DeepBrowse: Similarity-based Browsing through Large Lists (Extended Abstract)

Haochen Chen, Arvind Ram Anantharam, and Steven Skiena

Stony Brook University
Stony Brook, NY 11790, USA.
{haocchen, aranantharam, skiena}@cs.stonybrook.edu

Abstract. We propose a new approach for browsing through large lists in the absence of a predefined hierarchy. DeepBrowse is defined by the interaction of two fixed, globally-defined permutations on the space of objects: one ordering the items by similarity, the second based on magnitude or importance. We demonstrate this paradigm through our *WikiBrowse* app for discovering interesting Wikipedia pages, which enables the user to scan similar related entities and then increase depth once a region of interest has been found.

Constructing good similarity orders of large collections of complex objects is a challenging task. Graph embeddings are assignments of vertices to points in space that reflect the structure of any underlying similarity or relatedness network. We propose the use of graph embeddings (DeepWalk) to provide the features to order items by similarity.

The problem of ordering items in a list by similarity is naturally modeled by the Traveling Salesman Problem (TSP), which seeks the minimum-cost tour visiting the complete set of items. We introduce a new variant of TSP designed to more effectively order vertices so as to reflect longer-range similarity. We present interesting combinatorial and algorithmic properties of this formulation, and demonstrate that it works effectively to organize large product universes.

1 Introduction

Browsing is a form of information retrieval, where one does not know exactly what they want but hope to recognize it when they see it. Browsing through menus or lists of items is a very common component of user interface design for web and mobile applications. Menus are effective for presenting small sets of possible selections to the user, but rapidly become unwieldy and tedious to use beyond a dozen or so possibilities. Hierarchical systems, like faceted search or DAG-like structures help to efficiently navigate through large sets of possibilities, but constructing such taxonomies generally requires considerable effort and domain expertise.

We propose a new approach to list navigation, permitting serendipitous discovery over lists of hundreds of thousands of items without the need for a predefined hierarchy. Our approach *DeepBrowse* is based on two basic concepts:

1. We organize the universe of items in a fixed order, according to *similarity*. Thus local regions in the full list will be *coherent*, because each item should be similar to its neighbors.
2. To provide the diversity necessary for serendipitous discovery, we modulate the set of items visible from our current position by *significance*. At the highest level of navigation, we select from a broad array of the most popular or important items, but once we identify an item of interest we want to see more choices like that. Our interface enables the user to move between smaller and larger item universes at will.

These concepts and their presentation are extremely simple, but realizing them with minimum domain knowledge requires considerable technology under the hood. Consider the problem of constructing effective similarity orderings. How can we computationally measure the pairwise similarity between members of item-universes like books, movies, and music? We propose a general approach based on deep learning, namely the use of graph embeddings to construct similarity orderings.

How can we construct the most effective similarity order for browsing? We define an appropriate and novel optimization criteria for this task. Although it is NP-complete to construct the optimal order, we provide an approximation algorithm and heuristics which construct excellent orderings in practice. And finally, how can we order arbitrary items (e.g. books, movies, and music) by relative significance? We propose series of generally-available proxies to capture this notion.

This paper is organized as follows. We first introduce the *DeepBrowse* interface paradigm, and discuss its implementation through Android apps over three distinct item universes: Wikipedia pages, movies, and dictionary words. We then delve into the technical details of constructing similarity orderings through graph embeddings and combinatorial optimization. Finally, we report the results of a user study gauging the effectiveness of our interface, and review the research literature in several topics relevant to our work.

Specifically, our work makes the following contributions:

- **A Paradigm for List-Oriented Browsing.** We abstract the notion of browsing to the basic operations of *scanning* and *deepening*: scanning along a fixed similarity order, and deepening to expose more specialized items once a region of interest has been identified. We show that these operations permit us to access any list item of known position in $O(\log n)$ operations, assuming the two orderings are independent.
  We note that our approach can naturally be integrated with faceted search interfaces, by using the facet selection values as conditionals to block undesired items from appearing in the display window.
- **Implementation in Three Domains.** We have created three Android apps implementing the *DeepBrowse* paradigm in three distinct domains: Wikipedia pages (*WikiBrowse*), movies (*MovieBrowse*), and vocabulary words (*WordBrowse*). We have released these apps in the Android app store, and encourage the reader to play with it to get a feel for the interface in action.

