ShapeSearch: Flexible Pattern-based Querying of Trend Line Visualizations

Tarique Siddiqui, Paul Luh, Zesheng Wang, Karrie Karahalios, Aditya Parameswaran
University of Illinois (UIUC)
{tsiddiq2,luh2,zwang180, kkarahal,adityagp}@illinois.edu

ABSTRACT

Finding visualizations with desired patterns is a common goal during data exploration. However, due to the limited expressiveness and flexibility of existing visual analytics systems, pattern-based querying of visualizations has largely been a manual process. We demonstrate ShapeSearch, a system that enables users to express their desired patterns in trend lines using multiple flexible mechanisms — including natural language and visual regular expressions, and automates the search via an optimized execution engine. Internally, the system leverages an expressive *shape query algebra* that supports a range of operators and primitives for representing ShapeSearch queries. In our demonstration, conference attendees will learn how the various components of ShapeSearch help accelerate scientific discovery by automating the search for meaningful patterns in trend lines in domains such as genomics and material science.

PVLDB Reference Format:
Tarique Siddiqui, Paul Luh, Zesheng Wang, Karrie Karahalios, Aditya Parameswaran. ShapeSearch: Flexible Pattern-based Querying of Trend Line Visualizations. *PVLDB*, 11 (12):1962-1965, 2018.
DOI: https://doi.org/10.14778/3229863.3236235

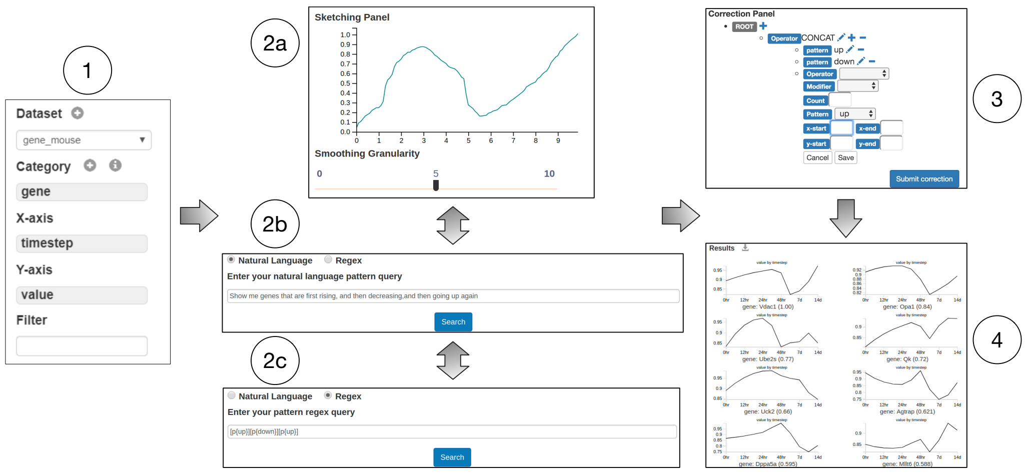
1. INTRODUCTION

While visual analytics systems such as Tableau [1] and Spotfire provide intuitive mechanisms to specify and generate visualizations, they do not provide ways to automate the search for desired visual patterns or trends. Due to this, analysts often spend hours examining many visualizations, all to find those that satisfy some desired visual pattern, e.g., a product whose sales is decreasing over time. Recent work, such as our tool Zenvisage [13] and Google Correlate [3], as well as older tools such as TimeSearcher [2] and Query-by-Sketch [14] aim to alleviate this burden by providing a "canvas" for users to sketch a visual pattern (or to drag-and-drop an existing visualization), with the system automating the search for visualizations that match that pattern, using an appropriate distance metric. We characterize such tools as *visual query systems* [6]. While these tools are a useful starting point in supporting the automated search for visualizations that match desired patterns, they offer limited flexibility in the pattern specification mechanism. In particular, if, instead of finding a product whose sales is decreasing over time, the analyst wanted to find a product whose sales is decreasing over some 3 month window (without specifying when), or if the analyst wanted to find a product whose sales has many increasing and decreasing portions (without specifying when these portions occur, their magnitude or their width), a precise "sketch" on a canvas can prove to be too rigid of a specification of the desired pattern. In such cases, the analyst may want more flexible mechanisms to specify the pattern of interest. Consider the following real-world example:

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/8.
DOI: https://doi.org/10.14778/3229863.3236235

Example. *Biomedical researchers often study changes in gene expression while investigating the impact of drugs on disease treatment. To do so, they often explore a large collection of trend line visualizations, one corresponding to each gene, with the x-axis as the time-stamps, and the y-axis as the expression values (depicted in Figure 1, explained subsequently), and search for desired patterns. For example, when influenced by an external factor, a gene can get induced (up-regulated), or repressed (down-regulated), or can have both induced or repressed pattern within a certain time window. Moreover, there can be further variations in the scale, repetitions, or the rate of change of these patterns depending on the type and influence of external factors. Based on their domain understanding and past experience, researchers first hypothesize the expected change in expression that an affected gene should depict. Lacking advanced programming skills, they then generate thousands of visualizations, one for each gene, using a domain-specific visualization tool, and manually inspect them for the hypothesized patterns, a cumbersome, time-consuming, and error-prone process. Visual query systems [2, 13, 14] partially mitigate the manually intensive search, by accepting a sketch of the desired gene expression profile as input, and automating the search for matching visualizations. Unfortunately, these systems tend to perform "exact" matching with the values of the drawn trend, ignoring what the researcher may instead be interested in—semantic features such as slope, smoothness, rate of change, and peaks in expression values.*

To address these challenges, we developed ShapeSearch, a flexible pattern querying system that supports multiple mechanisms for helping users express and search for desired patterns, with the following contributions:
ShapeQuery ***Algebra***. We developed a ShapeQuery *algebra* that abstracts key shape-based primitives and operators, encapsulating a variety of typical patterns that are often of interest in trend line visualizations. For developing this algebra, we used a corpus of real-world pattern queries, collected via Mechanical Turk.
Natural Language Interface. Since our typical end-users, such as our biomedical researchers, are often not proficient in programming, we built a natural language interface within ShapeSearch for the flexible specification of ShapeQueries, coupled with a sophisticated parser and translator for converting these queries to Shape-


Dataset

gene_mouse

Category

gene

X-axis

timestep

Y-axis

value

Filter

Natural Language [✓ Natural Language] Regex

Enter your natural language pattern query

Show me genes that are first rising, and then decreasing,and then going up again

Natural Language [✓ Regex]

Enter your pattern regex query

[p{up}][p{down}][p{up}]

Figure 1: ***The ShapeSearch Interface, consisting of six components. 1) Data upload and attribute selection, 2) Query specification: 2a) Sketching canvas, 2b) Natural language query interface, 2c) Regular expression interface, and 3) Correction panel, and 4) Top K results***

Query algebra. Unlike structured query languages, end-users do not need to know the syntax and semantics of the internal representation. One downside is that natural language queries can often be incomplete, and have subjective and/or ambiguous interpretation. ShapeSearch leverages a mix of automated and user-driven ambiguity resolution mechanisms to tackle these issues.

Regular Expression and Sketching. In addition to natural language, ShapeSearch supports a regular-expression-based interface for expert users, as well as a sketching interface (similar to typical visual query systems). These three interfaces can be used simultaneously, based on the complexity of the desired pattern, and users can switch between them as needed. All three interfaces ultimately compile down to ShapeQueries.

Scalable Evaluation. Naively matching a ShapeQuery to each visualization in a large collection of visualizations, wherein each visualization can be composed of thousands of datapoints, can take a really long time. ShapeSearch employs a scalable evaluation engine that facilitates efficient but approximate and perceptually-aware matching of visualizations to ShapeQueries, leveraging optimizations such as pruning and reuse of intermediate results.

