Behavior-Driven Visualization Recommendation

David Gotz
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10578 USA
dgotz@us.ibm.com

Zhen Wen
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10578 USA
zhenwen@us.ibm.com

ABSTRACT
We present a novel approach to visualization recommendation that monitors user behavior for implicit signals of user intent to provide more effective recommendation. This is in contrast to previous approaches which are either insensitive to user intent or require explicit, user specified task information. Our approach, called *Behavior-Driven Visualization Recommendation* (*BDVR*), consists of two distinct phases: (1) pattern detection, and (2) visualization recommendation. In the first phase, user behavior is analyzed dynamically to find semantically meaningful interaction patterns using a library of pattern definitions developed through observations of real-world visual analytic activity. In the second phase, our BDVR algorithm uses the detected patterns to infer a user's intended visual task. It then automatically suggests alternative visualizations that support the inferred visual task more directly than the user's current visualization. We present the details of BDVR and describe its implementation within our lab's prototype visual analysis system. We also present study results that demonstrate that our approach shortens task completion time and reduces error rates when compared to behavior-agnostic recommendation.

Author Keywords
Intelligent visualization, Information visualization, User behavior modeling, Visualization recommendation

ACM Classification Keywords
Algorithms, Human Factors

INTRODUCTION
Visualization has long been used to harness the power of human perception to uncover insights from large collections of data. However, it is impossible to create a "one-size-fits-all" technique for visualizing data because every task and data set has its own unique properties. As a result, the information visualization community has developed a wide range of novel visual metaphors designed to address the needs of a broad spectrum of different types of data and visual tasks (e.g., correlation, comparison, etc.).

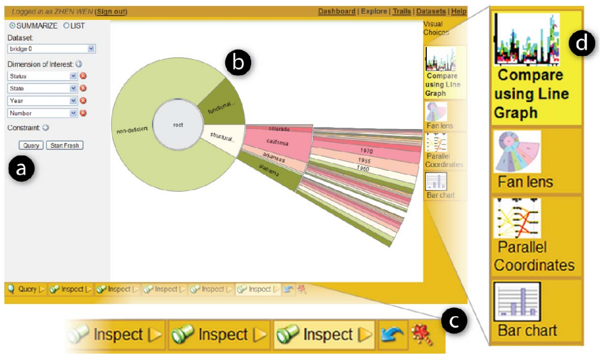


Figure 1. Behavior-driven visualization recommendation has been integrated into our lab's visualization system. Users can (a) issue queries and (b) interact with visualizations to analyze data. When a new recommendation is provided due to a user's behavior, he/she is notified via (c) a magic wand icon in the history panel and (d) a flashing segment on the recommendation sidebar. Users can accept the recommendation with a single click, or ignore it to continue uninterrupted.

Given the variety of options, how and when to use a particular visual metaphor requires a significant level of visual literacy. Unfortunately, average business users don't typically posses these skills. While domain experts within their own area, they usually have little or no training in visualization. Companies must therefore hire professional analysts (with visualization and analysis skills, but little domain knowledge) to generate reports that are in turn used by business-line employees to make decisions. This dramatically increases the cost of visualization-based solutions and places them beyond the reach of the legions of business users who might otherwise benefit from their capabilities.

Recognizing the challenge of supporting average users, several visualization systems have integrated intelligent algorithms to automatically compose or recommend effective visualizations given a user's task context. Existing systems can be generally classified into one of three categories. *Task-based systems* (e.g., [3]) use formal visual task descriptions as input to construct appropriate visual presentations. In contrast, *data property-based systems* (e.g., [14, 19]) focus on the data set being visualized and use features of the data itself as input to the visualization recommendation or composition algorithm. Finally, *hybrid systems* (e.g., [1, 24]) use a combination of both data properties and explicit representations of user intent to determine a proper visualization.

While existing approaches have helped reduce the skill barrier, visualization systems remain difficult to use. This is especially true for exploratory visual analysis tasks where users must interactively navigate through data to perform complex and data-driven analyses. In these cases, it is difficult for a user to explicitly describe his/her intent because it depends on what he/she finds during her task. Moreover, the user's intent can evolve as he/she examines the data and performs follow-up investigations. These factors make task-based or hybrid systems that require explicit task descriptions less appropriate. As a result, data property-based approaches are often used even though they don't incorporate any information about the user's visual task.

To overcome this limitation, systems often allow users to manually change visualization metaphors in the middle of a task. However, studies have shown that as a task evolves, users tend to remain with their current visualization as it becomes less useful despite the availability of better alternatives [12, 18]. This "sticky" behavior, which we call *visual inertia*, results in users working harder than needed because they are using the wrong visual tool for their task.

We hypothesize that *behavior-driven visualization recommendation (BDVR)*—an approach that actively monitors a user's normal analytic behavior to dynamically infer his/her ongoing visual task requirements to drive recommendations—can provide more accurate and task-relevant visualization recommendations when compared to previous approaches. Moreover, BDVR can provide dynamic recommendations as user intent evolves to help overcome the problems associated with visual inertia.

For example, a person comparing the price of hotels using a map-based visualization like Google Maps would need to iteratively open the callout for each hotel to view the required pricing information. This visual comparison requires significant user interaction. The user's iterative inspection of the callouts is just one instance of an important and common feature of visual analytic behavior, which we call *patterns*. Our observations of analytic activity have shown that users who remain with a sub-optimal visualization due to visual inertia often invent ad hoc patterns to accomplish their task [9]. Each pattern (e.g., the iterative inspection of the map's callouts) has a corresponding user intent (e.g., to compare hotel prices). BDVR would detect this behavior pattern and suggest an alternative presentation (e.g., a sorted bar chart) that could answer the user's question at a single glance.