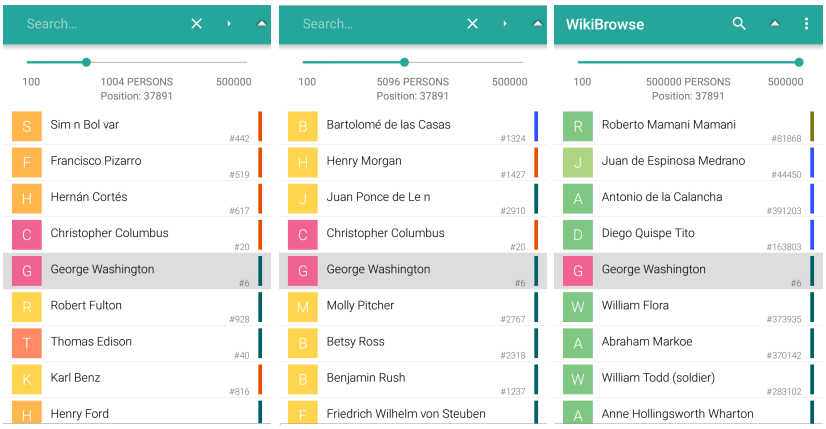

Fig. 1: Screenshots of the *WikiBrowse* interface, supporting searching through all historical figures appearing in Wikipedia. In this series, with *George Washington* selected as the central item, the universe size is expanded from the top 1000 historical figures to 5000, and finally 500,000 items, as we move from left to right. Each progressive shift successively exposes items of more specialized interest in the similarity order, those with closer connections to the central entity.

– **Deep Learning/Embedding Approach for Measuring Item Similarity.** The notion of graph embeddings (*DeepWalk*) serves as a unifying approach to measure pairwise similarity between classes of items as diverse as people, movies, and vocabulary words. Starting from a partial similarity network, *DeepWalk* learns a high-dimensional embedding for each vertex in the graph, generalizing its structure and reducing the problem of computing pairwise distance to a vector difference and dot product.

  We demonstrate that this approach scales to entity universes with hundreds of thousands of items, and provides a general and convenient abstraction for quantifying item similarity.

– **Similarity Orders for Effective Browsing.** The famous traveling salesman problem (TSP) seeks the minimum cost (maximum similarity) tour over a set of n items. However, the objective function in browsing is different than in transportation problems: we seek to maximize the similarity among items in each visible window, not just adjacent pairs.

  To improve browsing orders, we develop the notion of *k-robust* TSP tours, generalizing the traditional (or 1-robust) TSP problem. To the best of our knowledge, this variant of TSP has never been previously studied in the literature.

  We give efficient and effective heuristics to construct k-robust tours, and present experimental results that they achieve their objective of increasing categorical coherence with the parameter k.

– **User Study.** Browsing implies that the user does not have a well-defined task in mind, complicating the question of how to evaluate the success of their venture. Still, we perform a modest user study, demonstrating that our similarity order improved both performance and user experience on serendipitous discovery tasks over alphabetical order. We also demonstrate that user performance increases rapidly with exposure to the interface.

2 Related Work

Serendipitous Browsing. Serendipity is defined as the occurrence of something unexpected in a happy or beneficial way. André et al. [2,3] summarizes serendipity related research along two dimensions: the activity engaged in when encountering serendipitous information (directed browsing / non-directed browsing / none), and what type of information was found (relevant / not relevant to the goal). *DeepBrowse* falls into the category of non-directed browsing [13, 25] where users do not have a pre-defined goal while using it. [17, 21] show that organizing images according to similarity is useful for serendipitous browsing on images. StumbleUpon [14] allows users to stumble through the Web one (semi-random) page at a time. Bordino et al. [8] investigates the potential of entities in promoting serendipitous search from user-generated content (UGC). Clarke et al. [10] proposes a framework that systematically rewards novelty and diversity in information retrieval evaluation.

Graph Embeddings. Extensive literature has discussed different methods for graph embeddings. Multidimensional scaling [11], Laplacian eigenmaps [6] and IsoMap [24] all have good performance on small graphs, but the time complexity of these algorithms is too high to fit for a large-scale graph. Thus, these methods are not applicable to the network of Wikipedia people, which consists of about 500,000 vertices. Recently, methods are developed for building graph embeddings for large-scale graph. Deepwalk [20] presents an efficient online algorithm for learning representation of vertices in a network. By performing truncated random walk in the graph, Deepwalk treats the walks as sentences in a language model, and utilizes the Skip-gram model [18] on the random walks to train the vertex embeddings.

