LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK
LEHR- UND FORSCHUNGSEINHEIT FÜR
PROGRAMMIER- UND MODELLIERUNGSSPRACHEN

Institut für Informatik

Term-Propagation over Structured Data using Eigenvector Computation

Fabian Kneißl

Diplomarbeit

Beginn der Arbeit:	15. März 2010
Abgabe der Arbeit:	13. September 2010
Betreuer:	Prof. Dr. François Bry
	Klara Weiand, M.Sc.
	Dr. Tim Furche

Erklärung

Hiermit versichere ich, dass ich diese Diplomarbeit selbständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

München, den 13. September 2010 .....

Fabian Kneißl

Zusammenfassung

In dieser Diplomarbeit stellen wir PEST (term-**p**ropagation using **e**igenvector computation over **st**ructured data) vor, einen neuen Ansatz für approximative Suche auf strukturierten Daten. PEST nutzt die Struktur der Daten aus, um Termgewichte zwischen Objekten, die miteinander in Beziehung stehen, zu propagieren. Dabei gehen wir speziell auf solche strukturierten Daten ein, bei denen sinnvolle Antworten bereits durch die anwendungsbezogene Semantik gegeben sind, z.B. bei Seiten in Wikis oder Personen in sozialen Netzwerken. Die PEST-Matrix verallgemeinert die bei PageRank verwendete Google-Matrix durch Faktoren, die von den Termgewichten abhängen, und ermöglicht die unterschiedlich starke Gewichtung (semantischer) Ähnlichkeit für verschiedene Beziehungen in den Daten, z.B. Freund vs. Arbeitskollege in einem sozialen Netzwerk. Die Eigenvektoren dieser PEST-Matrix stellen die Verteilung der Terme nach der Propagation dar. Die Eigenvektoren aller Terme zusammen bilden einen Vektorraum-Index, der die Struktur der Daten einbezieht und der mit Standardtechniken des Information Retrieval gehandhabt werden kann. In umfassenden Experimenten mit einem Wiki aus dem wirklichen Leben zeigen wir, wie PEST die Qualität der Suchergebnisse im Vergleich zu mehreren existierenden Ansätzen verbessert. Außerdem stellen wir anhand von Experimenten mit einer sozialen Lesezeichen-Plattform dar, wie unterschiedliche Ansätze zur Auswahl der Beziehungen die Suchergebnisse beeinflussen.

Abstract

In this thesis, we present PEST (term-**p**ropagation using **e**igenvector computation over **st**ructured data), a novel approach to approximate querying of structured data that exploits the structure to propagate term weights between related data items. We focus on structured data for which meaningful answers are given through the application semantics, e.g., pages in wikis or persons in social networks. The PEST matrix generalizes the Google Matrix used in PageRank with a term weight dependent factor and allows different levels of (semantic) closeness for different relations in the data, e.g., friend vs. co-worker in a social network. Its eigenvectors represent the distribution of terms after propagation. The eigenvectors for all terms together form a vector space index that takes the structure of the data into account and can be used with standard information retrieval techniques. In extensive experiments on a real life wiki, we show how PEST improves the quality of the ranking over a range of existing ranking approaches. Furthermore, experiments on a social bookmarking site show how different approaches to relation selection influence the result rankings.

Acknowledgements

I want to thank everyone who supported me during my thesis.

First of all, I am deeply grateful to Prof. François Bry, Klara Weiand and Tim Furche for their outstanding and helpful supervision of this thesis. The deep integration into this research topic right from the beginning and the constant feedback helped me to advance quickly.

Moreover, I would like to thank my family for their kind all-time support and for making my studies possible.

Last but not least, many thanks also go to Tilman Dingler, Richard Röttger and Kevin Wiesner for proofreading my thesis. I really appreciate all these contributions.

Contents

1. Introduction

Mary wants to get an overview of software projects in her company that are written in Java and that make use of the Lucene library for full-text search. According to the conventions of her company's wiki, a brief introduction to each software project is provided by a wiki page tagged with *"introduction"*.

Thus, Mary enters the query for wiki pages containing *"java"* and *"lucene"* that are also tagged with *"introduction"*. The search engine in the semantic wiki KiWi [41] allows for such a search on structured data with different kinds of data items [13].

However, the results fall short of Mary's expectations for two reasons that are also illustrated in the sample wiki of Figure 1:

1. Some projects may not follow the wiki's convention (or the convention may have changed over time) to use the tag *"introduction"* for identifying project briefs. This may be the case for Document 5 in Figure 1. Mary could loosen her query to retrieve all pages containing *"introduction"* (but not necessarily tagged with it). However, in this case pages that follow the convention are not necessarily ranked higher than other matching pages.
2. Some projects use the rich annotation and structuring mechanism of a wiki to split a wiki page into sub-sections, as in the case of the description of KiWi in Documents 1 and 2 from Figure 1; or the mechanism to link to related projects or technologies (rather than discuss them inline), as in the case of Document 4 and 5 in Figure 1. Such projects are not included in the results of the original query at all. Again, Mary could try to change her query to allow keywords to occur in sub-sections or in linked to documents, but such queries quickly become rather complex or impossible with the limited search facilities most wikis provide. Furthermore, this solution suffers from the same problem as addressed above: Documents following the wiki's conventions are not necessarily ranked higher than those only matched due to the relaxation of the query.

Though we choose a wiki to introduce these challenges, they appear in a wide range of applications involving (keyword) search on structured data, e.g., in social networks, in ontologies, or in a richly structured paper repository. The common characteristic of these applications is that relevant answers (e.g., a wiki page or a data item describing a person in a social network) are not fully self-contained entities as in the case of standard web search, but obtain most of their relevance by virtue of their structural relations to other pages, persons, etc. At the same time, they are sufficiently self-contained to serve as reasonable answers to a search query, in contrast to, e.g., elements of an XML document.

Data items such as wiki pages tend to be less self-contained than common web documents. Therefore, PageRank and similar approaches that use the structure of the data merely for ranking of a set of answers do not suffice: As Figure 1 illustrates, also pages that do not contain the relevant keyword can be highly relevant answers due to their relations with, e.g., tags that prominently feature the keyword.

To address this challenge, not only the ranking, but also the selection of answers needs to be influenced by the structure of the data. That PageRank propagates the anchor text of

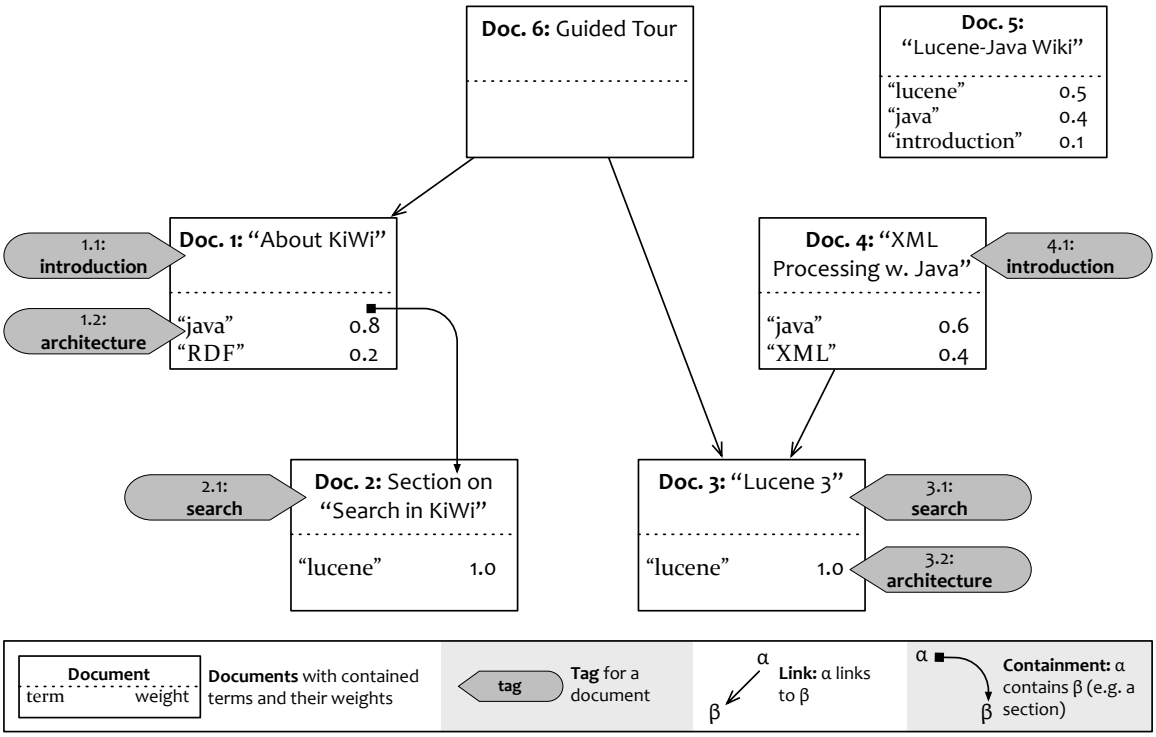


Figure 1: Link and containment graph for a sample wiki

links to a page as if they were content of that page is a first step in this direction.

In this thesis, we present a novel approach to **approximate matching over structured data**: PEST, short for term-**p**ropagation using **e**igenvector computation over **st**ructured data, is based on a unique technique for propagating term weights (as obtained from a standard vector space representation of the documents) over the structure of the data using eigenvector computation. It generalizes the principles of Google's PageRank [12], where links between web pages are employed to calculate the importance of the web pages and thereby connected pages influence each other's score (see Section 2.2). In PEST, this principle is applied on data where the content of a data item is not sufficient to establish the relevant terms for that item, but where rich structural relations are present that allow us to use the content of related data items to improve the set of keywords describing a data item.

In contrast to many other fuzzy matching approaches (see Section 3), PEST relies solely on modifying term weights – a measure for the importance of each word contained in a document – in the document index. Therefore, PEST requires no runtime query expansion, but can use existing document retrieval technologies such as Lucene [22]. Furthermore, the modified term weights already represent the importances of data items in terms of how well they are connected to others in the structured data. Therefore, one can omit a separate adjustment of the answer ranking as with PageRank.

As PEST is hence independent of the query, its computation can be performed entirely at index time. Yet, PEST is able to address all the above described problems in the context of structured data with meaningful answers such as a wiki, a social network, etc. To illustrate

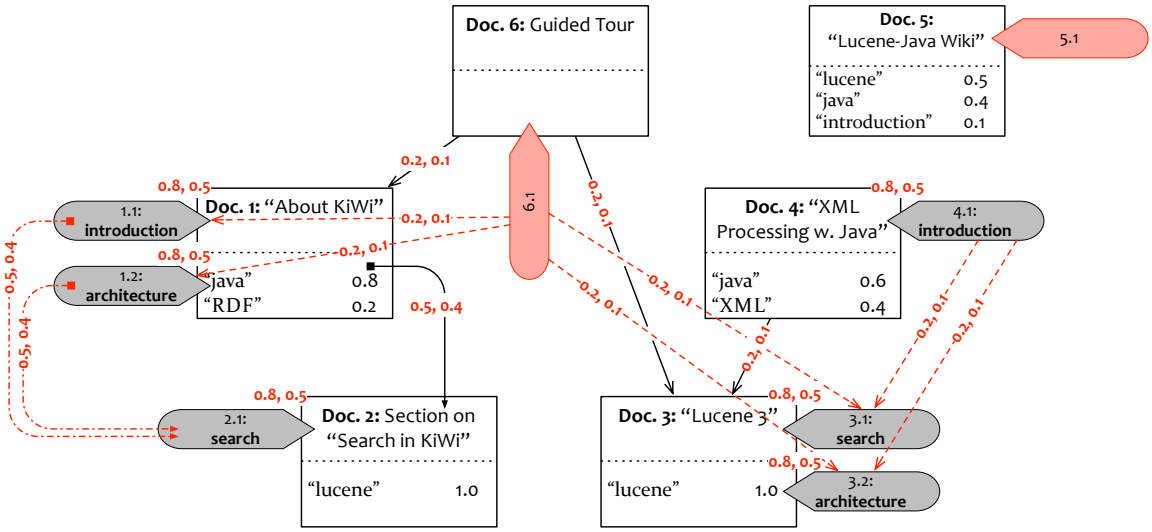


Figure 2: Edge weights and virtual nodes and edges for the graph in Figure 1

how PEST propagates term weights, consider again Figure 1. As in PageRank, the "magic" of PEST lies in its matrix, called the PEST *propagation matrix*, or PEST *matrix* for short. The PEST matrix is computed in four steps:

(1) Weighted Propagation Graph First, we extend and weight the graph of data items (here, wiki pages and tags): (a) For each wiki page without tags, a *dummy tag* (without term weights) is created. (b) For each link (resp. containment relation) between two wiki pages A and B, a link (resp. containment relation) between each pair of tags from A and B is added. These insertions are used to enable direct propagation between tags. Thus, we can configure how terms propagate between tags of related pages independently from term propagation between the pages. (c) Each edge in the resulting graph is assigned a pair of *weights*, one for traversing it from source to sink and one from sink to source. In general, if we have multiple types of relations and data items, this extension allows us to have strong connections between related properties of related items, e.g., between the classes of two highly related instances in an ontology.

The resulting graph for the sample wiki is shown in Figure 2. We have added the tags 5.1 and 6.1 and containment edges from tag 1.1 and 1.2 to tag 2.1, as well as link edges, e.g., from the tag 6.1 to tag $1.1, 2.1, 3.1$ and 3.2. In the following, we assign edge weights based solely on the type of the edge (link, containment, tagging), though PEST also allows for edge specific weights.

(2) Normalization of the Weighted Adjacency Matrix Thereafter, we create the weighted adjacency matrix from the weighted propagation graph, transpose and normalize it. Among several techniques discussed in Section 4.4, we choose to normalize the edge weights of each node by its number of outgoing edges as this gives the best results. Thus, the result is a sub-stochastic rather than a stochastic matrix.

(3) Transformation into the PEST Matrix So far, the matrix does not encode any information about the differences in *term distribution* in the original nodes, but only information about the structure of the wiki graph. To encode that information in the PEST matrix, we use an *informed leap*: In the case of PageRank, the probability for a random surfer to retrieve a web page, i.e., to leap to a node, is distributed uniformly; it is a *random leap*. In contrast, the surfer in PEST is *informed*, and therefore the probability is not equal for all nodes, but influenced by the original term weight distribution: A page with high term weight for term τ is more likely to be the target of a leap than one with low term weight for τ. For encoding this informed leap in the matrix, the remaining probabilities of the sub-stochastic matrix from the previous step together with a fixed leap parameter α (e.g., 0.3) are used. The resulting matrix is called the PEST matrix $\mathbf{P}_\tau$ for term τ. Note that it must be computed for each term individually, but does not depend on the query.

(4) Term Propagation using Eigenvector Computation Finally, the eigenvector $\mathbf{p}_\tau$ of the PEST matrix for term τ is computed using the power method, which iteratively calculates a vector converging to $\mathbf{p}_\tau$. By computing these eigenvectors for all terms, they can be combined to form the vector space representation, i.e., the document-term matrix, for the data items (here, wiki pages and their tags). Keyword *queries* can be evaluated on this representation with any of the existing information retrieval engines using a vector space model.

It is worth emphasizing that only steps 3 and 4 are term-dependent and, as shown in the experimental evaluation in Section 6.1, the time needed for the calculation of the term-dependent part of the evaluation on a wiki with 10,955 pages is in the low seconds on a single core for each term, even without sophisticated optimizations. Also, each term can be computed independently and, therefore, calculations can be parallelized easily. Thus, PEST's index computation scales well even for document collections containing hundreds of thousands of relevant terms (a significant portion of the English vocabulary).

To summarize, PEST improves on approximate keyword search approaches for structured data where meaningful answers are defined by the application in the following aspects: First, it is based on a *simple, but flexible* model for structured content that captures a wide range of knowledge management systems and (tree- or graph-shaped) structured data applications. The main contribution is the PEST matrix for propagating term weights over structured data. The PEST matrix allows the propagation of term weights at *index time* and yields a modified vector space representation that can be used by any information retrieval engine based on the vector space model. Though the PEST matrix is inspired by the Google Matrix used in PageRank, PEST deviates from and generalizes PageRank significantly:

- PEST uses the structure of the data not only for ranking but for *answer selection* as well: PEST also finds relevant answers that do not directly contain the considered keyword at all. Thus, it generalizes the limited form of term propagation in PageRank (from anchors to linked pages) and allows flexible propagation of terms between data items, based, e.g., on their type, the type of their relation etc.
- To achieve this term propagation, PEST uses an *informed leap based on term weight distribution* rather than a random leap. The employed technique is similar to that of

personalized PageRank, where the leap probability is based, e.g., on a user's preferences. However, goal and outcome are different: Where personalized PageRank only computes a ranking, PEST computes a *new vector space representation* of the data items that integrates propagated term weights with relevance based on the relations of a data item. This vector space representation can be used in any existing vector space search engine.

- Since the term weight distribution and thus PEST's informed leap is in general different for each term, PEST needs to consider term propagation for each term separately. Though this increases index time compared to PageRank, the term-dependent computation is only a small part of the overall indexing time and can be computed *independently for each term* and is thus easily parallelized.

Contributions

After introducing PEST in this section, we describe preliminary concepts which are necessary for PEST (Section 2): Graph and matrix propertis, Google's PageRank algorithm, the vector space model as well as the semantic wiki KiWi are covered.

We continue with an overview of current research in Section 3, providing insights into the two neighboring areas information retrieval and adapted PageRank algorithms. There, we also show how PEST differs from existing approaches.

In Section 4, we give a detailed explanation of the PEST algorithm including the computation of that matrix for a given graph of structured content. Moreover, we prove that any PEST matrix has 1 as dominant eigenvalue and that the power method converges with the corresponding eigenvector if applied to a PEST matrix. Also, variants of the algorithm are presented.

