Simplifying Filter/Flow Graphs by Subgraph Substitution

Florian Haag, Steffen Lohmann and Thomas Ertl
Visualization and Interactive Systems Institute (VIS), University of Stuttgart
Universitätsstraße 38 · 70569 Stuttgart · Germany
Email: {haag, lohmann, ertl}@vis.uni-stuttgart.de

***Abstract*—Although the filter/flow model is a useful concept for query visualization, the resulting graphs can be quite complex. We aim to reduce this complexity by substituting recurring subgraphs with more compact structures. Based on related work on extensions to the filter/flow concept, we have identified four recurring subgraphs that can be significantly simplified by substitutions. We describe these substitutions and discuss the preconditions that must be satisfied for their application. An example of news filtering illustrates how the substitutions can be used to simplify the overall structure of filter/flow graphs and increase their readability.**

I. Introduction

Tasks executed by information retrieval systems are becoming increasingly complex. Queries sent to such systems should limit the result set as precisely as possible. Various visualizations have been presented in an attempt to handle the complexity of such queries. In our work, we focus on the filter/flow visualization concept first proposed by Shneiderman [1], as it permits the representation of arbitrarily complex boolean expressions without the nessecity to restructure them into conjunctive or disjunctive normal forms.

In the filter/flow concept, a boolean expression is represented by a weakly connected directed graph. The graph is similar to a flowchart. The edges stand for the flow of data items. The nodes represent filter functions that may block certain data items. The boolean operations of conjunction and disjunction are expressed by connecting nodes sequentially and in parallel paths, respectively. Negations are expressed by inverted colors. An exemplary filter/flow graph is depicted in Figure 1. As seen in that figure, the concept can also be used to represent predicate logical expressions.

With this kind of graphs, arbitrary filter expressions can be represented. Users can trace the edges to understand how certain data items are evaluated by the expression. Typically, the repeated application of the same filter on a set of data items does not change the results, hence we will not consider cyclic graphs here.

As an example, Figure 2 shows a filter/flow graph that filters technology-related news reports. Only reports that were published less than one day ago are considered. Any reports published within the past hour are subject to some special restrictions: They must either contain at least three hyperlinks to external sites, or they must have been published by one of the user's favorite news sources. Eventually, the reports are sorted into two result sets based on a few keywords.

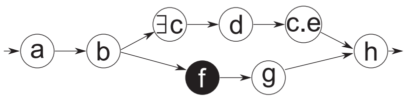


Fig. 1. A filter/flow graph that visualizes the expression $a \wedge b \wedge ((\exists c : d \wedge c.e) \vee (\neg f \wedge g)) \wedge h$. As suggested by Shneiderman [1] and Murray et al. [2], a negated filter is displayed in inverse colors.

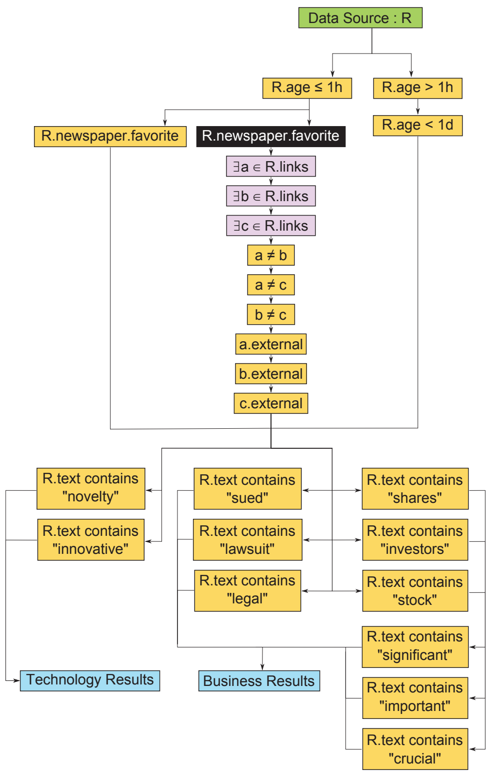


Fig. 2. An exemplary filter/flow graph. It filters technology-related news reports and sorts them into two result sets. The graph appears to be unnecessarily complex and contains various redundancies.

In the filter/flow graph, users can trace the arrows starting at the data source to find out whether and why a given report ends up in either of the result sets. Yet, the graph still appears unnecessarily complex and features some redundancies.

We will present an extended filter/flow graph notation in this work. Conventional filter/flow graphs can be transformed into the extended notation by substituting parts of the graphs.