Traveling Salesman Problem. Traveling salesman problem (TSP) is a widely studied algorithmic problem [15]. Given a weighted graph, the TSP problem seeks a minimum weighted Hamiltonian cycle. The Euclidean TSP is proved to be NP-complete [19], so the main interest is in developing approximation algorithms [22]. Heuristics like 2-opt [12], Lin-Kerninghan [16], all considerably improve the solution quality.

Variants of the standard TSP problem have also drawn researchers attention. The maximum-scatter TSP [4] is perhaps the most relevant work to our definition of k-robust TSP; it maximizes the minimum distance between each vertex and all of its neighbors which are at most m points away in the tour. Another related TSP variant is discussed by [7], which requires constructing a tour that minimizes $\sum_{i=1}^{n} l(i)$. Here, $l(i)$ is the distance traveled before the i-th vertex in the TSP

tour. These two problems both take the distance between each vertex and its close neighbors into consideration, which is similar to our notion of k-robust TSP. Although approximation methods provide strict theoretical bound for these problems, their time complexities are at least quadratic in n, which is not feasible for the large-scale graphs we consider here.

3 The DeepBrowse Paradigm

3.1 Formulation

The *DeepBrowse* search paradigm is defined over any universe U of n items by the interaction among two permutations (similarity and significance) by two operations on these permutations (scanning and deepening). *DeepBrowse* is designed to facilitate efficient browsing through very large item universes on small displays capable of representing only a few items simultaneously.

The similarity and significance permutations are defined as follows:

- *Similarity* – This permutation P_1 over U orders items by semantic similarity, so items $x = P_1(i)$ and $y = P_1(i + j)$ should be similar or related, for $1 \leq i \leq n - 1$ and $1 \leq j \leq w$, where w reflects the size of the display window. Similarity permutations can naturally be constructed given a pairwise-distance function $d(x, y)$, although other approaches can be built on clustering (pairs in the same cluster have smaller distance than intra-cluster pairs), text description similarity, customer co-purchase data, etc. In this paper, we will generally employ a method using L_2 distance on graph embeddings of the underlying universe. We preserve a single precomputed similarity order of items for all users.
- *Significance* – The permutation P_2 over U orders items by popularity, importance, or merit, so for items $x = P_2(i)$ and $y = P_2(i + 1)$ then x is deemed more significant than y, for $1 \leq i \leq n - 1$. Thus the most significance item is $P_2(1)$ and the least significance $P_2(n)$.
  The significance order of items is of great importance for efficient browsing. Item universes often exhibit power-law behavior, where a small fraction of the items command a large fraction of user interest and attention. The significance permutation explicitly encodes this into the search process. Natural measures of significance include sales, views, downloads, likes, frequency of use, critical reviews, and ranking functions built by a combination of such variables. Network centrality algorithms like PageRank provide potential ranking functions on similarity networks even in the absence of such metadata.

We note that these permutations are very small indexing structures, each requiring $n \lg n$ bits for an n-item universe. Each such permutation takes only 200KB for $n = 100,000$, and 2.5MB for $n = 1,000,000$, although more space may be necessary to turn this into an efficient index. These permutations are all precomputed for the given universe U.

The *scanning* and *deepening* operations rely on state variables m and p, both of which are bounded between 1 and n:

- The *significance horizon* m defines the size of the currently active universe, namely the set of the m items highest ranked by significance. The deepening operation increases or decreases this value m, further enlarging or restricting the size of this active universe.
- The *position* p defines the point currently of central interest in similarity permutation P_1. The scanning operation increases or decreases p, moving us forward or backward in this similarity permutation.

We presume that the display is capable of displaying a sequence of w elements at any given time. The central item displayed items is $P_1(p)$. The $w/2$ items above (below) it are those items from $P_1(p)$ to $P_1(p+x)$ such that $P_1(p+i)$ is displayed iff the rank of $P_1(p+i)$ in $P_2 \leq m$, and x is as small as possible. The $w/2$ items below $P_1(p)$ are defined analogously.