The implementation of the PEST algorithm is described in Section 5 where we not only show the details of the architecture but also take a closer look on the employed data structures.

In an extensive experimental evaluation of PEST on an entire real-life wiki (Section 6.1), we compare PEST with three existing keyword search approaches: a simple term frequency based ranking, the ranking used by Wikipedia, and the ranking returned by Google. The experimental evaluation demonstrates that (a) PEST significantly improves the ranking for each of the compared approaches over a range of keyword queries and (b) PEST achieves that improvement at the cost of an increase in index time that is notable, but also easily offset, as it is linear in the number of relevant terms with constants in the low seconds and can be well parallelized.

Thereafter, a second experiment (Section 6.2) with the social bookmarking site Delicious is carried through in order to evaluate methods using different kinds of relationships between URLs. The evaluation shows that not all techniques are recommendable equally but it depends on the focus of the query which method to choose.

Further in Section 6, important properties of the datasets processed by PEST are outlined and recommendations given for the choice of datasets for further evaluations.

Though the experimental results clearly show that PEST as discussed here can significantly improve existing keyword search in wikis as well as social networks, there are a number of open issues and further challenges to PEST summarized in Section 7.

2. Preliminaries

Before we can describe the PEST algorithm, preliminary matters have to be regarded that are necessary to explain PEST's details. Therefore, we will first look at important graph and matrix properties, second describe Google's PageRank algorithm, thereafter examine the vector space model, and lastly characterize the semantic wiki KiWi.

2.1. Graph and Matrix Properties

First, we will have a look at the definition of a graph, its adjacency matrix and, additionally, special properties of matrices required for Section 4.

2.1.1. Graph Definition

In the course of this thesis, the relations between documents are modeled as a graph. Therefore, we first need the precise definition of a directed graph:

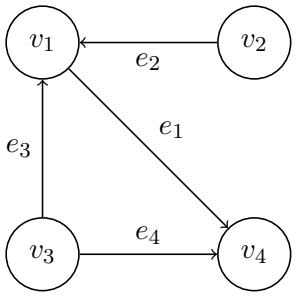
Definition 1. A **directed graph**, or **digraph**, is a tuple $G = (V, E)$ where V denotes a set of vertices (also called nodes) and $E \in V \times V$ a set of edges.

In contrast to an undirected graph, each edge has a specific direction. In the example of Figure 3, $G = (V, E)$ is a directed graph with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{e_1, e_2, e_3, e_4\}$ where $e_1 = (v_1, v_4)$, $e_2 = (v_2, v_1)$, $e_3 = (v_3, v_1)$ and $e_4 = (v_3, v_4)$. The edges' direction is depicted with an arrow. It is common to refer to all edges that originate from one node as its *outgoing edges*, and the edges that point to this node as its *incoming edges*. In the example, the only outgoing edge of v_1 is e_1 and its incoming edges are e_2 and e_3.

An extension of the directed graph is formed by adding weights to the edges of the graph:

Definition 2. A **weighted digraph** is a tuple $G = (V, E, w_e)$ describing a digraph (V, E) as in Definition 1 extended with a function $w_e : E \to \mathbb{R}$ for assigning weights to edges.

For example, the digraph from Figure 3 can be extended to a weighted digraph with $w_e(e_1) = 0.1$, $w_e(e_2) = 0.2$, $w_e(e_3) = 0.3$, and $w_e(e_4) = 0.4$.



$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 3: Directed graph with 4 vertices and 4 edges with its adjacency matrix A

2.1.2. Adjacency Matrix

For graphs as defined in the previous section, we have to consider a sensible representation to store the graph structure in the computer which fits our needs in terms of access and memory efficiency.

Not only the list of nodes has to be stored, but also a good way to access the edges is needed. The first option is to save the edges of a graph as a list, the so-called adjacency list. Then, for each vertice a list is created that contains the edges for this vertice. This is a very memory efficient way to store the graph as only those edges are stored that actually exist in the graph.

Another option is to create an adjacency matrix. Therefore, given a graph with n vertices, these have to be numbered from 1 to n. Then an $n \times n$ matrix is generated in which the i^{th} row and the j^{th} column is 1 if there is an edge from the i^{th} node to the j^{th} node, and 0 otherwise.

Definition 3. Given a digraph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, its corresponding **adjacency matrix** is defined as the matrix

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} = (a_{i,j})$$

with

$$a_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

For the sample graph of the previous section, its adjacency matrix is shown in Figure 3.

The adjacency matrix provides, similar to the adjacency list, a computationally fast way to access the edge information in a graph. It can, however, be quite memory-intensive to store this matrix as its memory usage is $O(n^2)$ for a graph with n vertices. Especially in the case of a sparse graph (i.e., a graph with relatively few edges), many matrix entries are 0. Using this property, one can apply compression algorithms to overcome this weakness in memory efficiency.

In this thesis, we use an adapted form of an adjacency matrix. We do not only store the information whether there is an edge between two nodes, but instead the weights of the edges. Nearly the same definition as above can be given for a weighted digraph:

Definition 4. Given a weighted digraph $G = (V, E, w_e)$ with $V = \{v_1, \ldots, v_n\}$, its corresponding **weighted adjacency matrix** is defined as the matrix

$$A_w = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} = (a_{i,j})$$

with

$$a_{i,j} = \begin{cases} w_e(e) & \text{if } e = (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

For a weight of 0, no edge exists between the two nodes. For the sample graph with the weights as given at the end of the last section, its weighted adjacency matrix is:

$$A_w = \begin{pmatrix} 0 & 0 & 0 & 0.1 \\ 0.2 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0.4 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

In this thesis, we use the adjacency matrix instead of an adjacency list for the reason that we introduce non-trivial weights between all nodes and thus end up at the PEST matrix that only has entries > 0 (as will be shown in Section 4.2.3). Hence, the adjacency list would contain very long edge lists, therefore lose its advantage of memory efficiency and even provide slower access times.

2.1.3. Special Properties of Matrices

We will take a closer look on some special types of matrices. Following up on the previous section, the adjacency matrix by its definition has the property of being a square matrix, i.e. the number of columns equals the number of rows:

Definition 5. Every $n \times n$ matrix is called a **square matrix**.

One further property that becomes important in this thesis is that all entries of the matrix are positive (non-negative) or even strictly positive:

Definition 6. A matrix $A = (a_{i,j}) \in \mathbb{R}^{m \times n}$ is called **positive**, if

$$\forall_{i=1,\ldots,m, j=1,\ldots,n} : a_{i,j} \geq 0.$$

Definition 7. A matrix $A = (a_{i,j}) \in \mathbb{R}^{m \times n}$ is called **strictly positive**, if

$$\forall_{i=1,\ldots,m, j=1,\ldots n} : a_{i,j} > 0.$$

Positive here denotes the property of each entry of the matrix containing a value ≥ 0. A special form of a positive square matrix is a column-stochastic matrix:

Definition 8. A positive square matrix $A = (a_{i,j}) \in \mathbb{R}^{n \times n}$ is **column-stochastic** if

$$\forall_{j=1,\ldots,n} : \sum_i a_{i,j} = 1.$$

In other words, column-stochastic means that each column forms a stochastic vector, i.e., the sum of the vector's values equals 1. The same property of a matrix can be defined for the rows where it is called **row-stochastic**. These types of matrices are especially important for the PageRank algorithm described in Section 2.2. A weaker form of the column-stochastic matrix consists in the column-substochastic matrix:

Definition 9. A positive square matrix $A = (a_{i,j}) \in \mathbb{R}^{n \times n}$ is **column-substochastic** if

$$\forall_{j=1,\ldots,n} : \sum_i a_{i,j} \leq 1.$$

Respectively, **row-substochastic** refers to a matrix where the row sums are ≤ 1. We use a column-substochastic matrix as an intermediate step to forming a column-stochastic matrix in the PEST algorithm.

Furthermore, the PEST algorithm can be simplified significantly by the usage of the transposed matrix A^T (compare step 2 of PEST in Section 4.2.1):

Definition 10. Given a matrix $A = (a_{i,j}) \in \mathbb{R}^{m \times n}$, its **transposed matrix** $A^T \in \mathbb{R}^{n \times m}$ is given as follows:

$$A^T = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix}^T = \begin{pmatrix} a_{1,1} & \cdots & a_{m,1} \\ \vdots & \ddots & \vdots \\ a_{1,n} & \cdots & a_{m,n} \end{pmatrix} = (a_{j,i}).$$

2.2. Google's PageRank Algorithm

An algorithm that received a lot of interest as part of the Google search engine is the PageRank algorithm developed by Page et al. [12, 35]. PEST extends this algorithm and applies it to approximate search on structured data. Therefore, we describe PageRank in the following before examining PEST more closely.

PageRank is based on the intuition that a link from one web page to another can be seen as an endorsement of the linked page. One can think of a random surfer who browses the web and repeatedly follows the links of web pages with equal probability for each link. Then, a page is often visited – and thus important – if many important pages link to it, even more so if the important pages only contain a few links. Furthermore, the random surfer gets bored after a while and jumps to a random web page. This probability is encoded as the random leap probability α (also called damping factor). Combining linking and random leap, the result of the random surfer browsing the web is the so-called PageRank, an importance score for each web page representing how often it is accessed by the surfer.

The PageRank idea is implemented by transforming the link graph into an adjacency matrix, also called transition matrix, which is augmented by a random leap component that ensures that the probability to jump from any web page to any other page is non-zero. As a consequence of introducing the random leap and setting α to a non-zero value, the normalized matrix is column-stochastic and strictly positive, which guarantees the existance of a unique PageRank vector as described below. In standard PageRank, the random leap factor operates using a uniform distribution, that is, the likelihood of a transition to a page is identical for all pages. For a set of N pages, the corresponding leap, or teleportation, vector m can be seen to consist of N entries with value $\frac{1}{N}$ each. Therefore, if A denotes the adjacency matrix (which has been normalized to be column-stochastic) without the random

leap component, the Google Matrix M becomes:

$$M = (1-\alpha)A + \alpha \underbrace{\begin{pmatrix} \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{pmatrix}}_{m} \cdot \underbrace{\begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}}_{N \text{ columns}} = (1-\alpha)A + \alpha \begin{pmatrix} \frac{1}{N} & \cdots & \frac{1}{N} \\ \vdots & \ddots & \vdots \\ \frac{1}{N} & \cdots & \frac{1}{N} \end{pmatrix}$$

$$= \begin{pmatrix} (1-\alpha)a_{1,1} + \frac{\alpha}{N} & \cdots & (1-\alpha)a_{1,n} + \frac{\alpha}{N} \\ \vdots & \ddots & \vdots \\ (1-\alpha)a_{n,1} + \frac{\alpha}{N} & \cdots & (1-\alpha)a_{n,n} + \frac{\alpha}{N} \end{pmatrix}$$

Once this modified transition matrix M has been set up, a vector, called PageRank vector, needs to be found which contains the importance scores of the web pages. This problem can also be formulated as an eigenvalue problem for M. Given a strictly positive square matrix (which we have), the Perron–Frobenius theorem asserts that a unique largest eigenvalue together with its eigenvector exists. Furthermore, it can even be proven that this eigenvalue equals 1 [14]. Then, its corresponding eigenvector equals the PageRank vector. This vector can be seen as the convergence of the web page importances of infinitely many random surfers who visit web pages based on their links.

In the original paper, an iterative method for solving the eigenvalue problem is suggested that is called the "power method" (see also Langville and Meyer [31], Golub and Van Loan [21]):

Algorithm 1 Pseudocode of the power method, adapted from [35]. p_i is the PageRank vector at iteration i, s the starting vector, δ the residual and ϵ the defined threshold where the iteration is terminated

$p_0 \leftarrow s$
$i \leftarrow 0$
repeat
  $p_{i+1} \leftarrow M p_i$
  $p_{i+1} \leftarrow \frac{p_{i+1}}{||p_{i+1}||_2}$
  $\delta \leftarrow ||p_{i+1} - p_i||_1$
  $i \leftarrow i + 1$
until $\delta < \epsilon$

Let $s \in \mathbb{R}^N$ be any vector with $||s||_2 = 1$, e.g., $\begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}^T$. This starting vector is used as the first vector p_0. Then, the matrix M is iteratively multiplied with the vector p_i and this vector is normalized after each iteration until the vector converges close enough to the eigenvector. To measure the convergence process, a residual is calculated as the difference between the two latest vectors: $||p_{i+1} - p_i||_1$. The termination criterion is fulfilled when the residual is below a custom threshold. Depending on the starting vector, the matrix, and the needed accuracy, Brin and Page report about 50 to 100 iterations till the designated accuracy is met [35]. In our experiments with PEST, even fewer iterations in the range of 10 to 30 showed to be sufficient for a maximum residual of 10^{-6}.

To sum up, the PageRank algorithm calculates importance scores for interlinked web pages. Relevant for this thesis is the detail that already in the original algorithm some term propagation is suggested: Anchor texts in hyperlinks between web pages are treated as if they belonged to the linked web page. Therefore, the context of the web page is added to the page's textual content and thus this technique results in a minimal form of fuzzy matching which only takes directly related anchor links into account. Compared to PEST, which uses a similar technique to PageRank, the scope as well as its aims differ, which will further be discussed in Section 3.2.

2.3. Vector Space Model

From a linguistic perspective, a further research area on which this thesis is built is information retrieval (IR). IR can best be described as a way to receive pieces of information (e.g., web pages or documents) upon submitting a search query [23]. A common example consists in entering a query in a web search engine and thereafter receiving a list with web pages that ideally contain the page looked for.

The vector space model (VSM) plays an important role for many IR applications, as it is versatile and provides a well established basis to build other systems on it, for example the Lucene search engine [3]. In the VSM, each document is represented as a vector of fixed length containing weights for all indexed terms. Therefore, terms have to be determined by tokenizing each document's content and – if necessary – processing the terms, e.g., as described in Section 5.1.3.

One standard method for determining the weights is the tf-idf method [40]; tf stands for *"term frequency"* and idf for *"inverse document frequency"*. The weight for term t and document d is then given as:

$$w_{t,d} = \mathsf{tf}_{t,d} \cdot \mathsf{idf}_t = \frac{\mathrm{freq}_{t,d}}{\sum_t \mathrm{freq}_{t,d}} \cdot \log \frac{N}{n_t}$$

where $\mathrm{freq}_{t,d}$ denotes the number of occurrences of term t in document d, N the number of documents, and n_t the number of documents containing term t. This way to compute the weights combines the relative importance of a term in a document (tf) with the rarity of this term over all documents (idf). As this method has already proven effective for similar applications, we incorporated it into PEST as well.

The vectors for all documents are computed and stored, e.g., in a *document-term matrix* where the columns depict the terms and the rows the documents. Upon query time, a vector is built using the same processing steps as for the documents and this query vector is compared to the document vectors. The quality of a match is calculated using a distance measure between each two vectors. A standard measure that is often regarded and that is also used in this thesis is the cosine measure: Given a query vector q and a document vector d, the measure assembles to $\cos(q,d) = \frac{\langle q,d\rangle}{|q|\cdot|d|}$ with the Euclidean scalar product $\langle q,d\rangle = \sum_i q_i d_i$ and the Euclidean L_2-norm $|q| = \sqrt{\langle q,q\rangle}$. This measure is independent of the Euclidean length of the vectors and only measures the angle between the vectors. As a consequence, two documents are very similar when many terms have similar weights.

Figure 4: Semantic wiki KiWi showing the content item *"KWQL"* containing another content item *"Some recent publications on KWQL"*

Thus, the most important questions are how to determine the weights of the vector and what distance measure to choose. In this thesis, we give an answer to how to compute the weights for approximate matching. Lucene is an example for a widely used and successful search engine which is based upon the VSM. This tool is also used in our implementation (Section 5).

2.4. KiWi as an Example for a Semantic Wiki

As already mentioned in the introduction, the PEST algorithm can be applied to a semantic wiki very well. To show an example of such a wiki, KiWi[1] is one with extended functionality in the areas of information extraction, personalization, reasoning, and querying [41]. See Figure 4 for a screenshot. KiWi relies on a modular conceptual model consisting of the following building blocks:

Content Items are composable wiki pages, the primary unit of information in the KiWi wiki. A content item consists of text or multimedia and an optional sequence of *contained* content items. Thus, content item containment provides a conventional way to structure

[1] `http://www.kiwi-project.eu`, showcase at `http://showcase.kiwi-project.eu/`

documents, for example a chapter may consist of a sequence of sections. For reasons of simplicity, content item containment precludes any form of overlapping or cycles.

Links are simple hypertext links and can be used for relating content items to each other or to external web sites.

Tags are meta-data that can be attached to content items and links, describing their content or properties. They can be added by users, but can also be created by the system through automatic reasoning.

To illustrate these concepts, consider again Figure 1: It shows a sample KiWi wiki using the above structuring concepts (for sake of familiarity, we call content items documents). For example, the content item (document) 1 "About KiWi" contains the content item 2 representing a section on "Search in KiWi" and is linked to by the content item 6 "Guided Tour". It is tagged with 1.1 *"introduction"* and 1.2 *"architecture"*.

This wiki can be represented as a directed graph as follows: Content items, links and tags are nodes whereas the linking, tagging and containment relationships are modeled as edges between the involved nodes, e.g. the attachment of a tag to a content item is formed by an edge from the tag to the content item. Note that we did not integrate links as nodes into the sample in Figure 1 for simplicity reasons.

Structure, within as well as between resources, thus plays an important role for expressing knowledge in the wiki, ranging from tags to complex graphs of links or content item containments. In this thesis, we exploit these structural relations for gaining different types of links in order to propagate terms along edges with different weights. For example, a mere linking between two documents can be regarded as a weaker relationship than the containment of one document in another one.

3. Related Work

Before looking at the algorithm itself, we identified research related to this thesis. Therefore, works from two different areas will be shown up: From the area of information retrieval, research taking the tree edit distance and adapted vector space models into account will be evaluated. Alongside, customized PageRank algorithms present the second research area of interest for this thesis.