To appear in:
2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2012); pp. 145 - 148
IEEE; doi: 10.1109/VLHCC.2012.6344501

Thereby, the complexity of the graphs is reduced and redundancies are avoided. We will concentrate on the syntactic aspects here; the semantic details are not discussed in this work. As an illustration, we will define several generic substitutions and apply them to the example graph from Figure 2.

II. Related Work

We have looked at query visualizations in general and concepts based on the filter/flow approach. We have attained an overview on alternatives to the basic filter/flow concept as well as on typical tasks and problems for user-defined queries.

Some of the approaches to visualize queries focus on representing boolean or predicate logical expressions, while others are described with set operations in mind. Essentially, these are equivalent in terms of expressivity, hence we will not distinguish between the two interpretations. The great variety of approaches includes Venn diagram-based concepts [3], [4], truth tables [5], [6], and custom graphical representations that rely on defined elements and alignments [7], [8]. These approaches often strive to represent all possible combinations of input variables, which increases visualization complexity for high numbers of input variables [9], or require a considerable learning effort.

The filter/flow concept has been built upon with various goals in mind. Kaleidoquery maps the full set of operations found in database query languages such as OQL, but partially abandons the flow-metaphor by allowing upstream effects in the flow [2]. In FindFlow [10] and DataMeadow [11], displaying intermediate results and unrestricted spatial layouts of the graphs are emphasized. Lark transfers flow-graphs onto a tabletop display [12]. Facet-Streams also uses a tabletop display and enriches the traditional concept with tangible components [13]. The mashup framework Yahoo! Pipes offers a variety of pre-defined node types for a filter/flow-like graph, though most of those types are not so much for filtering as for sorting and modifying news headlines, search results, or other data [14]. Other fields that query visualizations are used for range from digital libraries [3] over employee records [2] to geo-information systems [15], hotel and apartment search [13], [10], genetic ontologies [4] and car databases [5], [6].

The described approaches contain interesting ideas on how to extend the filter/flow concept. However, even though some of the related work implements improvements in single parts of the graphs, such as certain nodes with more than one inbound path in Yahoo! Pipes [14] or DataMeadow's filtering based on several parameters in a single node [11], few of the approaches focus on the underlying graph concept. Likewise, only few of the concepts support various result sets, a feature that is included in FindFlow [10], but is not possible in Yahoo! Pipes [14]. In order to incorporate all of these enhancements into a single concept, we consider it necessary to define generic ways of improving filter/flow graphs.

III. Improving Filter/Flow Graphs

The basic filter/flow concept found in all of the aforementioned variations can be formally defined as a connected directed graph $G = (V, E)$. As usual with directed graphs, V is the set of nodes (*filters*) and $E \subseteq V \times V$ is the set of edges (*flows*). Note that initial and final nodes are not explicitly defined: Any node (in particular, any number of nodes) may be used to display a result set, and nodes can be defined to output data without any input, so they act as data sources.

We have extended the filter/flow model in an attempt to simplify the graph structure. So far, there has been only one set of inbound flows and one set of outbound flows per node. In the extended filter/flow graph model, each node can have multiple *receptors* and *emitters*, each of which may be connected to an individual set of flows. In respect to the filter operation performed by the node, each receptor and each emitter is semantically different. Kaleidoquery [2] and Yahoo! Pipes [14] have implicitly done this in a similar fashion for a few node types, by providing additional emitters for nested sets and additional receptors for filter options.

A. Extended Graph Model

An extended filter/flow graph is defined as $\dot{G} = (\dot{V}, \dot{E}, I, O)$. I and O are the sets of receptors and emitters, respectively. $\dot{V} \subseteq \mathcal{P}(I) \times \mathcal{P}(O)$ is the set of nodes—each receptor and emitter can belong to only one node. As before, $\dot{E} \subseteq \dot{V} \times \dot{V}$ is the set of edges.

Two nodes v_1 and v_2 are called *directly connected*, iff there is an edge (o, i) with $o \in O_1, v_1 = (I_1, O_1)$ and $i \in I_2, v_2 = (I_2, O_2)$. This can also be written as $v_1 \to v_2$.

Any basic filter/flow graph $G = (V, E)$ can also be expressed as an extended filter/flow graph. Assume:

- $I = \{i_n \mid v_n \in V\}$
- $O = \{o_n \mid v_n \in V\}$
- $\dot{V} = \{(\{i_n\}, \{o_n\}) \mid v_n \in V\}$

Then $\dot{E}$ can be defined as $\{(o_n, i_m) \mid (v_n, v_m) \in E\}$. $\dot{G} = (\dot{V}, \dot{E}, I, O)$ is equivalent to $G = (V, E)$.