When m is large relative to n, a large fraction of the items are suitable for display, and it is efficient to simply walk past the items in P_1 of insufficient magnitude. However, for $m << n$, it may require a prohibitive amount of skipping to identify the nearest items for display. We recommend keeping two separate data structures for P_1: the first over the full universe and the second over only the top $\sqrt{n}$ items from P_2, and toggle between them for different values for m.

3.2 Implementation

We have implemented *DeepBrowse* as an Android application, so far instantiated over three separate search domains:

- *WikiBrowse* – Here we seek to identify interesting people to read about in Wikipedia. Our dataset here consists of the pages of $n = 496,614$ historical figures in Wikipedia, with a natural network defined by the links between their Wikipedia pages.
- *MovieBrowse* – Here we seek to identify interesting movies to watch from the Internet Movie Database (IMDB). Our dataset here consists of $n = 73,232$ films defining the vertex of a network, with an undirected edge (v_i, v_j) when IMDB recommends movie i (j) to viewers of movie j (i).
- *WordBrowse* – Here we seek to identify interesting words, worth checking the definition of in an on-line dictionary. Our dataset here consists of $n = 100,232$ English words, each associated with its vector representation from the Polyglot multilingual word embeddings [1].

In this section, we will introduce our basic user interface design, instantiated as *WikiBrowse* for a motivating example. Figure 1 presents several screenshots, which we use to illustrate the major components of our design: the magnitude slider and the content scroll.

The *magnitude slider*, on top of our user interface, modulates the effective size of the entity universe, here restricting focus to the top 1,004 people in significance order. The current selection, *George Washington* appears in 37,891th position in the full TSP tour over almost 500,000 historical figures, although all but the

top 1,004 are currently hidden from view. The size of the displayed universe can be modulated by sliding left or right.

These figures are accessible by scrolling up and down through the names, ordered by similarity. Thus, Spanish explorers Hernando Cortez and Francisco Pizarro appear as neighbors, as do inventors Robert Fulton and Thomas Edison, and automotive pioneers Karl Benz and Henry Ford. The names, here centered around George Washington, all have significance rank better than 1,004, as reflected by the number to the right of their names and color encoded in the box to the left of their name. Clicking on this box brings up the relevant Wikipedia page. The color strip on right reflects the category/cluster associated with each entity: for George Washington this is *Politicians – U.S.*

These screenshots also illustrate how the neighbors of a given item change as we slide right to increase the allowable significance rank. The neighborhood around George Washington gets progressively filled with lower-wattage figures of closer association, such fellow patriots of the American revolution Molly Pitcher and Betsy Ross. At its most expansive setting (right) Washington's neighbors are very obscure but relevant figures, e.g. William Flora, Abraham Markoe, and William Todd all served as soldiers under Washington during the revolution.

3.3 Search Complexity Analysis

Here we analyze the complexity of accessing a specific item $x = P_1(i)$ of known position i using the *scanning* and *deepening* operations, provided that the similarity permutation P_1 and the significance permutation P_2 are independent. If P_1 and P_2 were identical, searching for x would require $O(n)$ time. However, the correlation between P_1 and P_2 is usually very weak in practice. For *WikiBrowse*, *MovieBrowse* and *WordBrowse*, the Spearman correlation coefficient are 0.04, -0.18 and 0.19 respectively.

Under this independent assumption, it is easy to show that accessing x only takes $O(\log n)$ operations with high probability. Let $f(n)$ denotes the time complexity of accessing x in an n-item universe U, and $Y = \{P_1(j_1), P_1(j_2), \cdots, P_1(j_w)\}$ denotes the initial displayed items. By scanning through Y, we can locate $P_1(j_k)$ and $P_1(j_{k+1})$ such that $j_k \leq i \leq j_{k+1}$. Then, we perform a deepening operation to seek for x between $P_1(j_k)$ and $P_1(j_{k+1})$. Since P_1 and P_2 are independent, there are approximately $\frac{n}{w}$ items between $P_1(j_k)$ and $P_1(j_{k+1})$. This gives the following recursion:

$$f(n) = f(\frac{n}{w}) + O(w) \tag{1}$$

By applying the master theorem we have $f(n) = O(\log n)$ since w is a constant.

4 Constructing the Permutations

Here we describe a general pipeline for generating the significance and similarity permutations for n items, which works well for the three domains we describe: Wikipedia pages, vocabulary words, and movies.