3.1. Fuzzy Matching in Information Retrieval

Before we consider specific approaches of fuzzy matching, it is worth recalling that *approximate matching* (not only returning results that perfectly match the query, but also similar ones) and *ranking* (sorting results to be in a specific order) are closely related. Though they do not have to be used in conjunction, this is often the case, in particular to allow an approximate matching engine to differentiate looser results from results that adhere more strictly to the query. The full power of ranking and approximate matching is unleashed only in combination – approximate matching extends the set of results, ranking brings the results into an order for easier consumption by the user.

While approximate matching is widely used in web search and other IR applications, conventional query languages for (semi-)structured data such as XQuery [11], SQL [20] or SPARQL [38] do not usually employ approximate matching or rank results. These languages have been applied to probabilistic data, but this area is distinct from approximate querying: In probabilistic data management, the data itself introduces uncertainty, in approximate matching uncertainty is introduced under the assumption that the user is also interested in matches that do not quite match his query.

As the amount of structured web data increases and the semantic web continues to emerge, the need for solutions that allow for layman querying of structured data arises. Research has been dedicated to combining web querying and web search and introducing IR methods to querying, for example in the form of extensions to conventional query languages [38], visual tools for exploratory search [4, 33], an extension of web keyword search to include (some) structure [24], or keyword search over structured data [26]. With the arrival of these techniques, the need for approximate querying that does not apply solely to individual terms or phrases but that takes the data structure into account arises.

Approximate matching on data structure has mainly been researched in the context of XML [46]. The majority of work in this area can be divided into two main classes of approaches which are described in the following.

3.1.1. Tree Edit Distance

Tree edit distance [45, 16, 10] is a popular and well-researched approach for assessing the similarity of tree-shaped data. It is based on the concept of the edit distance for strings [32]. Given a pair of XML trees, a set of edit operations (typically at least node deletion and insertion), and a cost function, the tree edit distance is the cost of the cheapest sequence of operations that transforms one tree into the other. For a given XML tree, its best-

matching XML trees are those that have the lowest tree edit distance. Tree edit distance thus quantifies the similarity between the documents through the number of steps and types of operations needed to eliminate the differences between them. XML search can be straightforwardly formulated as finding XML documents that have a sufficiently low tree edit distance to a query represented as a labeled tree [42].

As finding the best-matching trees through the exhaustive computation of tree distances, itself costly, is computationally expensive, research has been focused on developing efficient distance algorithms [25, 48].

Amer-Yahia et al. [6, 7] present a conceptually related approach where queries in the form of tree patterns are gradually generalized by applying different types of pattern relaxation operations such as turning a parent-child node into an ancestor-descendant node. To limit the amount of necessary calculations, a procedure for the efficient generation of queries whose answers lie above a certain threshold is introduced.

Shasha et al. [44] present an approach where the distance between a query, represented as an unordered labeled tree, and an XML document is quantified by counting the number of root-to-leaf paths in the query that do not occur in the document.

In contrast to PEST, the described approaches are hard to generalize to graph data. Both pattern relaxation and tree edit distance require expensive calculations at query time, either a loop to relax the query and the evaluation of an often quite considerable number of relaxed queries or a significant number of distance calculations. PEST's computation on the other hand can be performed entirely at index time. Further, it is not obvious how different edge types, which are easily treated by PEST, affect tree edit distance and pattern relaxation.

Further, both procedures make the assumption that the query can be represented as a single tree. It is unclear how advanced features of a query language like disjunction, negation and vagueness in the query can be efficiently integrated into the approaches. This is in contrast to PEST which is independent of the query formalism being used.

Again, the contrast to PEST lies (a) in the limitation to tree-shaped data which would be hard to lift at least in the case of Shasha et al. [44] due to the reliance on paths and suffix trees and (b) in the need for a new query engine, where PEST can reuse existing information retrieval engines.

3.1.2. Adapting the Vector Space Model

Another class of approaches aims, like PEST, to adapt the vector space model to the application on XML data. In the vector space model, documents and queries are represented as vectors consisting of a weight for each term. Their similarity is then computed using for example the cosine measure between the two vectors. Different schemes can be used for calculating the term weights, the most popular one being tf-idf.

Pokorný [36] represents individual XML documents in a matrix instead of a vector, indicating the term weight for each combination of term and path. A query, also expressed as an XML tree, is transformed into a matrix of the same form. The score of a query with respect to a possible result is then calculated as the correlation between the two matrices. In an extension, the matrix is adapted to reflect also the relationship between paths.

In the work of Carmel et al. [15], document vectors are modified such that their elements are not weights for terms but rather weights for term and context pairs – the contexts of a term are the paths within which it occurs. The vector then consists of a weight for each combination of term and context. Queries, represented as XML fragments, are transformed into query vectors of the same form. Further, the similarity measure is modified by computing context similarities between term occurrences in the query and data. These are then integrated in the vector similarity measure.

Schlieder and Meuss [43] calculate adapted term frequency and inverse document frequency scores replacing textual terms by so-called structured terms, that is, all possible XML subtrees in a dataset. Each structured term can then be described by a vector of tf-idf scores. In contrast, query vector weights are defined by the user. The vector space model is applied to compare subtree and query vectors.

Activation propagation is used by Anh and Moffat [8] for approximate matching over XML structures. Here, a modified version of the vector space model is used to calculate similarity scores between query terms and textual nodes in the data. The calculation of term weights takes into account the structural context of a term as well as its frequency. In a second step, these scores are propagated up in the tree. Finally, the highest activated nodes are selected, filtering out some results which are considered to be unsuitable such as the descendants of results that have already been selected. This approach resembles ours in that activation propagation and the vector space are used to realize approximate matching over structure. However, unlike PEST, propagation happens upon query evaluation and is unidirectional. Like the other approaches in this class, it is also limited to tree-shaped data.

3.1.3. Differences to PEST

PEST differs from the majority of approximate matching approaches including those reviewed above in several important aspects:

- PEST does not realize fuzzy matching by defining a structural distance function and ranking results according to how close they are to a strict match. Instead, it uses the structure of the data to determine which terms are relevant to a document, regardless of whether or not they explicitly occur in it. As a consequence, not only new matches are introduced, but strict matches may also be re-ranked depending on their structural connections.
- PEST is designed for *graph-shaped data* rather than purely hierarchical data as most of the XML-based approaches discussed above.
- PEST can be used with any information retrieval engine based on the vector space model. The only modification to the evaluation process is the computation of the actual term weights. Otherwise, existing technology (such as Lucene or similar search engines) can be utilized. In particular, the PEST matrix is query-independent and thus can be computed at *index time*. No additional computations such as query transformations are needed during query evaluation.

3.2. Fuzzy Matching using Adapted PageRank Algorithms

Where the above approaches for approximate (keyword) search and querying on XML data are similar in aim, PageRank is closely related to PEST in technique, but differs considerably in aim and scope. The original PageRank article [12] suggests to exploit anchor-tags for web search. The anchor text of a link to a web page is treated as if it is part of the text of that web page. This suggestion can be seen as a special case of the approach suggested in this thesis where the only kind of propagation is that from anchor tags to linked pages and where link weights are ignored. The application of this approach is limited to anchor tags and does not apply to general graphs or generalizes to different link types. However, there are a number of extensions [9] of the original PageRank that share more of the characteristics of PEST.

3.2.1. Adapted PageRank Algorithms

One of these approaches consists in the personalized version of PageRank. Brin and Page [12] point out the possibility to realize such a version by using a non-uniform leap vector instead of a uniform vector. This vector can be used to customize the importance of certain pages, for example spam pages could be eliminated from the search results by setting the probability to jump to such pages to zero. Several schemes for improving the scalability of personalized PageRank have been presented in recent years [27, 29, 30, 39]. They are discussed in this section as well as in Section 7.1.2.

Topic-sensitive PageRank [27] builds on the idea of a personalized teleportation vector by introducing a number of topic-specific leap vectors, each assigning a uniform value to all pages relevant to the respective topic and 0 otherwise. The topic-dependent importance scores for all pages are calculated offline. At query time, a weighted classification of the query into topics is computed. A query-specific PageRank can then be calculated as a mixture of topic-specific scores. The motivation behind topic-sensitive PageRank is to avoid that generally important pages get highly ranked despite not containing information relevant to the query.

Query-dependent PageRank [39] is another extension of PageRank which is based on the idea that web pages matching a query that are connected to other matching pages should be ranked higher. The PageRank algorithm is adapted in such a way that the probability of a transition from one page to another is determined by how relevant the target web page is to the query. Towards this end, the distribution underlying the leap vector as well as the mode of calculating transition probabilities are adjusted. In both cases, the probability to transition to a web page is given as the proportion between that web page's relevance score and the sum of all matching pages' relevance scores. When a web page has no non-zero outlinks, the leap vector is used to jump to another web page. The transition matrix is not strictly positive and nodes which do not contain the relevant term are ignored. The PageRank vector is calculated for each term at index time, the scores for each term in the query are combined upon query evaluation.

3.2.2. PEST in Comparison to Existing PageRank Approaches

PEST differs from the described approaches in various ways. In contrast to PageRank (but similar to topic-sensitive and query-dependent PageRank), several matrices and eigenvectors are calculated, namely one per term, each using a term-dependent leap vector. In contrast to topic-sensitive PageRank as well as PageRank, the leap vectors do not use a uniform distribution.

Query-dependent PageRank overall makes little difference between matching web pages that are connected and those that are not, using the same functions to calculate leap and transition weights. On the other hand, only web pages which directly contain the term are exploited for the calculation, indirect connections between relevant pages via another page which does not contain the term are ignored.

Most importantly, PEST differs from all three approaches in the following ways:

- None of the approaches implements approximate matching over structured data or generally adds additional relevant results; they are purely approaches to *ranking* sets of web pages.
- The *assignment of edge weights in* PEST *is more flexible* in that edge weights can be set explicitly and individually or different weights can be chosen depending on the type of edge. In contrast, in PageRank and topic-sensitive PageRank edge weights are uniform and in query-dependent PageRank, edge weights are derived from keyword matches.
- While PEST can be used on web pages with linking as the only relation between pages, its versatile and extensible data model allows for the application to many *other types of graph-shaped data* such as a fine-grained modeling of structured web data.
- In PEST, the *probability of a leap is variable depending on the number and weight of outgoing edges of a node*, thus encoding the intuition that a user jumps to a new page when he cannot find what he is looking for by following links. The leap distribution is the combination of a uniform distribution (as in PageRank and topic-sensitive PageRank) but takes term distributions into account (as in query-dependent PageRank).

PEST can be seen as a generalization of at least PageRank and topic-sensitive PageRank: The results of classic PageRank can be reproduced by choosing edge weights in such a way that they are all identical after normalization and by using only the random leap component of the leap vector.

The behavior of topic-sensitive PageRank can be achieved by clustering all words belonging to a topic together, setting the edge weights as described above, and using only the informed leap component of the leap vector.

Query-dependent PageRank cannot directly be emulated as PEST relies on a strictly positive matrix and does not ignore nodes that do not (yet) contain the respective term. Further, different edge types corresponding to the different possible edge weights would have to be introduced.

4. The PEST Algorithm

After describing all necessary preliminaries and the related work, we will now focus on the PEST algorithm itself. Furthermore, we will explain the algorithm on an example and show up different techniques for normalizing the adjacency matrix.

4.1. Overview

Based on the model for a knowledge management system in Section 2.4, we now formally define the propagation of term weights over structured relations represented in a content graph by means of an eigenvector computation.

A document's tag is descriptive of the textual content of said content item – they have a close association. Similarly, the tags of a sub-document to some extent describe the parent document since the document to which the tag applies is, after all, a constituent part of the parent document. More generally, containment and linking in a wiki or another set of documents indicate relationships between resources. We suggest to exploit these relationships for approximate matching over structured data by using them to propagate resource content. A resource thereby is extended by the terms contained in other resources it is related to. Then, standard information retrieval engines based on the vector space model can be applied to find and rank results oblivious to the underlying structure of term weight propagation.

4.2. Detailed Steps of the Algorithm

Figure 5: The four steps involved in the PEST algorithm

The algorithm can be separated into four steps which are displayed in Figure 5:

1. The weighted propagation graph is generated from the graph,
2. its weighted adjacency matrix is created and normalized,
3. the PEST matrix is built by combining the normalized, weighted adjacency matrix with a custom leap matrix,
4. the eigenvectors are computed to produce a result ranking.

These steps will now be discussed further.

4.2.1. Weighted Propagation Graph

In this section, we formally define a generic graph-based model of structured content that is capable of capturing the rich knowledge representation features of a wide range of knowledge

management applications such as wikis, social networks, etc. We distinguish data items into primary (e.g., wiki pages or people in a social network) and annotation items (e.g., tags or categories). Therefore, we define a type structure as follows:

Definition 11. A **type structure** is a tuple $\mathcal{T} = (\mathcal{D}, \mathcal{A}, \mathcal{E})$ where $\mathcal{D}$ is the set of types for primary data items, $\mathcal{A}$ the set of types for annotation data items, and $\mathcal{E}$ the set of edge types.

Coming back to the wiki example KiWi, $\mathcal{D}$ and $\mathcal{A}$ each contain a single type, *"content item"* resp. *"tag"* and $\mathcal{E}$ contains two types, *"link"* (l) and *"containment"* edges (c). This suffices to represent the primary representation mechanisms of KiWi: A link edge from a content item to a tag represents a tagging and a link edge between two documents a (hypertext) link. A containment edge between two content items represents the nesting of these documents. Containment between a content item and a tag is not present in KiWi and thus is disregarded.

Definition 12. A **content graph** is a tuple $G = (\mathcal{T}, V, E, \mathit{type}, T, w_t)$ where $\mathcal{T}$ is its type structure, V the set of vertices (data items) in G and $E \subseteq V \times V \times \mathcal{E}$ its set of typed edges. $\mathit{type} : V \to \mathcal{D} \cup \mathcal{A}$ assigns a type to each node. The textual content of data items is represented by a set T of terms and the term weight function $w_t : V \times T \to \mathbb{R}^+$ that assigns a weight to each pair of a node and a term. The term weights for each node v are required to form a stochastic vector (i.e., $\sum_{\tau \in T} w_t(v, \tau) = 1$).

For the sample wiki from Figure 1 on page 14, the six documents 1 to 6 (with type *"content item"* from $\mathcal{D}$) and the tags 1.1, 1.2, 2.1, 3.1, 3.2, 4.1 (with type *"tag"* from $\mathcal{A}$) form V, $(1, 2, l)$, $(6, 1, l)$, $(6, 3, l)$, $(4, 3, l)$, $(1, 1.1, l)$, $(1, 1.2, l)$, …, $(4, 4.1, l)$ and $(1, 2, c)$ form the set of all edges E, the set of all terms in the wiki form T, and $w_t(1, \text{"java"}) = 0.8$, …, $w_t(2.1, \text{"search"}) = 1$, …

To be able to control where propagation will be performed and to what extent, we first augment the content graph with a number of additional edges and vertices and, second, assign weights to all edges in that graph.

Definition 13. A **weighted propagation graph** is a content graph extended with a function $w_e : E \to \mathbb{R}_0^+$ for assigning weights to edges that fulfills the following conditions:

- Each primary data item carries at least one annotation of each annotation type: For each node $v_d \in V$ with $\mathit{type}(v_d) \in \mathcal{D}$ and each annotation type $t_a \in \mathcal{A}$, there is a node $v_a \in V$ with $\mathit{type}(v_a) \in \mathcal{A}$ and two edges $(v_d, v_a, t), (v_a, v_d, t') \in E$ for some edge types t and t'.
- For each edge between two primary data items there is a corresponding edge between each two annotations of the two primary data items, if they have the same type: For each $v_d, w_d, v_a, w_a \in V$ with $\mathit{type}(v_d) \in \mathcal{D}$, $\mathit{type}(w_d) \in \mathcal{D}$, $\mathit{type}(v_a) \in \mathcal{A}$, and $\mathit{type}(w_a) \in \mathcal{A}$ with $(v_d, w_d, t), (v_d, v_a, t_a), (w_d, w_a, t_a) \in E$ and $\mathit{type}(v_a) = \mathit{type}(w_a)$, there is an edge $(v_a, w_a, t) \in E$.

Edges are usually created in each direction, as they describe an asymmetric relationship where the weights differ in their values. In this thesis, we often give the weights for the two

edges as one pair of numbers, one for traversing the edge in its direction, one for traversing it against its direction.

In the context of KiWi, the first condition of Definition 13 requires that each document must be tagged by at least one tag. The second condition ensures that tags of related documents are not only related indirectly through the connection between the documents, but also stand in a direct semantic relation. For example, a document which contains another document about a certain topic trivially also is about that topic to some extent, since one of its constituent parts is. As information about the content or properties of a document is contained in its tags, we claim that edges must be added between tags of documents that share an edge.

Proposition 14. *For every content graph, a weighted propagation graph can be constructed by (1) adding an empty annotation node of type a to each primary data item that does not carry an annotation of type a via an edge of arbitrary type and (2) copying any relation between two primary data items to its same-type annotation vertices.*

For the sample wiki from Figure 1 on page 14, the resulting weighted propagation graph is shown in Figure 2 on page 15. The graph contains two "dummy tags" (5.1 and 6.1) as well as a number of added edges between tags of related documents.

We call a weighted propagation graph *type-weighted*, if for any two edges $e_1 = (v_1, w_1, t), e_2 = (v_2, w_2, t) \in E$ it holds that, if $\mathit{type}(v_1) = \mathit{type}(v_2)$ and $\mathit{type}(w_1) = \mathit{type}(w_2)$, then $w_e(e_1) = w_e(e_2)$. In other words, the weights of edges with the same type and with start and end vertices of the same type respectively must be the same in a type-weighted propagation graph. In the following, we only consider such graphs.

Also the example graph in Figure 2 on page 15 is type-weighted, e.g., the weights for a link relation from one content-item to another content-item are always 0.2 in forward and 0.1 in reverse direction.