With this extended definition, complex filter functions can be represented by single filter nodes. This reduces the overall number of nodes and edges, avoids redundancies and thereby improves the overview of the graph.

B. Substitutions

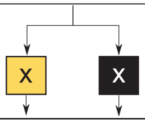
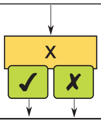
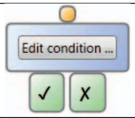
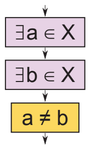
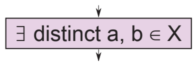
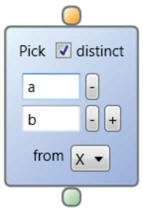
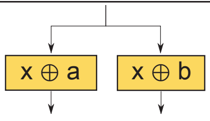
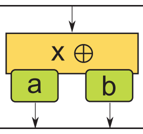
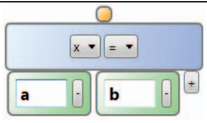
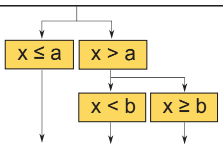
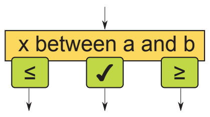
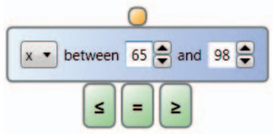
Before applying any substitution rules to a filter/flow graph, the order of some subgraphs may be swapped. This is possible for any subgraph $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{I}, \tilde{O})$ of $\dot{G}$ that fulfills the following condition:

- The subgraph must be connected to the rest of the graph by only one receptor and one emitter, i.e.
  $|\{i \mid (i \in \tilde{I}) \wedge (o_x \notin \tilde{O}) \wedge ((o_x, i) \in \dot{E})\}| \leq 1$ and
  $|\{o \mid (o \in \tilde{O}) \wedge (i_x \notin \tilde{I}) \wedge ((o, i_x) \in \dot{E})\}| \leq 1$

Two such subgraphs $\tilde{G}_1$ and $\tilde{G}_2$ have to meet the following requirements to be swapped:

- $v_1 \to v_2, v_1 \in \tilde{V}_1, v_2 \in \tilde{V}_2$, and the connection represents a commutative concatenation (here: a conjunction).
- Any outbound connection from $\tilde{G}_1$ leads to $\tilde{G}_2$, and any inbound connection from $\tilde{G}_2$ comes from $\tilde{G}_1$:
  $\forall (o_x, i_x) \in \dot{E} \setminus \tilde{E}_1 \setminus \tilde{E}_2 : (o_x \in \tilde{O}_1) \Leftrightarrow (i_x \in \tilde{I}_2)$

TABLE I
EXEMPLARY SUBSTITUTION RULES FOR A FILTER/FLOW GRAPH.

	Basic Filter/Flow Model	Extended Filter/Flow Model	Possible Representation in a User Interface
Conditional filters			
Nested set instances Note: Subgraphs with a higher number of variables (i.e. more $\exists$ and $\neq$ nodes) will be substituted accordingly.			
Binary operator filters Note: Any number of right-hand operands can be integrated into one resulting node.			
***Between* filters** Note: If any of the single filter nodes is not present (e.g. $x \leq a$), the respective emitter in the resulting node (e.g. $\leq$) will not have any outbound flows.			

The reordering helps to include a particular set of nodes in a connected subgraph that can then be substituted.

Subsequently, rules for the substitution of an arbitrary subgraph $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{I}, \tilde{O})$ with a subgraph $\tilde{G}' = (\tilde{V}', \tilde{E}', \tilde{I}', \tilde{O}')$ can be defined. The only requirement for such rules is that for each receptor i_x in $\tilde{G}$, there is an equivalent (with respect to the filter function of the node) receptor i'_x in $\tilde{G}'$, and for each emitter o_x in $\tilde{G}$, there is an equivalent emitter o'_x in $\tilde{G}'$. The substitution in a graph $\dot{G} = (\dot{V}, \dot{E}, I, O)$ will then yield a graph $\dot{G}' = (\dot{V}', \dot{E}', I', O')$ as follows:

- Any nodes that are part of $\tilde{G}$ are replaced:
  $\dot{V}' = \dot{V} \setminus \tilde{V} \cup \tilde{V}'$
- The same is done to receptors and emitters:
  $I' = I \setminus \tilde{I} \cup \tilde{I}'$
  $O' = O \setminus \tilde{O} \cup \tilde{O}'$
- The resulting filter/flow graph contains all of the original edges except those in the substituted subgraph plus those in the new subgraph. Moreover, all edges that connect the substituted subgraph with the rest of the graph are now connected to the new subgraph, that is:
  $\dot{E}' = \dot{E} \cup \tilde{E}' \setminus \{(i, o) \mid (i \in \tilde{I}) \vee (o \in \tilde{O})\}$
  $\cup \{(i, o'_x) \mid ((i, o_x) \in \dot{E}) \wedge (i \notin \tilde{I}) \wedge (o_x \in \tilde{O})\}$
  $\cup \{(i'_x, o) \mid ((i_x, o) \in \dot{E}) \wedge (i_x \in \tilde{I}) \wedge (o \notin \tilde{O})\}$

IV. Substitution Rules

We have identified some generic subgraphs that occur repeatedly in the literature about filter/flow graphs. These subgraphs can be substituted with single nodes. The substitutions are depicted in Table I and will be described in the following. As the subgraphs in the basic filter/flow model are matched with a certain degree of flexibility, only examples of the subgraphs are depicted. The table also shows possible user interface representations of the resulting nodes in order to illustrate the correlation between the extended graph structure and a user interface.

1) Conditional Filters: Conditional execution is a crucial concept in any form of programming. Nonetheless, building a filter/flow graph that performs two different actions based on a condition proves unnecessarily complicated. The filter nodes that make up the condition have to be duplicated and negated to express the *else*-path of the conditional flow. Among others, this is the case in Yahoo! Pipes [14]. Once nodes can have two emitters, the redundant negated filter can be skipped and one of the emitters can be used to output any data items that do not meet the condition.

2) Nested Set Instances: Frequently, instances of sets nested within the examined data items have to be considered. As such operations occur in relational and object-oriented databases, Kaleidoquery is an example for a filter/flow extension with support for this feature [2]. Since several distinct instances might be examined, we have included the restriction of inequality into the quantifier node. This reduces the required number of nodes as a single *distinct* flag can express the same restriction for all instances.

3) Binary Operator Filters: FindFlow [10], Facet Streams [13] and Seifert's work on digital library browsing [16] all use binary comparison operators in a way that the same left-hand value (without loss of generality) is compared to several right-hand values at a time. In the extended filter/flow model, this can be condensed to a single filter node that has one emitter for each desired right-hand operand.

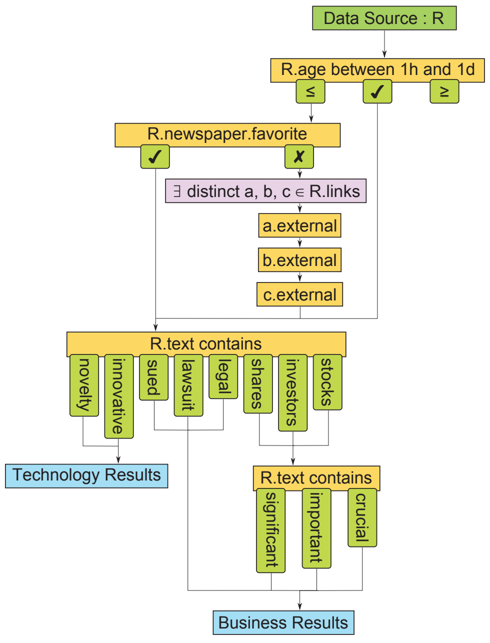


Fig. 3. An extended filter/flow graph that represents the same filter as Figure 2, after the substitutions outlined in Table I have been applied.

4) Between Filters: Ordinal values frequently have to be checked for inclusion in contiguous ranges of values, as happens in Kaleidoquery [2] and DataMeadow [11]. Several chained filters for lower and upper bounds of those ranges can be replaced with a single *between* filter. It can offer additional emitters for data items that were outside of the desired bounds, again reducing graph complexity and avoiding inverted duplicates of *greater than*/*less than* comparison nodes.

Figure 3 shows the more comprehensible filter/flow graph, after the substitutions from Table I have been applied to the graph presented in Figure 2. The number of nodes has been significantly reduced, as has the number of duplicate, negated and inverted filter nodes for the same restriction. As a result, the overall clarity of the graph has obviously been improved.

V. Conclusion and Future Work

We have presented a concept that simplifies the readability of filter/flow graphs by replacing recurring subgraphs. As an extension of the original filter/flow concept was used, we have formally described how any filter/flow graph can be expressed as an instance of the extended model. By defining appropriate substitution rules, subgraphs in the extended filter/flow graph can then be replaced with more comprehensible filter nodes according to our substitution concept. This simplifies the overall graph structure and avoids redundant filter settings. To illustrate the concept, we have presented four substitution rules for frequently used subgraphs. These four rules have been applied to an exemplary filter/flow graph, which was successfully simplified.