4.1 Ranking Construction

The ranking permutation P_2 is best constructed from domain-specific metadata for the given entity universe. For our three initial search domains:

- *WikiBrowse* – To measure the significance of people in Wikipedia, we use the historical rankings published in [23]. It uses Wikipedia as its main data source, performing a statistical factor analysis of criteria such as PageRank, article readership (hits), length, and editing history – although each of these component variables defines an independent ranking. Details of this analysis, including corrections to reflect historical aging, appear in [23].
- *MovieBrowse* – To measure the significance of movies, we use the number of votes the film received in IMDB to indicate its importance. Alternate permutations might be defined by box-office gross, critical scores, quality ratings, or some combination of these variables.
- *WordBrowse* – To measure the significance of words, we sorted them according to their frequency of appearance in the English edition of Wikipedia. Weighting these by TF-IDF score or Google search frequency would give other reasonable criteria.

4.2 Similarity Permutation Construction

As a general approach to measuring pairwise similarity between arbitrary pairs of items, we start with a partial domain-similarity network of items:

- *WikiBrowse* – Here we start with the network where the vertices are Wikipedia pages, and the edges are links between pages: (a, b) implies that page a refers to page b.
- *MovieBrowse* – Here we start with the network where the vertices are movies from IMDB, and the edges are recommendation links: (a, b) implies that movie b is recommended to people who liked movie a.
- *WordBrowse* – Here we use the Polyglot word embeddings space directly: (a, b) implies that word a has word b as one of its k-nearest neighbors.

To generalize from these partial graphs, we employ the *DeepWalk* [20] technique to construct a high-dimensional vector representation for each item. *DeepWalk* performs random walks over this graph to generate sequences of vertices, which can be interpreted "sentences" over the "vocabulary" of vertices. Using this formalism, Skip-gram embeddings can be constructed to build a vector representation for each vertex.

This high-dimensional representation allows for the fast computing of item similarity. We used the Euclidean distance between the high-dimensional representations of the two vertices, but the cosine distance / vector dot product could alternately be used. Specifically, more details about the similarity measurement between Wikipedia figures can be found in [9].

4.3 *k*-robust TSPs

The problem of optimizing similarity order has a natural connection to the famous traveling salesman problem (TSP). In particular, if we define the similar order over the vertices of a graph/network, the natural optimization goal is to minimize the total distance between adjacent vertices in the tour, i.e. find the TSP.

However, since phone screen is capable of displaying multiple items at the same time, it is desirable that in our interface mutually similar items appear throughout the same screen, not merely as neighboring elements. Thus the TSP objective function does not result in a browsing order which is optimally visually appealing.

To address this issue, we propose (to our knowledge) the novel combinatorial notion of a k-robust TSP tour. Given a graph $G = (V, E)$, the k-robust cost $C(k, T)$ of tour $T = \{t_1, t_2, \ldots t_{|V|}, t_1\}$ is defined as:

$$C(k, t) = \sum_{i=1}^{|V|} \sum_{j=1}^{k} d(t_i, t_{i+j}) \tag{2}$$

This cost is the weight of the kth power of the tour in a graph theoretic sense. Thus we take into account the cost between a vertex and all the vertices within a window size of k.

Fig. 2: The optimal k-robust TSP tour over all the U.S. state capital cities, for $k = 1$ (left), $k = 2$ (center), and $k = 3$ (right). The tours show greater regional coherence and more zigzags with increasing k.

Fig. 3: All Hamiltonian cycles of the unit grid graph are minimum cost TSP tours for the grid. However, the space-filling Hilbert curve defines a better 2-robust tour than the conventional snake order, by a factor of $\sim \sqrt{2}$.

For $k = 1$, this objective function is the same as a standard TSP tour, but this is not the case for larger values of k. Figure 2 illustrates this by showing the optimal k-robust tours of U.S. state capital cities for $1 \leq k \leq 3$. These tours clearly differ. Further they show greater geographic coherence as k increases.

This is more easily appreciated by considering the points on a $\sqrt{n} \times \sqrt{n}$ unit grid graph, as shown in Figure 3. The crinkly space-filling Hilbert curve has a bend at every possible location, as opposed the straight-edged snake tour, which minimizes bends. The cost $d(t_i, t_{i+2})$ is thus $\sqrt{2} = 1.414$ for the Hilbert curve vs. $d(t_i, t_{i+2}) = 2$ for the snake tour.

