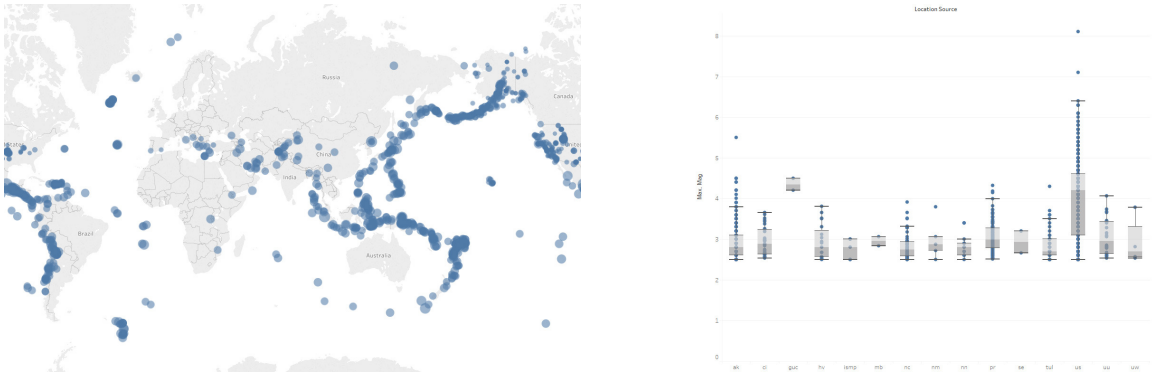
Rapid Sequence Matching for Visualization Recommender Systems

Shaoliang Nie*
North Carolina State University

Christopher G. Healey†
North Carolina State University

Rada Y. Chirkova
North Caralina State University

Juan L. Reutter
Pontificia Universidad Católica de Chile, Santiago, Chile



(a) (b)

Figure 1: Earthquake magnitude visualizations: (a) locations, magnitude *mag* → size; (b) boxplots for *mag* by region

Abstract

We present a method to support high quality visualization recommendations for analytic tasks. Visualization converts large datasets into images that allow viewers to efficiently explore, discover, and validate within their data. Visualization recommenders have been proposed that store past *sequences:* an ordered collection of design choices leading to successful task completion; then match them against an ongoing visualization construction. Based on this matching, a system recommends visualizations that better support the analysts' tasks. A problem of scalability occurs when many sequences are stored. One solution would be to index the sequence database. However, during matching we require sequences that are *similar* to the partially constructed visualization, not only those that are identical. We implement a locality sensitive hashing algorithm that converts visualizations into set representations, then uses Jaccard similarity to store similar sequence nodes in common hash buckets. This allows us to match partial sequences against a database containing tens of thousands of full sequences in less than 100 ms. Experiments show that our algorithm locates 95% or more of the sequences found in an exhaustive search, producing high-quality visualization recommendations.

Index Terms: Information systems—Data access methods; Similarity measures; Human–centered computing—Visualization theory, concepts and paradigms

1 Introduction

With the ever increasing amount of data being captured, new methods to store, manage, and analyze this data are critical. One promising approach is to *visualize* the data, allowing users to apply their human visual system to explore a dataset through high-level tasks like discovery, hypothesis testing, and pattern detection. Although the area of visualization was formally proposed in 1987 [32], the line and bar chart were first presented in the late 1700s by Playfair [34].

Visualization has proved invaluable across a range of areas including physical science, economics, and cybersecurity [14, 15, 21]. As data sizes grow and problems become more complex, however, it is difficult for domain experts to construct *effective* visualizations. Fully automatic approaches have struggled to manage context, domain-specific knowledge, and ambiguity. Collaborations between domain and visualization researchers have led to systems for specific domains [11, 40]. Access to visualization experts is limited, however. Because of this, visualization recommender systems have been proposed to allow domain experts to construct high-quality visualizations independently.

Fig. 1 shows 1,989 earthquakes recorded from July 2–September 23, 2017 [41]. The task is to locate outliers by magnitude. An intuitive solution is to display the earthquakes on a map with magnitude represented by size, then visually search for outliers (Fig. 1a). Unfortunately, this approach works poorly. A better strategy is to use boxplots that are designed to highlight outliers in non-normal distributions (Fig. 1b). A recommender could match a novice user's initial design choices to successful visualization sequences, then recommend a boxplot visualization.

To build a database of successful sequences, recommender systems often use crowdsourcing to combine visualizations by both expert and novice users [13, 28, 38]. A standard construction method is to choose an initial data–feature mapping: a visual representation for some or all of the data attributes in the dataset; then evaluate the visualization's ability to solve an analytic task. Improvements are added iteratively: change an aspect of the visualization, evaluate it, and continue until a successful visualization is constructed. The design is captured as a single-path graph (a *sequence*), where nodes represent visualizations and edges represent iterative operations (*e.g.*, changes to the data–feature mapping, filtering, aggregation, and so on). Sequences that generate "successful" visualizations are stored

*e-mail: snie@ncsu.edu
†e-mail: healey@ncsu.edu

in a sequence database, where individual sequences are combined into a composite graph representation of all sequences.

Our main goal in this paper is to develop a sequence database for rapid comparison, storage, and retrieval of sequence patterns for visualization recommender systems, specifically:

1. Sequences similar to (as opposed to identical to) a target sequence can be rapidly located.
2. Storage scales to thousands or tens of thousands of sequences with a near-constant retrieval time.

An important challenge is the design and implementation of an efficient sequence matching algorithm. During discussions with database experts, it was stated that a query response threshold of 100ms is recommended for any interactive system such as ours. We integrated a locality sensitive hashing (LSH) algorithm into a recommendation system we are developing. Given a similarity algorithm and a crowdsourced sequence database, we can match nodes from an initial design accurately and in real-time to nodes in successful sequences. These are presented to the user as recommendations to guide them to an efficient and effective completion of their task. Our novel contributions in this paper include:

1. A method to represent a specific visualization state in a sequence node using a simple data–feature mapping.
2. Transformations that convert visualization representations into set-based notations for Jaccard similarity.
3. LSH to place similar nodes in common hash buckets for rapid retrieval.
4. Experimental analysis showing that LSH retrieval needs only a few tens of milliseconds, even for large hash tables.
5. Discussion of converting common visualization representations into set notation, to demonstrate generalizability.