Related Work. Our work draws on prior work on search and retrieval of temporal data, and natural language interfaces for data exploration. Most existing tools, including visual query systems [2, 3, 13] leverage distance measures such as Euclidean distance that perform "exact" matching between the query and target trend lines, or Discrete Time Warping [11] and cross-correlation-based metrics [9] that admit more "tolerant" matching by transforming the query trend line to the target trend line. None of these metrics allow users to explicitly control the local and global features of the trend lines being identified, such as the slope of the trend, number of peaks, and their combinations. The Shape Definition Language (SDL) [10] lets users search for basic shapes in trend lines using a more structured keyword-based language but offers users limited flexibility in terms of shape primitives and operators. Moreover, trend lines in SDL are preprocessed and indexed in advance, whereas ShapeSearch enables ad-hoc pattern matching, and uses adaptive pattern-specific query optimizations at runtime for improving performance. A number of keyword and natural language interfaces for querying databases [5, 7] and generating visualizations [4, 12] have been developed recently. However, since the underlying ShapeQuery algebra in ShapeSearch is different from relational algebra, existing parsing and translation strategies from the database-specific work cannot be easily adapted; the natural language interfaces for visualizations in prior work do not make use of visual patterns or features to facilitate search, instead opting to be an interface to a typical visual analytics system such as Tableau.

2. SYSTEM OVERVIEW

We now briefly describe the usage scenario of ShapeSearch for our genomics application, along with the systems architecture.

2.1 Usage Scenario

Figure 1 depicts the ShapeSearch interface, with an example query from genomics where a biomedical researcher wants to search for genes whose expression values follow a specific pattern: first rising, then going down, and finally rising again. Only genes that are influenced during a treatment depict such changes in their expression values. In order to search for the desired pattern, the researcher first loads the dataset (mouse gene data) via form-based options on the left (Box 1), and then selects the space of visualizations to explore by setting category as gene, X axis as time, and Y axis as expression values. Next, the researcher enters their intended pattern search query using natural language (Box 2b). When the researcher types in their query, ShapeSearch also recommends potential phrases based on historical queries via auto-complete. Alternatively, if the query consists of a simple trend, they can draw the corresponding sketch (Box 2a), and if it is too complex to be expressed via either natural language or a sketch, they can issue a visual regular expression query (Box 2c). On submitting, the query is parsed and translated to a structured representation consisting of ShapeQuery operators and primitives. The structured representation is shown to user as part of correction panel (Box 3). Simultaneously, the system executes the current best guess for the ShapeQuery and visualizes the top-k matching results in the results panel (Box 4). In case of ambiguity or incorrect translation, the researcher can make corrections with the help of correction panel, or rephrase their query.

2.2 System Architecture

Figure 2 depicts the architecture of ShapeSearch. The system is designed as a web application, with a front-end as described in the previous subsection. All front-end queries are issued to the back-end using a REST protocol, which then parses and translates it into an intermediate ShapeQuery algebra representation. The back-end supports an ambiguity resolver that uses a set of rules to resolve syntactic and semantic ambiguities, for making corrections, and for adding missing values to the user queries. The corrected queries are also displayed on the front-end for user-driven validation and additional corrections. The validated query is finally optimized and executed by the execution engine. The top visualizations that best match the ShapeQuery are finally returned to the front-end in the the JSON format.

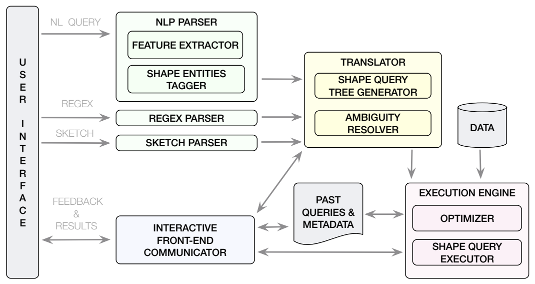


Figure 2: ***System Architecture***

3. SHAPE-QUERY ALGEBRA

We now describe the syntax and semantics of a ShapeQuery, an internal structured representation of flexible user queries issued via any one of the three querying mechanisms. The execution engine optimizes and executes each ShapeQuery for finding visualizations that best match the query.

At a high-level, a ShapeQuery consists of one or more *subcomponents*, called ShapeSegments, interconnected via *operators*. A ShapeSegment defines a pattern over a sub-region of the visualization, and can have three primitives—*Location, Pattern, and Modifier*. Operators interconnect one or more ShapeSegments and help combine multiple ShapeSegments to define a complex pattern. As an example, consider a simple query to find genes whose expression patterns first rise and then go down. Here, the query consists of two ShapeSegments, first for the rising pattern, and second for the decreasing, interconnected with an operator that indicates that the two ShapeSegments are to be concatenated. To help express a variety of patterns, the system currently supports six operators, developed after analyzing a large corpus of pattern queries we collected via Mechanical Turk. We describe the basic primitives and operators supported by ShapeQuery algebra in Table 1.