Approximating Optimal *k*-Robust Tours That the complexity of finding the optimal k-robust tour is NP-complete follows directly from the hardness of TSP for $k = 1$. This motivates the question of finding provable approximations to the optimal k-robust tour.

In fact, the optimal TSP tour approximates the optimal k-robust tour to within a factor of k:

Theorem 1. *The optimal TSP tour $T = (t_1, \ldots, t_n)$ serves as a $\Theta(k)$ approximation to the optimal k-robust tour, on metric graphs where n is relatively prime to $k!$.*

Proof. First, observe that the k-robust distance function on a metric graph satisfies the triangle inequality. This gives a trivial bound on $d_{i,i+k}$, namely

$$d_{i,i+k} \leq \sum_{j=1}^{k} d(t_i, t_{i+j}). \tag{3}$$

Let OPT be the TSP cost of optimal tour T. Thus $OPT = \sum_{i=1}^{n} d(i, i+1)$, giving a simple upper bound on the k-robust cost of T is $C(k, 2) \times OPT$. This suggests an $O(k^2)$ approximation ratio.

But we can tighten this bound by observing that the edges of the form $(i, i+j)$ for all $1 \leq i \leq n$ form a closed tour visiting all n points, if j is relatively prime to n. The cost of this tour must be $\geq OPT$, or else it would have defined the optimal tour. Thus the sum of the edges in the kth power of T must be at most $k \times OPT$, yielding the result.

4.4 Heuristic Optimization for *k*-robust TSP

We use a nearest neighbor based heuristic for building the initial TSP tour. We start from a random vertex in the graph, and repeatedly prepend the nearest neighbor of the current TSP tour to it. Formally, if the current partial TSP tour is $T = (v_1, v_2, \cdots, v_n)$, the nearest neighbor u to this TSP tour is defined as:

$$arg\min_{u} \sum_{i=1}^{k} d(u, v_i), u \in V - T \tag{4}$$

The algorithm runs in $O(D|V|^2)$ (D is the dimensionality of vertex representation), but we can speed it up to $O(mD|V|)$ by sampling only m candidate vertices for consideration for insertion, at some cost in quality. Here, we sample the m

nearest neighbors of each vertex as candidates by constructing a ball tree on the graph. The algorithm for querying the m nearest neighbors in a ball tree runs in $O(D|V|log|V|)$, thus the whole algorithm runs in $O(mD|V| + D|V|log|V|)$. In the experiments below, we choose $m = 500$, $D = 64$.

Then, we adapt the widely used 2-opt heuristic to improve the resulting greedy tour. The 2-opt approach repeatedly swaps a pair of vertices (v_i, v_j) in the TSP tour, thus reversing the tour between v_i and v_j. We accept a swap if it reduces the k-robust cost. This process is repeated until the tour is locally optimal. For large-scale graphs, we only perform 2-opt until the tour cost reaches a predefined threshold.

We use two metrics to assess the k-robustness of our tours:

- *Label consistency rate* (LC) – Items in a real-world universe tend to occur into natural domain-specific clusters. A good browsing order should respect these clusters, positioning items in the same cluster close to each other in the tour. The *label consistency rate* measures the ratio of neighbors of v which share the same cluster label. Each vertex v_i in $G = (V, E)$ is assigned a cluster label c_i. The label consistency rate l of tour $T = (v_1, v_2, \cdots, v_{|V|}, v_1)$ is defined as:
$$l = \frac{\sum l_{i,j}}{2k|V|}, 1 \leq i \leq |V|, i - k \leq j \leq i + k \tag{5}$$
where:
$$l_{i,j} = \begin{cases} 0, c_i = c_j \\ 1, c_i \neq c_j \end{cases} \tag{6}$$
- Average k-neighbor distance (D) – Here d_k measures the average distance between each vertex and all its neighbors in a window of size k, so
$$d_k = \frac{1}{|V|} \sum_{i=1}^{|V|} \sum_{j=1}^{k} d(t_i, t_{i+j}) \tag{7}$$

4.5 *k*-robust TSP: Experimental Results

We evaluated our k-robust TSP heuristic on each of the three datasets associated with our WikiBrowse, MovieBrowse, and WordBrowse apps. The cluster label for each dataset was generated by the K-means++ algorithm [5], with the number of clusters is set to 8 for each dataset.