2 BACKGROUND

We briefly review existing visualization recommender systems and visualization representations, then present an overview of locality sensitive hashing, Jaccard similarity, and their relationship to traditional hash tables.

2.1 Visualization Recommenders

Numerous visualization recommendation systems have been proposed, using historical data to drive their recommendations, or precomputed knowledge models based on dataset and task.

Vartak et al. discuss requirements for visualization recommendations [42]: *relevance*, *surprise*, *non-obviousness*, *diversity*, and *coverage*. Koop presents VisComplete, a system that creates a *provenance database* of visualization pipelines (*i.e.*, crowdsourced historical data), then matches this against partially created pipelines to provide "recommendations by consensus" [27]. VisComplete uses graph theoretic algorithms to perform partial-to-full pipeline matching, making it a good candidate for our algorithms. Work in our laboratory developed ViA [19] by combining human perception [18] and mixed-initiative planning to rapidly locate visualizations that "fit" a user's analytic tasks. Wongsuphasawat et al. introduce Voyager 2, a mixed-initiative system where users and a recommendation algorithm collaborate to identify charts that: (1) provide broad coverage for data overviews; and (2) support relevant recommendations for specific question answering [48].

2.2 Interest Weights

Recommenders often depend on assigning an "interestingness" score to different visualizations. Measuring user interest is complicated, particularly when it must be automatic [7, 24, 25]. Users who are asked to rank their interest in a visualization quickly resort to a default answer (*e.g.*, "Not interesting"), or decline to provide any answer after a few initial responses. Moreover, interests are not static: they can change, often dramatically, as exploration unfolds.

We have explored the idea of *preference elicitation* to track a user's interests in a dataset as they form and change [9, 17]. As the user explores, an interest engine updates rules in real-time, deprecating those for past interests, and defining new rules or strengthening existing ones as the user's explorations change.

2.3 Visualization Representations

A visualization representation maps data to visual properties to present to a viewer. Many techniques draw on Bertin's *Semiology of Graphics* [2], a classic work that describes information, graphic systems, and relationships between the two. Examples include MacKinlay's work on automating visualizations of relational data [31], and Healey et al.'s integration of psychophysics and mixed-initiative planning to recommend perceptually optimal visualizations [19]. Both approaches use formal descriptions of a visualization to generate data-to-visual feature mappings.

More recently, effective visual encodings have been combined with data transformations and graphic algebra to produce more expressive visual representations. One example is Wilkinson's *The Grammar of Graphics* [47]. Commercial and open-source systems have taken inspiration from Wilkinson's work. Protovis and its successor, D3 [3], provide ways to create flexible and expressive visualizations using the notion of *representational transparency* of the visualization representation. Vega also uses a visualization grammar based on marks and data and view manipulation operations to define visualizations [37]. Vega-Lite, building on Vega, combines a grammar of graphics and a grammar of interaction to generate interactive visualizations [36].

Although these representations differ in specific details, they all provide formal specifications that separate a visualization from its construction. They also share several important features.

1. The formalizations are declarative, so implementations are independent in the choice of a visualization front-end.
2. The formalizations are concise yet expressive, supporting a wide range of visualization designs.
3. Visualizations are defined and built from the formalizations.

These features make the formalizations suitable candidates for our algorithms. The visualization representation we designed is based on this work, but extended into a set notation to support similarity-based hashing.

2.4 Indexing by Similarity

Hashing can approximate similarity search in large databases [43,44]. There are two main hashing approaches: (1) locality sensitive hashing, and (2) learning to hash. LSH uses a distance metric to assign similar items similar hash values with a certain probability. Alternatively, learning to hash optimizes an objective function based on the underlying data to produce its hash function. The advantage of learning to hash is that the hash function incorporates data information, so it has performance gains for targeted datasets. However, the data is not always known and there can be discrepancies between training data and new data. LSH imposes no assumptions on the data and has theoretical guarantees on performance. Based on this, and to allow our method to be widely applicable, we chose LSH. We use the min-hash family [4] to formulate a visualization as a set. Since our focus is to investigate the feasibility of min-hash to support scalable visualization recommendations, we chose its most basic form. Extended min-hash algorithms exist [4, 23], as well as ways to optimize hash table search efficiency [1, 30]. These improvements could be applied with minimum modification to our approach to improve its performance.

3 REPRESENTATION

We represent a visualization state as a set of three objects: data transformation T, geometry G, and data–feature mapping M. Given a dataset D with n attributes $A = (a_1, a_2, \ldots, a_n)$, T are operations on D such as grouping, filtering, or computing derived values. G is the type of visualization used (*e.g.*, boxplot, isarithmic map, and so on). M realizes the abstract T and G by assigning a set of visual properties $V = (v_1, v_2, \ldots, v_n)$ to each data attribute, $(a_i, v_i, f(a_i))$ using function $f(a_i)$. $f(\cdot)$ operate on a specific data attribute, while T are applied to all elements in D. A set-based representation has two important advantages: (1) it is compatible with indexing-by-similarity like LSH, and (2) it generalizes to other common representations in the visualization community. Formally, the visualization representation is defined as

$$\{\, T, G, (a_i, v_i, f(a_i)\ \forall i \in 1, \ldots, n)\,\} \tag{1}$$

In this representation, there are zero or more T, one G, and one or more $f(a_i) : a_i \mapsto v_i$. The representation is independent of implementation, allowing for user-chosen visualization front-ends. For example, in Fig. 1 $T = \{\,\text{translate}\,\}$, $G = \{\,\text{circle}\,\}$, and $M = \{\, mag, size, f(mag) : mag \mapsto radius = 0.1\, mag\,\}$.

3.1 Generalizability

Generalizability can be viewed from two perspectives: the ability to generalize to other visualization domains, or the ability to generalize to non-visualization problem environments. We demonstrate the first type of generalizability by applying our approach to different visualization formalizations. We convert a well-known representation into a set format suitable for LSH by demonstration how we can extend Vega-Lite [36] (Sec. 2.3).