4.2.2. Normalization of the Weighted Adjacency Matrix

As a second step to generating the PEST matrix, we compute the weighted adjacency matrix:

Definition 15. Let us assume that all nodes of the weighted propagation graph G are numbered as $v_1, \ldots, v_n$. Then the **weighted adjacency matrix** $\mathbf{A}_w$ of G is given as

$$\mathbf{A}_w = \left(\sum_{e=(v_i,v_j,t)} w_e(e) \right)_{i,j}$$

with $i, j = 1, \ldots, n$, $e \in E$, $v_i, v_j \in V$ and $t \in \mathcal{E}$.

Before using this matrix in the PEST algorithm, we have to make sure that all entries of the matrix are between 0 and 1. Thus, we normalize $\mathbf{A}_w$ and furthermore transpose it for easier handling by PEST. Thereupon, we obtain the transition matrix $\mathbf{H}$ for G as follows (where $\mathit{outdegree}(v)$ denotes the number of outgoing edges of v):

$$\mathbf{H} = \left(\frac{1}{\mathit{outdegree}(v_i)} \left(\mathbf{A}_w^T \right)_{i,j} \right)_{i,j}$$

By normalizing with the number of outgoing links rather than the total weight of the outgoing edges, edge weights are preserved to some extent. At the same time, nodes with many outgoing edges are still penalized. Normalization with the out-degree proved the most effective in our experiments compared to, e.g., normalizing with the maximum sum of outgoing term weights (over all nodes) or with the sum of outgoing term weights for each node. Different choices for normalization preserve different properties of the original matrix and it depends on the use case which one should be used. For our purposes, the described approach worked best. But for other applications of the PEST algorithm, another method might be more suited. For a more detailed discussion, have a look at Section 4.4.

4.2.3. Transformation into the PEST Matrix

To propagate term weights along structural relations, we use a novel form of transition matrix, the PEST propagation matrix. In analogy to the *random surfer* of PageRank, the term weight propagation can be explained in terms of a *semi-random reader* who is navigating through the content graph looking for documents relevant to his information need expressed by a specific term τ (or a bag of such terms). He has been given some – incomplete – information about which nodes in the graph are relevant to τ. He starts from one of the nodes and reads on, following connections to find other documents that are also relevant for his information need (even if they do not literally contain τ). When he becomes bored or loses confidence in finding more matches by traversing the structure of the wiki (or graph in general), he jumps to another node that seems promising and continues the process.

To encode this intuition in the PEST matrix, we first consider which connections are likely to lead to further matches by weighting the edges occurring in a content graph. The transposed, normalized adjacency matrix $\mathbf{H}$ of the resulting graph has been calculated in the last two sections. Second, we discuss how to encode, in the leap matrix $\mathbf{L}_\tau$, the jump to a *promising* node for the given term τ – rather than to a random node as in PageRank.

In the following, we will show how to compute the leap matrix $\mathbf{L}_\tau$. Given a leap factor $\alpha \in (0, 1]$, a leap from node j occurs with a probability

$$P(\text{leap}|j) = \alpha + (1 - \alpha)(1 - \sum_i \mathbf{H}_{i,j}).$$

The second summand of this equation is needed to fill up the matrix's columns in order to arrive at a column-stochastic matrix, as will be discussed in the next section.

A leap may be *random* or *informed*. In case of a random leap, the probability of jumping to some other node is uniformly distributed and calculated as $l^{\text{rnd}}(i, j) = \frac{1}{|V|}$ for each pair of vertices (i, j).

An informed leap by contrast takes the term weights, that is, the prior distribution of terms in the content graph, into account. It is therefore term-dependent and given as $l^{\text{inf}}_\tau(i, j) = \frac{w_t(i,\tau)}{\sum_k w_t(k,\tau)}$ for a $\tau \in \mathcal{T}$. Thus, when the probability of following any of the outgoing edges of a node decreases, in turn a leap becomes more likely.

In preliminary experiments, a combination of random and informed leap, with heavy bias

towards an informed leap, proved to give the most desirable propagation behavior. The overall leap probability is therefore distributed between that of a random leap and that of an informed leap occurring according to the factor $\rho \in (0, 1]$, which indicates which fraction of leaps are random leaps.

Therefore, we obtain the leap matrix $\mathbf{L}_\tau$ for term τ as

$$\mathbf{L}_\tau = \Big(P(\text{leap}|j) \cdot ((1-\rho) \cdot l_\tau^{\text{inf}}(i,j) + \rho \cdot l^{\text{rnd}}(i,j)) \Big)_{i,j}.$$

Lemma 16. *$\mathbf{L}_\tau$ is strictly positive.*

Proof. By choosing $\rho \in (0, 1]$, the following is true for all i, j:

$$(1-\rho) \cdot l_\tau^{\text{inf}}(i,j) + \rho \cdot l^{\text{rnd}}(i,j) = \underbrace{(1-\rho) \cdot \frac{w_t(i,\tau)}{\sum_k w_t(k,\tau)}}_{\geq 0} + \underbrace{\rho \cdot \frac{1}{|V|}}_{>0} > 0.$$

Also, the probability for a leap from each node j is > 0:

$$P(\text{leap}|j) = \underbrace{\alpha}_{>0} + \underbrace{(1-\alpha)(1 - \sum_i \mathbf{H}_{i,j})}_{\geq 0} > 0.$$

□

Definition 17. Let $\alpha \in (0, 1]$ be a leap factor, $\mathbf{H}$ be the normalized transition matrix of a given weighted propagation graph and $\mathbf{L}_\tau$ the leap matrix for $\mathbf{H}$ and term τ with random leap factor $\rho \in (0, 1]$. Then the **PEST matrix** $\mathbf{P}_\tau$ is the matrix

$$\mathbf{P}_\tau = (1-\alpha)\mathbf{H} + \mathbf{L}_\tau.$$

Each entry $m_{i,j} \in \mathbf{P}_\tau$ represents the probability of transitioning from node j to node i (as the matrix has been transposed). It can easily be seen that $m_{i,j}$ is determined primarily by two factors: the normalized edge weights of any edge from j to i and the term weight of τ in j.

4.2.4. Term Propagation using Eigenvector Computation

The final step in the PEST algorithm consists in computing the eigenvector. As we need a unique solution, it is important that we arrive at a column-stochastic matrix (where each column sums up to 1) for processing.

Theorem 18. *The PEST matrix $\mathbf{P}_\tau$ for any content graph and term τ is column-stochastic and strictly positive.*

Proof. First, we will prove that $\mathbf{P}_\tau$ is strictly positive. $\mathbf{H}$ is positive (but not necessarily strictly positive) because all edge weights are ≥ 0 and the normalization of the adjacency matrix is a multiplication with a number ≥ 0. Further, $\mathbf{L}_\tau$ is strictly positive because of

Lemma 16 and, thus, $\mathbf{P}_\tau$ is strictly positive. Therefore, there is a non-zero random leap probability from each node to each other node.

$\mathbf{P}_\tau$ is column-stochastic, as for each column j

$$\begin{aligned}
\sum_i (\mathbf{P}_\tau)_{i,j} &= \sum_i ((1-\alpha)\mathbf{H}_{i,j} + (\mathbf{L}_\tau)_{i,j}) \\
&= (1-\alpha)\sum_i \mathbf{H}_{i,j} + \Big(\big(\alpha + (1-\alpha)(1 - \sum_l \mathbf{H}_{l,j})\big) \cdot \\
&\big((1-\rho) \cdot \underbrace{\sum_i l_\tau^{\inf}(i,j)}_{=1} + \rho \underbrace{\sum_i l^{\mathrm{rnd}}(i,j)}_{=1}\big)\Big) \\
&= (1-\alpha)\sum_i \mathbf{H}_{i,j} + (1-\alpha)(1 - \sum_l \mathbf{H}_{l,j}) + \alpha \\
&= 1 - \alpha + \alpha = 1
\end{aligned}$$

□

Then it can be guaranteed that the solution of the eigenvalue problem yields the unique eigenvalue 1 with eigenvector $\mathbf{p}_\tau$ with $\sum_i \mathbf{p}_{\tau,i} = 1$, as proven by Bryan and Leise [14]:

Corollary 19. *The* PEST *matrix* $\mathbf{P}_\tau$ *has eigenvalue* 1 *with unique eigenvector* $\mathbf{p}_\tau$ *for each term* τ*.*

The resulting eigenvector $\mathbf{p}_\tau$ gives the new term weights for τ in the vertices of the content graph after term weight propagation. It can be computed, e.g., using the power method (which is guaranteed to converge due to Theorem 18). The power method is also used in Google's original PageRank algorithm to solve the eigenvalue problem and retrieve the eigenvector. For more details about the PageRank algorithm and the power method, refer to Section 2.2.

Using the entries of eigenvector $\mathbf{p}_\tau$ as document weights (as $\mathbf{p}_\tau$ contains the propagated term weights), we arrive at the result ranking by sorting the documents according to their weight in $\mathbf{p}_\tau$. This ranking can easily be extended to multi-term queries by computing the eigenvectors of all terms and combining them using, e.g., a cosine measure, as it is done in standard vector space models.

Furthermore, if the eigenvectors for all terms are computed, we arrive at the vector space representation of the content graph after term weight propagation, which is a document-term matrix using the propagation vectors $\mathbf{p}_\tau$ for each term τ as columns.

4.3. Structure Propagation with the PEST Matrix: An Example

In order to confirm that the described propagation approach performs as expected, the sample wiki from Figure 1 on page 14 has been analyzed. In a prototype implementation of the PEST algorithm, the resulting vector space representation after term weight propagation was computed which will be described here more detailed.

Edge type	Start node type	End node type	Weight	Reverse weight
link	content item	content item	0.2	0.1
containment	content item	content item	0.5	0.4
link	tag	content item	0.8	0.5

Table 1: Edge weights by edge and node types used for the graph in Figure 2 on page 15

	1	**1.1**	**1.2**	**2**	**2.1**	**3**	**3.1**	**3.2**	**4**	**4.1**	**5**	**5.1**	**6**	**6.1**	$\sum$
1	0.14	0.48	0.48	0.40	0.10	0.18	0.21	0.21	0.25	0.18	0.28	0.24	0.30	0.13	3.65
1.1	0.20	0.00	0.00	0.01	0.19	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.08	0.60
1.2	0.20	0.00	0.00	0.01	0.19	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.08	0.60
2	0.16	0.00	0.00	0.01	0.31	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.00	0.60
2.1	0.00	0.16	0.16	0.20	0.00	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.00	0.64
3	0.00	0.00	0.00	0.01	0.00	0.00	0.32	0.32	0.09	0.00	0.01	0.01	0.08	0.00	0.91
3.1	0.00	0.00	0.00	0.01	0.00	0.20	0.01	0.01	0.01	0.08	0.01	0.01	0.01	0.08	0.49
3.2	0.00	0.00	0.00	0.01	0.00	0.20	0.01	0.01	0.01	0.08	0.01	0.01	0.01	0.08	0.49
4	0.11	0.13	0.13	0.16	0.08	0.18	0.16	0.16	0.19	0.45	0.21	0.18	0.17	0.10	2.45
4.1	0.00	0.00	0.00	0.01	0.00	0.00	0.04	0.04	0.20	0.00	0.01	0.01	0.01	0.00	0.41
5	0.07	0.09	0.09	0.11	0.05	0.09	0.11	0.11	0.13	0.09	0.14	0.43	0.11	0.06	1.76
5.1	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.01	0.01	0.00	0.20	0.01	0.01	0.00	0.33
6	0.04	0.00	0.00	0.01	0.00	0.04	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.31	0.52
6.1	0.00	0.04	0.04	0.01	0.00	0.00	0.04	0.04	0.01	0.00	0.01	0.01	0.20	0.00	0.49
$\sum$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

Table 2: The PEST matrix for the sample wiki from Figure 1 for the query term *"java"* with $\alpha = 0.3$ and $\rho = 0.25$. The last row shows the column sums and the last column the row sums

As parameters for the algorithm, we used a leap factor of $\alpha = 0.3$ and a random leap factor of $\rho = 0.25$. Using these factors, the PEST matrix is computed for each term $\tau \in \{$*"java"*, *"lucene"*, $\ldots\}$. The edge weights are derived by intuition of the authors as shown in Table 1. The resulting PEST matrix for the term *"java"* is shown in Table 2.

Note that the matrix contains high probabilities for propagation to 1 and 4 throughout thanks to the informed leap. This preserves their higher term weight for *"java"* compared to other nodes that do not contain *"java"*.

Using the PEST matrix, we compute for each term the resulting PEST vector $\mathbf{p}_\tau$. Together, these vectors form a new document-term matrix representing the documents and tags in our wiki, but now with propagated term weights, as shown in Table 3.

To verify the veracity of our approach, let us consider a number of desirable properties an approach to approximate matching on a structured knowledge management systems such as KiWi should exhibit:

1. Documents containing a term directly (e.g., *"java"*) with a significant term weight should still be ranked highly after propagation. This should hold to guarantee that direct search results (that would have been returned without approximate matching) are retained.
   Indeed Documents 1, 4, and 5, all containing *"java"*, are ranked highest for that term, though the tags of Document 1 come fairly close. This is desired, as Document 1

	RDF	*XML*	*architecture*	*introduction*	*java*	*lucene*	*search*
1	**0.46**	0.03	**0.11**	**0.11**	**0.26**	**0.08**	0.07
1.1	**0.11**	0.02	0.05	**0.23**	0.07	0.04	0.07
1.2	**0.11**	0.02	**0.24**	0.04	0.07	0.04	0.07
2	**0.10**	0.02	0.05	0.05	0.06	**0.21**	**0.09**
2.1	0.06	0.02	0.06	0.06	0.04	0.06	**0.24**
3	0.02	**0.08**	**0.09**	0.04	0.04	**0.22**	**0.09**
3.1	0.02	0.04	0.03	0.04	0.02	0.06	**0.22**
3.2	0.02	0.04	**0.23**	0.04	0.02	0.06	0.03
4	0.01	**0.53**	0.02	**0.08**	**0.17**	0.02	0.02
4.1	0.01	**0.12**	0.02	**0.22**	0.04	0.02	0.02
5	0.01	0.02	0.01	0.03	**0.11**	**0.11**	0.01
5.1	0.01	0.01	0.01	0.02	0.03	0.03	0.01
6	0.03	0.02	0.03	0.02	0.03	0.03	0.02
6.1	0.03	0.02	0.04	0.03	0.02	0.02	0.03

Table 3: Document-term matrix for the sample wiki from Figure 1 after term weight computation (columns = terms). All values ≥ 0.8 are highlighted as the most relevant matches for each column

contains *"java"* with high term weight and tag-document associations are among the closest relations.

2. A search for a term τ should also yield documents not containing τ but directly connected to ones containing it. Their rank should depend on the weight of τ in the connected document and the type (and thus propagation strength) of the connection. Again, just looking at the results for *"java"*, the two tags of Document 1 as well as the contained Document 2 receive considerable weight for term *"java"*.

3. Searching for documents that contain *"architecture"* and *"introduction"* should also rank documents highly that do not include these terms, but that are tagged with *"architecture"* and *"introduction"*.
   Document 1 is such a case and is indeed the next highest ranked document for such a query after the three documents directly containing *"architecture"* or *"introduction"* (using either boolean or cosine similarity).

This brief evaluation illustrates the effectiveness of the proposed term weight propagation approach for approximate matching and therefore we felt encouraged to show its effectiveness also for larger and more diverse document collections. This will be done in Section 6.

4.4. Normalization Techniques for the Adjacency Matrix

When having a look at the graph structure and its nodes and weights, the edge weights correspond to the reader's opinion about where further matches might be found. Both node and edge weights are encoded in the transition matrix: For each state, the weight of the outgoing edges indicates the transition probability. However, there is one problem to be solved: Outgoing edge weights can add up to a value greater than 1. But it is necessary to transfer the adjacency matrix into a sub-stochastic matrix – where each column sum (i.e., sum of outgoing edge weights) is ≤ 1 – for further processing as described in Section 4.2.3. In some of the situations, the sum of edge weights of one node can be less than 1. This

	Formula	Normalize	by
1	$w'_{i,j} = \frac{1}{m} \cdot w_{i,j}$ with $m = \max_k n_k$	matrix	maximum of n_j
2	$w'_{i,j} = \frac{1}{n_j} \cdot w_{i,j}$	column	n_j
3	$w'_{i,j} = \frac{1}{m} \cdot w_{i,j}$ with $m = \max_i s_i$	matrix	maximum of s_j
4	$w'_{i,j} = \begin{cases} \frac{1}{s_{e,j}} \cdot w_{i,j} & \text{if } s_j > 1 \\ w_{i,j} & \text{otherwise} \end{cases}$	column	by s_j (only if sum > 1)
5	$w'_{i,j} = \begin{cases} \frac{1}{s_j} \cdot w_{i,j} & \text{if } s_j \neq 0 \\ w_{i,j} & \text{otherwise} \end{cases}$	column	by s_j (if possible)

Table 4: Normalization techniques for the adjacency matrix showing which part of the matrix (*"Normalize"*) is normalyzed by what factor (*"by"*). $w_{i,j}$ (resp. $w'_{i,j}$) describes the weight of the edge from node j to node i before (resp. after) normalization (i.e., $w_{i,j}$ is the entry of the matrix at position $[i, j]$), n_i the number of outgoing edges of node i, and s_i the sum of the weight of all outgoing edges of node i

case does not pose a problem for normalization, as the remaining value from 1 is used for the random leap. If the sum is greater than 1, however, we have to normalize at least this column. Simply normalizing all outgoing edges so their weights total 1 will not accurately maintain the differences in weights between different nodes. For example, if two nodes both only had one outgoing connection, their edge weights would always be identical after normalization, even if the actual weights are vastly different.

To remedy this problem, several approaches were examined, as also summarized in Table 4.