We employed our heuristic to built k-robust TSP tours for each $1 \leq k \leq 4$, and measure the cost of each of these tours under the distance and label consistency metrics for various k' values from 1 to 16. The heuristic explicitly seeks to minimize the distance, but implicitly seeks to maximize label consistency.

Our results are shown in Table 1. The dominant results for each column are shown in bold, and highlight along the main diagonal for both evaluation metrics. In particular, tours optimized for k-robustness tend to perform best for the k they were optimized for, which confirms the soundness of our optimization heuristic. Further, designs for $k = 4$ outperform tours optimized for smaller k when tested for higher levels of robustness, namely $k' = 8$ and $k' = 16$.

WikiBrowse	D@1	D@2	D@3	D@4	D@8	D@16	LC@1	LC@2	LC@3	LC@4	LC@8	LC@16
1-Robust Tour	**1.62**	1.73	1.81	1.87	2.04	2.25	**93.0**	91.9	91.0	90.2	87.8	84.2
2-Robust Tour	1.67	**1.71**	1.77	1.82	1.96	2.13	92.7	**92.3**	91.7	91.1	89.3	86.6
3-Robust Tour	1.69	1.73	**1.76**	1.81	1.93	2.09	92.5	92.1	**91.8**	91.3	89.9	87.6
4-Robust Tour	1.71	1.75	1.78	**1.80**	**1.91**	**2.05**	92.4	92.0	91.7	**91.4**	**90.2**	**88.2**

Table 1: Experimental result for k-robust TSP tour optimization for the Wikipedia dataset. D@k denotes the average k-neighbor distance for the given tour, while LC@k denotes the k-neighbor label consistency rate. Tours designed to optimize robustness for a given $1 \leq k \leq 4$ dominate the performance for the criteria they were designed for under both measures of quality. Smaller values of $D@k$ and larger values of $LC@k$ indicate better performance.

5 User Study

Evaluating a user interface for serendipitous browsing is complicated by the nature (or lack) of the task. Success is achieved when the user finds something interesting to them, not a particular item proposed by the investigator.

To provide a baseline for comparison and assess the value of the similarity order, we constructed two versions of the *WikiBrowse* app for both user studies. The standard version uses our 4-robust TSP tour to provide conceptual orders. The alternate version uses alphabetical order instead. we asked our subjects to browse on the app for 5 minutes, and mark all the items they find interesting enough to read its Wikipedia page. Since by default the app displays the 100 most significant people who are already well-known to most subjects, we asked the participants to mark only those people who are not within the top 100. 16 different subjects (9 males and 7 females) were recruited to participate in this study. All the subjects were students at a local university, all of whom have normal or corrected-to-normal eye vision. Each participant was given a brief questionnaire about their experience at the conclusion of their task. To rule out order effects, half of the participants tried the similarity order version first, while the other half used the alphabetical order version first.

For the serendipity discovery task, the average number of interesting people found within five minutes of browsing is 17.7 ($SD = 7.95$) in the similarity order app, versus 11.6 in the alphabetical order app ($SD = 6.20$), which means the similarity order app yields a much higher chance of encountering interesting people. Also, subjects who used the alphabetical order app first made significantly more serendipitous discoveries when they switched to the similarity order app (from 11.5 to 20.6 in the mean). In contrast, subjects who used the similarity order app first showed degraded performance with alphabetical order (from 14.75 to 11.75 in the mean). Thus, the similarity order app proves superior to the alphabetical order app in encouraging serendipitous discoveries.

Table 2 reports the results of our user satisfaction survey, graded on a 7-point Likert scale. Subjects clearly noticed that the neighboring items were more similar

Question	Alpha	Sim
I notice the similarities between the type of people appearing near each other.	1.8 (0.66)	**5.9** (1.17)
The interface is good for discovering interesting Wikipedia people.	4.1 (1.27)	**5.4** (1.00)
The interface is good for finding a specific person in Wikipedia.	**4.9** (1.39)	4.0 (1.06)
It was easy to learn how to use this interface.	5.9 (1.03)	**6.1** (0.70)

Table 2: Average scores for the user interface satisfaction questionnaire, using a 7-point Likert scale, with 1=strongly disagree, 7=strongly agree. Alpha denotes the average score and standard deviation for the alphabetical order app, while Sim stands for that of the similarity order app.

with the k-robust tour than alphabetical order. By contrast, the alphabetical order was deemed better for locating specific figures, exactly the expected result since there is no way to locate specific items by name in the similarity order short of exhaustive search, versus binary search for alphabetical order. Questions addressing user satisfaction generally yielded approval. Subjects generally felt the interface as easy to use and good for discovering interesting people.

