups of table rows) of the input tabular data, so that brushing and linking are defined unambiguously by assigning rendering properties to the transmitted data subsets. In VisFlow the user may directly interact with the data items and perform interactive queries through the embedded visualizations. VisFlow visualizations update reactively on user interactions, and output user selections explicitly, so that the subsets being analyzed can be easily tracked, compared, and understood. We implement the VisFlow framework prototype that provides the user with easily accessible user interface and intuitive interactions for editing the flow diagrams. With a focus on data subsets, our flow model also reduces the diagram complexity and mitigates the learning overhead of a computational data flow.

The contributions of this paper can be summarized as follows:

- We propose a visualization framework called VisFlow for tabular data that explores the option of using the subset flow model to address the interactivity and complexity issues from previous data flow systems. VisFlow largely enhances the interactivity of data flow for the purpose of visual data exploration and interactive queries, by supporting immediate interaction feedback, data selection, brushing and linking, etc.
- We implement the web-based VisFlow framework prototype that includes the user interface and controls for efficiently creating and editing the VisFlow data flow diagrams. It is easy to get started with the system using a web browser. The system is publicly available online and as an open source project.
- We demonstrate by case studies with domain data analysts how VisFlow is able to complete a considerable set of visual data analysis tasks, and be applied to practical domains.

We first survey the related work, introduce the concept of data flow diagrams and other data flow systems in Section 2. We then describe the VisFlow subset flow model in Section 3 and the VisFlow framework in Section 4. Since only high-level concepts of the VisFlow model and framework can be covered in the paper due to the page limit, we refer interested readers to the VisFlow prototype documentation[1] for more details. Two case studies with practical analysis scenarios are presented in Section 5. Discussions on the design choices and limitations of VisFlow, along with its future work and conclusions, are presented in Section 6 and 7.

[1] https://visflow.org/doc.html

2 Related Work

There are a large number of tools for visualization and analysis of tabular data. We lay our emphasis on the approaches that explicitly expose data flows to the user.

2.1 Data Flow Diagrams

Data flow is a diagrammatic representation of the data transfer within a system. It originated as a graphical method for analyzing system structures [18]. The VisFlow flow diagram is defined as a way for data transmission, so that the transmission direction must be uniquely specified without ambiguity. This makes it different from illustrative flow diagrams where the name "data flow diagram" (DFD) may have other indications [18]. The graph-based visualization approaches with relational diagrams are not data flows. For example, Domino [20] focuses on tabular data subset comparison. It visualizes bidirectional relational edges that are not however flow edges. Other non-data-flow examples include the visualization schema by North et al. [29], the Improvise system with shared-object coordination [44], the network-based approach by Liu et al. [25]. Data flow systems provide visualization construction interface. Yet many of the visualization construction tools including iVisDesigner [32] and Quadrigram [6] have different mechanisms for managing data pipelines and do not present a flow diagram. Analytical or executional paths have implicit data flows with a single path, such as Victor's creative demo of drawing dynamic visualizations [42] and the Lyra [37] system. However this paper focuses on flow flexibility and multiple flow branches.

2.2 Data Flow Systems

We identify systems employing data flow models by their major application scenarios and functionality.

2.2.1 General Computational Systems

Data flow models have been widely adopted to model computations [21, 28]. They are also effective for system modeling, *e.g.* in the domains of parametric modeling [2], signal and media processing [1, 7]. Using data flows a system may provide analysis capability including data transformation, data filtering, statistics computation, and visualizations. For example, the user is able to employ a comprehensive set of computational nodes to perform machine learning and data mining in the IBM SPSS Modeler [4] and KNIME [5]. Taverna [45] and the Yahoo Pipes [9] provide workflows for web services and web content processing respectively.

The aforementioned systems are designed for general-purpose computation and analysis. However, visualization capability and queries by user selections are lacking in those systems. Most of these systems do not emphasize the interactive queries that are possible in plots, as visualization nodes have no outputs and serve only as a summary of the computation results or statistics. Computational data flows often yield a complicated flow diagram due to the complexity of computations. In this paper rather than aiming at general computations we focus on enhancing the user's interactive control over the visualized data in a data flow. More particularly, achieving interactive analysis with brushing and linking in the above systems is not straightforward as derived data generated from computational modules introduces the ambiguity of tracing data items. The design goal of VisFlow is to overcome this limitation, while preserving the flexibility of using data flow diagrams.

2.2.2 Visualization Systems

There are a large number of visualization systems based on data flows, which have been shown to be effective in constructing flexible rendering pipelines. Early systems emerged mostly for interactive construction of scientific problem solving environments [46], *e.g.* steering geometric modeling and performing volume rendering [16, 30, 33, 41]. In the Application Visualization System (AVS) [41], SCIRun [30, 31],

ConMan [22], SmartLink [40], and VISSION [39], program modules are provided as modular blocks to be interconnected to achieve tunable custom rendering. Ross et al. [35] proposed a data flow workspace HIVE for exploring multi-dimensional scaling algorithms. VisTrails [11] generates multi-view visualizations from the specification of a pipeline and provides the interface to ease the manipulation and management of different pipelines. Other related systems include Voreen [27] and Kepler [26]. However in those systems the advantages of data flows are mostly exploited in the application construction phase to perform a specific type of rendering or algorithm. There is a lack of interaction support on the rendered result (*e.g.* in case of volume rendering only view navigation is provided), and explicit re-execution is required to update visualizations. Additionally, modifying the pipelines often requires expert knowledge on the system modules. VisFlow focuses on applying data flows with simpler usage for interactive data analysis, rather than constructing rendering pipelines. Waser et al. [43] presents an effective data flow design that supports interactive flood simulation steering, in which the parametric connections fit the particular domain. VisFlow has a subset flow model that facilitates interactions over data flow visualizations, which is generalizable and not domain-specific.

2.3 Data Flow with Subsets

Connecting programming modules (*e.g.* VTK in [11]) in a data flow often results in complex flow diagrams that make it hard for users to use. An option to make it easier is to constrain the data types transmitted in the data flow. One of the approaches is to focus on data subset manipulation which effectively provides view juxtaposition [19] over a large collection of data items. The Waltz system [33, 34] passes subsets between system modules for volume segmenting and rendering, and consequently produces a simpler tree-form flow structure to shorten the visualization workflow. Data flows with subsets are relatively easier to follow and understand. They also naturally represent sequential steps of data filtering. ExPlates [23] presents an information visualization workflow system that supports interactive subset extraction by expanding the flow diagram upon user selection, and shows embedded visualizations in the diagram nodes.

In this paper we extend the subset data flow concept to allow for interactive tabular data analysis with data flows. The Waltz system essentially targets at volume rendering and its subset filtering/slicing and view displays are dedicated to only volume data. Compared with other information visualization data flows like ExPlates, the VisFlow model makes a key distinction that it does not generate derived data within the flow to make subset brushing and linking unambiguously defined (see Section 3). However, ExPlates performs table join operations and thus does not sufficiently support visual linking of data items, as there is no straightforward way to represent data items by rendering properties. With our model constraint we develop a corresponding set of node categories, data primitives and transmission rules for VisFlow, which are specifically catered to effectively assisting the user to edit, compare, and understand tabular data subsets. Furthermore, with the proposed model VisFlow is able to produce a simpler flow diagram (discussed in Section 6.2). The advantages of user interaction resulting from working exclusively with tabular subsets in a data flow framework have not yet been explored in the previous literature. We believe this model to be well suited for a visual data analysis environment.