Regular expressions can be built on top of primitives and operators from ShapeQuery algebra to express a wide variety of ShapeQueries. A regular expression for ShapeQuery has the following format: $[S]$ OP $[S]$ $OP \ldots OP$ $[S]$. Here $[S]$ denotes a ShapeSegment, and OP denotes an operation that combines two ShapeSegments. Without going into the formal specification, we now provide a few examples to illustrate the syntax and semantics of regular expression.

Example 1: $[\mathbf{p}\{up\}, \mathbf{xr}\{1,3\}]$ $[\mathbf{xr}\{4,8\}, \mathbf{yr}\{7,5\}]$

The expression represents a ShapeQuery consisting of two ShapeSegments, the first one depicting an upward trend between the x range 1 to 3, and the second one a downward trend (not mentioned in the query, but implicit from the specified x and y ranges) with y range 7 to 5 and x range 4 to 8. Between two ShapeSegments, the CONCAT operator (,) is assumed to be the default operator.

Example 2: $[\mathbf{p}\{peak\}, \mathbf{q}\{2,4\}, \mathbf{xr}\{1,12\}]$

This query searches for 2 to 4 peaks between x range 1 to 12. Instead of specifying both minimum and maximum number of desired peaks, via $q : \{2,4\}$, one can specify the exact number of matches, or use symbols such as $*$, ? or $+$ for any, zero or one, or at least one matches respectively.

Example 3: $[\mathbf{p}\{up\}, \mathbf{xr}\{1,10\}] \cdot [\mathbf{p}\{peak\}, \mathbf{xr}\{3,5\}]$

Here, we are looking for an overall upward trend between x values 1 to 10 with a peak between 3 to 5. The AND operation (.) between the two ShapeSegments denotes that both the patterns should be matched simultaneously.

4. NATURAL LANGUAGE TRANSLATION

There are multiple steps involved in the translation of natural language queries to their intermediate ShapeQuery representation.

Shape Primitives and Operators Recognition. A natural language query consists of a sequence of words, where each word either maps to one of the entities (ShapeQuery primitives or operators), or is a noise word. We follow a three-step process in tagging words to their corresponding entities. In the first step, we use a standard Parser [8] for POS tagging and parsing dependencies between the words. Next, using a set of rules, we annotate each word to be either noise or non-noise. For non-noise words, if they match with high confidence (e.g., low edit distance) to a frequently-used word for any entity (expanded via WordNet synsets), we add another annotation called "predicted primitive". In the final step, we extract a set of predefined features (in addition to annotations) for each of the non-noise words, and use a conditional-random field (CRF) model to predict their corresponding entities. For training the CRF, we collected and tagged 250 natural language queries via a Mechanical Turk study, where users were asked to describe patterns in trend line visualizations. On cross-validation, the model had an F1 score of 81% in accuracy.

ShapeQuery **Tree Generation.** We use a context-free grammar to represent structural relationships between entities. Using the grammar, ShapeQuery tree generator groups related entities into a ShapeSegment, and combines multiple ShapeSegments using operators. ShapeSegments correspond to the leaves in the query tree, whereas operators correspond to the intermediate nodes. If no valid ShapeQuery tree is identified, the ambiguity resolver module tries to predict the nearest possible valid ShapeQuery tree.

Ambiguity Resolution. Ambiguity can occur at various levels in the query, both in the structure (syntax) as well as the meaning (semantics). Common causes of structural inconsistencies are grammatical errors, and missing connector words, all of which can lead to incorrect tagging of entities. For example, there could be two patterns within the same ShapeSegment, or multiple missing entities in the same segment. Similarly, semantic inconsistencies occur when there are multiple conflicting meanings of the same phrase. For instance, a parsed ShapeSegment might represent an increasing pattern from y=10 to y=2, where the pattern type conflicts with y values. In case of such ambiguities, the system tries to predict the best possible intermediate representation, using rules (constructed based on our analysis of training corpus and past queries), and then displays the parsed ShapeQuery in the correction panel (as depicted in Figure 1 Box 4) for edits from the user.