In this paper, we describe a novel behavior-driven approach to visualization recommendation that automatically detects, interprets, and reacts to a user's natural behavior patterns. In contrast to previous work, we forgo explicit task descriptions and instead use implicit task information obtained by monitoring users' normal behavior. To the best of our knowledge, this is the first work on visualization recommendation that uses dynamically detected behavior features as implicit signals of users' task requirements. Specifically, our work on BDVR offers the following contributions:

- **A set of common visual analytic patterns.** We define four common pattern types that we observed across a wide range of users and tasks. We describe both the behavior that marks each pattern as well as the implied intent.
- **A two-stage algorithm for behavior-driven visualization recommendation.** We present a novel BDVR algorithm with two distinct phases. First, it monitors user interactions throughout the user's task and uses a rule-based approach to to detect semantically meaningful interaction patterns. Second, the detected patterns are used as input to our recommendation algorithm which infers user intent in terms of common visual tasks (e.g., comparison) and suggests visualizations that better support the user's needs.
- **An evaluation of BDVR's effectiveness.** We include results and analysis from a 20-person study of BDVR as applied within HARVEST, our lab's prototype visualization application [7]. The findings show that BDVR improves task completion time and reduces task error rate when compared to behavior-agnostic recommendation.

RELATED WORK

Our work on behavior-driven visualization recommendation is related to several different areas of research including studies of visual analytic behavior, prior work in behavior-driven interfaces, and previous approaches to intelligent visualization.

Visual Analytic Behavior

Various studies have been conducted to better understand user behavior during sensemaking tasks in general [20] and during visual analysis in particular. Several of those that focus on visualization [9, 12, 18] have observed significant levels of visual inertia in which users of a particular visualization are reluctant to change to alternative views.

For example, Kobsa [12] noted that, rather than switching views, users spent significant time attempting to use a visualization provided by default even when it was a poor match for their tasks. As captured in the performance model proposed by Plumlee and Ware [18], the additional interaction steps steps taken by these users can have a strong negative impact on their performance. Our work on BDVR, which is designed to reduce the negative impact of a user's visual inertia, is directly motivated by these studies.

In other work, several researchers have developed tools for modeling and capturing a visualization user's behavior. Such systems have been used for a wide variety of applications, including to enable re-use of complex analysis processes [2, 11], to support the re-visitation of previously viewed visualization states [4, 7, 13, 22], and to help ensure a comprehensive analysis for long-term tasks [17]. Our work similarly captures a model of the user's visual analytic behavior. However, in contrast to previous work, our algorithm actively analyzes the captured model to gather implicit input for dynamic visualization recommendation.

Behavior-Driven Interfaces

While behavior-driven techniques have not been previously applied to visualization recommendation, they have been explored in other domain areas. For example, TaskTracer [5] captures low level interaction events (e.g., focus events, text selection, etc.) and allows users to tag them as part of a specific task. This builds a task profile that can be used to assist in interruption recovery. More recently, TaskTracer has been extended to eliminate the manual tagging process to allow fully automated creation of task profiles [21]. In similar work, the SWISH system [16] monitors user desktop activity as a stream of windows events. Using a metric for "relatedness," SWISH can determine groups of windows that related to a single task.

Plan-based interfaces and implicit recommender systems are also related. Similar to BDVR, plan-based techniques (e.g., [6]) monitor semantic activity events to determine when a system should adapt to meet a user's changing needs. However, their plans consist of pre-defined finite automata that represent tasks (e.g., a classroom lecture). In contrast, we do not require any explicit task model. Recommender systems based on implicit feedback are also related. However, such systems often model aggregate populations (e.g., [15]) and/or employ user models based on information content rather than user interaction behavior (e.g., [10]). In contrast, we offer personalized behavior-based recommendation.

Intelligent Visualization

A large number of intelligent algorithms have been designed to automatically compose or recommend effective visualizations given a user's task context. Existing systems can be generally classified into three categories: (1) task-based, (2) data property-based, and (3) hybrid systems.

Task-based systems (e.g., [3]) use formal visual task descriptions as input to construct appropriate visual presentations. These tools typically rely on *a priori* knowledge of the user's visual task which must be explicitly defined. In contrast, *data property-based systems* (e.g., [14, 19]) focus on the set of data being visualized and use features of the data set as input to the visualization recommendation algorithm. These tools have no task model, and are therefore limited in their ability to customize intent-appropriate visualizations.

Combining these approaches are *hybrid systems* (e.g., [1, 24]). They use both data properties and explicit representations of user intent to determine a proper visualization. Our work on BDVR is most closely related to this category. However, it is impractical to require that users explicitly specify or pre-define models that can describe their evolving intent during dynamic visual analytic tasks. Our approach therefore actively monitors a user's behavior to automatically infer user visual task requirements at runtime.

PATTERNS DURING VISUAL ANALYSIS

To support our approach to behavior-driven visualization recommendation, we must be able to infer a user's analytic goals based on his/her behavior over the course of a task. Only then can we recommend task-appropriate visualizations. In this section, we discuss user action *patterns*, a key structural aspect of visual analytic behavior that maps directly to end-user analytic goals. Patterns are the key enabler for our behavior-driven recommendation algorithm. We first review the definition of patterns and discuss why they occur. We then illustrate a number of commonly observed patterns through real-world examples.

Patterns of Behavior

To better understand how users perform visual analysis tasks, we conducted a study in which users were asked to work on realistic analysis tasks using one of two commercial-grade visualization tools. We recorded video of each user's task and manually captured logs of their analytic activity. While many of the study results are beyond the scope of this paper [9], we focus on is one finding that motivates much of our work on BDVR: action *patterns*.

In our analysis of both the recorded video and activity logs, we modeled user behavior in terms of a user's analytic *actions* [8]. Actions provide a vocabulary (e.g., *Inspect*, *Filter*, and *Bookmark*) for modeling a user's visual analytic steps at a semantic level that is domain independent (e.g., independent of the tool-specific sequence of clicks, drags, and key-press events required to perform a specific action).

In our analysis, we found that nearly every user performed short, iterative sequences of analytic steps in order to accomplish a specific low-level analytic goal, such as visual comparison. For example, one user conducting a travel-related task performed an iterative sequence to compare hotel room rates for a set of hotels displayed on a map. The user clicked on each hotel's icon to open a call-out with the corresponding hotel rate and other details. We call this behavior an *Inspect* action. To find the cheapest rate, the user *Inspect*ed each hotel one after the other. The same action sequence was performed multiple times throughout the user's analysis, though often with different parameters (e.g., hotels in a different city, or restaurants instead of hotels).