It will be interesting to compare different sets of both generic and application-/user-dependent substitution rules based on our concept in terms of user performance. We are in the process of implementing a prototypical filter/flow implementation in order to conduct user studies. The presented substitution capabilities will be included in that implementation, and we hope to take advantage of existing works about graph transformation in the process [17]. Ultimately, this may provide intriguing insights on how to make complex flow-like structures more comprehensible to users. Moreover, it is our hope that the formal definition of the filter/flow extension paves the way for further transformations to such graphs beyond subgraph substitutions.

References

[1] B. Shneiderman, “Visual user interfaces for information exploration,” Digital Repository at the University of Maryland (USA), Tech. Rep., 1991.

[2] N. Murray, N. Paton, and C. Goble, “Kaleidoquery: A visual query language for object databases,” in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ser. AVI ’98. New York, NY, USA: ACM, 1998, pp. 247–257.

[3] S. Jones, “Graphical query specification and dynamic result previews for a digital library,” in *Proc. of the 11th annual ACM Symposium on User Interface Software and Technology*, ser. UIST ’98. New York, NY, USA: ACM, 1998, pp. 143–151.

[4] H. A. Kestler, A. Müller, T. M. Gress, and M. Buchholz, “Generalized Venn diagrams: A new method of visualizing complex genetic set relations,” *Bioinformatics*, vol. 21, no. 8, pp. 1592–1595, 2005.

[5] J. Huo, “KMVQL: A visual query interface based on karnaugh map,” in *Proc. of the Working Conference on Advanced Visual Interfaces*, ser. AVI ’08. New York, NY, USA: ACM, 2008, pp. 243–250.

[6] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, “Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals,” *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 29–53, 1997.

[7] J. F. Pane and B. A. Myers, “Improving user performance on boolean queries,” in *CHI ’00 EA on Human Factors in Computing Systems*, ser. CHI EA ’00. New York, NY, USA: ACM, 2000, pp. 269–270.

[8] A. Spoerri, “InfoCrystal: A visual tool for information retrieval & management,” in *Proceedings of the Second International Conference on Information and Knowledge Management*, ser. CIKM ’93. New York, NY, USA: ACM, 1993, pp. 11–20.

[9] M. Chui and A. Dillon, “Speed and accuracy using four boolean query systems,” in *Proceedings of the Tenth Midwest Artificial Intelligence and Cognitive Science Conference*, ser. MAICS-99, 1999, pp. 36–42.

[10] T. Hansaki, B. Shizuki, K. Misue, and J. Tanaka, “FindFlow: Visual interface for information search based on intermediate results,” in *Proc. of the 2006 Asia-Pacific Symposium on Information Visualisation*, ser. APVis ’06. Darlinghurst, Australia: ACS, 2006, pp. 147–152.

[11] N. Elmqvist, J. Stasko, and P. Tsigas, “DataMeadow: A visual canvas for analysis of large-scale multivariate data,” in *IEEE Symposium on Visual Analytics Science and Technology*, ser. VAST ’07, Washington, DC, USA, 2007, pp. 187–194.

[12] M. Tobiasz, P. Isenberg, and S. Carpendale, “Lark: Coordinating co-located collaboration with information visualization,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1065 –1072, 2009.

[13] H.-C. Jetter, J. Gerken, M. Zöllner, H. Reiterer, and N. Milic-Frayling, “Materializing the query with facet-streams: a hybrid surface for collaborative search on tabletops,” in *Proc. of the 2011 Annual Conference on Human Factors in Computing Systems*, ser. CHI ’11. New York, NY, USA: ACM, 2011, pp. 3013–3022.

[14] P. Sadri, E. Ho, J. Trevor, K. Cheng, and D. Raffel, “Yahoo! Pipes,” February 2007. [Online]. Available: http://pipes.yahoo.com/

[15] A. Morris, A. Abdelmoty, B. El-Geresy, and C. Jones, “A filter flow visual querying language and interface for spatial databases,” *GeoInformatica*, vol. 8, no. 2, pp. 107–141, 2004.

[16] I. Seifert, “A pool of queries: Interactive multidimensional query visualization for information seeking in digital libraries,” *Information Visualization*, vol. 10, no. 2, pp. 97–106, 2011.

[17] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer, *Fundamentals of Algebraic Graph Transformation*. Berlin/Heidelberg, Germany: Springer, 2006.