Vega-Lite builds on Wilkinson [47] and visualization systems like Tableau [39]. A *unit* specifies data, transforms, geometric marks to display the data, and visual encodings for data attributes. This is similar to our $\{T, G, M\}$ representation (Eq. 1). Vega-Lite also supports analytic operations like sorting, aggregation, and binning. These can be included in T, assuming the visualization front-end understands how to preprocess the data.

Vega-Lite extends a unit visualization using *layer* (overplotting), *concatenation* (side-by-side views), *facet* (multiple visualizations of data subset), and *repeat* (multiple visualizations with different unit representations). Our representation can embed unit sets and an associated operation in a top-level set to support these hierarchies.

$$\{\text{op:}\mathit{layer}, \{u_{1,1}, \ldots, u_{1,n}\}, \ldots, \text{op:}\mathit{facet}, \{u_{m,1}, \ldots, u_{m,n}\}\} \tag{2}$$

Vega-Lite supports *nested views* to construct dashboards from visualization collections. Finally, Vega-Lite added formalisms to describe interaction, something our representation currently does not support.

4 LOCALITY SENSITIVE HASHING

Our goal is to identify collections of similar visualizations in a visualization sequence. We turn to LSH to address this need [22]. LSH attempts to place elements in buckets such that similar elements have a higher probability of being stored in a common bucket, versus dissimilar elements. Wang et al. provide an excellent overview of LSH and the different methods used to achieve it [43].

Our approach for sequence storage is: (1) decompose a sequence into its constituent nodes; (2) use LSH to store nodes, where each node references its parent sequence. For retrieval: (1) decompose an initial exploration path into its m most recent nodes n_i; (2) use LSH to retrieve parent sequences S_j of nodes similar to n_i; (3) construct $\bigcup S_j$ to search for recommendations. We implement a *c-approximate near neighbor* solution.

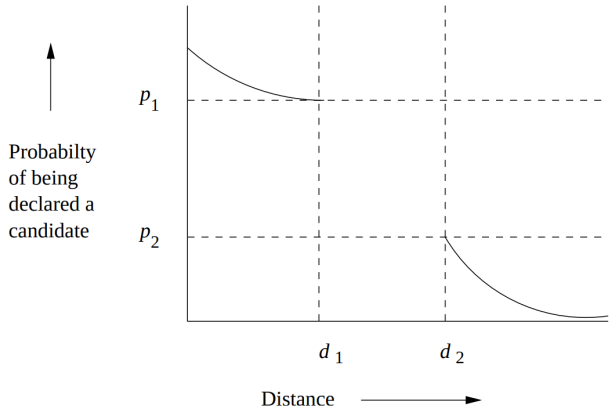


Figure 2: Illustration of locality sensitive hashing [29]. Items with distances below d_1 will be hashed to the same value with probability at least p_1, while items with distances above d_2 will be hashed to the same value with probability at most p_2.

Definition 4.1. c-Near Neighbors. Given a set of points P, a query point q, and a closest neighbor point p distance $R = \text{dist}(q, p)$ from q, find all points X that are within distance cR of q, $X = \{x \mid \text{dist}(q, x) \leq cR, x \in P\}$ for a user-chosen c.

Definition 4.2. c-Approximate Near Neighbors. Given P, q, and $0 < \delta \leq 1$, report each point x that is a c-near neighbor of q with probability $1 - \delta$.

Locality sensitive hashing builds on these foundations by defining a family of hash functions H such that:

Definition 4.3. H is (R, c, P_1, P_2)-sensitive $\iff$ for any q, p

- if $\text{sim}(q, p) \leq R$ then $\Pr[h(q) = h(p)] \geq P_1$
- if $\text{sim}(q, p) \geq cR$ then $\Pr[h(q) = h(p)] \leq P_2$

where $\text{sim}(q, p)$ is the similarity between q and p. The expectation is that $P_1 > P_2$. To increase the distance between P_1 and P_2, L separate hash tables are constructed. The hash value $g_j(p)$ for point p in each table is a concatenation of k hash functions h_i: $g_j(p) = (h_{1,j}(p), h_{2,j}(p), \ldots, h_{k,j}(p))$, $1 \leq j \leq L$ where each h_i is selected at random from H. This means a point p is stored L times, once in each of the L hash tables.

To query a point q, L buckets $g_1(q), g_2(q), \ldots, g_L(q)$ are identified and their points are stored in a similar node set. Recall that a bucket $g_\cdot(q)$ is expected to contain q and points similar to q. Two common approaches are used to terminate the search.

1. Stop after finding the first L' similar points, often $L' = 3L$.
2. Retrieve all similar points from all L non-empty buckets.

Since similarity plays a critical role in LSH, a similarity algorithm must be chosen. Numerous candidates exist (*e.g.*, l-norm, cosine similarity, or Hamming distance). We apply a specific algorithm, min-hash. Min-hash uses Jaccard set similarity, which integrates well with many visualization representations, since they can be converted to set-based formats. The Jaccard coefficient J between two sets A and B defines their similarity.

$$J(A, B) = \frac{|\, A \cap B \,|}{|\, A \cup B \,|} = \text{sim}(A, B) \tag{3}$$

In this implementation, H is a family of hash functions h_i that randomly assign an integer value to each point in P. The algorithm

was originally proposed by Broder to identify similar web pages a year prior to the use of the term LSH [4]. The goal is to estimate $J(A,B)$ without explicitly computing $A \cap B$ and $A \cup B$.

Definition 4.4. Estimating $J(A,B)$. Hash function $h_i \in H$ maps set members to distinct integers. Let $h_{\min}(A)$ be the member in A with the minimum hash value. If $h_{\min}(A) = h_{\min}(B)$, the minimum element in $A \cup B$ is also in $A \cap B$. By definition, this probability is the Jaccard coefficient $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \Pr[h_{\min}(A) = h_{\min}(B)]$.

To estimate $\Pr[m_{\min}(A) = m_{\min}(B)]$, multiple hash functions are employed, since a single hash function will evaluate to $\Pr = (0 \mid 1)$, a poor estimate. Given k hash functions $h_i \in H$ that represent a random permutation of the set member-to-integer mapping (*i.e., H* is the family of random permutations), $J(A,B)$ is defined as:

$$J(A,B) \approx \frac{1}{k} \sum_{i=1}^{k} (h_{i,\min}(A) = h_{i,\min}(B)) \tag{4}$$

Broder showed that the random permutation family H satisfies Defn. 4.3 with $\mathrm{sim}(A,B)$ replaced by the Jaccard coefficient similarity [5]. Thus, as long as a visualization representation can be converted to a set notation, it can use Jaccard-based LSH to store similar visualization nodes in common hash buckets.