We refer to these iterative behaviors as *patterns*. Each pattern consists of a repetitive sequence of actions performed by a user to accomplish a corresponding analytic goal. Using the example above, the user's pattern of iterative *Inspect* actions over a set of similar objects (e.g., hotels) reflects the user's intent to visually compare a specific object property (e.g., hotel rates). We found from our logs of user behavior that over 96% of users performed at least one type of pattern three or more times during a 30 minute task. To put this number in perspective, a randomly sequenced list of actions with length similar to those of the logs in our study would have similar structures only 9.6% of the time. This provides strong evidence ($p < 0.01$ for one-tailed Binomial test) that patterns are a structure of visual analytic behavior and do not occur by chance [9].

Not only are patterns widespread, but they occur at various scales. Most basic are *simple patterns* which occur when users perform a single type of analytic action repeatedly (e.g., the example describe above: $Inspect(hotel_a)$, $Inspect(hotel_b)$, $Inspect(hotel_c)$). More complex *compound patterns* occur as users repeat entire chains of actions to accomplish an analytic goal rather than repeating just a single type of action. For example, a user might perform

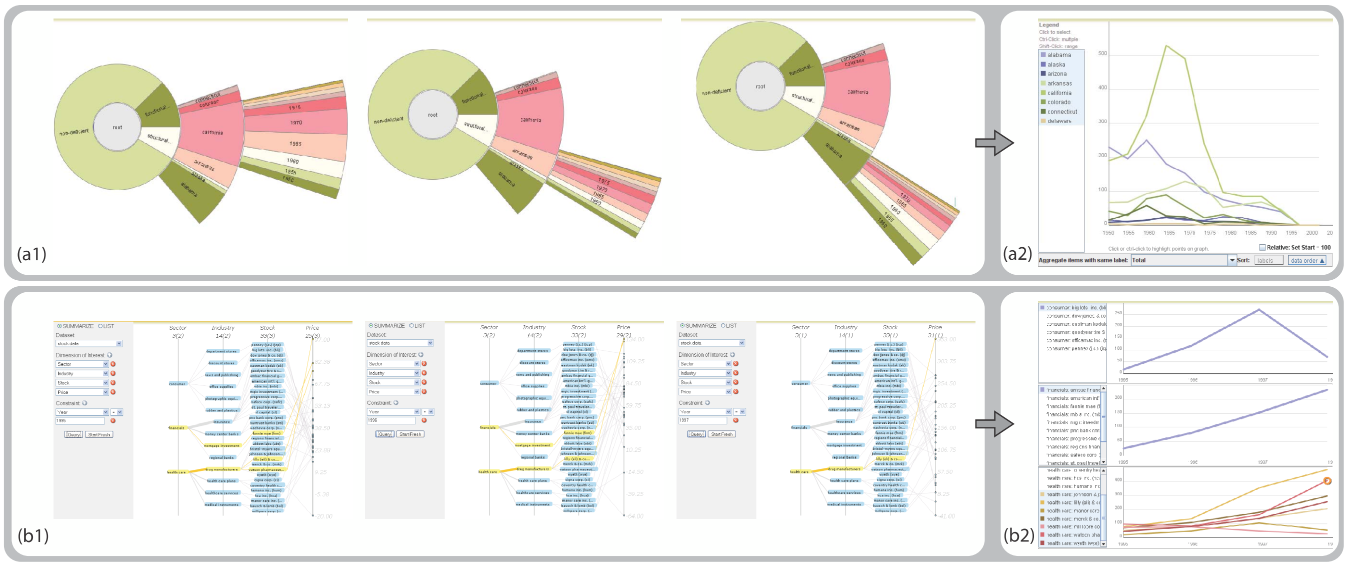


Figure 2. Examples of BDVR in HARVEST. After (a1) an iterative *Scan* pattern of temporal data, the system recommends (a2) a time chart as a more appropriate alternative given the current task context. Similarly, the system responds to (b1) a *Flip* pattern of various years of data by recommending (b2) a set of time charts that directly facilitate the user's intended comparison.

a complex pattern to compare hotels along multiple dimensions (e.g., room rate, user ratings, and amenities) by using multiple types of actions to gather relevant data for each hotel. We focus in this paper on simple patterns because they are less ambiguous in terms of user intention, though we consider compound patterns as important future work.

Based on the evidence gathered in our study, we believe that patterns are executed primarily to compensate for real or perceived limitations in the visual tool being used to perform a task. Because users cannot obtain the desired analytic result directly, they invent iterative procedures to perform the required task. These ad hoc procedures form patterns. For instance, the user in the hotel rate example performed a pattern to compare prices because the map-based visualization did not facilitate such a comparison directly. In contrast, no pattern was required to determine the geographical relationships between hotels because this comparison was directly supported by the map metaphor. Rather than switch to an alternate view of the hotels that would allow a direct visual comparison of the rates, the user's visual inertia kept him using the map despite its lack of functionality.

Common Patterns

The types of patterns performed by participants in our study settings vary widely. This is especially true for compound patterns which provide more room for variation. However, a handful of simple patterns were commonly observed across many users and tasks. In this section, we define the four most widespread pattern types: *Scan*, *Flip*, *Swap*, and *Drill-Down*.

Scan Pattern. A *Scan* pattern takes place when users iteratively perform *Inspect* actions over a series of visual objects that represent similar data objects. A *Scan* pattern indicates a user's intent to visually compare attributes of the objects being scanned. For example, consider the hierarchical FanLens visualization of bridge inspection data shown in Figure 2(a1). A user that needs to determine which state has the largest increase in structural bridge problems over any 5 year period would need to iteratively expand sections of the visualization for each state (e.g., California, then Arkansas, etc.) to expose the temporal data for comparison. This user is performing a *Scan* pattern over *States* to compare bridge failures over *Time*.