3 VisFlow Flow Model

In this section we introduce the VisFlow data flow model, *i.e.* the *subset flow model*. We start with the input data of the model, and then give the flow diagram definition, interaction methods and model constraints.

3.1 Tabular Data

VisFlow aims at visualizing and analyzing tabular data. The tabular data applied in VisFlow matches the entity-relationship database table definition. We consider each table row to be a meaningful data *item*. A group of data items is a data *subset*. Table 1(a) shows a few entries from the HealthStats table of the World Bank Data repository with the "SP.DYN.LE00.IN" indicator[2]. The year columns give the expected life at birth for the countries. For this table each data item is a country. VisFlow may also work with series data, in which case one table dimension defines the series order, and each data item represents a data point within the series. Table 1(b) gives an example of series data in a table, where the series order is defined by the years.

Country	2011	2012	2013
Australia	81.895	82.046	82.198
Brazil	73.347	73.618	73.886
China	75.042	75.200	75.353

(a)

Country	Year	Value
Australia	2011	81.895
Australia	2012	82.046
Australia	2013	82.198
Brazil	2011	73.347
...	...	...

(b)

Table 1. Tabular data example. Table (b) shows the same data for table (a) represented in the series form. Series transpose operation (Section 4.4) transforms table (a) into (b) on the primary key "Country", where the transformed attributes are the values for each year. Note that the transposed table (b) has 9 rows that are not all shown.

3.2 Flow Diagram

We now introduce the subset flow model employed by VisFlow. Figure 2 illustrates the model concepts.

3.2.1 Nodes, Ports and Edges

A flow diagram in VisFlow consists of nodes and edges. A *node* is a VisFlow module that loads, processes, filters or visualizes the data. Nodes expose input and output *ports* that accept and transmit incoming and outgoing data. Input and output ports are shown on the left and right side of a node respectively (see Figure 2). An *edge* is a directed connection from an output port to an input port. A *single port* (one dot ▪) accepts at most one edge. A *multiple port* (three dots ⁘) does not have edge number restriction. Topologically a VisFlow data flow diagram is a directed acyclic graph (DAG).

Downflow and upflow. We define *downflow(x)* and *upflow(x)* to be the set of nodes that are reachable from node x following the edges, in their original and reverse directions respectively.

3.2.2 Primitive Elements

The VisFlow data flows transmit only two types of primitive elements: *subsets* and *constants*. Ports are categorized exclusively into *data ports* (white background ▫) that transmit subsets, and *constant ports* (gray background ▫) that transmit constants. Ports must be type-matched when connected by edges.

Subsets. A subset transmitted by the flow is a collection of table rows from some input table as defined previously. A subset virtually preserves all the dimension information of its source table and the attribute values of its data item members, which both can be retrieved from a node receiving the subset. A subset is denoted by a pair of brackets containing the IDs of its member items in Figure 2.

Rendering properties are associated with data items in VisFlow. Those properties are transmitted together with the data items. Each data item has a *rendering property object* that maps a set of rendering parameters to their assigned values, e.g. $\{color : red,\ size : 5\}$. VisFlow currently supports five types of rendering properties: *color, size, border color, border width, opacity*. Rendering properties are set and modified by the nodes along the flow so that a same data item may have different rendering properties at different nodes. The property binder (Section 3.3) in Figure 2 assigns **red** color property to the data items in the subset $\{a, b\}$, so that a and b have red color throughout their downflow.

Constants. We define constants to be an ordered list of constant values. The values are either specified by user input directly or extracted

[2] http://data.worldbank.org/topic/health

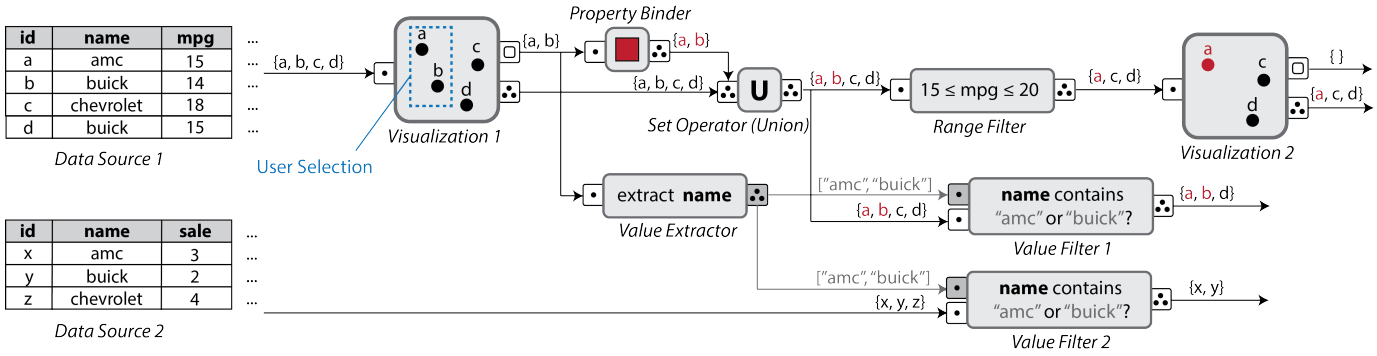

Fig. 2. Illustration of the key concepts of the VisFlow subset flow model. Node types are labeled in the diagram. The subsets are denoted by letter IDs within brackets. Assigned rendering properties are shown by **red** font color. Transmitted constants are shown in **gray**.

from attribute values of data items. Constants can be numbers, strings or dates. In Figure 2, two string constants "amc" and "buick" are extracted from the names of the user selected data items (a and b) at the value extractor (Section 3.3).

3.2.3 Subset Flow

The name *subset flow* naturally comes from the fact that subsets are the major primitives transmitted by the VisFlow data flow. Within a subset flow a data item from a subset must correspond to one of the input table rows so that it can be uniquely identified and given rendering properties. Note that although most data flow systems implicitly support generating subsets, this correspondence constraint defines VisFlow's unique system behavior and makes the VisFlow subset flow model different from a general data flow capable of subset generation. More particularly, VisFlow does not produce derived data within the flow such as a joined table. Such model has two key advantages:

Brushing and linking definition. By constraining the transmitted data to be subsets, brushing and linking operations in VisFlow are defined by the rendering properties bound to the data items. The subset flow model allows rendering properties to be uniquely associated with data items throughout the flow, and prevents the ambiguity of inheriting rendering properties when new types of data items are derived and generated, *e.g.* by a table join. It is not straightforward to define the behavior of brushing and linking with derived and mutated data. This is because there exist multiple possible results. For example, when a data item colored red is joined with a heterogeneous data item colored blue, the system cannot tell which color should be inherited. Therefore it has to ask the user for a decision, which would consequently increase the usage complexity and introduce confusion.

User perception. With the complexity and confusion resulting from inheriting rendering properties when derived data is involved, it is hard for the user to mentally trace, compare, and understand data subsets. The subset flow model works exclusively with data subsets as to improve the user's understanding of the data being visualized. Interactive queries are more intuitive as the user will be able to tell which data items he/she is selecting, from which answers to analytical questions can be derived. However, selecting derived table rows, *e.g.* from a joined table, is less intuitive as there might exist arbitrary new types of subsets with varying dimensions.

Further discussions on our design choices can be found in Section 6.

3.3 Node Categories

We briefly review the VisFlow node categories.

Data sources. Data sources load tabular data from data files. They do not have input ports. A data source always produces a single output subset from its input table(s). It may optionally include utilities to combine multiple homogeneous tables into one, or transform the input table for the convenience of analyzing series (Section 4.4).