5 Sequence Storage and Query

To perform sequence matching, past sequences are stored and queried using Jaccard-based LSH. Sequences are saved using a graph database optimized for rapid graph operations [20, 33, 45]. Nodes are assigned a unique sequence ID that is stored in the LSH tables together with a reference to the parent sequence.

Graph databases support rapid path traversal using index-free adjacency [16]. Given a target node, its data description includes the disk locations of its parent and child nodes. This means reading a sequence of length n requires one index lookup to retrieve the target node, followed by (up to) n disk reads to retrieve the node's sequence. Graph databases will also try to take advantage of any blocking the storage device or operating system provides. Together with LSH, this allows us to read nodes in $\mathrm{O}(1)$ and sequences of n nodes in $\mathrm{O}(n)$.

6 LSH Implementation

Although the theoretical description of LSH makes it seem straightforward for sequence matching, in practice the LSH parameters must be carefully selected to ensure optimal performance.

Based on the description in Section 2, a set representation S_i is hashed to a *signature* $\mathrm{sig}(S_i)$ with a specific length using the min-hash algorithm. Given $\mathrm{sig}(S_1)$ and $\mathrm{sig}(S_2)$ for two sets, min-hash guarantees the probability of a digit at a common position in $\mathrm{sig}(S_1)$ and $\mathrm{sig}(S_2)$ is equal to $J(S_1,S_2)$. By counting how many digits are identical, we estimate the Jaccard similarity and therefore the probability of the similarity between S_1 and S_2. The longer the signature, the better it will approximate the true probability.

Signatures are divided into b bands of r digits per band, so $\|\mathrm{sig}\| = br$. A high-quality hash function and number of buckets is chosen independently to hash the digits in each band, producing b separate hash tables. For a given threshold θ, this ensures that pairs of sets with estimated similarity below θ will have a very low probability of hashing to the same bucket in any of the b hash tables, while pairs of sets with similarity above θ will have a very high probability of hashing to similar buckets. By querying all b hash tables, we can find a set of similar candidates. Based on this, b, r, θ, and the length of a signature are related as follows [29].

$$\|\mathrm{sig}\| = br, \quad \theta = \left(\frac{1}{b}\right)^{\frac{1}{r}} \tag{5}$$

Theoretically, as long as these equations hold we should expect good performance from LSH. From Eq. 5, only two variables among b, r, and θ are independent: fixing any two determines the third. From Eq. 5, the signature length depends on b and r. Our LSH implementation uses b and θ as free parameters, selected to calculate the length of the signature and perform LSH. This guarantees that no matter what values we assign to b and θ, for the chosen threshold θ if $|\,\mathrm{sim}(S_1) - \mathrm{sim}(S_2)\,| \geq \theta$ then the probability of S_1 and S_2 hashing to common buckets is high, but if $|\,\mathrm{sim}(S_1) - \mathrm{sim}(S_2)\,| < \theta$ then the probability is low. There are several critical caveats to this theoretical description.

1. **θ influences the uniformity of hash value distribution.** If the similarity of most pairs of sets are above θ, most items will concentrate in a few buckets, increasing search from $\mathrm{O}(1)$ to $\mathrm{O}(n)$. If most similarities are below θ, sets are distributed randomly, which gives good lookup performance but poor search results. Therefore, we need to choose θ based on the underlying dataset. One approach is to calculate all pairs of similarities, but this requires $\mathrm{O}(n^2)$ time. Instead, we sample several θ_i and perform LSH on each. The uniformity of the resulting hash tables reflects the quality of θ_i. Since hash table construction is rapid, this approach works well.
2. **θ and b determine r, and r determines the total number of possible inputs for a hash function.** b and r are constrained by θ (Eq. 5): if b increases, r must increase, and vice-versa. Therefore, if b is not sufficiently large, r will be limited in the number of input values inp_r it can represent. If inp_r is smaller than the number of buckets in a hash table, some buckets are guaranteed to be empty. Increasing the size of the table will not help, since this does not change inp_r. To avoid this, we must choose a sufficiently large b.
3. **A large b is necessary to maintain scalable hash functions, but this increases the number of hash tables.** If on average each bucket holds k sequences, the total number of similar sequences returned is $\sim kb$. Since the time required to generate recommendations is proportional to kb, we want k as small as possible. A perfect hash function with $k = 1$ is, in practice, impossible to define. However, experimenting with numerous hash functions revealed that Java's Arrays.hashcode module produces small k. It also has a very low false negative rate, ensuring k does not increase due to dissimilar sequences hashing to a common bucket.
4. **min-hash is probabilistic.** Using probabilistic min-hash has two implications. First, the quality of the signature varies between applications of min-hash. More subtly, sometimes many pairs of signatures are similar in certain bands, meaning many sequences hash to a common bucket in the hash tables responsible for those bands. This is not due to the hash function we use, but instead because (within these bands) the inputs are seen as highly similar. We cannot guarantee this similarity actually exists in the dataset, since min-hash is probabilistic. To address this, we run min-hash and LSH several times and choose the result with the best distribution and performance.
5. **Total recommendation time depends on many factors other than the quality of the LSH implementation.** The time needed to return a set of recommendations depends on LSH, but also on θ, the query node, and the topology of the database. For example, retrieving recommendations from a large dataset may take longer simply because the dataset has many more nodes that are similar to the query node, based on the chosen threshold θ.

In summary, to produce comparable run times for different databases, we must choose θ and b for each database such that the hash tables for each band store sequences uniformly. We also try

to ensure that the number of similar nodes kb returned for any query are similar. If these numbers are comparable and overall similarity accuracy is high, the total time to return recommendations will be nearly equal across the different databases.

7 Recommendation Algorithm

The performance of a recommender includes both the time for sequence retrieval and the time to convert similar sequences into recommendations. Both operations need to be measured during simulation experiments. Given a rapid method to retrieve similar nodes, our current recommendation algorithm proceeds as follows.