Flip Pattern. A *Flip* pattern occurs when users iteratively change filter constraints along a particular dimension to alter the set of data on display within a visualization. A *Flip* pattern indicates a user's intent to visually compare multiple sets of data. For example, consider the parallel coordinates visualization of stock data shown in Figure 2(b1). A user intending to identify the best single year stock performance by industry might iteratively change the year constraint of his/her query (e.g., from 1995 to 1996 to 1997, back to 1995, etc.) to explore the full space. This user is performing a *Flip* pattern over *Years* to do the intended task for each industry.

Swap Pattern. A *Swap* pattern occurs when users repeatedly re-arrange the order in which dimensions of data are presented within a visualization (e.g., swapping data dimensions on axes within a scatter plot or re-ordering axes within a parallel coordinates visualization). A *Swap* pattern corresponds to a user's intent to compare correlations between various dimensions of data. For example, consider a parallel coordinate visualization of real estate data. A user that needs to search for correlations between dimensions (e.g., house style vs. selling price, or school quality vs. taxes, etc.) would iteratively re-order the axes in search of correlations between neighboring dimensions. This user is performing a *Swap* pattern over *Real Estate Attributes* to achieve the intended correlation comparison.

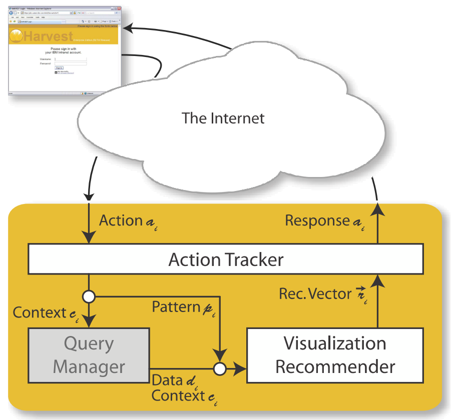


Figure 3. A simplified architectural view of our BDVR system.

Notation	Definition
a_i	The action performed in exchange i.
a_i^r	The overall system response to action a_i
c_i	The task context calculated by the action tracker module.
d_i	Data returned based on the task contact following action a_i.
$p_i =< t_p, G >$	The pattern report created after recording action a_i, where t_p is the type of pattern and G is the set of generalized parameters.
$\vec{r}_i$	The ranked list of visualization recommendations in response to action a_i.
τ_{active}	The active trail sequence, representing the chain of actions a_i that makes up a user's current line of inquiry.

Table 1. Notation used in the definition of our behavior-driven visualization recommendation algorithm.

Drill-Down Pattern. A *Drill-Down* pattern occurs when users repeatedly filter down along orthogonal dimensions of a data set. A *Drill-Down* pattern corresponds to a user's intent to narrow his/her analytic focus to a targeted subset of a data collection based on facets of metadata. For example, consider a user who wishes to examine only hotels with four-star restaurants, ratings of 3 stars or higher, and with fitness facilities. The user would iteratively filter the set of hotels three times, once for each criterion, to restrict the displayed options to only those which meet his/her requirements. This user is performing a *Drill-Down* pattern over *Hotels* to isolate the targeted subset of lodging options.

BEHAVIOR-DRIVEN VISUALIZATION RECOMMENDATION

Our behavior-driven approach to visualization recommendation exploits the patterns of interaction behavior that occur naturally during visual analysis to provide recommendations that better match a user's evolving analytic needs. In this section, we first provide a brief overview of HARVEST, our lab's visual analytic system within which our recommendation algorithm has been developed. We then describe the two main phases of our approach: (1) pattern detection, and (2) visualization recommendation. Finally, we describe how recommendations are conveyed to users and how the system responds when users accept a suggested alternative.

The HARVEST Visual Analytics System

Our algorithm for BDVR is built into HARVEST, a visual analytics system designed in our lab to help everyday users derive insights from large amounts of data [7]. HARVEST embraces intelligent user interface techniques to support a general business user population that has no special training in visualization or analysis. It contains a set of smart visualization components, a visualization recommendation engine, and an action tracking module that captures a semantics-based record of a user's analytic activity for reuse and insight provenance. A screenshot of the HARVEST system is shown in Figure 1, and the remainder of this subsection provides a brief overview of the system architecture.

HARVEST is a thin-client web application that provides access rich server-side functionality (see Figure 3). The HARVEST client can run in standard web browsers including Firefox and Internet Explorer. Through a combination of client technologies (HTML, Java, Javascript, and AJAX), the user interface allows users to issue queries and directly interact with visualizations to perform interactive tasks. For example, users can interactively select individual wedges in the chart shown in Figure 1 to filter down to a subset of data. Alternatively, users could click on one of the visualization choices shown in the sidebar to quickly change the display to an alternative visual metaphor.

In response to a user's interaction with the system, an *Action* a_i is reported by HARVEST's smart visualization components [7] and forwarded over the network to the server where it is processed first by the *Action Tracker* module. Actions are semantically meaningful units of users' visual analytic activity, such as *Query*, *Filter*, or *Bookmark* [8]. The Action Tracker records each new action in two distinct data structures. First, it maintains a graph-based structure that represents the logical sequence of discreet analytic steps in a user's unfolding analysis, which we call a trail graph. Second, the action tracker builds a summary of the user's task context c_i which represents an aggregated view of the user's current line of inquiry (e.g., a set of data constraints and the user-selected visual metaphor). The Action Tracker then analyzes the trail graph in search of semantic behavior patterns. If a pattern is found, a pattern report p_i is generated and sent forward through the HARVEST pipeline. The pattern detection process is described in detail later in this section and is a key element of our behavior-driven approach to visualization recommendation.

The task context c_i built by the Action Tracker is forwarded to the *Query Manager* which translates the data constraints portion of c_i to SQL and issues a query against an external content store. It sends the retrieved data d_i together with c_i to the next component in our pipeline.