Visualizations. Visualization nodes render the input subsets using visualization metaphors. To facilitate the interactivity of data flow visualizations, plots are embedded in the visualization nodes by default. Interactive selection can be made directly in an embedded visualization and sent to other nodes through a dedicated selection port (square icon ▢). In Figure 2, the user selects a and b in Visualization 1. The selection port of Visualization 1 outputs the selected subset $\{a,b\}$. Additionally a visualization node also has a data pass-through forwarding port (a multiple port denoted by ∴) that simply outputs its input. The redundancy is included to reduce diagram clutter. In Figure 2 Visualization 1 passes through the entire input set $\{a,b,c,d\}$ to the downflow through the forwarding port. Otherwise the downflow nodes must be connected to the data source to receive the entire input set.

Visualizations in VisFlow must always respect the rendering properties of data items. Visualization 2 in Figure 2 renders data item a in red color, as assigned by the upflow property binder. Note that different visualization metaphors may present the rendering properties differently. While a scatterplot (*e.g.* Figure 1(q)) renders the dots directly in the items' assigned colors, a heatmap (Figure 1(k)) sets the font colors of row labels to the items' colors so as not to interfere with the color mapping used by the heatmap cells for the items' attribute values. The user is able to tune the rendering parameters of the visualizations through the user interface.

Value generators. Value generators produce constants that are used as filtering parameters. The value extractor in Figure 2 extracts the names of data items a and b ("amc" and "buick"), which are constants used by the downflow value filters.

Filters. Filters examine attribute values of data items and perform attribute filtering. Filtering parameters are constants and can be user specified, or retrieved from value generators. The value filters in Figure 2 find the items with names being either "amc" or "buick". A filter can also be a *sampler* that down-samples an input subset on a user-selected sampling dimension.

Property binders. Property binders assign rendering properties to its input data items. The property binder in Figure 2 assigns **red** color to its input subset items a and b. A property binder may also encode the attribute values of the input subset using a mapped visual channel (*e.g.* a color scale in Figure 3). The user may assign distinguishable visual representations to important subsets so that they can be identified and linked across multiple plots. Rendering properties can be overwritten by downflow property binders.

Set operations. Set operation nodes take two or more subsets from a same table to produce a new subset using a mathematical set operation. The rendering properties of a same data item are merged at a set operation node. The union node in Figure 2 merges $\{\boldsymbol{a}, \boldsymbol{b}\}$ with $\{a,b,c,d\}$ and preserves the colors of $\boldsymbol{a}$ and $\boldsymbol{b}$. The last connected input subset has higher priority in case of a rendering property merge conflict.

In the VisFlow prototype we have designed a few example node types for each category, as listed by the node creation panels in Figure 4. It is possible to add new node types to the categories or extend the categories on demand as long as the added nodes meet the VisFlow model requirements.

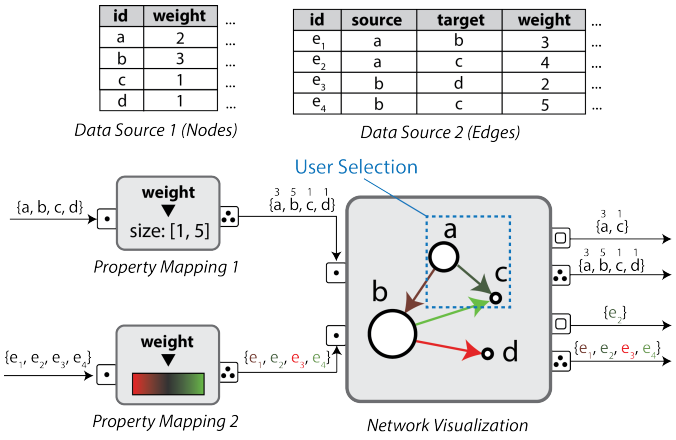

Fig. 3. Network visualization takes two heterogeneous subsets as inputs, for nodes and edges respectively. There are property mappings of node weights to node sizes, and edge weights to edge colors. Sizes bound to the nodes are shown on top of the node IDs. Colors bound to the edges are denoted by the font colors of the edge IDs. The network correspondingly renders the nodes and edges, and have four outputs: two for the user's node selection and forwarding of nodes; two for the user's edge selection and forwarding of edges.

3.4 Interactions

In VisFlow, presented data in the visualizations can be directly selected and extracted as subsets for further queries or manipulations. An important task for visual analytics is to be able to perform *brushing and linking*.

Brushing. VisFlow supports brushing by binding rendering properties to the subsets. A subset will be shown consistently according to its rendering properties. For example, the user selects items *a* and *b* from Visualization 1 in Figure 2 and passes them through the property binder, which brushes the items in red.

Linking. The same subsets in VisFlow are automatically linked across nodes with the same rendering properties based on the nature of data flow. A downflow visualization in Figure 2 (Visualization 2) receives the data items with associated rendering properties, so that ***a*** is highlighted in red. Note that Visualization 2 does not necessarily need to apply the same visualization metaphor as Visualization 1. Combined with the filter, the example flow diagram in Figure 2 effectively brushes and highlights the selected items that satisfy $15 \leq mpg \leq 20$.

3.5 Heterogeneous Data

VisFlow supports heterogeneous data by links between heterogeneous tables, or visualizations specifically designed for heterogeneous data.

Link between heterogeneous data. Heterogeneous data items can be linked by value extraction and filtering. Attribute values are first extracted from some data items as keys. Those keys are then used to relate heterogeneous data items of the user's choice, which can be further brushed and presented in linked visualization styles in VisFlow. In Figure 2, a second data source loads a different type of table that can be filtered with the extracted constants from the subset selected from the first table.

Visualization for heterogeneous data. A visualization may directly render heterogeneous data. One of the examples is a network visualization, in which data of nodes and edges are both required. Figure 3 illustrates the concept. A network visualization accepts two input subsets, for the nodes and edges respectively. Nodes and edges can be assigned different rendering properties. In Figure 3 the node weights are encoded by sizes, while the edge weights are encoded by colors from a red-green color scale. The network renders the nodes and edges respectively according to their rendering properties. User selections and forwarding of nodes and edges are output separately. Therefore a network node has four output ports.

3.6 Diagram Example with the Auto MPG Data

Figure 1 provides a comprehensive example of composing visualization web application using the VisFlow data flow model. The diagram applies the well-known Auto MPG dataset[3] (loaded by data source (a)) that consists of 392 cars (excluding cars missing attributes) and their information on 9 dimensions, including name, mpg, displacement, etc.

A scatterplot (b) first shows the relation between the dimensions "displacement" and "mpg". Two outliers, in terms of the overall negative correlation, are identified and selected. An interesting task could be to find all those cars that are produced in the same years as the two outliers. We define this subset of cars to be *S*. A value extractor (e) extracts the "model.year"s of the selected outliers and a value filter (f) performs the query for *S* by filtering the whole car collection. On the other hand, a parallel coordinates plot (d) helps provide an overview of value distribution, in which lines are color encoded (c) depending on the mpg values. The user selection in the parallel coordinates (cars having 5 cylinders) are brushed in **blue** (r) and unified (h) with *S*, while the outliers chosen above are brushed in **red** (g). Three visualizations, a table (i), a histogram (j) and a heatmap (k) are used to render *S* along with the selection from the parallel coordinates.