1. For the subsequence from the current node p_{cur} backwards m nodes along the user's exploration path, retrieve sequence sets $S_i = \{s_{i,0}, \ldots, s_{i,n_i}\}$ as parent sequences of similar nodes in $p_{\text{cur}-i}$'s LSH buckets, $0 \leq i \leq m$.
2. Intersect the sequence sets to find sequences $S = S_1 \cap \cdots \cap S_m$ that contain all m nodes from the current subsequence. Optionally remove sequences where the position of $n_{\text{cur}-p}$ comes before $n_{\text{cur}-q}$, $p > q$, that is, where nodes do not occur in the same order as in the current sequence.
3. Assign an interest weight w to every node in the remaining sequences. Choose a threshold interest weight $w_{\min}$ to define a node as "interesting" when $w \geq w_{\min}$.
4. Identify the position p_i of p_{cur} in each sequence s_i. Walk forward from p_i in s_i until an interesting node q_i is found, or the last node q_i of s_i is reached.
5. Once all sequences are searched, return the n top q_i nodes sorted by their interest weights w_i as recommendations.

It would be possible to return the final node of each sequence $s_i \in S$ as a recommendation. We avoid this because: (1) this assumes that the user is completing the task supported by s_i, which may not be true, especially when visualization construction has just started; and (2) moving to intermediate nodes allows users to choose new, novel exploration paths, potentially enriching the sequence database with new ways to solve the given task.

7.1 Inferential Interest Weights

We chose to automatically calculate inferential interest weights, since this places no burden on the user. An inferential system can also leverage the crowdsourced nature of our sequence database.

A number of properties are available that leverage the graph the sequences form. Although experimental testing would be needed to confirm their utility, we believe these properties may offer important advantages over other measurements based on individual users in isolation. Current candidates include:

- **In-degree** $\deg_i$: **out-degree** $\deg_o$ **ratio**. An increase in $\deg_i / \deg_o$ represents an increase in *convergence:* numerous exploratory paths meet at a common node. This suggests the node represents an inflection point: a visualization that identifies the steps needed to reach a final task solution.
- **Average termination distance**. For sequences from a recommended node p_{rec}, the average remaining sequence length indicates how "close" a user is to a solution; recommendations closer to a solution could be favored.
- **Average termination similarity**. All reachable sequence termination nodes have pairwise similarity scores. High similarity between termination nodes means that similar solutions were constructed from p_{cur}, so the shortest available path will lead most quickly to task resolution.

We chose to combine the the last three interest candidates with equal weights of $w = \frac{1}{3}$, except when a user is following a new exploration path (*i.e.*, no termination nodes exist). When this happens, we use the first two candidates with $w = \frac{1}{2}$. More sophisticated w are being considered as an area of future work.

Operation	Allowable Values
T	translate; rotate; scale
G	bar; line; pie; scatterplot; choropleth map
a	year; temperature; pressure; rain; snow; radiation; cloud coverage; wind speed; direction; velocity
v	x; y; hue; saturation; size
$f(a)$	all pairs of $a \mapsto v$

Table 1: Transformations T, geometry G, data attributes a, visual features v, and data–feature mapping $f(a)$

8 Simulation Experiments

We constructed a collection of simulated sequence databases of varying sizes to test performance. Since our focus is on the technical side of rapid sequence matching, the emphasis is accuracy and speed. For this purpose, simulated databases provide a more controlled, experimental and demonstrative environment to evaluate our approach. We must evaluate the feasibility of our approach prior to instrumenting visualization tools to construct sequence databases.

To construct the simulated databases, nodes used our visualization representation (Section 2.3), and values for different variables are defined in Table 1. Since our recommendation prototype does not require real sequences to test its performance for lookup or recommendations, we built sequences with random collections of node states to stress the scalability of our system. The one concession we made was to define sixteen separate termination nodes from four clusters of "similar" node states, to allow the average termination similarity measure to return meaningful values. For all other nodes, we randomly choose a state from the possible states defined by Table 1.

Once a sequence construction approach was defined, we built six separate sequence databases of sizes 100, 250, 1000, 2500, 5000, and 10000 sequences each (producing between 1174 and 115005 unique nodes). Sequence databases of more than 10000 paths (or 100000 nodes) are rare, so this represents an appropriate range of sizes to investigate scalability. Sequence lengths were uniformly chosen over a range of ten to fifteen nodes, with the terminating node uniformly selected from the sixteen possible candidates. $b = 20$ hash tables with 1097 buckets were initially built for $n = 100$ sequences, then increased from this starting value based on appropriate choices for θ, b, and r (see Section 6). Table 2 shows the number of sequences and nodes in each database, θ, b, r, and the hash table size, the time to retrieve similar nodes, to choose recommendations from the resulting sequences, the total time to present recommendations, and the recall and precision of sequences found as a percentage of all similar sequences produced by exhaustive search. Recall represents the fraction of all possible similar sequences returned. Precision represents the fraction of sequences returned that are, in fact, similar [46].

To test retrieval and recommendation times, we randomly selected 2000 unique visualization states, then requested recommendations for $m = 6$ node subsequences. We recorded both the time to retrieve similar sequences, and the time to process the sequences into recommendations, averaging to obtain a final result. For recall and precision testing, we randomly selected 300 states, then compared the sequences returned to the known set of similar sequences.