The *Visualization Recommending* module receives the data d_i and task context c_i each time an action flows through the

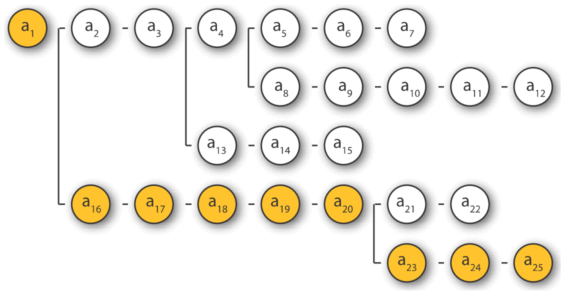


Figure 4. The Action Tracker maintains a graph-based data structure representing the logical structure of the user's activity. The user's active trail τ_{active} (highlighted nodes) represents a linear summary of the user's current line of inquiry.

server. In addition, whenever a behavior pattern is detected by the action tracker, a report p_i is passed as additional input to the recommender. From these inputs, the visualization recommendation algorithm generates a ranked vector of data- and behavior-appropriate visualization recommendations, $\vec{r}_i$. Each recommendation specifies both (1) a visual metaphor and (2) a description or instantiating the visual metaphor (e.g., a data-mapping that connects the data in d_i to the visual slots exposed by the specified metaphor). The recommendation vector is passed back through the action tracker which records the system's response a_i^r before sending it on to the client for rendering through the user interface. We describe how recommendations are conveyed to users in more detail later in this section.

As this flow demonstrates, both the Action Tracker and Visualization Recommender modules play a key role in our approach to behavior-driven recommendation. We therefore focus on these two components as we define the BDVR algorithm.

Detecting Patterns

The Action Tracker module is responsible for monitoring a user's visual analytic activity and detecting semantically meaningful patterns of behavior. As shown in Figure 3, user behavior is reported in the form of semantic actions. In response to each incoming action a_i, the Action Tracker updates its internal graph-based representation of the logical flow of the user's analysis activity. Most relevant to BDVR, this representation includes a linear sequence of user actions that we call the user's *active trail* τ_{active}, which represents the logical sequence of actions corresponding to the user's current line of inquiry. The pattern detection is applied to the sequence of actions in the active trail.

For example, Figure 4 shows a high-level overview of the graph-based action tracker representation of a user's session that contains 25 user actions. The actions are numbered based on the temporal order in which they occurred, while the five branches in the structure represent five logical lines of inquiry explored by the user. Highlighted in the figure are nine actions that make up the user's active trail, τ_{active}.

$$\tau_{active} =< a_1, a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{23}, a_{24}, a_{25} > \quad (1)$$

```
<pattern_definition type="flip">
        <symbol_definition action="filter" symbol="F"/>
        <symbol_definition action="query" symbol="Q"/>
        <sequence regex="F{3}?" parameter_constraint="identical"/>
        <sequence regex="Q{3}?" parameter_constraint="identical"/>
</pattern_definition>
```

Figure 5. Our rule-based pattern detection algorithm uses declarative XML-based definitions (such as this one for the *Flip* pattern) to allow the definition of new patterns without any code changes.

The algorithm for constructing the graph-based data structure is motivated by our action-based model of visual analytic activity [8] and is beyond the scope of this paper.

Pattern Definitions

After integrating a newly reported action a_i into τ_{active}, the action tracker initiates its search for semantically meaningful action patterns. It employs a rule-based approach to pattern detection based on a library of pattern definitions. The library contains one definition for each supported pattern type (e.g., *Scan* or *Flip*). Each pattern definition includes two parts: (1) one or more *regular expressions* that encode the action sequences that form a particular pattern, and (2) a set of *pattern feature guidelines* that specify requirements on the parameters of each action in a matching sequence.

For example, the *Scan* pattern in Figure 2(a1) occurred when a user performed iterative *Inspect* actions that focus on similar visual objects (e.g., structural bridge problems in California, Arkansas, Alabama, etc.). Therefore, a *Scan* can be defined by the regular expression "$I\{4\}?$" (which matches sequences of four or more *Inspect*s at the end of τ_{active}) and pattern feature guidelines that require each *Inspect* to reference the same type of data (e.g., structural bridge problems in various states).

We create similar definitions for each pattern type. A pattern definition can have more than one regular expression if multiple distinct action sequences can correspond to the same user intention. For example, the *Flip* pattern (see Figure 5) has two regular expressions and occurs via either a sequence of queries (from our query tool), or a sequence of filters (performed through direct interaction with a visualization tool).

Pattern feature guidelines are just as critical as regular expressions in defining patterns and can by themselves help distinguish between different user intents. For example, the *Drill-Down* pattern is defined using the exact same regular expressions as *Flip*, but with different pattern feature guidelines. *Drill-Down* requires all filters to reference unique data types (e.g., $Filter[Status = \mathbf{StructuralProblem}]$, $Filter[Year = \mathbf{2006}]$, and $Filter[State = \mathbf{Alabama}]$), while *Flip* requires that all filters reference identical data types (e.g., $Filter[Year = \mathbf{2006}]$, $Filter[Year = \mathbf{2007}]$, and $Filter[Year = \mathbf{2006}]$).

Detection Algorithm

Pattern detection is performed via a three phase process. In the first *expression matching* phase, τ_{active} is compared against the regular expressions associated with each definition in the pattern library. For example, if the final four actions in τ_{active} were all *Inspect* actions, it would match the *Scan* definition. Definitions with at least one matching regular expression are forwarded to the next stage of the process.

The second phase is *feature validation*. For each definition forwarded from the first phase, the matching actions in τ_{active} are vetted against the pattern feature guidelines to determine if a valid pattern has been found. This is performed by analyzing the parameters for all actions in the matching sequence. For instance, consider the *Inspect*s illustrated in Figure 2(a1). These actions all focus on the subset of data for bridges of $Status = \textbf{StructuralProblem}$ across various *State* values. This constitutes a *Scan* because all steps reference the same dimensions ($Status$ and $State$) as required by the pattern definition. In contrast, if each *Inspect* focused on different dimensions they would not constitute a *Scan*.