This example also includes a heterogeneous table (loaded by data source (l)) that contains the projected MDS coordinates of the cars, *i.e.* dimensions "mds_x", "mds_y", and (car) "name". The MDS coordinates are generated by the metric SMACOF algorithm using the Minkowski Model in Euclidean distance [15], on all the dimensions with a maximum of 1,000 iterations.

An important task of studying an MDS plot is to identify the distribution of a subset. Here we highlight the distribution of *S* in the MDS plot (q) by linking heterogeneous data, as introduced in Section 3.5. The car names of *S* are extracted (m) and used to retrieve a corresponding subset with the MDS coordinates from the MDS table (by filter (n)) to be highlighted. Highlighting of *S* in the MDS plot is easily achieved by assigning **yellow** color (o) to *S* and unifying *S* (p) with the other cars from the MDS table.

Recall that visualizations in VisFlow render the data items in their rendering properties, but potentially in different ways. Histogram (j) visualizes the "horsepower" distribution of the cars, with the number of highlighted items shown proportionally in their bins. Heatmap (k) on the other hand renders the row labels in the data items' colors, while its cells use a separate color scale to encode the attribute values.

As a summary, this flow diagram finds the cars that were produced in the same years as those selected outliers in the scatterplot (b), as a set *S*. It visualizes the distribution of *S* in an MDS plot. Meanwhile, additional cars selected from the parallel coordinates are highlighted together with *S* for comparison, in the visualizations (i), (j) and (k). This example demonstrates how brushing and linking are achieved in VisFlow, and how VisFlow works with heterogeneous data.

4 VisFlow Framework

We develop the VisFlow framework prototype that implements the VisFlow data flow model and demonstrates its applications. The framework prototype is available online[4]. The source code for VisFlow is available as open source at GitHub[5].

4.1 User Interface

The VisFlow framework interface is designed to assist efficient perception and manipulation of the VisFlow data flow diagrams in a web browser. Figure 4 shows the diagram editing interface. The flow diagram is drawn and manipulated on a virtual infinite canvas. Nodes can be created, resized and re-positioned, in an intuitive drag-and-drop manner. A node creation side panel on the left guides the user towards node addition, while a pop-out node creation panel controlled by hot key appears around the mouse cursor to closely follow the editing focus. Dragging from a port to another port or a node creates an edge.

[3] http://archive.ics.uci.edu/ml/datasets/Auto+MPG
[4] https://visflow.org/
[5] https://github.com/yubowenok/visflow

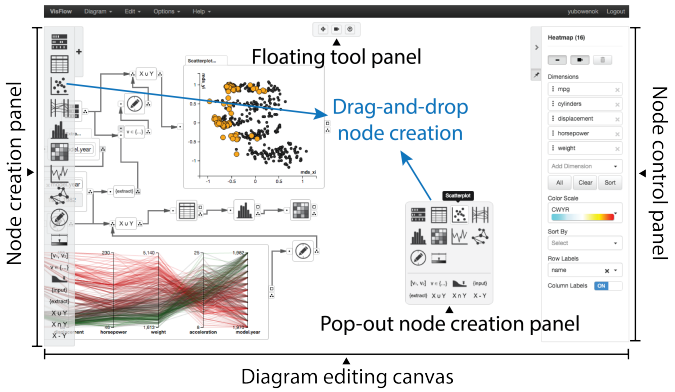

Fig. 4. The VisFlow framework prototype interface. Nodes are created in a drag-and-drop manner onto the infinitely large canvas. The node creation panels list the node types supported by the VisFlow prototype framework sequentially: data source; (visualizations) table, scatterplot, parallel coordinates, histogram, heatmap, line chart, network; (property binders) property editor, property mapping; (attribute filters) range filter, value filter, sampler; (value generators) value maker, value extractor; (set operations) union, intersect, minus.

VisFlow automatically selects the first available port that satisfies connection constraints when an edge being created is dropped onto a node. A node control panel on the right allows the user to set node specific parameters, including the dimensions to be rendered in visualizations, the color scale of a heatmap, etc. Visualizations are embedded into the rectangular node areas on the canvas, in which interactive selections can be performed. Nodes can be either shown in detail (*e.g.* Figure 1(b), (c), (d), (f), etc., in Diagram Editing) or collapsed into icons to save screen space (*e.g.* Figure 1(a), (g), (j), (n), etc., in Diagram Editing). A floating tool panel provides navigation control, VisMode toggle (Section 4.2), and data uploading.

4.2 Visualization Mode (VisMode)

The *visualization mode (VisMode)* of the VisFlow framework hides diagram edges and ports and presents only a user-selected set of nodes. The sizes and positions of the nodes in the VisMode can be configured separately from the diagram editing mode. Figure 1 illustrates the correspondence between the diagram editing mode and the VisMode. Views labeled in blue in the VisMode correspond to the labeled node with the same letter in the diagram editing mode. The visualizations are re-arranged in a more compact layout to present a cleaner interface for data analysis, like an off-the-shelf application. We provide smooth transition of the diagram elements when the VisMode is toggled, to help the user perceive the node correspondence between the two modes.

4.3 Reproducibility and Collaboration

The VisFlow prototype supports the reproducibility of the flow diagrams and the collaborations between VisFlow users. Diagrams can be saved at and loaded from the server, and shared with other VisFlow users for co-editing. A workflow designer can produce a ready-to-use interactive web visualization application without its internal data flow details exposed to its audience by sending a diagram in the VisMode. VisFlow preserves the diagram interaction states (*i.e.* user selections and navigation) when loading a saved diagram, so that analysis results can also be shared with proper highlighting.

4.4 Series Data Support

As tabular data may contain column attributes of ordered series, *e.g.* the year columns in Table 1(a), we further provide a *series transpose* operation that can be performed at the data source nodes to help analyze tabular data with series. In this case, each data point in the series is of analytical interest. A data point in the series represents a new type of data item that is different from the original table items. The series transpose operation takes a set of primary key attributes and a set of dimensions, and transforms the input table into rows of series points. Dimension names are written in an attribute column, and the original table values are stored in a third column. Using "Country" as the primary key, and the years as the dimensions, series transpose produces Table 1(b) from Table 1(a). Series transpose only serves as a shortcut for the user to apply series data in VisFlow. It provides an extra utility for the user's convenience, which satisfies the subset flow constraint and is equivalent to the user transforming the table outside VisFlow and loading the transformed table into VisFlow. A table like Table 1(a) with series can then be visualized, *e.g.* by a VisFlow line chart.

4.5 Computation

VisFlow avoids repeated execution and data storage redundancy to ensure performance. A connected input port only makes a reference to the transmitted subset or constants coming from its upflow output port, without acquiring additional copy of the subset or constants. Our prototype keeps minimally one copy of each table dataset in memory, and stores the row indices of the data items as their IDs in the transmitted subsets, so that attribute values are not duplicated. Though data items may have one-to-many relations with their rendering properties instances, we only make copies of rendering properties at the property binders that modify the properties and store object references elsewhere. User operations such as flow diagram editing, data item selection in visualizations and filtering parameter updates, lead to reactive changes in the downflow nodes. We propagate the changes in the flow in topological order starting from the node where a change occurs. The propagation at every node stops immediately when no change is detected, *e.g.* the output of a set intersection may remain the same after an item is added to its input subset.