Sequences	Nodes	θ	b	r	Table n	Retrieve	Recommend	Total	Recall	Precision
100	1174	0.5	20	6	1097	0.2 ms	2.7 ms	2.9 ms	98%	42%
250	2894	0.5	30	6	2861	0.2 ms	5.4 ms	5.6 ms	98%	36%
1000	11577	0.5	50	7	11491	0.6 ms	15.1 ms	15.7 ms	95%	30%
2500	28772	0.6	60	10	28771	0.7 ms	11.6 ms	12.3 ms	99%	43%
5000	57651	0.6	80	10	57649	0.7 ms	17.2 ms	17.9 ms	98%	38%
10000	115005	0.65	90	12	115001	1.0 ms	17.3 ms	18.3 ms	95%	46%

Table 2: Number of sequences and unique nodes in each test database, θ, b, r, and hash table size for the databases; time for node retrieval, building recommendations from nodes retrieved, and total recommendation time; percentage of all similar sequences returned (recall), and percentage of sequences returned that were similar (precision)

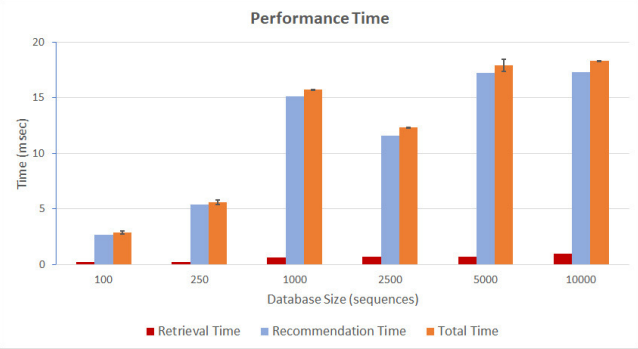


Figure 3: Time for retrieval, recommendation, and total time for databases of varying sequence sizes

8.1 Results

The time required to retrieve similar sequences through LSH was dominated by the times for other actions, averaging 0.57 ms (Fig. 3, red bar). Identifying a set of recommendations within the similar sequences was also rapid, averaging 11.6 ms (Fig. 3, blue bar), for a total average recommendation time of 12.2 ms, nearly an order of magnitude below our 100 ms threshold (Fig. 3, orange bar). We saw slight increases in retrieval and recommendation times as the database size grew from 100 to 1000, then near-constant performance past that point. We hypothesize that this is because smaller databases have fewer hash tables to query (*i.e.*, b is smaller), and the total number of similar nodes returned is less than for databases with 1000 or more sequences.

Recall (accuracy) ranged from 95% to 98% relative to an exhaustive search for similar sequences. LSH will include sequences which are not "similar" by our definition, and exclude sequences that are. However, we feel a lower bound on recall of 95% is sufficient to provide high quality lists of recommended visualizations to an analyst.

Precision percentages were lower, ranging from 30% to 46%. This means that approximately two-thirds of the sequences retrieved did not meet our query similarity threshold θ_q. Recall increases as θ_q increases, but if $\theta_q > \theta$ for the threshold θ used to build the hash tables, nodes with similarities $\theta \leq \text{sim} \leq \theta_q$ will have a high probability of hashing to a common bucket, even though they do not satisfy $\text{sim} \geq \theta_q$, producing false positives.

We accept this recall–precision tradeoff because: (1) we favor recall over precision, since this provides the majority of similar sequences and any non-similar sequences returned will be eliminated during recommendation selection, and (2) when a node with $\text{sim} < \theta_q$ is retrieved, we ignore it, so the only time spent is in retrieving the node and not its visualization sequence, producing a very small time penalty.

To determine statistical significance, we analyzed the null hypothesis H_0 of equality of mean recommendation times across database sizes. An important question is whether to group individual query times, and if so, how that should be done. Probability of equal means p is highly affected by sample size, since (all other things constant) a larger n produces a lower variance, and therefore a lower p value. Our size of $n = 2000$ is more than large enough to skew results based on a standard $\alpha = 0.05$ significance threshold.

One option is to use tables that adjust α based on n, but our n falls outside the range of most tables [35]. We chose to use power analysis to compute the minimum number of samples $n_{\min}$ needed to justify any significance found [8]. We collapsed individual times into groups of size $n_{\min}$, and used the group average in an analysis of variance (ANOVA) of group times across database size. Although many power analysis formulas are available, we chose a simple approach based on confidence interval (margin of error) Z, confidence level α, and standard deviation σ.

$$n_{\min} = \frac{Z^2 \sigma (1-\sigma)}{\alpha^2} \tag{6}$$

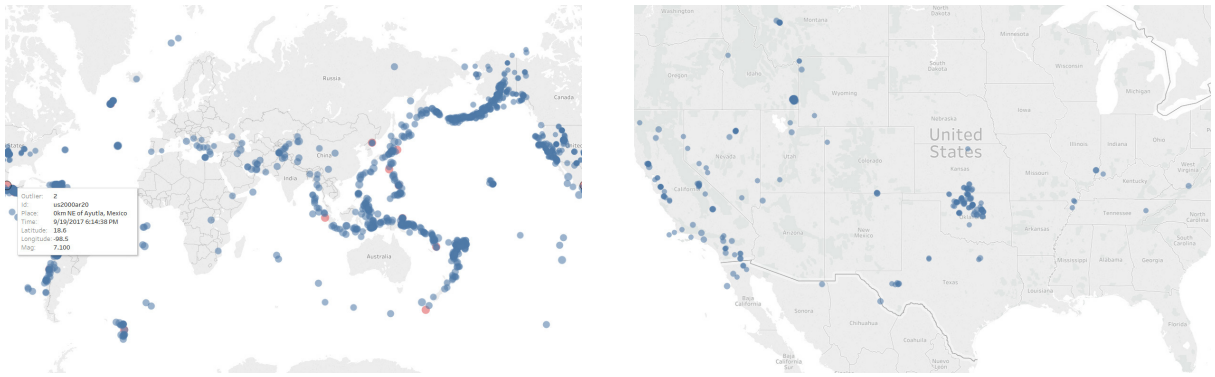
For a confidence interval of 5% ($Z = 1.96$), a confidence level $\alpha = 0.05$, and an estimated standard deviation $\sigma = 0.5$, the minimum number of samples for justifying significance is $n_{\min} = 385$.

Bartlett's test for equality of variance reports $p < 0.001$ (unequal variances), so we applied Welch's ANOVA to correct for this, producing $F(5, 10) = 5576$, $p < 0.001$. Not surprisingly, H_0 is rejected, implying mean times across database sizes are not equal. More interesting is a corrected post-hoc analysis of significant differences in mean times by pairs of database sizes. Here, the pairs 1000–5000, $p = 0.13$, 1000–10000, $p = 0.29$, and 5000–10000, $p = 0.99$ showed no significant difference in means. This mirrors Fig. 3, where the recommendation times and corresponding query times rise as database size increases to a ceiling of approximately 17–18 ms. Interestingly, database size 2500 showed significant differences in mean times versus the 1000, 5000, and 10000 sequence databases.