1. A graph-wide normalization constant m is determined as the highest number of outgoing edges for one node. All edge weights are then divided by m. Using this type of normalization, the difference in edge weights is preserved, but many weights can get very small if one node has many outgoing edges. Therefore, only a few terms are propagated through the graph and PEST's effect on the result ranking is relatively small.
2. An alternative consists in normalizing the matrix column-wise by the number of outgoing edges for the node representing this column. Then, the difference in edge weights is not preserved between different nodes anymore, but still between edges originating at the same node.
3. As in the first method, a graph-wide normalization constant m is determined, but this time as the highest sum of outgoing edge weights for one node. This leads to an effect similar to the one in method 1, only that here, at least one column is normalized to 1 whereas this is in general not the case for method 1. So the weights are in general higher than in method 1.
4. Another variant is to normalize a column only if it is necessary that it is normalized, i.e., if the sum of its outgoing edge weights is greater than 1, which has the advantage that edge weights are preserved if possible. Problematic about this approach is that nodes which have only a few and light-weighted edges are penalized in that they do

not propagate much of their term weight to other nodes compared to nodes with a edge weight sum of 1.

5. For the sake of completeness, also the simple method that just normalizes the outgoing edge weights of each node to 1 has been analyzed. Nodes without outgoing edges were ignored as $m = 0$ would result in a division through 0.

All of these methods were not only examined theoretically, but also evaluated as part of an experiment in Section 6.1.4. As a brief result, the method that worked best in our tests was method 2 which is therefore used as standard throughout this thesis.

5. Implementation

This chapter presents an overview of the implementation and provides guidance for possible extensions of the program. It further gives insights into the organization of the source code along with interesting aspects of the implementation.

To establish a common understanding about the basic implementation, the technologies that have been used are described. Second, the architecture is examined and its details discussed according to each processing step. Finally, some data structures that are important for the algorithm is examined more closely.

As an introduction into the kinds of data that are to be processed, we briefly describe the domains of the experiments that are carried through in Section 6: The first experiment is conducted on a real-life wiki, a MediaWiki[2], that focuses on data from the Simpsons TV series. The dataset therefore contains wiki pages that contain text as well as links to other pages in the form of XML data. Examples for the single pages are characters, places or episodes of the TV series. The second dataset is a dump of the social bookmarking site Delicious. Here, users bookmark URLs where additionally tags describing the web page at the given URL can be attached. Thus, users, bookmarks, and tags form a linked graph.

5.1. Applied Technology

So let us take a look at the applied technology first. We describe the main technologies we used and why we chose them. Furthermore, we examine the areas of parsing and accessing data and of text analysis more closely.

5.1.1. Java as Programming Language

As programming language, Java 5 was used for several reasons: First, it offers easy access to many interfaces like MySQL, XML, and Lucene and further allows a relatively easy implementation of a graphical user interface which is used for quickly checking experiment results. In addition, it is cross-platform compatible and is integrated into development environments like Eclipse which are freely available. The downside of Java is that the size and implementation of data structures cannot be controlled down to bits and bytes. Furthermore, some benchmarks suggest that programming languages that have more control over data structures, e.g., C or C++, are more efficient when used correctly. However, Prechelt [37] shows that the difference in performance between the implementations of different programmers is much higher than between different programming languages like Java and C++ and thus the human factor plays a higher role than the chosen programming language. Because of this fact and as runtime optimization is not the focus of this thesis, we are confident in having chosen the right language.

[2]MediaWiki is a free wiki software which is used by Wikipedia and many other wikis, see `http://www.mediawiki.org/`

5.1.2. Parsing and Accessing the Data

Several Java technologies are used for easier access of data. First, to parse our custom input files that follow the DOT-file definition [2], we used the parser generator ANTLR [1] which will be described in Section 5.3.1.

For accessing XML dumps of, e.g., the Simpsons wiki (see Section 6.1.1), a SAX[3] parser is used. Here, the XML data is read as a continuous stream and because we do not require seeking to various positions in the data stream, this linear form of parsing is suited for our purpose. This is in contrast to a DOM[4] parser where the structure of a document tree is built in memory which can be navigated through. But as we do not need this feature, the use of the SAX API is the more efficient choice.

To store data in a database, we used the JDBC[5] technology and focused on using a MySQL database. It would not pose a problem to use a different database, e.g., the program has already been adapted for another purpose to use the PostgreSQL database. Only the database driver has to be exchanged and perhaps some minor changes to some SQL statements have to be made for compatibility reasons. For simplicity reasons, we restricted ourselves to the freely available and efficient MySQL database. More details on the storage of data in the database will be given in Section 5.3.2.

5.1.3. Text Analysis with Lucene

Not only data access was simplified by using Java technologies, also the complexity of text analysis could be reduced by using the Lucene libraries [3, 22]. Lucene is originally a search engine based on the vector space model (see also Section 2.3), but as it contains all necessary parts for text analysis and processing, it is advisable to use it. Two parts were included into our implementation: A *tokenizer* splits the input text, e.g., a wiki page, at specific symbols like blanks, commas, brackets, etc. into a stream of tokens, usually a list of words. Then, *filters* are used to modify this list and drop some tokens or replace them by other content.

The filters are the more interesting part, as they can be used to substantially decrease the number of terms that are used in the PEST algorithm while at the same time improving the result relevance because of more decisive terms. For example, when filtering the tokens of wiki pages from the Simpsons wiki, we applied five filters:

- A lower case filter modifies each token to contain only lower case characters by transforming each upper case character into its lower case equivalent,
- an ASCII folding filter replaces non-ASCII characters with their ASCII equivalent (if they have one),
- an ASCII filter removes all remaining characters that could not be transcribed using the previous filter,
- a stopword filter removes all tokens which are contained in a predefined list of words

[3] **S**imple **A**PI for **X**ML provides an interface to parse XML *streams*

[4] **D**ocument **O**bject **M**odel represents an XML document as a *tree structure*

[5] **J**ava **D**ata**b**ase **C**onnectivity is a standardized interface for accessing *SQL databases* which many existing databases implement

Package name	Description	Section
parser	Preprocessing and parsing of data sources	5.2.1
matching	Implementation of the PEST algorithm	5.2.2
ranking	Comparison of result rankings	5.2.3
ui	Presentation of results	5.2.4
model	Data structures	5.3
utils	Helper classes for common tasks	–

Table 5: Overview of the Java packages with a short description

(usually function words, i.e., words with little lexical meaning like "the" or "and"),

- and a stemmer filter reduces all tokens into their normalized form (e.g., "transformed" ⇒ "transform", "transformation" ⇒ "transform", etc.).

This filtering chain ensures that the number of terms is reduced to a minimum without throwing away important information.

5.2. Overview of the PEST Architecture

To give a brief overview of the implementation, the software components are depicted in Table 5. All preprocessing of data takes place in the Java classes of the *parser* package, the *matching* package contains the main routines for the PEST algorithm, and the classes in the *ranking* package postprocess the results and create a comparison of different rankings. The *ui* package contains classes for the graphical user interface, the data structures which are described in Section 5.3 are located in the *model* package, and in *utils* additional common helper classes are situated. These parts will be discussed in more detail now.

5.2.1. Preprocessing and Parsing of Data Sources

Before being able to run the PEST algorithm on some specific data, the researcher has to parse the data and transform it into a format which is readable by this PEST implementation. Considerations about which data to use for what kind of data item in PEST have to be made carefully in the way it will been described in Section 6.3.1. Furthermore, an appropriate input format has to be chosen as there are two possibilities: a DOT-file format which is suited especially for testing smaller examples and a database-stored format which offers an improved scalability. These two formats are described in more detail in Section 5.3.

By means of the example of the Simpsons wiki, we will describe the process of parsing a data source and converting it into a database-stored format. The data source for our parser is an XML dump of the wiki. Note that it can be any MediaWiki XML dump, which makes this parser applicable also for other experiments using MediaWikis. In Figure 6, a UML diagram depicts the main classes for parsing together with the connections between them. The *WikiDumpParser* is the main class controlling the whole parsing process. After initializing the database and creating the necessary tables, two preprocessing steps and two page processing steps involved in parsing the XML dump of the wiki are triggered.

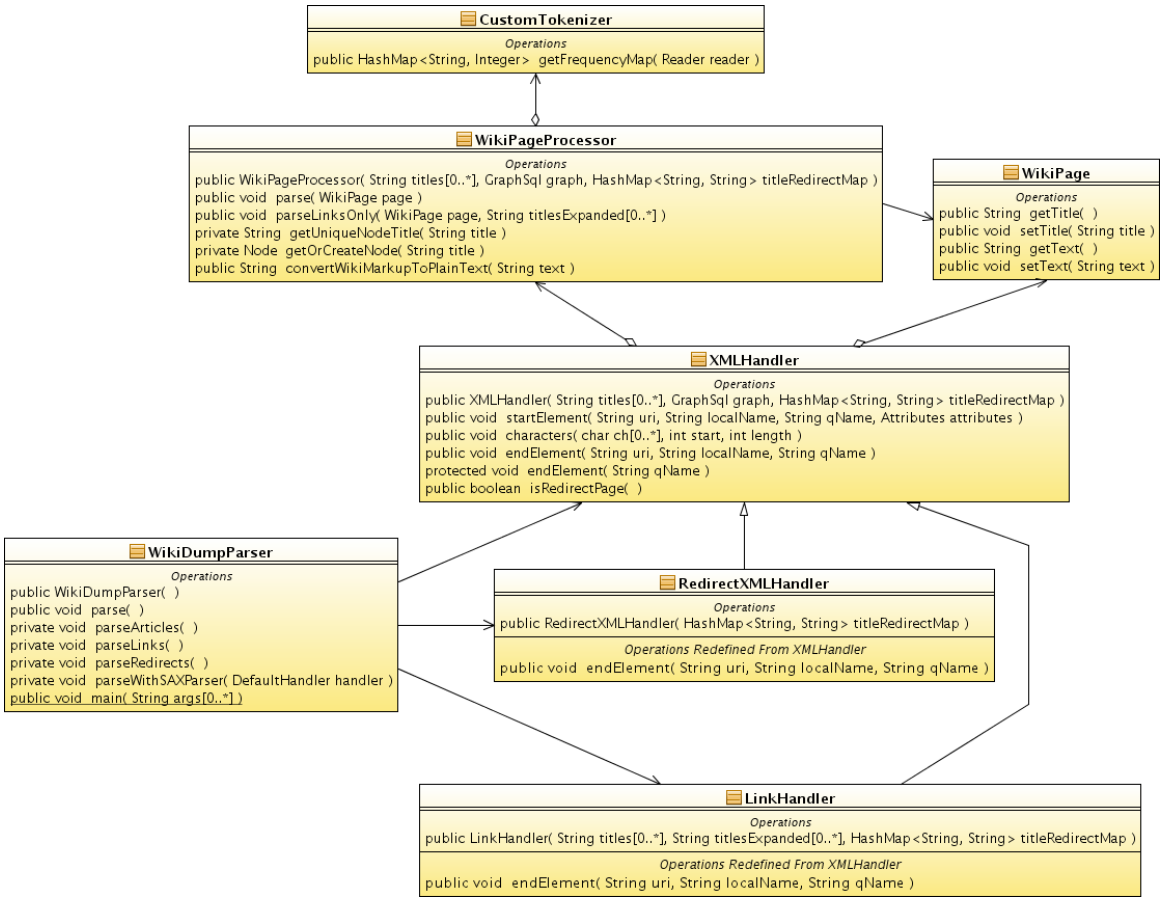


Figure 6: UML class diagram of the main classes responsible for parsing a MediaWiki dump

Preprocessing In the first preprocessing step, pages that have no content but directly link to another page (e.g., "Bart" → "Bart Simpson"), so-called "redirects", are read using the *RedirectXMLHandler*. This is necessary at the beginning in order to prevent the duplication of wiki pages and thus to create a list of unique pages. The *RedirectXMLHandler* class is a subclass of *XMLHandler* and only differs in the action that is executed. The *XMLHandler* class uses a SAX parser to extract single wiki pages from the XML stream into *WikiPage* objects and executes the method *endElement(...)* after each page is completely read. The *RedirectXMLHandler* overwrites this method: If a redirect page is found, then the page's title is inserted into a hash table together with the destination page title for fast look-up during the next steps. Furthermore, we implemented a caching method which saves the hash table to a file on the hard disk and loads it when the parsing process is started another time. This saves time especially when experimenting with different settings during the last step of the parsing process.

The second step is necessary when only a part of the wiki should be parsed. If a list of wiki page titles is given, it is possible to expand the list to also include pages that are linked by pages in the original list. Then, the *LinkHandler* parses the given pages and adds links to new pages, which were previously not included, to the page list. Note that the hash table generated in the previous step is used to resolve redirect pages to their original page title.

Page Processing More important for PEST than the preprocessing however, is the real parsing of the wiki pages. This is done by the *XMLHandler* which calls the *parse(WikiPage page)* method of the *WikiPageProcessor* and furthermore creates a node in the graph for each real, i.e., non-redirect, wiki page.

Afterwards, the wiki markup text is converted into plain text using several regular expression substitutions and fed into the *CustomTokenizer*'s *getFrequencyMap(Reader reader)* method. This is a sophisticated method which receives a text as input and returns a list of terms with their respective number of occurrences. This process, which uses parts of the Lucene project, has already been discussed in Section 5.1.3 and forms the first core part of the parsing process, i.e., the term frequency creation. These term frequencies are stored in the database for further use by the PEST algorithm.

The second core part consists of parsing the outgoing links of each wiki page and thus resembles the creation of edges between nodes in the graph. This is done by using a regular expression adapted to matching links in the wiki markup text. Also, the redirect hash table from the first step is used here to resolve recursive links. The links are saved in the database as edges between page nodes.

Altogether, a complete graph structure together with a term frequency list for each node is computed during the described processing steps. Though representing a complex application flow, breaking the chain down into smaller steps helps getting a better understanding and encourages to try this on one's own data.

5.2.2. Implementation of the PEST Algorithm

The heart of the software is the implementation of the PEST algorithm, which is mainly situated in the *PEST* class with some help from the *MatrixConstruction* classes. In this section, we will describe which steps are executed and furthermore where the computation takes place.

The first thing that is done upon running the PEST algorithm is the **verification of query terms**. For each term in the given query term list, it is checked whether the term is included in the set of terms in the data structure. If it is not, the term is fed into the *CustomTokenizer* class whereupon the returned term, if existing in the set of terms, is used as query term instead of the original term. After doing this for each query term, one arrives at a revised term list which is then used for computation.

Following the algorithm outlined in Section 4.2 and depicted in Figure 5, the **adjacency matrix** is computed. This task is done by either the *MatrixConstructionSql* or the *MatrixConstructionImpl* class as the generation of the matrix very much depends on the type of data source used. The *Sql* class uses the database backend whereas the *Impl* class uses the internal representation from which the DOT-file is parsed. Furthermore, during the matrix construction it makes a difference whether the weights are set using standard values based on the type of connection, e.g., an edge "CI"→"TAG" of type "TAGGING" gets weight 0.6, or defined on a per-edge basis in the data source. Either the method *getTransitionMatrixWithStandardWeights* (former) or *getTransition* (latter) is called. When using standard weights, the weights are read from a file which by default is `default.settings` and can be overriden using the `-s` command line switch. The construction of the matrix itself is pretty straightforward as it just iterates through the nodes' edges and builds the matrix according to the weights or types of these edges. The type of matrix which is created is determined by the *MatrixFactory* class and can be a *MemoryArrayMatrix*, a *MemoryMappedFileMatrix*, or a *FileMatrix* where the memory representation is the preferred one if enough memory is available. These different storage methods of the matrix will be further discussed in Section 5.3.

Now that we have generated the adjacency matrix, the matrix is **normalized**, which takes place in *PEST*'s *initializeMatrix()* method. As described in Section 4.4, several techniques can be applied. These were tested in this method and are still available in the source code for further tests.

Until here, the computation is the same for all terms. But now, one after the other, the PEST vector for each term is computed. Therefore, the following steps are executed once for each term.

The **computation of the PEST matrix** is done in the *compute(...)* method of the *PEST* class. Before modifying the matrix, the normalized adjacency matrix is cloned so that it does not have to be computed again for each term. Furthermore, the frequency distribution over the nodes for the current term is read from the data source, normalized to 1 and saved in a vector which helps to create the PEST matrix and, moreover, provides a good starting vector for the power method in the next step. Now, the PEST matrix is computed in the way defined by the parameters. As we tried different techniques for normalizing, several parameters can be tuned: First, one can choose to use a self loop algorithm which

allocates the remaining amount for each column to the own node (each column of the matrix represents the outgoing edges of a node and thus this allocation is an edge to the own node) instead of distributing it according to the term frequency vector (as we suggest it in our algorithm which has been proven better). Moreover, the *randomLeapFactor* α and the *uniformDistributionFactor* ρ parameters, which have already been described in Section 4.2, are used here for the PEST matrix computation. For this computation, each entry of the matrix is examined column-wise and transformed into a new value so that the resulting matrix is column-stochastic.

Now that we have the PEST matrix, the **eigenvector computation** is the final step in the algorithm. This is done using the power method which iteratively computes a new result vector that gets closer and closer to the real eigenvector. When the difference between the norm of the last vector and the norm of the new vector is below a defined threshold, the iteration is stopped and the current vector represents the PEST vector $\mathbf{p}_\tau$ which is saved for the current term. After that, the computation of the PEST matrix for the next term is started.

5.2.3. Comparison of Result Rankings

Although the computation of the PEST vector $\mathbf{p}_\tau$ is the most important part of the software, the comparison of the result ranking with a base ranking is not to be disregarded. The requirements for such a comparison are:

- It should allow a quick review of the results when experimenting with different queries and parameters.
- The changes in the ranking should be easily visible.

We chose to display the ranking in a table with five columns: PEST ranking position, PEST score, node title, original ranking position, and change in position. The last column displays either the change in position, e.g., "+3", or "NEW" if the node is not contained in the original ranking or nothing if there is no change. Therefore, the user can easily recognize how much the ranking was changed by the PEST algorithm.