4.6 Implementation

The VisFlow prototype framework employs a client-server architecture. The client runs the flow computations and renders the diagram and user interface within the user's browser. The server provides access to data, stores and loads the flow diagram configurations upon client requests. Custom tabular data can be uploaded from the user's local machine to the server. We implemented the VisFlow framework front-end in JavaScript. Rendering is performed by D3 [12] in SVG for the convenience of listening for element interactions. Since the analysis and visualization capability of VisFlow are defined by its integrated node types, we applied an objected-oriented code architecture using JavaScript prototypes for easy extensions. Extended node types inheriting a node base class can be written independently and then added to VisFlow. More details on the implementation can be found at the GitHub repository. The current codebase contains around 20K lines of Javascript code.

5 Case Studies

In this section we present two use cases that demonstrate VisFlow's capability of enabling interactive visual data analysis through flexible visualizations and interactive queries in different task domains.

5.1 Gene Regulatory Network Analysis

Understanding the regulations between genes, *e.g.* the presence of a gene repressing or activating another, is an important goal in biology. Representing the regulations between genes by the regulatory networks, which are derived from biologists' lab experiment results using mathematical models, is a key approach to studying the complicated relations between genes. One of the example networks, Th17, supports immune cell fate specification and is computed from a lineage differentiation model system [14]. Previous research [47] has yielded domain application that assists biologists to analyze such networks along with their ground-truth lab experiment results. We show that VisFlow is able to generate, with a small number steps, a similar gene regulatory network analysis environment for computational biologists. We worked with a group of computational biologists who perform regulatory network analysis as one of their research tasks.

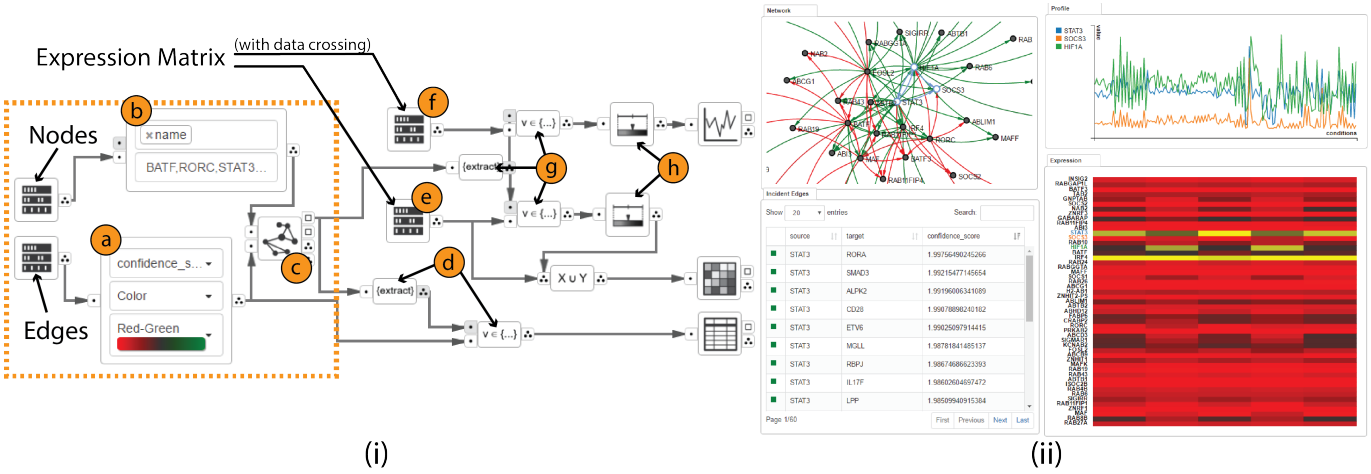

Fig. 5. Regulatory network analysis workflow and its corresponding application generated by VisFlow. The flow diagram (i) derives the interactive regulatory network analysis application (ii) in the VisMode. The expression profiles retrieved from the expression matrix rows are plotted in the line chart. Row labels of the heatmap for selected genes in the network are highlighted in the same colors as their profiles that appear in the line chart.

The biologists involved usually write script for the same set of tasks to adapt their network data to other existing visualization tools, and have moderate experience with programming. After a short introduction of VisFlow, we collaborated with the biologists on the flow diagram configuration. Figure 5 gives one of the resulting flow diagrams in (i) and its corresponding VisMode application in (ii).

5.1.1 Regulatory Network

A gene regulatory network is a directed weighted graph, in which nodes are genes and edges are sourced from master regulators named Transcription Factors (TFs). The weight of an edge denotes the confidence score of a regulation prediction. Rendering such network with encoded edge weights is a first-step task in the analysis (see Figure 5). The network visualization node in VisFlow takes two input tables, one for nodes and the other for the edges, as described in Section 3.5. The network node renders the input nodes and edges using a force-directed layout provided by D3[6]. Edge weight encoding is achieved by a property mapping node (a). Biologists typically explore the network bottom-up by adding to the network known genes they are interested in. A value filter with user input gene names is added (b) so that the network only shows the genes specified by the biologists. The nodes and edges are then connected to a network visualization (c). The dotted section in Figure 5(i) shows the five diagram nodes for a network visualization at the top-left of Figure 5(ii).

Biologists would search for incident edges of interest that are currently not shown in the network, and then potentially expand the network through the incident edges. An incident edge table is added to the diagram by adding a gene name filter on either the source gene name or the target gene name of the edges, combined with a value extractor (d) that retrieves the names of the user selected genes in the network. This conveniently lists incident edges upon gene selection in the network (bottom-left of Figure 5(ii)).

5.1.2 Expression Matrix and Gene Profiles

The biologist uploads a gene expression matrix to VisFlow as the supporting ground-truth data. The matrix rows represent genes and columns represent experiment conditions. Each matrix cell contains the gene's responsiveness value under one experiment condition. The visualization requirement is to render the expression matrix as a heatmap, and additionally show selected gene profiles (rows of the heatmap) in a line chart, ordered by the matrix columns. In this case, the experiment conditions can be thought of as a series and we can apply the VisFlow line chart. Lines are created by first grouping the series points by a grouping dimension "genes", and then rendering the lines along the series dimension "experiment conditions". Since the input matrix does not contain such series points as data items, we apply the series transpose operation (Section 4.4) to the input matrix. After series transpose the new table gets rows of type *(gene, experiment condition, value)*, which are applicable in the line chart. As rendering the heatmap requires the original matrix, two data sources (e) and (f) are applied, producing the original and transformed matrices respectively. We filter the matrix rows (g) and encode gene names using categorical color scales (h), which assign colors based on the hash values of gene names to provide linked visualizations, *i.e.* the heatmap row labels are linked with the rendered lines by their colors.

Figure 5(ii) shows the resulting regulatory network analysis application in the VisMode, which supports the visualizations of the regulatory network and the experiment matrix, as well as unidirectional linked queries from the network to the matrix (a subset of functionality from the Genotet system [47]). The biologists performing this experiment commented that VisFlow could save time and efforts of biologists as a query workflow can be directly composed in VisFlow, otherwise they have to write custom script to parse and plot the data. This case study shows that VisFlow is able to assist domain data analysis with the composition of a small number of data flow nodes, and does not require programming to use.

5.2 Baseball Pitch Analysis

We present a second case study in the area of sports analytics for baseball games, where we interacted with an expert in baseball analytics who is a statistician working on applied problems in sports. Baseball is a highly data-driven sport. Since pitching style is potentially the most interesting and important aspect of baseball games, statistics for various metrics of pitches have been available for many decades. The particular interest of the analyst is to understand the behavior and style of pitchers by studying their movement data. We demonstrate how VisFlow is able to adapt to this realistic scenario to quickly develop a tool that an expert can use for the analysis, *e.g.* to identify how pitches differ in their deliveries.