5. SHAPEQUERY EXECUTION

The query executor takes the validated ShapeQuery as input, and scores each visualization, returning the top-k ones. Conceptually, to score a visualization given an ShapeQuery tree, we need to proceed in a bottom-up fashion, starting from scoring the ShapeSegments (leaves), and combining the scores as we move up the tree based on the operators in the intermediate nodes. Since ShapeSegments can be aligned in multiple ways, especially when the location primitives are missing, the nodes in the tree will record multiple scores, one for each possible alignment. The maximum score at the root node is the score for the visualization.

Scoring. Each visualization is assigned a score between -1.0 (poor match) and 1.0 (perfect match). A score for the ShapeSegment is a simple-weighted sum of two sub-scores. The first sub-score measures how well the portion of the visualization matches the specified pattern semantically. Patterns (e.g., up, down, flat) are scored using a function that depends on slope and residual sum of squares, both derived from fitting a linear regression model to the portion

Table 1: ***ShapeQuery Algebra basic primitives and operators***

Name	Type	Description
LOCATION	Primitive	Defines the boundaries of the ShapeSegment between which the pattern is matched, consisting of four sub fields: X start position (xs), X end position (xe), Y start position (ys), and Y end position (ye).
PATTERN	Primitive	Defines a characteristic pattern that the user is looking for in the ShapeSegment. The system supports basic patterns that are commonly used for characterizing trend lines such as up, down, flat, noise, as well as complex patterns such as peak that are made out of the basic patterns.
MODIFIER	Primitive	Defines the way a pattern is matched. A modifier can be subjective specifying how the pattern should change over time (e.g., sharply, gradually), or can be quantitative enumerating the number of times a pattern should occur in a ShapeSegment (e.g., at least, at most, between).
CONCAT	Operator	Concatenates two ShapeSegments, the second one appended after the first one.
AND	Operator	Simultaneously matches two ShapeSegments. Unlike in concat, the two ShapeSegment can share the same sub-region of the visualization (e.g., genes that are going up overall and also have a small peak at some point, or ones that are both flat and noisy.
OR	Operator	Indicates a choice between two ShapeSegments. The systems picks the ShapeSegment that matches better (e.g., genes whose expressions are either up-regulated or down-regulated.
NEGATE	Operator	Matches the opposite of the pattern expressed in the ShapeSegments (e.g., instead of rising or falling patterns, one can say not flat pattern.
GROUP	Operator	Scopes and assigns precedence between two or more ShapeSegments.

of visualization. The second sub-score measures how close the x and y values are to the one in the ShapeSegment, using L1-norm distance.

Naive Approach. One naive way of aligning a ShapeQuery to a visualization is to try all possible partitions of the visualization, and for each partition candidate, calculate the score. Not surprisingly, the time complexity of this approach is prohibitively large, i.e, $O(n^k)$ where n is number of possible partitions, and k is the number of ShapeSegments in the ShapeQuery. A comparatively faster solution is to use dynamic programming that reuses intermediate computations. Still, the time complexity is quadratic ($O(n^2k)$), and does not scale when n is large.

Divide and Conquer. We note that for a large number of patterns such as up, down, and flat, the score of the parent node can be computed from the statistics of the children nodes (and thus we can avoid calculating the scores for the intermediate nodes from scratch), and secondly, patterns in the ShapeQuery follow a strict order, that can be leveraged for pruning intermediate computations. Based on these observations, we first calculate scores for all possible ShapeSegment subsequences from the ShapeQuery for each of the leaf nodes, and then for intermediate nodes we merge subsequences from children nodes, pruning out subsequences which are absent in the ShapeQuery. Moreover, after merging we only need to retain the maximum score for a subsequence. Overall, there are as many merges as there are nodes in the tree ($2n$), and therefore we can calculate the score for each visualization in linear time ($O(nk^4)$, k is typically small).