From the implementation side, the classes in the *ranking* package are responsible for the output. Objects that are subclasses of the *Ranking* class can be fed into the *RankingComparison* class which calculates the table described above. For example, the *DefaultRanking*, i.e., the PEST ranking, and the *OriginalRanking*, i.e., a pure tf ranking, can be compared which results in a table like those shown in Appendices A.3 and A.4. The actual calculation of the position changes is as simple as searching the position of the node in the original ranking and determining the change for each position in the new ranking. No further optimization has been done as this calculation is only performed in experiments and thus does not influence the actual runtime. The *Ranking* and *RankingComparison* classes can return the result table in two ways: Either they output it as plain text in a table or they provide a *TableModel* which can be used by the graphical user interface to display the table directly in a graphical component.

One further aspect of the rankings exists in the *Adjustment* class. For the wiki experiments of Section 6.1, the pure tf ranking yielded very poor results and thus it had to be adjusted

to perform better. Therefore, we looked at the implementation of the search in MediaWiki and built a similar ranking enhancement by using the function

$$score = orig\_score \cdot length\_norm \cdot title\_boost \cdot link\_boost$$

where

$$length\_norm = \begin{cases} 1.0005 - 0.0005 \cdot term\_count & \text{if } term\_count \leq 1000 \\ 0.5 & \text{otherwise} \end{cases}$$

$$title\_boost = \begin{cases} 3.0 & \text{if page title contains query term} \\ 1.0 & \text{otherwise} \end{cases}$$

$$link\_boost = \frac{1 + incoming\_edge\_count}{15.0}$$

This modification of the ranking can be turned on by passing an instance of the *Adjustment* class to the constructor of the relevant *Ranking* class. Therefore, we only need one class for modifying the scores of both rankings and furthermore enable the easy creation of other ranking enhancements for which it suffices to create a subclass of the *Adjustment* class.

5.2.4. Presentation of Results

Once the comparison has been computed, it has to be presented in some way. There are two ways to interact with the PEST software: either through passing command line parameters to the program or by using a graphical user interface (GUI). The parameters are clearly documented when adding `-h` to the command line. This documentation is also shown in Appendix A.1.

The GUI (Figure 7) is split into three main parts. On the top, the query terms can be entered, the type of ranking chosen, and the PEST algorithm can be started to either solely show the ranking ("Search") or compare the ranking with another ranking ("Compare"). The middle part is filled with tabs for each result table. In each table, the data can be sorted by clicking on a column header. On the bottom, we placed a status bar which displays log messages. For example, it shows the current status of the PEST calculation, and it also provides more information about one specific result item when the user clicks on it, i.e., the detailed calculation of the adjustment if it has been enabled.

From the implementation perspective, the GUI on the one hand manages the PEST algorithm part by calling the methods inherited by the *CommandLine* class. On the other hand, it creates the graphical components and provides appropriate user feedback. First and foremost, the ranking result is displayed using a *JTable* component and feeding it with a *TableModel* created by one of the ranking classes, as described in the previous section. By using the Java-provided GUI classes, the design could be kept clean and the implementation complexity reduced to a minimum. The log messages of the status bar are displayed using a customized *Handler* and using the log methods of the *java.util.logging* package. Thereby,

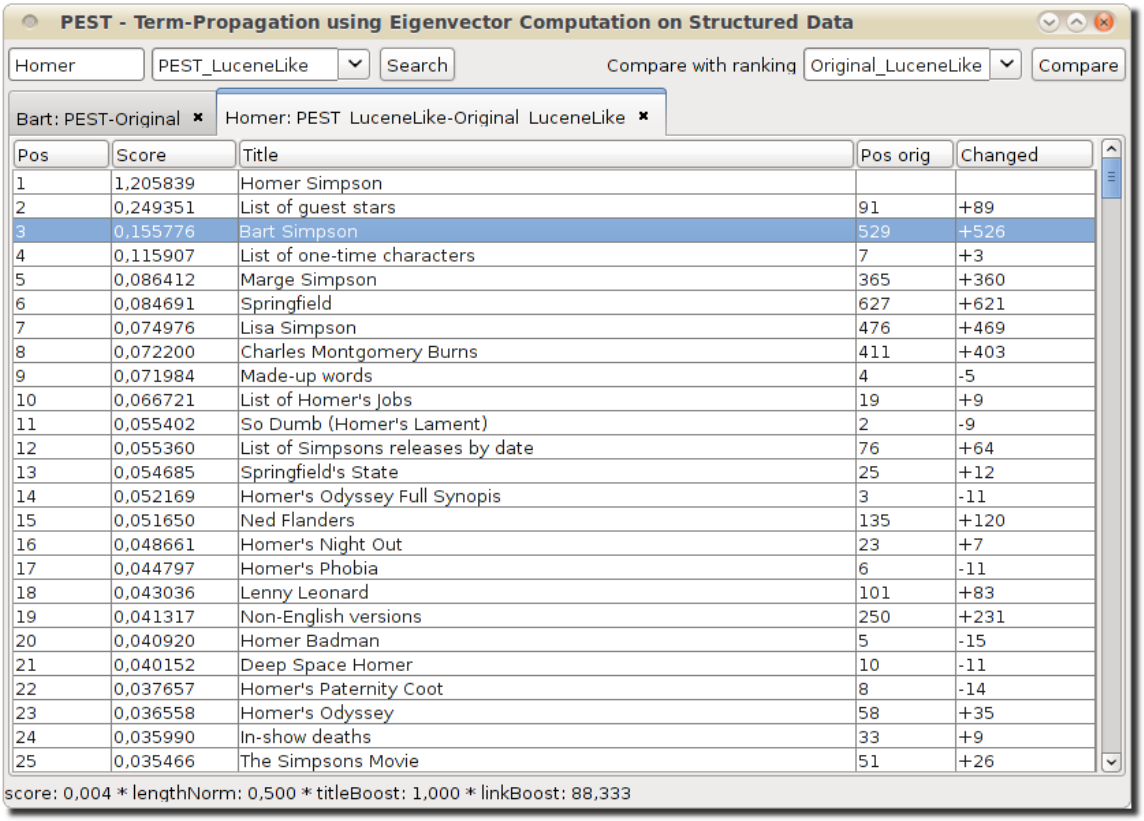


Figure 7: PEST GUI with ranking details for the wiki page “Bart Simpson”

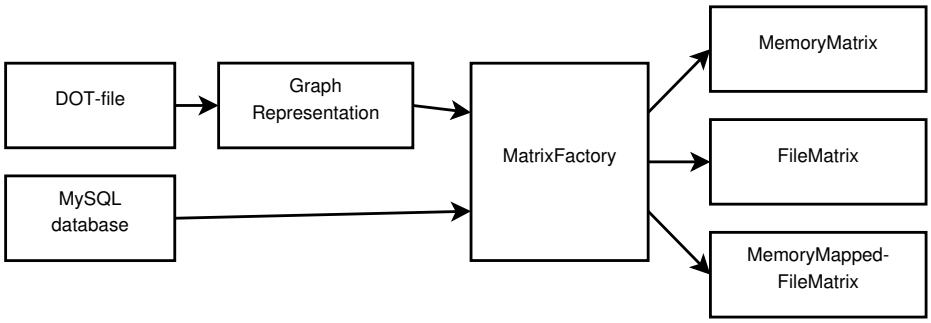


Figure 8: Usage of data structures during the different steps

two functions are achieved at once without duplicate implementation: a clean way of logging events on the command line and graphical feedback to the user via the status bar.

The software implementation follows the steps of the PEST algorithm very closely with only few divergences. This makes sense because a good code structure eases the understandability for readers of the source code and following a clearly defined algorithm in this way helps to achieve this goal.

5.3. Data Structures

Besides showing up the software structure as we have done in the previous section, we will highlight some data structures that have a great impact on the software and are worth a further investigation. Therefore, we follow the stream of events as depicted in Figure 8: First, we will have a look at the DOT-file like input format and its representation of the graph, continue with the MySQL format and finally look at the different ways the matrix can be buffered.

5.3.1. DOT Language Input Format

The simple input format is a file written in a language similar to the DOT language. DOT is used to describe graphs that are easily readable for humans as well as computer programs. It is part of the Graphviz package where the DOT language is defined[6]. Here is an example of a DOT-like file used in PEST:

```
digraph sample {
    doc    [type="ci", terms=((java, 0.7), (search, 0.3))];
    luceT  [type="tag", terms=((lucene, 1.0))];

    doc -> luceT [type="linking", weight=0.3];
}
```

A node is created by declaring its name, e.g., *doc*, and appending options in square brackets such as the type or a list of terms together with their frequencies. Directed edges are created

[6] http://www.graphviz.org/doc/info/lang.html

through the `doc -> luceT` mechanism, where a type and weight can be given as options. Therefore, the example above describes a graph with two nodes *doc* and *luceT* and an edge from *doc* to *luceT* together with the respective options. Compared to the DOT language, the only incompatibility consists in the use of the `terms=(...)` option, as no option is allowed where values are grouped by brackets. However, we cannot use quotation marks instead of brackets as also a term *like "DOT, language"* is a possible valid term containing a comma and a blank and its implementation would on the one hand confuse humans when reading or writing the file and on the other hand require more effort for parsing the file correctly. Therefore, the language was modified in order to support readability for humans.

The parsing of this input format is implemented using the ANTLR tool. It provides a framework designed to create parsers from grammatical descriptions in several target languages. The idea is to write abstract grammar rules which are translated into source code of the target language, in our case Java. The rules on the one hand define the syntax of the file and on the other hand the actions that are executed, e.g., snippets of Java code. Thus, it is possiple to parse the file at the same time as generating the graph representation of it. The ANTLR file for this grammar consists of only 139 lines of code and includes the methods for creating the graph representation whereas the two output files generated by ANTLR have 1530 lines of code. So it can be seen that this tool provides a good choice for writing grammars like this. For a graphical description of the grammar rules, refer to Appendix A.2.

From the DOT-file, its corresponding graph is constructed. As we have not yet defined it in terms of its implementation, we will do this now. Naturally, a graph consists of a list of nodes which is also how we implemented it in Java: The *GraphImpl* object contains an *ArrayList* of *NodeImpl* objects. Further, each *NodeImpl* contains a HashMap for the edges where the destination *NodeImpl* acts as key pointing to an *EdgeImpl* object as value. In this *EdgeImpl* object, the type as well as the weight of the edge is saved. For storing the term frequencies, a second HashMap is created which contains each term together with its frequency. Each class contains methods for easier access of the single data items and thus a simple, yet powerful in-memory representation of the graph is produced.

5.3.2. MySQL as Input Source

The other choice to store the data for further processing through PEST is a MySQL database. The class that manages the database access is *GraphSql*. Here, a method exists that creates all four tables of the database: nodes, edges, terms, and termcounts. These table definitions together with their attributes are depicted in Figure 9.

It is worth noting that the *type* attribute of nodes is unique and therefore no two nodes with the same title can exist. The same applies for terms having a unique *term* attribute. Also, there can only be one edge of a given type between two nodes for a matter of simplicity which, however, does not pose any restrictions. The weight of an edge can be *NULL*, then the weight is either 0 or – if defined – set using standard weights by the type of connection as described in Sections 4.2.1 and 5.2.2. The termcounts table defines the term frequencies for the nodes. For experiment purposes, the *count* attribute contains the number of occurrences of the given term in the defined node. Whereas it is not required for the PEST algorithm itself,

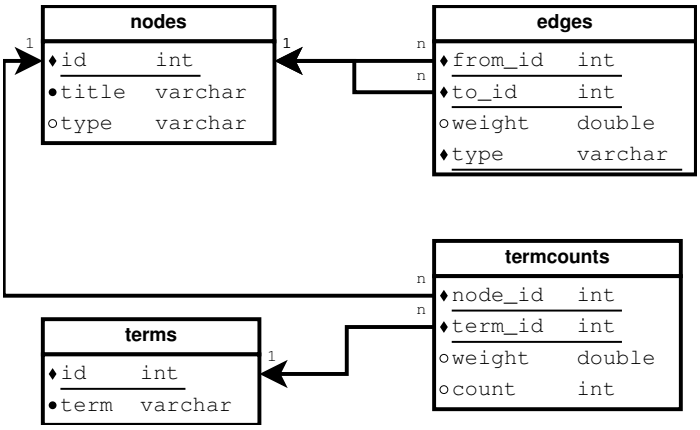


Figure 9: Tables for the graph structure as stored in the database. Underlined attributes depict primary keys, a filled diamond a unique and not-*NULL* attribute, and a filled circle a not-*NULL* attribute

it has proven useful for checking the correctness of the term frequency calculations. From the count attribute, the *weight* attribute is calculated using the *normalizeTermFrequency(int nodeId)* method in *GraphSql*.

5.3.3. In-Memory Matrix

The two main methods how to store the matrix are in-memory and in-file. First, we will look at the in-memory representation: Here, the matrix is most easily stored as a two-dimensional array in which the values, usually double precision values, reside. This has the advantages that it is easy to implement and has a great performance and the disadvantages that it uses a large amount of memory and also uses space for empty entries. Another method would be to use a representation where only entries of the matrix that contain values are stored. Thereby, the used memory becomes smaller for sparse matrices, but in contrast, it requires more effort in organizing these entries which results in a more complex implementation and possibly a reduced performance (this depends on the sparseness of the matrix). To give an example, a quadtree offers these properties: It recursively splits the matrix into four sub-matrices and saves space for sparse matrices as splits are only performed if at least one element of the sub-matrix is not empty [5]. Another option, which can be combined with any of the two mentioned methods, is to compress the data additionally and thereby save memory space.

Thus, the question is how sparse our matrix will be. Up to the normalized adjacency matrix, the matrix usually is sparse. Therefore, a representation storing only used entries is a good option. However, the PEST matrix stores a value in every entry and thus the matrix is not sparse anymore. Hence, the sparse matrix representation does not make sense as the overhead in organization is bigger than the gain in memory space. Thus and for facts of simplicity, the in-memory matrix is stored as a two-dimensional array.

5.3.4. In-File Matrix

As for storing the matrix on the hard disk, the problem can be divided into two separate questions: How and in which order to store the entries of the matrix and how to create an efficient data access model. For storing the entries, a simple method is to store them linearly row by row (or column by column), so that entry $a_{i,j}$ of an $m \times n$ matrix is stored at position $n \cdot i + j$ (respectively $i + n \cdot j$). This is also the method we chose for our in-file matrix implementation. There are plenty of other storage possibilities, for example the quadtree method described in the previous section. In this thesis though, these methods have not been examined for suitability for the PEST algorithm.

Also for the second question on an efficient data access model, plenty of different methods exist where some are intertwined with the data storage model. Here, some caching methods will be regarded. The simplest one is to read each value when it is needed and to not perform any caching. This is how it is done in the *FileMatrix* implementation. Though not having any overhead for reading data that is not used, the magnitude of tiny data accesses is not the best option for reading from hard disks. Therefore, an adapted version of this technique has been implemented in the *MemoryMappedFileMatrix*. Here, a certain number of rows of the matrix is read at once and cached for faster reading and writing of data. The overhead for managing the cached rows is small and probably outweights the disadvantage of perhaps reading too much data. No extensive benchmarking has been done, but from brief experiments it could be seen that this method performs better than the *FileMatrix* implementation. Nevertheless, also this method is subject to further improvements which exceed the scope of this thesis.

To sum up, the methods implemented as part of this thesis are designed to demonstrate that PEST is also suited for larged datasets.

6. Experiment Results

In order to confirm that the described propagation approach performs as expected, a prototype implementation of the PEST matrix construction has been implemented and experiments computing the resulting vector space representation after term weight propagation have been conducted. Much effort was put into these experiments with different datasets and the two applications that were most promising will be discussed in this chapter. Therefore, we will first show the results of a wiki, namely the Simpsons wiki, and afterwards regard a social bookmarking site, Delicious. Furthermore, we will describe properties that are PEST inherent to put it into a wider application area.

6.1. Simpsons Wiki

As an example we chose a real world wiki, the Simpsons wiki[7], which is self-contained, focused on a specific topic and at the same time small enough to easily judge which pages are relevant for a certain query and to perform a large number of experiments. At the time of this writing, it includes 10,955 pages (without redirects, talk and special pages).

6.1.1. Experiment: Setup and Parameters

Wiki pages are first stripped of any markup annotations. The resulting text is normalized, stopword filtered using the list from `ranks.nl`[8] and stemmed using the English snowball filter[9]. This process of text analysis is also covered in Section 5.1.3. The resulting wiki page collection contains 22,407 terms.

To keep the example simple and to allow for reliable human relevance judgements, we use only one type of edge, namely linking between pages. For computing the adjacency matrix $\mathbf{H}$, the weight of a link in outgoing direction is set to 0.2, its reverse weight to 0.1. As parameters for computing the PEST matrix, we use a leap factor of $\alpha = 0.15$ and a random leap factor of $\rho = 0.25$.