One of the analysis tasks is to compare different pitchers across multiple games. Our data comes from MLB.com Statcast[7]. The data is organized in many separate tables, including a list of team matchups, a player roster for each team, a list of game plays (*i.e.* individual pitches) for each game, etc. For each game play, there are measures of metrics for the pitch. The metrics include the spin rate and speed of the ball, the time it took the pitcher to release the ball, pitcher extension, and the result of the play, which can be a Ball, a Strike, or different types of Foul balls.

[6] http://d3js.org

[7] http://mlb.com/statcast

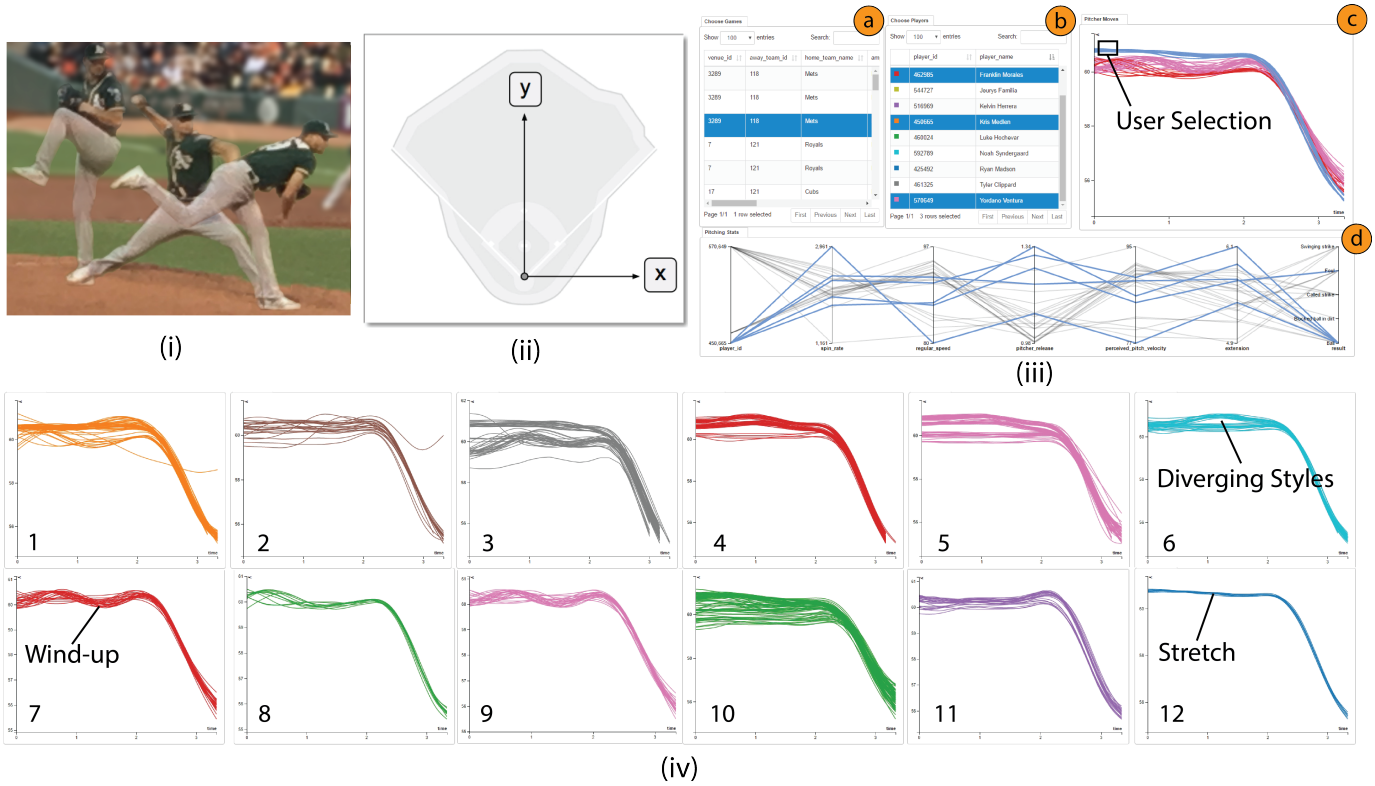

Fig. 6. Applying VisFlow to baseball pitch data analysis. (i) The pitching movement; (ii) The Statcast coordinate system illustration (from [24]); (iii) The analysis environment for the baseball pitch analysis generated by VisFlow; (iv) Plots of pitching movements of 12 players. A categorical color scale is applied to render each player's pitches in a uniform color.

Despite the data being spread over multiple table files, in a small amount of time the analyst can compose a VisFlow flow diagram that enables us to visualize the game plays (see Figure 6(iii)). The top-left table (a) lists the games described by the data and allows the user to select one or more games to study, with each game having around 300 pitches. The table in the middle (b) has each pitcher's name and player ID, from which the user may choose a list of players he/she is interested in. Upon user selection of a set of players, the parallel coordinates at the bottom (d) reactively displays the metrics statistics for the pitches of the selected players. In Figure 6(iii), the user currently selects 1 game and 3 pitchers. Note that selecting multiple games is also possible, in which case the presented plots will show the data for multiple games, and the statistics are shown for all pitches in those games.

In addition to the data described above, the Statcast system collects the pitch movement data. Statcast optically tracks the players at 30 Hz, which gives us unprecedented details on the players' movements on the field (Figure 6(i)). The optical tracking system is illustrated in Figure 6(ii). The tracking system uses a coordinate system in which $(0,0)$ is at the home plate where the batter is; the y axis points to the pitcher's mound; the x axis is orthogonal to it in a right-handed coordinate system. For each player, there is a sequence (x_t, y_t), of 2D positions, recorded at a series of timestamps (t's). The recorded coordinates for a player is of the approximated pitcher center of mass during his movement. The data contains 90 samples for each pitch, that is, $(x_1, y_1), \ldots, (x_{90}, y_{90})$. The analysis of pitcher movements typically focuses on the delivery styles, for which the first 3 seconds of the movements are of key importance. Since pitchers have negligible movement in the x direction, and pitchers' mound is exactly at $y = 60.5$ feet from the home plate, viewing the movement data with y values between $[55, 65]$ would allow the analyst to focus on the essential tracking records. With VisFlow filters, such visualization and attribute filtering requirement can be easily achieved by creating a few nodes in the diagram. Range filters are applied to remove tracking records after 3 seconds from the start of the delivery, as well as those records that appeared outside the interested y value range $[55, 65]$. A line chart visualization (Figure 6(iii)(c)) is used to render the series (t, y_t). Players can be encoded by categorical colors, so that the line chart renders the pitching movements of a same player in a uniform color. Using VisFlow brushing and linking, the user can easily relate the pitchers' movements with the pitch statistics. For any user selected pitch movements in the line chart (c), the parallel coordinates plot (d) instantly highlights the metrics statistics corresponding to the selected pitches. The analyst can therefore easily observe the speeds, velocities, and results of selected pitchers' movements. Such interactive queries largely help the analyst derive relations between the delivery styles and the players' performance.