The final phase is *generalization*. When a pattern definition has been satisfied, generalization is performed by examining the shared parameters of the matching actions to determine the scope of the pattern. At the end of this stage, a pattern report $p_i =< t_p, G >$ is forwarded to the visual recommendation module. Here, t_p is the pattern type and G is the set of generalized parameters. Continuing with the *Scan* example from Figure 2(a1), the generalization process would detect that all actions reference $Status = \textbf{StructuralProblem}$ but vary along the $State$ dimension (e.g., California, Arkansas, Alabama). As a result, the pattern report defined below would be passed on to the recommendation module.

$$p_i =< Scan, \left[\begin{array}{c} Status = \textbf{StructuralProblem} \\ State = x \end{array} \right] > \tag{2}$$

Visualization Recommendation

Given a detected user behavior pattern, our visualization recommendation engine automatically recommends the top-N suitable visualizations to the user. This engine extends our previous effort in using example-based learning to automate visualization generation [24] in two respects. First, the engine infers a user's intention from detected patterns to recommend alternative visual metaphors. Second, the engine tailors the instantiation description of the visual metaphor to the inferred visual task.

Recommendation Based on Detected Patterns

A user's intention in visual tasks dictates the type of visualization that is most appropriate to use. However, it is difficult for users to explicitly specify their intention, particularly during complex analyses in which the user's intention dynamically evolves as new insights are discovered. Therefore, our recommendation engine utilizes the pattern reports produced by our pattern detection algorithm as implicit signals of users' intended visual tasks.

The first stage of our visual recommendation algorithm is to infer a user's *visual task* from both the pattern report p_i and the context our engine maintains about the current visualization (e.g., the data d_i and visual metaphor presented to the user at the time the pattern was performed). We represent a visual task as: $VisualTask =< t_v, D >$. Here, t_v is the type of visual task. We map the implied intent of our defined patterns to specific visual tasks such as visual comparison, one of the most common types of visual tasks [3]. D is the set of data dimensions associated with the visual task. Each data dimension in D is represented as a 4-tuple: $Dimension =< n_d, t_d, C_d, r_d >$. Here, n_d is the name of the dimension, t_d is the data type of the dimension, C_d is a set of constraints on the dimension, and r_d describes the role of the dimension in the intended visual task.

For each dimension in d_i, our engine first determines its role r_d based on p_i. r_d can be one of three values: *constraint*, *differentiating*, or *characterizing*. A *constraint* dimension is used to limit the scope of visual comparison (e.g., $Status = \textbf{StructuralProblem}$ in Equation 2). A *differentiating* dimension distinguishes the objects being compared (e.g., the $State$ dimension in Equation 2). Both of these dimension types can be obtained from the G in a pattern report. In contrast, a *characterizing* dimension is a characteristic of the objects being compared and is determined by the visualization context following a rule-based approach. For example, in the *Scan* pattern that triggered the report in Equation 2 there are two *characterizing* dimensions: $Year$ and $Number$. These correspond to the number of bridge problems in a particular year, the attributes of the $State$ wedges being compared by the user in Figure 2(a1). After r_d is determined for a dimension, its name n_d and data type t_d can be obtained from the data description for d_i. The set of constraints C_d is a union of the constraints in the user's task context c_i and the G in a pattern report.

The inferred $VisualTask$ is then used together with the properties of data d_i to retrieve a list of potentially useful visual metaphors from a visualization example corpus [24]. In this corpus, the visual metaphors are annotated with the visual tasks and data properties for which they are suitable. For example, the line graph metaphor is annotated as suitable for comparing numerical values along time. During the retrieval of visual metaphors, the annotations are used to compare with the given $VisualTask$.

Tailoring Visualization Instantiations

For each of the suitable visual metaphors ranked by the recommendation engine, we generate an *instantiation description*. The description includes information on: (1) appropriate data transformations for d_i; and (2) a data-mapping that connects the transformed data to the visual slots exposed by the metaphor.

The descriptions are tailored by our engine to satisfy the inferred visual task using the list of *characterizing* data dimensions in r_d. However, for a given $VisualTask$, there may be too many characterizing dimensions to be effectively visualized by a single visualization. For example, in Figure 2(b2) a line graph can be used to compare the $Price$ dimension along the $Year$ dimension. However, the line graph cannot effectively visualize the $Sector$ and $Industry$ data at the same time as the other dimensions.

To handle this issue, our engine selects and groups characterizing data dimensions so that the data can be visualized using multiple instances of the visual metaphor (e.g., multiple line graphs). For this purpose, we use a content selection algorithm [23] to group data dimensions. Consider the example in Figure 2(b1), where the $Stock$ dimension (a stock's name) is highly related to the $Price$ dimension because there is a one-to-one mapping. In contrast, the $Sector$ and $Industry$ dimensions have a one-to-many relationship with $Stock$ and are therefore less tightly related. As a result, our algorithm places $Stock$ data on individual line graphs. Of the remaining dimensions, $Sector$ has better grouping properties (i.e., a smaller number of groups) and is used to partition stocks into multiple time charts. This is encoded in an instantiation description along with the needed data transformations to produce the set of line graphs in Figure 2(b2).

Conveying Recommendations to Users

A critical component of any recommendation algorithm is conveying the appropriate alternatives to the user. HARVEST uses a passive approach to notify users of new visualization recommendations that occur as a result of behavior patterns. First, the system displays a magic wand icon near the user's navigation history to indicate that their behavior has caused a response by the system. Second, a flashing segment on the recommendation sidebar is illuminated to display the suggested alternative. This interface is shown in Figure 1.

The passive approach was chosen to avoid forcefully disrupting a user's visual analysis behavior. Unexpected changes to the visual presentation can break a user's visual momentum and slow his/her progress. However, the disadvantage of the passive approach is that users may miss or ignore the notification and stay with a sub-optimal visualization. Our current approach was chosen based on informal experimentation. More analysis of this passive/active tradeoff in user interface design is an important area of future study.

Accepting Recommendations

A user can accept a recommendation with a single click on the flashing recommendation segment shown in Figure 1. In response to the user's click, the HARVEST client sends a *ChangeView* action to the HARVEST server specifying the clicked recommendation.

On the server side, the action is processed first by the Action Tracker which records the action and updates the user's task context c_i if required. The context c_i will change whenever there are new data constraints specified in the data transformation description section of the recommendation. If c_i has changed, the Query Manager module issues a query to get an updated set of data records. The updated data is then passed to the Visualization Recommender to instantiate the accepted recommendation. Finally, the HARVEST server sends out the overall system's response a_i^r, which includes both the instantiated visualization and updated c_i.