To judge the relative effectiveness of PEST, we compare with three ranking schemes:

- Luc: The first ranking is a basic tf-idf ranking with cosine similarity. It is similar to the default scoring used by Lucene[10]. Note that the idf value before propagation is used, since after propagation every term is contained in every wiki page (though mostly with a very low weight).
- Wik: The second ranking is based on the ranking algorithm used in MediaWiki (and thereby on Wikipedia)[11]: It compensates for the differing length of wiki pages by gradually decreasing the document score with increasing document length. Further, each page is boosted on the basis of its number of incoming links. Finally, if the title

[7] `http://simpsons.wikia.com`
[8] `http://www.ranks.nl/resources/stopwords.html`
[9] `http://snowball.tartarus.org/algorithms/english/stemmer.html`
[10] `http://lucene.apache.org/java/3_0_1/api/all/org/apache/lucene/search/Similarity.html`
[11] `http://www.mediawiki.org/wiki/Extension:Lucene-search`

	PEST score	Page title	Wik		Goo	
1	1.273	Bart Simpson	2	+1	1	0
2	0.222	List of guest stars	95	+93	-	*new*
3	0.185	Homer Simpson	590	+587	235	+232
4	0.121	Lisa Simpson	186	+182	202	+198
5	0.117	Bart Gets an F	7	+2	2	-3
6	0.112	Bart Simpson (comic book series)	3	-3	45	-39
7	0.101	List of one-time characters	29	+22	555	+551
8	0.092	Bart the General	4	-4	38	+30
9	0.089	List of Bart Episodes in The Simpsons	1	-8	115	+106
10	0.079	List of Simpsons releases by date	45	+35	-	*new*
11	0.078	Springfield	1112	+1101	-	*new*
12	0.070	Marge Simpson	1116	+1104	-	*new*
13	0.064	Made-up words	18	+5	278	+265
14	0.056	The Bart Book	5	-9	83	+69
15	0.049	Bart Sells His Soul	15	0	3	-12
16	0.048	Bart Gets Hit by a Car/Full Synopsis	6	-10	10	-6
17	0.048	Bart vs. Thanksgiving	9	-8	14	-3
18	0.048	Bart vs. Lisa vs. the Third Grade	8	-10	16	-2
19	0.048	Charles Montgomery Burns	1113	+1094	299	+280
20	0.046	Radio Bart	14	-6	12	-8

Table 6: Top 20 Wik+PEST ranking for query *"Bart"* ("-": page is not returned as answer at all) including the Wik and Goo ranks together with their positional changes compared to PEST

of the wiki page contains a query keyword, this page is boosted by a certain factor, in our case by 3. For details on this computation, refer to Section 5.2.3.

- Goo: Where possible, we also compare with the ranking returned by a Google query restricted to the Simpsons wiki, i.e., containing `site:simpsons.wikia.com`.

We denote with Luc+PEST the ranking obtained by applying PEST to the wiki and using the basic tf-idf ranking on the modified vector space model created by PEST. Analog for Wik+PEST, we use a PEST ranking modified by the ranking adjustment algorithm from MediaWiki. For Google, we also compare with Wik+PEST, as the ranking algorithm used by Google is not openly available.

6.1.2. Comparing PEST

In the following, we use two queries for comparing the ranking produced by PEST with those discussed above: First, we look at single word queries, e.g., *"Bart"*. Second *"Moe beer"* is used to illustrate the effect of PEST in the presence of queries consisting of multiple keywords.

Table 6 shows the top 20 answers for the single keyword query *"Bart"* using PEST with MediaWiki-style term weights (Wik+PEST ranking). In the last two columns, we compare

that ranking with the one returned by MediaWiki without PEST (Wik ranking) and with the ranking returned by Google (Goo ranking). Further experiment results can be found in Appendix A.3.

There are a number of striking observations in this comparison:

1. Both the unmodified Wik and Goo ranking do not return any of Bart's family members (Homer, Lisa, Marge) or Bart's hometown "Springfield" as highly relevant for the query *"Bart"*. This is clearly due to the fact that these pages infrequently contain the term *"Bart"*. In contrast, PEST returns all of these pages for highly related characters or locations among the top 20 matches for the query *"Bart"*. The significant difference also to the Goo ranking shows that PageRank alone can not attribute for the improvements demonstrated by PEST.
2. This effect is particularly noticeable for "Marge" and "Springfield", which occur at a rank below 1000 for Wik and do not occur at all in the Goo ranking.
3. Some hub-like lists of links like "List of guest stars" are further promoted by PEST. This is in many cases a desirable behavior (e.g., "List of Bart Episodes" is clearly relevant). In case of the query *"Bart"* however, these pages are not only highly connected but also contain the term *"Bart"* very often as it occurs in many episode titles. For other document collections, this effect is not as prominent.

Applying PEST to a basic tf-idf ranking such as Luc yields even greater improvements as shown in Appendix A.3.2, where we compare the ranking of the top 20 pages using PEST with Luc term weights to Luc ranking without PEST. The "Bart Simpson" page gets pushed from position 325 to pole position and Bart's direct relatives on the positions following shortly behind. This demonstrates that the PEST algorithm is capable of achieving significant improvements if applied to a range of existing ranking schemes.

For the the multi-term query "*Moe beer*", the application of PEST also significantly improves the ranking as shown in Appendix A.3.3. Here, we start with the Wik ranking. For example, "Homer Simpson", a frequent visitor of "Moe's Tavern" and a big consumer of beer, is ranked 6^{th} compared to rank 108 without PEST. For completeness, the relevance measures are given in Table 7.

In addition to the top 20 ranking, we also considered the top 100 answers for each of the above rankings for many queries including *"Bart"* and *"Moe beer"*. The main results of this comparison are summarized in Table 7:

The number of relevant pages that are introduced by PEST into the top 100 answers is significantly high in each case. At the same time, most of the pages that are dropped from the top 100 are irrelevant and at worst a relevant page is dropped by about 50 ranks. For *"Bart"*, nearly no irrelevant pages are introduced, for *"Moe beer"*, only a few irrelevant pages are added to the top 100 answers, but without affecting relevant pages significantly.

All previous top 100 pages are still included in the first 140 results in the PEST ranking and the lowest position not included in the new top 100 is the former position 65. Thus, only few relevant pages are discarded or moved further to the end of the ranking than necessary.

	"Bart"		*"Moe beer"*	
	Wik rank	**Luc rank**	**Wik rank**	**Luc rank**
Relevant	22	17	20	20
Irrelevant	4	2	7	9

Table 7: Number of relevant and irrelevant pages in the top 100 ranking added by PEST compared to the Wik and Luc ranking

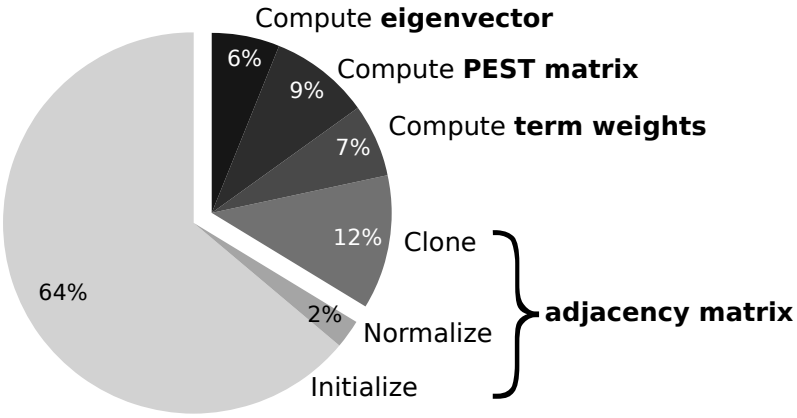


Figure 10: Percentual processing time for indexing a single term

6.1.3. Run Time Performance Evaluation and Benchmarks

The comparative evaluation of the quality of the rankings with and without PEST shows the significant improvements PEST can contribute to keyword search. However, the downside is that these improvements decrease the run time performance. We will show in the following that the algorithm is capable of calculating the index in an adequate amount of time.

To quantify the performance of PEST, we ran a large number of keyword queries on a Intel Core 2 Duo E8400 with 8GB Ram running Sun Java 6 on a 32-bit installation of Ubuntu 9.10. The structure of the data as well as the terms are stored in a MySQL database. The algorithm is not parallelized and runs entirely on a single core. As a dataset, the Simpsons wiki with 10,955 pages and 22,407 terms is used, as discussed above. We do not use any partitioning or segmentation techniques, but hold all matrices in memory at once, using about 2 GB of main memory.

Once the modified vector space index computed by PEST is created, the query time depends only on the used information retrieval engine. Therefore, we focus here on the time PEST spends for indexing a given structured dataset.

Unsurprisingly, the indexing time for PEST scales like that of PageRank in the number of pages, i.e., linearly, as shown in Figure 11.

Figure 10 shows the percentages of each of the steps needed for indexing a single term with PEST: The term independent part, i.e., the initialization and normalization of the transposed, normalized adjacency matrix $\mathbf{H}$ for the underlying weighted propagation graph, takes on average 16 s, about 66% of the processing time of a single term. If several terms

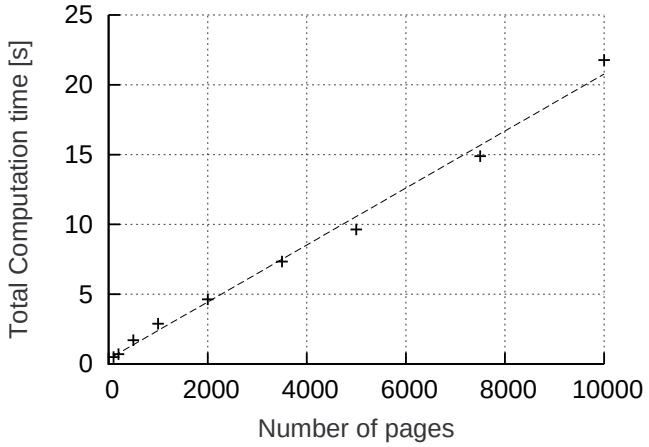


Figure 11: Indexing time over dataset size

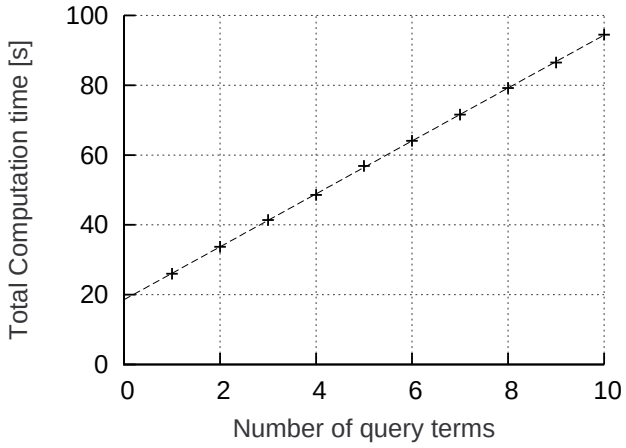


Figure 12: Indexing time over number of indexed terms

are indexed at once, this part needs to be executed once for all terms only. For each term, we need to create a copy of **H**, compute the term weights (here using Wikipedia-style term weights), combine the copied matrix **H** with the resulting leap matrix and compute the eigenvector of the resulting PEST matrix. Overall, this part takes on average 8 *s* and thus about 33% of the time for processing a single term.

It is worth emphasizing this result: Only about 8 *s* are needed to process each term. Furthermore, we can compute the PEST matrix for each term independently, even on different cores or computers. Figure 12 further emphasizes this result: If we increase the number of index terms, the total computation time on a single core increases, but only linearly with small constants. Figure 13 shows further that the number of unique terms in a document collection such as the Simpsons wiki increases only fairly slowly with an increasing number of pages once a threshold of 5000 to 10000 terms is reached. Thus, even for large document collections, we can expect a number of index terms in the range of tens of thousands for which the PEST matrix and index can be quickly computed by a small number of CPUs,

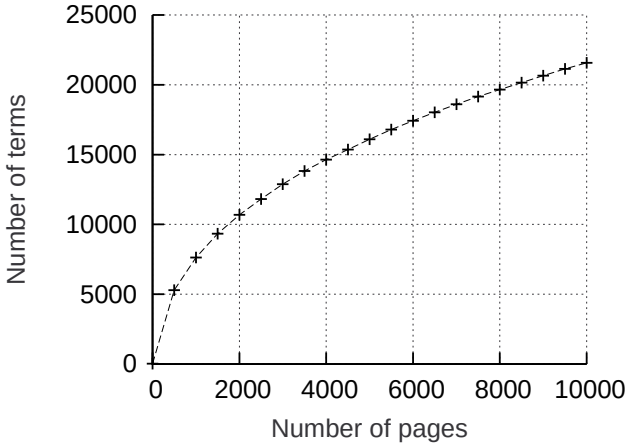


Figure 13: Number of unique terms over dataset size

even with the fairly unoptimized version of PEST discussed here.

Initially, this calculation has to be performed only once per term. When documents are edited, deleted or added, only the PEST vectors of the involved terms need to be re-calculated.

The scalability challenge that PEST faces is also less severe than that of personalized PageRank discussed in Section 3 as an individual PageRank has to be computed not per user but per term: The number of terms in a document collection is not directly proportional to the number of terms; instead, as the number of documents grows, most newly added documents only add few new words to the overall set of terms. This applies even more to a set of thematically homogeneous set of documents. Figure 13 shows the change in the number of unique terms as the number of documents in the dataset increases.

6.1.4. Results from the Different Normalization Techniques

In addition to the results described above, we performed tests on the different normalization techniques described in Section 4.4. The result rankings for the query term *"Bart"* can be seen in Appendix A.3.

The normalization method that worked best in our tests was method 2. Although it does not preserve the proportion between all edge weights, its use yields the best result. Methods 1 and 3 both have the weakness that the ranking is not changed very much, as in the example rankings the constant m was 1351 for method 1 (resp. 226 for method 3) and thus the edge weights were diminished very highly. Methods 4 and 5 in contrast changed the ranking too much and furthermore pushed irrelevant results too far into the beginning of the ranking, e.g., method 4 pushes the document *"Lenny Leonard"* from position 105 in the method 2 ranking to position 25 while this document has a small correlation with the entered query *"Bart"*. This became especially apparent for nodes with very few edges, as they are especially pushed. To mention it, the convergence of the power method is much slower for methods 4 and 5, which is probably due to the quite uniform distribution of high edge weights.

These test results suggests that normalizing not graph-wide increases performance compared to graph-wide normalization and on the other hand normalizing too much has a perhaps too high impact on the ranking, at least for wikis. For other types of data, another normalization technique might be better suited.

6.2. Delicious

Besides using a wiki as dataset, also other kinds of data are of great interest, especially social networks. In these, many connections are in place which convey useful information about the user and his shared information. These links can be used by PEST to propagate terms in the graph and optimize search queries. However, using Delicious[12] as a dataset, the connections turned out to be a little bit more difficult for a good ranking. We will describe our approach and the encountered problems in this section.

6.2.1. Description of the Dataset

Delicious describes itself as a "social bookmarking website". Users can post URLs and describe them via tags. The popularity of a URL can be seen in how often it is bookmarked by users. Furthermore, the tags give valuable information about what information can be found on the posted website.

As no official dataset is available from Delicious, we had to use another source. The two possibilities included crawling data by ourselves or using an existing dataset. Data dumps are available from the *DAI-Labor*[13]. Although holding data dated earlier than 2008, which is a bit old in terms of the fast-paced Internet development, it is still suited for our purpose and reduces the effort for gaining data.

The description for this dataset is also given by Wetzker et al. [47], but we will recall its most important properties. It consists of several files each containing data for one month from 9/2003 to 12/2007. The captured data are rows of the format:

```
[date] [user-id] [url] [tag]
```

Therefore, it contains the most important pieces of information: the user, the URL and the tags. Note that several tags belonging to the same URL are given as several rows each with one tag. What is missing in the dataset are the relationships between users which would have been a good additional source for inserting edges in our graph.

6.2.2. Experiment Setup

Several different methods were chosen to be evaluated on the same dataset. We used only a portion of the available data, i.e. 10,000 different URLs, as this size is easily manageable by PEST and experiment results were available very quickly. In the following, we will describe the different methods. They are also summarized in Table 8. Furthermore, small graphs depicting the type of connections which are used in each method can be seen in Figure 14.