Finally, a potential issue related to the b hash tables is that numerous large hash tables may not fit in main memory. This can be addressed with a dynamic hash table method like extendible hashing [10], which guarantees: (1) the hash table buckets grow and shrink to an optimal size without the need to re-hash their entries; and (2) any bucket can be retrieved with at most two disk seeks.

8.2 Dynamic Sequence Databases

The values of θ, b, and r are tied to a static database. In a recommendation system, however, the database size grows as new approaches are explored to solve analytic tasks. To support this, we choose the initial parameters for a specific simulated database, then monitor



(a) **(b)**

Figure 4: Earthquake visualizations: (a) earthquake locations, *mag*→ size, *outlier* → color: red for $mag > Q_3 + 1.5\,\text{IQR}$, purple for $mag < Q_1 - 1.5\,\text{IQR}$, blue otherwise; (b) a cluster of earthquakes in Oklahoma and Kansas

performance to adjust the parameters as needed.

Initial θ, b, r. To choose the initial parameter values, we construct a simulated database of $n = 10000$ sequences of length 10, by default selected uniformly from the space of all possible visualizations. If the user has a specific set of tasks to complete, we can approximate a more specific distribution to sample when building our initial database. Although the θ, b, and r we select will not be optimal $n < 10000$, query and recommendation is rapid for small databases, so even a non-optimal distribution of nodes across the b hash tables should still produce acceptable performance.

Although unlikely, if n exceeds 10000, we can monitor recommendation time to maintain our 100 ms threshold. We expect this to persist for a reasonable period of time, since: (1) our experience suggests initial performance will be an order of magnitude below the 100 ms threshold; (2) many nodes in new sequences will already exist in the database; and (3) for a larger database, multiple new nodes will be needed before performance is pushed above the threshold. If performance degrades past 100 ms, we must reorganize the hash tables used for sequence queries. The key parameter is θ, since it is used to subdivide the database into similar and dissimilar sequence pairs. We sample new values θ' to identify an appropriate replacement. This will be faster than during initial database construction, since we know whether θ is too low or too high by examining the distribution of nodes in the current hash tables. Once a θ' is selected, corresponding b' and r' are chosen as discussed in Section 6.

9 Practical Example

As a practical example, consider a USGS dataset of 1,989 earthquakes (Fig. 1) [41]. We focus on latitude, longitude, and magnitude *mag*, with a simple analytic task of identifying outlier *mag* in both the strong and weak directions.

Analysis generally proceeds in one of two directions. More novice users attempt to use a map (Fig. 4a) to visually identify outliers. More expert users analyze *mag* directly, using calculations based on interquartile range (IQR, the distance between the first and third quartiles) to define upper and lower thresholds for outliers (Fig. 4b). For an outlier threshold of 1.5 IQR, any $mag < 1.95$ or > 5.68 is an outlier. Figure 1a visualizes earthquakes as circles with $mag \mapsto$ size and $outlier \mapsto$ color: purple for weak outliers, red for strong outliers, and blue for non-outliers. We see a number of strong outliers (*e.g.*, the recent magnitude 7.1 earthquake in Mexico), but no weak outliers.

When novices request recommendations, the system uses their last six visualizations to locate sequences with nodes similar to the user's previous six nodes. We do not enforce ordering of nodes in similar sequences, so the system can backtrack to the initial map visualizations of *mag* (used by both novices and experts), then follow alternative paths to identify visualizations used to define outlier thresholds (*e.g.*, Fig. 4b). Novices use the recommendations to: (1) recognize the strategy of threshold calculation; and/or (2) determine what boxplot visualizations represent, then use this understanding to move to a threshold+highlighting approach.

Other interesting phenomena can be identified, for example, a large cluster of small magnitude earthquakes occurring in Oklahoma and Kansas. Oklahoma has seen a significant increase in earthquakes of $mag \geq 3$ over the last seven years, possibly due to wastewater wells produced by the oil and gas industry [26].

With only a few thousand nodes stored in a few tens of sequences, our example does not require pattern matching scalability. However, in a real-world domain with more complex tasks and a much larger sequence database, our experimental results show we can continue to return recommendations in only a few tens of milliseconds.

10 Conclusions

This paper describes supporting sequence-based visualization recommenders with LSH. Jaccard-based similarity can be used to store similar nodes in common hash buckets. This allows us to retrieve common sequences very rapidly, then analyze them for potentially interesting nodes (visualizations) to recommend to a user. Experimental results showed that by carefully choosing the parameters used in LSH, we can retrieve nodes in 1 ms or less, and complete the entire recommendation operation in 20 ms or less.

A number of important practical findings were discovered during our experiments, in terms of choosing effective LSH parameters θ, b, and r. One area of future work is how to update these parameters dynamically. Another area of interest is how interaction can be added to our visualization grammar, similar to Vega-Lite.

A further interesting aspect to look into is effectiveness. As a crowdsourcing system, the quality of retrieved visualizations essentially depends on the quality of the database. There is no guarantee of the effectiveness of the recommendations. To mitigate this issue, an effectiveness score can be considered to be assigned to the retrieved visualizations based on perceptual and design principles, and let the system only recommend those with higher scores. In this way, instead of relying on the quality of the crowdsourced database, we have a built-in mechanism to retrieve and identify visualizations both similar and effective.

REFERENCES

[1] M. Bawa, T. Condie, and P. Ganesan. LSH forest: Self-tuning indexes for similarity search. In *Proceedings of the 14th International Conference on World Wide Web*, pp. 651–660. Chiba, Japan, 2005.

[2] J. Bertin. *Sémiologie Graphiques: Les Diagrammes, les Réseaux, les Cartes*. Gauthier-Villars, Paris, France, 1967.

[3] M. Bostock, O. Ogievetsky, and J. Heer. D^3 data driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[4] A. Z. Broder. On the resemblance and containment of documents. In *Proceeings of the Compression and Complexity of Sequences 1997 (SEQUENCES '97)*, pp. 21–29. Washington, D.C., 1997.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8–13):1157–1166, 1979.

[6] J. Brutlag. Speed matters for google web search, 2009.