Optimizations across visualizations. Even though the above approach is linear in the number of the candidate partitions, the execution time can still be non-interactive, especially when the number of visualizations is large. In order to improve the performance, we further apply a pruning-based optimization based on the following observation. The score of a subsequence at a given node is bounded between the maximum and the minimum scores of the subsequences from the children nodes, with the scores decreasing montonically as we move up the tree. Thus, we prune the search space of visualizations by maintaining upper and lower bounds of scores for nodes in the ShapeQuery tree, using which the visualizations whose scores can never be in the top-k are pruned.

6. DEMONSTRATION WALKTHROUGH

As part of the demonstration, attendees will be able to (1) see the limitations of existing visual querying systems [2, 3, 13] in the flexible search for patterns, (2) understand the features and capabilities of ShapeSearch using hands-on examples on multiple real world datasets from different scientific domains. In particular, we will demonstrate how natural language queries and visual regular expressions can help in issuing more expressive pattern search queries, and how ShapeSearch can effectively match a variety of patterns; and finally (3) learn about the internals of ShapeSearch, especially the natural language parser, and query optimizations.

In order to achieve the above objectives, we will be using three real world datasets. 1) *Genomics dataset*. This dataset consists of gene-gene and protein interactions used by biomedical researchers at the NIH sponsored genomics center at Illinois for understanding relationships and interactions among genes, and changes in gene expression patterns before and after a clinical trial. We will demonstrate how our system can help accelerate pattern discovery by quickly and effectively finding genes with different shapes and number of peaks. 2) *Battery dataset*. This dataset consists of properties of different chemicals explored by battery scientists at Carnegie Mellon University for understanding and correlating properties of chemicals for designing more efficient Lithium-Ion batteries. ShapeSearch can help scientists find chemicals that follow a specific behavior or pattern on change in physical properties such as temperature, pressure, or when associated with other chemicals. 3) *Real Estate dataset:* This dataset consist of housing prices for different cities in the US from 2004 to 2014. Attendees will use ShapeSearch on this dataset for finding cities and states showing opposite trends in their housing prices, especially during the period of economic downturn.

Acknowledgements. We thank the anonymous reviewers for their valuable feedback. We acknowledge support from grant IIS-1513407, IIS-1633755, IIS-1652750, and IIS-1733878 awarded by the National Science Foundation, and funds from Microsoft, Adobe, Toyota, Google, and the Siebel Energy Institute. The content of this document is solely the responsibility of the authors and does not necessarily represent the official views of any funding agencies or other organizations.

7. REFERENCES

[1] Tableau public (www.tableaupublic.com/). [Online; accessed 3-March-2014].
[2] P. Buono et al. Interactive pattern search in time series. In *Visualization and Data Analysis 2005*, volume 5669, pages 175–187, 2005.
[3] M. et al. Google correlate whitepaper. 2011.
[4] T. Gao et al. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *UIST'15*, pages 489–500, 2015.
[5] R. J. L. John, N. Potti, and J. M. Patel. Ava: From data to insights through conversations. In *CIDR*, 2017.
[6] D. J.-L. Lee, J. Lee, T. Siddiqui, J. Kim, K. Karahalios, and A. Parameswaran. Accelerating scientific data exploration via visual query systems. *arXiv preprint arXiv:1710.00763*, 2017.
[7] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1):73–84, 2014.
[8] C. D. Manning et al. The stanford corenlp natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014.
[9] J. Paparrizos and L. Gravano. k-shape: Efficient and accurate clustering of time series. In *SIGMOD'15*, pages 1855–1870. ACM, 2015.
[10] R. A. G. Psaila and E. L. Wimmers Mohamed &It. Querying shapes of histories. *Very Large Data Bases. Zurich, Switzerland: IEEE*, 1995.
[11] L. Rabiner et al. Considerations in dynamic time warping algorithms for discrete word recognition. *IEEE TASSP*, 26(6):575–582, 1978.
[12] V. Setlur et al. Eviza: A natural language interface for visual analysis. In *UIST'16*, pages 365–377. ACM, 2016.
[13] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.
[14] M. Wattenberg. Sketching a graph to query a time-series database. In *CHI '01 Extended Abstracts*.