Once the client receives a_i^r, it updates the visualization canvas as well as the data constraints in the query panel (Figure 1a). The updated query panel informs the user of the inferred data constraints which have been automatically integrated into c_i and the new visualization. If the inferred constraints don't match users' actual intent, the user can interactively change the constraints directly through the query panel.

EVALUATION

We have tested the BDVR approach extensively using the HARVEST visual analytics system [7] on a set of investigative tasks over multiple data sets. In the study reported here, the implementation of BDVR focused on two patterns: *Scan* and *Flip*. Our study quantitatively and qualitatively compared user task performance using HARVEST under two settings: (1) with BDVR enabled, and (2) with BDVR disabled. When BDVR is disabled, HARVEST uses only data properties to recommend suitable visualizations.

Study Design and Methodology

We designed six visual analysis tasks using three distinct data sets. Each of the following data sets was selected from a real-world analytic task:

- **Data set 1:** data on the status of bridges in various U.S. states from 1950 to 2005 (1836 records, 4 dimensions).
- **Data set 2:** social network data of a set of employees (138 employees, each employee has 6 attributes).
- **Data set 3:** data on stock prices for Standard & Poor's 500 companies from 1995 to 2001 (3500 records, 5 dimensions).

For each data set we designed two tasks, each of which asked a user to find a set of targets that met certain criteria (e.g., stocks that belong to particular sectors and satisfy certain performance criteria). By design, all tasks required users to perform multiple data comparisons. To achieve a balanced, within-subject comparison, two similar but not identical tasks were designed for each data set. For example, for the stock data set, the two tasks focused on different sectors and time ranges and used different performance criteria.

We recruited twenty users for our study, thirteen of whom were male while seven were female. Their ages varied from mid 20s to early 50s. They had some experience with general visualization tools (e.g., Google Maps), but were not visualization experts. We used a within-subject comparison methodology and asked each user to perform all 6 tasks. Among the 6 tasks, 3 were performed using HARVEST with BDVR enabled, and the other 3 with BDVR disabled. To avoid potential biases such as learning effects, we permuted the order and conditions of tasks.

Each user was given a 15-minute tutorial on HARVEST. For each task, the user was first given the task description and the question to be answered. They were then given access to HARVEST which had the data set loaded and an initial set of visualization recommendations. The top recommendation was instantiated by default and the user was able to switch to alternative visualizations with a single click. Because HARVEST knew nothing of the user's analytic intent at the beginning of a task, its initial visualization recommendations were based on data properties alone.

We allotted 10 minutes for each task and recorded the actual task completion time, which was counted from the moment when the user started using HARVEST to the moment when the user claimed that the task was completed. During each task, we also recorded the visualizations used and behavior patterns detected. At the end of each task, we collected subjective feedback via two questions. First, we asked the user to rate the provided visualization recommendations on how well the visual metaphors supported the tasks using a scale of 1 (least) to 5 (best). We then asked users to comment on the least liked and most liked aspects of HARVEST's action tracking and visualization recommendation features.

Results and Analysis

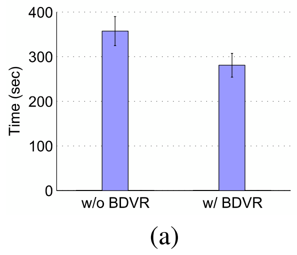
The data collected in our study shows that BDVR can effectively detect user behavior patterns in 41 trials out of all 60 trials with BDVR enabled (69%). Moreover, users accepted the visualization recommendations made based on the detected patterns in 36 out of those 41 trials (88%).

We further analyzed both objective and subjective data to assess the impact of BDVR on user task performance. We first computed the mean task completion time over all the tasks and users. With BDVR enabled, the mean completion time was significantly improved ($p < 0.001$ in ANOVA test) from 357 seconds to 281 seconds, amounting to a 21% reduction (Figure 6a). Based on our observations, the significant reduction in time can be attributed to a reduction in users' visual inertia with BDVR enabled. The initial visualization provided by HARVEST based on data properties alone was usually sufficient to give users an overview of the data. However, the initial visualizations became less suitable later as users focused on specific visual comparisons. In such situations, alternative visualizations were tried in 90% of the trials with BDVR enabled versus 78% in the trials with BDVR disabled. Moreover, of those that did explore alternative visualizations, users with BDVR chose task-appropriate alternatives (e.g., rating higher than 3) more often than those without BDVR. Accordingly, users were more likely to use task-appropriate visualizations with BDVR enabled (82% of the trials) than with BDVR disabled (55% of the trials). One user commented:

"When it was flashing [to notify me of new recommendations], I realized that I was not using the best [visualization]. So I switched to other visualizations [to find a better one]. [It turned out] the flashing recommendation was pretty good."

We also measured task error rate by the percentage of questions that the users correctly answered. Our results indicated a statistically significant difference in task error rate between the two settings ($p < 0.01$) (Figure 6b). The mean error rate across all tasks and users was reduced from 26.7% (without BDVR) to 5.0% (with BDVR), an 81% reduction. We attribute this sharp reduction to BDVR's ability to help push users towards visualizations that were more appropriate for the visual comparisons they needed to perform in the tasks.

Users also indicated a subjective preference for behavior-



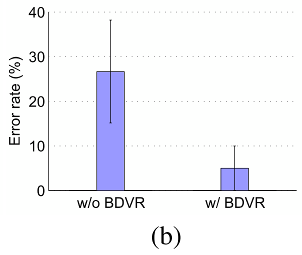


Figure 6. Mean and 95% confidence interval of (a) task completion time, and (b) task error rate with BDVR disabled or enabled

Visualization Type	% of Detected Patterns
FanLens	52%
Social network diagram	80%
Parallel coordinates	69%
Bar chart	0%
Line graph	0%
Scatter plot	0%

Table 2. Six types of visualization used in the study.

driven recommendations because they felt that BDVR helped them when they became *"stuck"* while using a particular visualization. Over all the tasks, users overwhelmingly favored the top visualization recommendation with BDVR enabled (mean rating of 3.75 out of 5) over the recommendations with BDVR disabled (mean rating of 2.65).