The constructed analysis environment may help conclude baseball findings. For instance, it is easy to recognize the different types of deliveries, *i.e.* wind-up (y distance increases and decreases alternately in the beginning) v.s. stretch (y distance remains stable in the beginning). It is also possible to visually compare the movement of different players. It can be observed that some players have fairly uniform movements, while others' movements vary widely. Figure 6(iv) shows the pitches of 12 pitchers, numbered from 1 to 12. The pitchers in the first row (number 1-6) have clearly two movement patterns, as the movements diverge in the middle. The pitchers in the second row have a single movement pattern. Among the second row, pitchers on the left half (number 7-9) have wind-up styles, and the others (number 10-12) use a stretch delivery. Other observation, such as the variation of pitcher 10's starting y distance is significantly larger than the other players who solely use stretch deliveries, can also be made.

The data involved in this case study has multiple heterogeneous tables. It also has a large volume of recorded movement positions. In the diagram developed for the case study, 20 games are loaded, which contain 100K+ data points for the pitcher movements and other asso-

ciated records, from 5 heterogeneous tables. Around 30 pitches and 500 movement points are to be rendered for each player per game. A VisFlow sampler can be used to scale up to higher data volume, *e.g.* spanning a larger number of games. The diagram applies 28 nodes. The VisMode of the application generated from the diagram is shown in Figure 6(iii). It took the analyst less than an hour to complete the diagram after a 30-minute introduction of VisFlow. This case study shows with the flexibility provided by VisFlow it is possible to directly analyze such complex domain data interactively. The analyst also thinks that VisFlow provides an easy and effective solution to analyzing heterogeneous tabular data, which can be generalized and applied to other domains as well. He later independently created a flow diagram for basketball shot analysis.

6 Discussion and Limitations

We discuss the design choice and limitations of the VisFlow system.

6.1 Design Philosophy

Due to the subset flow model constraint, VisFlow is not able to perform some of the analysis tasks listed in [10] such as clustering. Therefore it is arguably true that VisFlow has less data processing power than general computational data flow systems such as KNIME [5], or systems with programming toolkits such as VisTrails [11]. However we believe it is a justifiable design choice for VisFlow because:

1) The design of VisFlow focuses on integrating interactive data exploration into data flows and facilitating subset identification, manipulation and comparison. VisFlow fills the absence of interactivity in computational data flows. Data processing, however, is not a primary aim of VisFlow.
2) We observe that under many circumstances the data to be interactively analyzed has already been processed by other tools, *e.g.* a data mining suite [8]. Therefore it is possible to leave data processing tasks, including data generation, clustering, aggregation, etc., for specialized tools.
3) At this time, there exists no single solution that achieves all the necessary goals, *i.e.* data processing power, visualization interactivity and usage simplicity. We believe VisFlow presents a reasonable trade-off between these aspects. Nevertheless, VisFlow preserves considerable visualization and analysis capability, *e.g.* all 7 tasks from [38] and 8 of the 10 tasks from [10] can be reformulated with plotting, interactive selections, and attribute filtering queries, and are thus effectively addressed by VisFlow. Meanwhile, VisFlow yields diagram complexity that is relatively lower and makes the system easier to use. The case studies in Section 5 show that VisFlow achieves satisfactory results in domain data analysis, without requiring the user to receive extensive training on the system usage, or spend a significant amount of time on diagram editing.

By the data state model [13] VisFlow essentially provides visual mapping transformations of information visualization that integrate data transformations via subset manipulations. The embedded visualizations also present data states and help support workflow provenance [17].

6.2 Usage Simplicity

VisFlow applies the subset flow model in which nodes are well-known visualization metaphors and filters, and its edges only transmit subsets or constants. The model intuitively follows the user perception of data items. VisFlow enables data exploration without the hassle of considering programming input/output parameters and makes it possible for the end user to modify the workflow for different queries with a low learning overhead. The subset flow model yields a less cluttered diagram compared with module programming systems such as [2, 11]. Its diagram is also simpler compared with the interactive workflow ExPlates [23], which has significantly more edges in its flow diagram due to the complexity of table joins and the interface design of dimension-wise connections. More particularly, for each unique data type there is at most one edge between any pair of VisFlow nodes. Additionally, the integration of interactive selection into visualizations and the exposition of selection ports result in fewer filtering nodes in the diagram. Otherwise interactive filtering has to be externalized as filter nodes (as in ExPlates), which would increase the number of nodes. It can be seen that most of the workflows presented in this work preserve planar or almost planar graph layout. The VisMode of VisFlow employs an idea similar to the VisTrails extension VisMashup [36] that optionally hides the workflow details that are not directly related to the analysis. The VisFlow framework also provides reactive responses upon diagram changes and user selections, making diagram editing, brushing and linking more convenient compared with systems like VisTrails that require explicit re-execution to update the visualizations.

6.3 Limitations

We identify some tasks that are more difficult to achieve in VisFlow:
Workflow loops. A workflow might need to be executed repeatedly (*e.g.* to refine the selection in a plot). However, data flow diagrams are DAGs and downflow subset cannot be added back to the upflow input. Existing data flow approaches all bear this limitation. We mitigate this limitation in VisFlow by allowing subsets to be exported into data files from ports, so that exported subsets can be re-loaded in the upflow to create an analysis cycle.
Iterative filtering. Some filtering operations such as finding a connected component in a graph requires the filter to check a condition recursively or repeatedly (*i.e.* whether an unvisited vertex is reachable from a visited vertex). The value filters in VisFlow perform one-stage filtering. To achieve multiple levels of filtering we must have multiple filtering nodes, which in the graph traversal case depends on the number of graph vertices. Such number is unknown in advance and could be large. It is possible to add new types of filters to VisFlow to support such types of filtering.

6.4 Scalability

The current VisFlow implementation assumes moderate data sizes that fit into the browser memory, which means the system is able to render up to millions of data items onto the HTML page. Upon larger data sizes Level-of-Detail (LoD) rendering may be desired, which presents aggregated statistics or samples of the original data. Though we can provide LoD entirely at the client-side visualization nodes, a more scalable alternative is to perform server-side flow execution that requires a non-trivial client-server communication protocol for cooperative rendering and interaction handling within a data flow context.

7 Conclusions

In this work we designed VisFlow, a web-based visualization framework and its *subset flow model*, for interactive visualizations and analyses of tabular data. VisFlow enhances data flow interactivity for data exploration by working exclusively with subsets. Our proposed model yields simpler flow diagrams and mitigates the system learning overhead. We justified our design choices and showcased the application value of VisFlow by case studies on data analyses with domain experts. The analysts were able to develop VisFlow diagrams that effectively provide an interactive data analysis environment that adapts to their domain specific tasks.

We have a number of areas that we would like to work on as future work. First, our current prototype is limited by the computation and memory capability of the browser environment. We plan to develop server-side rendering with automatic Level-of-Detail and a new flow execution mechanism with a client-server communication protocol. Another interesting area is to develop a recommendation system to help novice users with suggested flow diagrams.

Acknowledgments

We would like to thank Rich Bonneau and his lab for help with Section 5.1, and Dan Cervone, Carlos Dietrich, Tiago Etiene, Marcos Lage, and MLB.com for Section 5.2. This work was supported in part by an IBM Faculty Award, the Moore-Sloan Data Science Environment at NYU, the NYU Tandon School of Engineering, AT&T, NSF CNS-1229185, CCF-1533564, and CNS-1544753.

REFERENCES

[1] Cycling74. https://cycling74.com/.

[2] Grasshopper3D. http://www.grasshopper3d.com/.

[3] IBM OpenDX. http://opendx.org/.

[4] IBM SPSS Modeler. http://www.ibm.com/software/products/en/spss-modeler.

[5] KNIME data analysis platform. http://www.knime.org/.