[7] M. Claypool, P. Le, M. Waseda, and D. Brown. Implicit interest indicators. In *Proceedings of the 6th International Conference on Intelligent User Interfaces (IUI 2001)*, pp. 14–17. Santa Fe, New Mexico, 2001.

[8] J. Cohen. A power primer. *Psychological Bulletin*, 112(1):155–159, 1992.

[9] B. M. Dennis and C. G. Healey. Assisted navigation for large information spaces. In *Proceedings of the 13th IEEE Visualization Conference (Vis 2002)*, pp. 419–426. Boston, Massachusetts, 2002.

[10] R. Fagin, J. Nievergelt, and N. Pippenger. Extendible hashing—a fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.

[11] N. Ferreira, L. Lins, D. Fink, S. Kelling, C. Wood, J. Freire, and C. Silva. BirdVis: Visualizing and understanding bird populations. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2374–2383, 2011.

[12] J. D. Foley, F. D. Van, A. Van Dam, S. K. Feiner, J. F. Hughes, J. HUGHES, and E. ANGEL. *Computer graphics: principles and practice*, vol. 12110. Addison-Wesley Professional, 1996.

[13] D. Gotz and Z. Wen. Behavior-driven visualization recommendation. In *Proceedings of the 14th international conference on Intelligent user interfaces*, pp. 315–324. ACM, 2009.

[14] L. Hao, C. G. Healey, and S. A. Bass. Effective visualization of temporal ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):787–796, 2015.

[15] L. Hao, C. G. Healey, and S. E. Hutchinson. Flexibile web visualization for alert-based security analysis. In *Proceedings Visualization for Cyber Security (VizSec 2013)*, pp. 33–40. Atlanta, Georgia, 2013.

[16] C. G. Healey. *Disk-Based Algorithms for Big Data*. CRC Press, Boca Raton, Florida, 2016.

[17] C. G. Healey and B. M. Dennis. Interest driven navigation in visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1744–1756, 2012.

[18] C. G. Healey and J. T. Enns. Attention and visual memory in visualization and computer graphics. *IEEE Transactions on Visualization and Computer Graphics*, 18(7):1170–1188, 2012.

[19] C. G. Healey, S. Kocherlakota, V. Rao, R. Mehta, and R. St. Amant. Visual perception and mixed-initiative interaction for assisted visualization design. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):396–411, 2007.

[20] F. Holzschuher and R. Peini. Performance of graphic query languages: Comparison of cypher, gremlin, and native access in Neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pp. 195–204. Genoa, Italy, 2014.

[21] J. P.-L. Hsiao and C. G. Healey. Visualizing combintorial auctions. *The Visual Computer*, 27(6–8):633–643, 2011.

[22] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC '98)*, pp. 604–613. Dallas, Texas, 1998.

[23] J. Ji, J. Li, S. Yan, Q. Tian, and B. Zhang. Min-max hash for Jaccard similarity. In *IEEE International Conference on Data Mining (ICDM 2013)*, pp. 301–309. Dallas, Texas, 2013.

[24] D. Kelly and J. Teevan. Implicit feedback for inferring user preference: A bibliography. *ACM SIGIR Forum*, 37(2):18–28, 2003.

[25] H.-R. Kim and P. K. Chan. Learning implicit user interest hierarchy for context in personalization. *Applied Intelligence*, 28(2):153–166, 2008.

[26] M. Koerth-Baker. How the oil and gas industry awakened oklahoma's sleeping fault lines. `https://fivethirtyeight.com/features/how-the-oil-and-gas-industry-awakened-oklahomas-sleeping-fault-lines/`, 2016.

[27] D. Koop. VisComplete: Automating suggestions for visualization pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1691–1698, 2008.

[28] D. Koop, C. E. Scheidegger, S. P. Callahan, J. Freire, and C. T. Silva. Viscomplete: Automating suggestions for visualization pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1691–1698, 2008.

[29] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Data*. Cambridge University Press, New York, New York, 2014.

[30] Q. Lv, W. Josephson, A. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007)*, pp. 950–961. Vienna, Austria, 2007.

[31] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, 1986.

[32] B. H. McCormick, T. A. DeFanti, and M. D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6):1–14, 1987.

[33] Neo4j Technology, Inc. Neo4j 3.2. `https://www.neo4j.com`, 2017.

[34] W. Playfair. *The Increase of Manufacturers, Commerce, and Finance, with the Extension of Civil Liberty, Proposed in Regulations for the Interest of Money*. G. J. & J. Robinson, London, United Kingdom, 1785.

[35] A. E. Raffery. Bayesian model selection in social research. *Sociological Methodology*, 25:111–163, 1995.

[36] A. Satyanaraya, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A grammer of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017.

[37] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technologhy (UIST '14)*, pp. 669–678. Honolulu, Hawaii, 2014.

[38] D. P. Singh, L. Lisle, T. Murali, and K. Luther. Crowdlayout: Crowdsourced design and evaluation of biological network visualizations. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, p. 232. ACM, 2018.

[39] Tableau. Tableau 10.3. `https://www.tableau.com`, 2017.

[40] K. A. Thomson, W. J. Ingraham, M. C. Healey, P. H. LeBlond, C. Groot, and C. G. Healey. Computer simulations of the influence of ocean currents on Fraser River sockeye salmon (*oncorhynchus nerka*) return times. *Canadian Journal of Fisheries and Aquatic Sciences*, 51(2):441–449, 1994.

[41] United States Geological Survey. Earthquake catalog. `https://earthquake.usgs.gov/earthquakes/search`, 2017.

[42] M. Vartak, S. Huang, T. Siddiqui, S. Madden, and A. Parameswaran. Towards visualization recommendation systems. *ACM SIGMOD*, 45(4):34–39, 2016.

[43] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey, 2014.

[44] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(99):1–1, 2017.

[45] J. Webber. A programmatic introduction to Neo4j. In *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity (SPASH '12)*, pp. 217–218. Tucson, Arizona, 2012.

[46] Wikipedia. Precision and recall. `https://en.wikipedia.org/wiki/Precision_and_recall`, Last accessed,r2017.

[47] L. Wilkinson. *The Grammar of Graphics*. Springer, New York, New York, 2005.

[48] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. MacKinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specificiation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*, pp. 2648–2659. Denver, Colorado, 2017.