Our data also provided insight on the influence of visualization types on BDVR. Overall, there were 6 types of visual metaphors used in the tasks. For each visualization type, we computed the percentage of trials where a pattern was detected out of all the trials where the visualization type was used and BDVR was enabled (Table 2). The results show that behavior patterns were more likely to be detected for more sophisticated visualizations, which often encoded complex data sets and required more user interaction to understand the data.

Finally, our study exposed three limitations of our current work. First, some users felt that the system's notification of BDVR recommendations might be too subtle to notice. Currently, to avoid interrupting a user's cognitive process, we passively notified users by flashing the recommendation on the right side of the browser. This was sometimes missed by users who were concentrated on the visualization at the center. Second, users wanted HARVEST to detect more complex patterns. For example, users sometimes intermixed *Filter* actions with *Inspect* actions to compare sets of stocks, creating patterns that are more complex than those captured by our current set of pattern definitions. Finally, we observed a few occasions where users abandoned a system-provided recommendation to return to their original view. However, because our recommendation was passively provided, users did not complain. Moreover, despite these cases BDVR still provides a statistically significant improvement in performance.

CONCLUSION

In this paper, we described a novel approach to visualization recommendation that detects and reacts to semantically meaningful user behavior patterns during visual analysis. Our approach, *Behavior-Driven Visualization Recommendation* (*BDVR*), can recommend more accurate and task-relevant visualizations than previous solutions. Moreover, our behavior-driven approach can help overcome visual inertia by providing dynamic recommendations throughout a user's ongoing task that better match his/her evolving visual analytic intent.

We described our two phase BDVR algorithm and its implementation within the HARVEST system. The first phase monitors user interactions throughout a user's task and analyzes the recorded behavior to detect semantically meaningful patterns. The second phase recommends visualizations that more effectively support the user's implied visual task which we inferred from the detected patterns.

Finally, we presented the results of a user study that demonstrates the effectiveness of our approach. In comparison to visual recommendations computed without behavior-based feedback, BDVR yields statistically significant improvements in both task completion time and task error rate. In addition, subjective user feedback suggests that users strongly prefer BDVR recommendations over those provided when behavior-based feedback is not used.

While our initial results are promising, there remain several areas to explore in future work. In particular, more study must be conducted to understand how aggressively recommendations should be conveyed to the user. In general, recommendations that are too subtle are likely to be ignored by users, while those that are too aggressive can interrupt the user's cognitive process. Finding the proper balance remains a challenge. In addition, providing an explanation for why a particular recommendation was suggested would make the system's behavior more transparent to users. Another topic for future study is the incorporation of higher-level pattern detection and generalization.

REFERENCES

1. E. Andre and T. Rist. *Intelligent User Interfaces*, chapter 4 (The Design of Illustrated Documents as a Planning Task), pages 94–116. AAAI Press/The MIT Press, 1993.
2. L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: Enabling interactive multiple-view visualizations. In *IEEE Vis*, 2005.
3. S. M. Casner. A task-analytic approach to the automated design of graphic presentations. *ACM Trans. on Graph.*, 10(2):111–151, April 1991.
4. M. Derthick and S. F. Roth. Data exploration across temporal contexts. In *IUI*, 2000.
5. A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI*, pages 75–82, New York, NY, USA, 2005. ACM.
6. D. Franklin, J. Budzik, and K. Hammond. Plan-based interfaces: keeping track of user tasks and acting to cooperate. In *IUI*, pages 79–86, New York, NY, USA, 2002. ACM.
7. D. Gotz, Z. Wen, J. Lu, P. Kissa, M. Zhou, N. Cao, W. H. Qian, and S. X. Liu. Harvest - visualization and analysis for the masses. In *IEEE InfoVis Poster*, 2008.
8. D. Gotz and M. X. Zhou. Characterizing users' visual analytic activity for insight provenance. In *IEEE VAST*, 2008.
9. D. Gotz and M. X. Zhou. An empirical study of user interaction behavior during visual analysis. Technical Report RC24525, IBM Research, 2008.
10. Y. Hijikata. Implicit user profiling for on demand relevance feedback. In *IUI*, 2004.
11. T. Jankun-Kelly, K.-L. Ma, and M. Gertz. A model for the visualization exploration process. In *IEEE Vis*, 2002.
12. A. Kobsa. An empirical comparison of three commercial information visualization systems. In *Proc. of InfoVis*, 2001.
13. M. Kreuseler, T. Nocke, and H. Schumann. A history mechanism for visual data mining. In *IEEE InfoVis*, 2004.
14. J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. on Graph.*, 5(2):110–141, 1986.
15. D. W. Oard and J. Kim. Implicit feedback for recommender systems. In *AAAI Workshop on Recommender Systems*, 1998.
16. N. Oliver, G. Smith, C. Thakkar, and A. C. Surendran. Swish: semantic analysis of window titles and switching history. In *IUI*, 2006.
17. A. Perer and B. Shneiderman. Systematic yet flexible discovery: Guiding domain experts through exploratory data analysis. In *IUI*, 2008.
18. M. D. Plumlee and C. Ware. Zooming versus multiple window interfaces: Cognitive costs of visual comparisons. *ACM Trans. on Computer-Human Interaction*, 13(2), 2006.
19. S. F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein. Interactive graphic design using automatic presentation knowledge. In *CHI*, 1994.
20. D. M. Russell, M. J. Stefik, P. Pirolli, and S. K. Card. The cost structure of sensemaking. In *CHI*, 1993.
21. J. Shen, L. Li, T. G. Dietterich, and J. L. Herlocker. A hybrid learning system for recognizing user tasks from desktop activities and email messages. In *IUI*, 2006.
22. Y. B. Shrinivasan and J. J. van Wijk. Supporting the analytical reasoning process in information visualization. In *CHI*, 2008.
23. M. X. Zhou and V. Aggarwal. An optimization-based approach to dynamic data content selection in intelligent multimedia interfaces. In *UIST*, 2004.
24. M. X. Zhou and M. Chen. Automated generation of graphical sketches by example. In *Proceedings of IJCAI*, pages 65–74, 2003.