[6] Quadrigram. http://www.quadrigram.com/.

[7] vvvv. http://vvvv.org/.

[8] Weka 3: Data mining software in java. http://www.cs.waikato.ac.nz/ml/weka/.

[9] Yahoo pipes. http://real.pipes.yahoo.com/pipes/.

[10] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *IEEE Symposium on Information Visualization (InfoVis'05)*, pages 111–117, 2005.

[11] L. Bavoil, S. P. Callahan, C. E. Scheidegger, H. T. Vo, P. Crossno, C. T. Silva, and J. Freire. VisTrails: Enabling interactive multiple-view visualizations. In *IEEE Visualization Conference*, pages 135–142, 2005.

[12] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics (InfoVis'11)*, 17(12):2301–2309, 2011.

[13] E. H.-h. Chi and J. Riedl. An operator interaction framework for visualization systems. In *IEEE Symposium on Information Visualization (InfoVis'98)*, pages 63–70, 1998.

[14] M. Ciofani, A. Madar, C. Galan, M. Sellars, K. Mace, F. Pauli, A. Agarwal, W. Huang, C. N. Parkurst, M. Muratet, K. M. Newberry, S. Meadows, A. Greenfield, Y. Yang, P. Jain, F. K. Kirigin, C. Birchmeier, E. F. Wagner, K. M. Murphy, R. M. Myers, R. Bonneau, and D. R. Littman. A validated regulatory network for Th17 cell specification. *Cell*, 151:289–303, 2012.

[15] J. de Leeuw. Modern multidimensional scaling: Theory and applications (second edition). *Journal of Statistical Software, Book Reviews*, 14(4):1–2, 9 2005.

[16] D. Foulser. IRIS Explorer: A framework for investigation. *ACM SIGGRAPH Computer Graphics*, 29(2):13–16, 1995.

[17] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. *Managing Rapidly-Evolving Scientific Workflows*, pages 10–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[18] C. Gane and T. Sarson. *Structured Systems Analysis: Tools and Techniques*. McDonnell Douglas Systems Integration Company, 1979.

[19] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4):289–309, Oct. 2011.

[20] S. Gratzl, N. Gehlenborg, A. Lex, H. Pfister, and M. Streit. Domino: Extracting, comparing, and manipulating subsets across multiple tabular datasets. *IEEE Transactions on Visualization and Computer Graphics (InfoVis'14)*, 2014.

[21] J. Gurd, W. Bohm, and Y. M. Teo. Performance issues in dataflow machines. In *Future Generations Computer Systems. Elsevier Scientific*, pages 285–297, 1987.

[22] P. E. Haeberli. ConMan: A visual programming language for interactive graphics. *ACM SIGGRAPH Computer Graphics*, 22(4):103–111, June 1988.

[23] W. Javed and N. Elmqvist. ExPlates: Spatializing interactive analysis to scaffold visual exploration. *Computer Graphics Forum (Proc. EuroVis)*, 32(2):441–450, 2013.

[24] M. Lage, J. H. Ono, D. Cervone, J. Chiang, C. Dietrich, and C. Silva. Statcast dashboard: Exploration of spatiotemporal baseball data. *IEEE Computer Graphics & Applications*, 2016, to appear.

[25] Z. Liu, S. Navathe, and J. Stasko. Network-based visual analysis of tabular data. In *IEEE Visual Analytics Science and Technology (VAST'11)*, pages 41–50, Oct 2011.

[26] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency Computat.: Pract. Exper.*, 18(10):1039–1065, Aug. 2006.

[27] J. Meyer-Spradow, T. Ropinski, J. Mensmann, and K. Hinrichs. Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations. *IEEE Computer Graphics and Applications*, 29(6):6–13, Nov 2009.

[28] W. A. Najjar, E. A. Lee, and G. R. Gao. Advances in the dataflow computational model. *Parallel Computing*, 25(13-14):1907 – 1929, 1999.

[29] C. North, N. Conklin, K. Indukuri, and V. Saini. Visualization schemas and a web-based architecture for custom multiple-view visualization of multiple-table databases. In *Information Visualization*, pages 211–228, 2002.

[30] S. G. Parker and C. R. Johnson. SCIRun: A scientific programming environment for computational steering. In *ACM/IEEE Conference on Supercomputing*, Supercomputing '95, 1995.

[31] S. G. Parker, D. M. Weinstein, and C. R. Johnson. *The SCIRun Computational Steering Software System*. Birkhäuser Boston, 1997.

[32] D. Ren, T. Hllerer, and X. Yuan. iVisDesigner: Expressive interactive design of information visualizations. *IEEE Transactions on Visualization and Computer Graphics (InfoVis'14)*, 20(12):2092–2101, Dec 2014.

[33] J. Roberts. Waltz - an exploratory visualization tool for volume data, using multiform abstract displays. In *Visual Data Exploration and Analysis V, Proc. SPIE*, volume 3298, pages 112–122, 1998.

[34] J. C. Roberts. On encouraging coupled views for visualization exploration. In *Visual Data Exploration and Analysis VI, Proc. SPIE*, pages 14–24, 1999.

[35] G. Ross and M. Chalmers. A visual workspace for hybrid multidimensional scaling algorithms. In *IEEE Symposium on Information Visualization (InfoVis'03)*, pages 91–96, Oct 2003.

[36] E. Santos, L. Lins, J. Ahrens, J. Freire, and C. Silva. VisMashup: Streamlining the creation of custom visualization applications. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1539–1546, 2009.

[37] A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. *Computer Graphics Forum (Proc. EuroVis)*, 2014.

[38] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *In IEEE Symposium on Visual Languages*, pages 336–343, 1996.

[39] A. Telea and J. J. van Wijk. *VISSION: An Object Oriented Dataflow System for Simulation and Visualization*, pages 225–234. Springer Vienna, Vienna, 1999.

[40] A. Telea and J. J. van Wijk. *SmartLink: An Agent for Supporting Dataflow Application Construction*, pages 189–198. Springer Vienna, Vienna, 2000.

[41] C. Upson, J. Faulhaber, T.A., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The application visualization system: a computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

[42] B. Victor. Drawing dynamic visualizations. https://vimeo.com/66085662, February 2013.

[43] J. Waser, H. Ribii, R. Fuchs, C. Hirsch, B. Schindler, G. Blschl, and M. E. Grller. Nodes on ropes: A comprehensive data and control flow for steering ensemble simulations. In *IEEE Transactions on Visualization and Computer Graphics*, number 12, 2011.

[44] C. Weaver. Building highly-coordinated visualizations in Improvise. In *IEEE Symposium on Information Visualization (InfoVis'04)*, pages 159–166, 2004.

[45] K. Wolstencroft, R. Haines, D. Fellows, A. R. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. N. de la Hidalga, M. P. B. Vargas, S. Sufi, and C. A. Goble. The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research*, pages 557–561, 2013.

[46] H. Wright, K. Brodlie, and M. Brown. The dataflow visualization pipeline as a problem solving environment. In *Proceedings of the Eurographics Workshop on Virtual Environments and Scientific Visualization*, pages 267–276, 1996.

[47] B. Yu, H. Doraiswamy, X. Chen, E. Miraldi, M. Arrieta-Ortiz, C. Hafemeister, A. Madar, R. Bonneau, and C. Silva. Genotet: An interactive web-based visual exploration framework to support validation of gene regulatory networks. *IEEE Transactions on Visualization and Computer Graphics (VAST'14)*, 20(12):1903–1912, Dec 2014.