6 Conclusion

We have proposed a new design for browsing-oriented user interfaces. Implementing a domain-specific DeepBrowse interface can be reduced to the problem of constructing two permutations, one measuring the similarity between items, and the other the relative significance of each item over the universe. We show how to construct these permutations in a systematic way using graph embeddings and combinatorial optimization. Finally, we report a user study which demonstrates that our interface meets its basic design objectives.

References

1. Al-Rfou, R., Perozzi, B., Skiena, S.: Polyglot: Distributed word representations for multilingual nlp. CoNLL-2013 p. 183 (2013)
2. André, P., Teevan, J., Dumais, S.T.: From x-rays to silly putty via uranus: serendipity and its role in web search. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2033–2036. ACM (2009)
3. André, P., Teevan, J., Dumais, S.T., et al.: Discovery is never by chance: designing for (un) serendipity. In: Proceedings of the seventh ACM conference on Creativity and cognition. pp. 305–314. ACM (2009)
4. Arkin, E.M., Chiang, Y.J., Mitchell, J.S.B., Skiena, S.S., Yang, T.: On the maximum scatter TSP. SIAM Journal on Computing 29(2), 515–544 (2000)
5. Arthur, D., Vassilvitskii, S.: k-means++: The advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 1027–1035. Society for Industrial and Applied Mathematics (2007)
6. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: NIPS. vol. 14, pp. 585–591 (2001)

7. Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, B., Raghavan, P., Sudan, M.: The minimum latency problem. In: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing. pp. 163–171. ACM (1994)
8. Bordino, I., Mejova, Y., Lalmas, M.: Penguins in sweaters, or serendipitous entity search on user-generated content. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management. pp. 109–118. ACM (2013)
9. Chen, Y., Perozzi, B., Skiena, S.: Vector-based similarity measurements for historical figures. Information Systems, Special Issues on Selected Papers from SISAP 2015 64, 163–174 (2017)
10. Clarke, C.L., Kolla, M., Cormack, G.V., Vechtomova, O., Ashkan, A., Büttcher, S., MacKinnon, I.: Novelty and diversity in information retrieval evaluation. In: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval. pp. 659–666. ACM (2008)
11. Cox, T.F., Cox, M.A.: Multidimensional scaling. CRC press (2000)
12. Croes, G.A.: A method for solving traveling-salesman problems. Operations research 6(6), 791–812 (1958)
13. De Bruijn, O., Spence, R.: A new framework for theory-based interaction design applied to serendipitous information retrieval. ACM transactions on computer-human interaction (TOCHI) 15(1), 5 (2008)
14. Hauff, C., Houben, G.J.: Serendipitous browsing: Stumbling through wikipedia. In: Searching4Fun! workshop (2012)
15. Hoffman, K.L., Padberg, M., Rinaldi, G.: Traveling salesman problem. In: Encyclopedia of Operations Research and Management Science, pp. 1573–1578. Springer (2013)
16. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations research 21(2), 498–516 (1973)
17. Liu, H., Xie, X., Tang, X., Li, Z.W., Ma, W.Y.: Effective browsing of web image search results. In: Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval. pp. 84–90. ACM (2004)
18. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
19. Papadimitriou, C.H.: The euclidean travelling salesman problem is np-complete. Theoretical Computer Science 4(3), 237–244 (1977)
20. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710. ACM (2014)
21. Rodden, K., Basalaj, W., Sinclair, D., Wood, K.: Evaluating a visualisation of image similarity as a tool for image browsing. In: IEEE Symposium on Information Visualization. pp. 36–43. IEEE (1999)
22. Rosenkrantz, D.J., Stearns, R.E., Lewis, II, P.M.: An analysis of several heuristics for the traveling salesman problem. SIAM journal on computing 6(3), 563–581 (1977)
23. Skiena, S.S., Ward, C.B.: Who's Bigger?: Where Historical Figures Really Rank. Cambridge University Press (2013)
24. Tenenbaum, J.B., De Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. science 290(5500), 2319–2323 (2000)
25. Toms, E.G.: Serendipitous information retrieval. In: DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries. Zurich (2000)