[12] `http://delicious.com`
[13] `http://www.dai-labor.de/en/irml/datasets/delicious/`

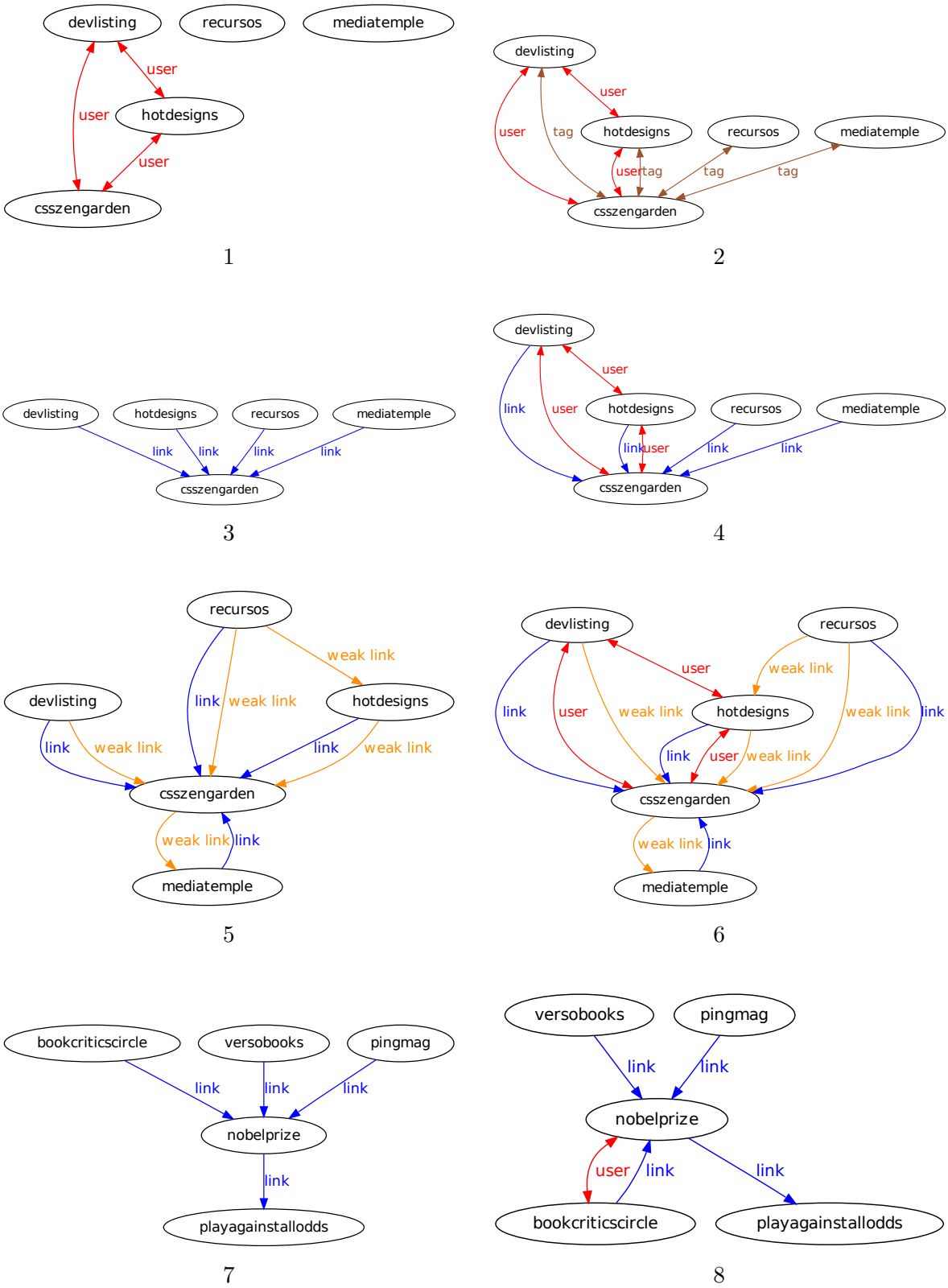


Figure 14: Small sample graphs showing which types of connections are used in each method

	Nodes	Edges	Terms
1	URLs	Same user	Tags
2		Same user or same tag	Tags
3		Web graph	Tags + content
4		Web graph or same user	Tags + content
5		Web graph or weak links	Tags + content
6		Web graph or weak links or same user	Tags + content
7	Domains	Web graph	Tags + content
8		Web graph or same user	Tags + content

Table 8: Evaluated methods for the Delicious dataset

1. As a first, simple experiment, we used the URLs as nodes in our graph and connected URLs posted by the same user with edges. The intuition is that each user has an area of expertise where he is interested in and bookmarks many web pages in this area. Thus, these URLs have some common ground and propagating between them is sensible. The edge weight is set to $[0.1, 0]$ (0.1 depicting the weight of the edge and 0 the weight of the reverse edge), where it has to be noted that these edges exist once for each direction and thus each edge's propagation weight is 0.1. As terms, the tags gave a natural basis describing the URL in a reasonable way with user-generated knowledge. The same normalization filter is applied to these tags as described in Section 5.1.3 in order to reduce the number of tags and bundle them into more meaningful terms. If several users tag the same URL with one common term, the weight of this term is scaled linearly to the number of taggings, i.e. the weight describes the frequency of a term in the URL's bulk of tags.
2. In the next step, also URLs with the same tag were connected in our graph and thus used for propagation. They are weighted higher than the user edges, $[0.3, 0]$. Multiple identical tag assignments to a URL are treated as one single allocation and thus only one edge is inserted. Here, the notion consists in exploiting the similarity of URLs when being tagged with the same tag.
3. Besides the evaluations of the first two experiments, which only take data from the given data dumps into account, we focused on the evaluation of links between URLs. Therefore, we downloaded the web pages from the URLs and parsed their content which was examined in terms of anchor links to other URLs contained in our set of URLs. This resulted in a graph where all these links exist as edges, a so-called web graph. As only thematically related web pages are linked in such a web graph, propagating along these edges gives an intuitive kind of spreading term weights. The weights were chosen as $[0.8, 0.6]$. No other edges were regarded in this method.
   Furthermore, we could now include the content of each web page as terms which we gained by converting the HTML data to plain text and then tokenizing the text. The same normalization filter is applied to these terms as well as the tags as described in Section 5.1.3.
4. The same method as in 3 is applied, with the difference that also the social connection between URLs is regarded, i.e. edges between URLs tagged by the same user are

inserted. The weights are the same as in 1.

5. Similar to 3, the web graph is considered, and furthermore also *weak links* are used as edges. We define them as normal links to URLs with the difference that only the domain-part of the link matters and the rest of the URL is discarded. This results in a much larger set of edges which are weighted less than the web graph edges, namely $[0.1, 0.05]$. User links again are not used. The notion is to also relate domains that link to each other, as the content of a web page on a specific domain often correlates to other pages on the same domain.
6. To examine the differences to a socially informed graph like in 3 and 4, also here user connections are regarded in addition to the web graph, with the same weight as in 1.
7. Two more experiments were evaluated that deal with an abbreviated version of the URLs which only uses the domain-part – like it is done with weak links. Therefore, also only weak links are regarded. Apart from that, the setup is the same as in 3. The edge weights of links are set to $[0.8, 0.6]$. Here, we attempt to get the significance of domains for one problem area.
8. Similar to the previous experiment with the addition of social linking.

In general, the edge weights were chosen to follow the intuition of the authors. Also, a bit of edge weight tuning was done to achieve better results. The edge weights given here are the final ones used for the experiments.

6.2.3. Results

The methods above were evaluated using the two queries *"art"* and *"photography"*. These two query terms are similar enough to return some of the same URLs and at the same time they differ enough to yield a set of web pages which should be returned by one query but not by the other one.

For evaluation, we chose to compare the PEST rankings with a normal tf-measure ranking. The most obvious way would have been to compare the results with the search results given by Delicious' own search. However, these results are based on a different dataset and thus cannot be compared.

Here, a summary of the experiment results will be given, for detailed results, refer to Appendix A.4.

1. The first method yielded a poor ranking compared to a tf-measure. The most plausible reason for this failure is that users post a lot of URLs which do not necessarily have something in common but, more likely, form groups of related URLs, which in between each other are not very similar at all. Therefore, the propagation of tags to the other URLs of a user does not transfer information in a valuable way and PEST does not give good results. This applies to both ranking methods.
2. When the second method is applied, the results are quite similar to the normal tf-ranking and no big changes in positions appear. This is due to the fact that terms are propagated to URLs which have the same tags and thus no real propagation happens at all (besides the user propagation). This duplicate usage of the term information does

not yield an improvement of the ranking. Therefore, we concluded that propagating between URLs with the same tags is not sensible and should be omitted.

3. Using the edges of the web graph for propagation, it appears that a more appropriate way of propagating terms is found. On the one hand, many URLs are ranked according to their tf-ranking. But on the other hand, some valuable URLs are pushed to a higher position. For example, some blog articles about *"photography"* are pushed from 37 to 22, and the Wikipedia entry about Nicéphore Niépce, one of the inventors of photography [34], is brought up by 43 ranks to position 39. As can be seen, the results are better and it can be noted that the web graph therefore gives a much better basis for the application of PEST.

4. Having the social connections in addition to the web graph, this method again promotes web pages that do not necessarily correlate with the query, the way it is in method 1. Here, for example, the homepage of "Synergy2", a user interface tool for operating systems, came into the *"photography"* ranking at position 26 as a new hit although not having to do anything with photography. Therefore, it can again be seen that using tags from the same user to establish links does not work well, due to the diverse interests of many users.

5. When taking weak links into account in addition to normal links, domains that are linked often are ranked higher. The idea is that important domains have a higher amount of incoming links and thus are more relevant for the user. For example, http://africanpainters.blogspot.com/ is ranked at position 28 (formerly 284) and it fits well into the *"art"* query. But the downside is that also irrelevant domains are promoted, for example, http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html as a link collection for network monitoring tools ranks 36^{th}. Furthermore, when comparing the results for *"art"* and *"photography"*, many URLs that are highly promoted appear in both rankings at about the same position. This is also an effect of the use of weak links on domains. To sum up, weak linking has both positive and negative effects, but the negative ones prevail in our results.

6. Adding social connections to our graph does not result in big changes compared to the previous method. The only change consists in the lesser extent to which important domains get boosted. They are still visible in the top 40 ranking of *"photography"*, but not of *"art"*.

7. Using only the domain-part of the URLs, we tried to get some insights into the importance of websites. And as expected, domains that are linked a lot appear in the top ranks and are promoted by PEST very much. This happens to be very favorite for some URLs like http://www.photoethnography.com or http://www.flickr.com for the *"photography"* query, but of course also common high-linked websites are pushed higher, like http://www.facebook.com or http://www.amazon.com, which do not have so much in common directly with *"art"* or *"photography"*. However, it is interesting to see how high-traffic websites are ranked in the two different queries. For example, http://www.youtube.com is ranked 18^{th} for *"art"* and 49^{th} for *"photography"* where you can see the difference in importance which is justified in our opinion. Therefore, more the difference between several queries could yield interesting results and should be researched more.

8. Including links between URLs by the same user, the ranking difference to the tf-ranking is diminished. The additional edges in our graph take some of the propagation power of the web graph links and therefore soften the effects noticed in the previous method. Therefore, it does not yield as good results compared to the other methods.

To sum up, interesting results could be achieved by experimenting with the Delicious dataset. In the case of method 3, relevant pages were promoted and search results could be improved. The focus on links taken from the web graph and content of the web pages proved to be effective and underlines the performance of PEST for the application on structured web data. Adding a social component in terms of links between pages bookmarked by the same user introduces the following problem: As users are often interested in many diverse subjects, interlinking these subjects incorporates relations into the propagation graph that diminish the value of this graph. To remedy this problem, the links would have to be created on a per-subject basis which is what we propose to do in a follow-up experiment on a larger dataset. Using only the domain-part of URLs results in an importance score for each website, similar to PageRank with the difference of being term-dependent.

Although not all methods resulted in an improvement of the ranking in terms of being a useful search result for a user, their evaluation helped to gain insights into the requirements for the data which can be processed by PEST as will be described in the following section.

6.3. Properties of Datasets Processed by PEST

Not all kinds of datasets are suitable for the application of the PEST algorithm. The most obvious constraint is the need for data which can be transformed into a graph structure. This means that data items (nodes) and links between them (edges) have to be identified in the data source and chosen sensibly. Furthermore, as terms are propagated in the graph structure, some text must exist that is used for forming the terms. This can be just a title or a longer article which has to be processed into a term frequency vector which contains all relevant terms together with their frequency occurring in the text. In addition, our experiments highlight that other properties of the data, e.g., the number of links in comparison to the number of nodes, have a great impact on the results. These properties, which have not been clear in the beginning, will be described in the following.

6.3.1. Identification of Data Items

The first step that has to be done when experimenting with data is identifying which data to use for what purpose. In our PEST algorithm, three kinds of data exist:

- Data items, e.g., documents, tags, wiki pages, music artists
- Links between data items, e.g., hyperlinks, taggings, containment, between artists playing at the same concert
- Terms, e.g., titles, text, tags, artist names

One problem that came up during our experiments was that it is not always clear which type of data should be chosen for a certain dataset property. For example, tags can be used as terms or links can be created between documents which contain the same tag. Both

		All nodes		Only nodes with edges	
	# edges	**# nodes**	**Mean**	**# nodes**	**Mean**
Simpsons	52977	10955	4.8	4032	13.1
Delicious 3	2770	10000	0.3	1915	1.4
Delicious 4	318172	10000	31.8	8675	36.7

Table 9: Measures of edge distribution among the nodes of the Simpsons and Delicious experiments. The *"Mean"* column shows the mean number of edges per node. The values are given for all nodes as well as only for nodes that have outgoing edges

ways make sense, and it has to be considered which way is the better one. As a first hint, we learned that using the same property in both ways at once does not give better results. Moreover, it depends on the wanted search queries. If the search for a certain tag should be possible, then tags have to be used as terms in order to propagate them in the graph. However, for propagating other terms between similar documents, i.e., documents being tagged the same way, the second approach works better. It has to be considered though that the amount of links between documents with the same tag can get quite large as it rises quadratic to the number of documents (more exactly, for one tag, if n documents have the same tag, there are $\frac{n \cdot (n+1)}{2}$ links).

Besides finding a sensible division into the different kinds of data, also the amount of information which is to be included into the evaluation has to be determined. This again depends on the search queries one wants to examine on the dataset. For example when emulating a search engine for a wiki, it makes sense to include the whole text (split up into words) as terms in order to allow searching for every single word. On the other hand, when music artists are interlinked and similar artists are to be found, only using the artist names – and thus one term per node – can be sufficient.

All in all, it is clear that this is not a trivial problem and it has expansive consequences on the quality of PEST's results. Therefore, we recommend to determine the selection of the right data items precisely. In Section 6.2, we have an example of how to evaluate different choices of data items.

6.3.2. Connectivity between Data Items

The second criterion which affects the quality of the result ranking is the connectivity of the graph. Here, we use connectivity as synonym for the edge to node ratio of the graph. Each input graph has to be connected to a certain extent in order to allow terms to be propagated through it. Although we did not focus on determining if there is an upper boundary for a minimal connectivity (resp. a lower boundary for a maximum connectivity), we can give some estimations about a suitable network connectivity. In Table 9, we show how many edges and nodes exist in the graphs of the examples.

It can be seen from Table 9 that in the Simpsons wiki every node has on average 4.8 outgoing edges whereas in the Delicious datasets this value is 0.3 respectively 31.8 for two sample variants of the experiment. While the Simpsons example yielded good results, the results from Delicious are of mixed quality. The connectivities of the Delicious experiments seem

to be either too low or too high whereas the ones from the Simpsons experiment lie in the middle. A too low connectivity does not provide enough links for propagation. On the other hand, a too high connectivity results in a graph where many non-relevant propagation edges exist, and if the edge weights are not close to 0, a noticeable amount of the term weights is distributed in meaningless ways. Thus we suggest to use graphs with roughly 5 times as many edges as nodes, where about half of all nodes should have outgoing edges.

6.3.3. Distinctiveness of Terms

Also, the right selection of terms plays an important role in the algorithm as it determines for which keywords results can be retrieved. Furthermore, the number of terms impacts the distinctiveness of the results, e.g., how the ranking of two syntactically different, but semantically similar queries differ. These objectives are the same as for vector space model search algorithms and, thus, analog methods can be applied which include removing of stopwords, stemming and (more complex) finding synonyms. On the other hand, what exceeds the traditional search engines is the fact that in PEST the terms are the objects that get propagated through the graph. However, this is only decisive in the way the kinds of data are chosen, as described in Section 6.3.1, whereas the terms in PEST serve the same purpose as in traditional VSMs. Only the weights are customized to reflect the approximate matching approach. Therefore, it is sufficient to apply text analyzing methods also employed for other search engines.

7. Outlook and Conclusion

Having covered the main aspects of PEST, we will now give some hints how PEST can still be improved and afterwards recapitulate this thesis in the conclusion.

7.1. Outlook

Although this thesis provides a solid explanation of term-**p**ropagation using **e**igenvector computation over **st**ructured data, there is also a wide body of further work to refine and extend PEST. We will first discuss how the algorithm itself can be improved, continue with approaches to increase its run time efficiency and finally discuss how to deal with an index that is periodically updated.

7.1.1. Improvements to the PEST Algorithm

We are currently using rough estimates for α and ρ as well as for the edge weights rather than empirically validated observations. A guide to choosing these values might be possible to derive from studying the behavior of PEST on data with varying characteristics. Edge values, in particular, could also be amenable to various machine learning approaches, for example, using average semantic relatedness as a criterion, or to semi-automatic approaches through user-feedback. If edge weights are to be determined based, e.g., on the semantic similarity of a typed edge and a term (as determined through their Google distance or distance in an *ontology*), also the transposed, normalized adjacency matrix $\mathbf{H}$ becomes term-dependent and has to be computed once for each term.

We have also considered a number of different algorithmic approaches to term weight propagation, e.g., where propagation is not based on convergence but on a fixed number of propagation steps. Techniques for spreading activation [18, 19] might be applicable and a comparison study is called for. Furthermore, the computation of the PEST matrix is just one of several alternatives to finding a stochastic propagation matrix.

There are also a number of *specific areas for improving* PEST*:*

- In PEST, propagation between data items of different types influence each other: E.g., a document with many tags propagates a relatively smaller amount to its children than a document with few children. For extreme cases, a model where each of these propagations is at least given a minimal amount might prove superior to the basic version of PEST described here.
- Links to *external resources* such as Linked Open Data or ontologies are currently not considered in PEST. Their inclusion would allow to enrich the content graph and thereby enhance the results of term propagation. This extension seems particularly promising in combination with aforementioned typed links.
- At the moment, PEST makes no distinction between terms based on where they occur in the document. One simple and obvious extension would be for example to represent the anchor text for each link, strongly propagating the respective terms to the linked document. This particular scheme mirrors a feature of classic PageRank, but a wide

range of further possibilities for modifying edge or term weights based on the position of the text in the document exists.

- Another, wiki-specific, extension is observing how the term scores of a document change over several *revisions* and taking this into account as a factor when ranking query answers.
- Any approximate matching approach suffers from non-obvious *explanations* for returned answers: In the case of a boolean query semantics, the answer is obvious, but when term propagation is used, a document might be a highly-ranked query result without containing any query terms directly. In this case, providing an explanation, for example, that the document in question is closely connected to many documents containing query terms, makes the matching process more transparent to users. However, to automatically compute good minimal explanations is far from a solved issue.
- A further aspect that actually started the thought process about implementing an approximate matching algorithm is the possible integration into the search engine Lucene. In order to achieve this, the calculated propagated term frequencies have to be stored into Lucene's database. However, also position information of the terms is stored there and, furthermore, an inverse index is not trivial anymore as the terms are now contained in all documents.

With the increasing size of the linked open data cloud, data providers require convenient means to sift through that data to discover relevant concepts and published instances for linking with their data. To find such concepts and instances, the structure of the involved RDF data is crucial, and thus existing search engines are insufficient. At the same time, formal (e.g., SPARQL) queries over ontologies require extensive training and knowledge of the structure of the involved ontologies. Here, a semantic version of PEST would provide publishers with an easy and familiar means to discover relevant concepts and individuals in large-scale ontologies by taking the structure of the data into consideration to return the most relevant matches to keyword searches.

7.1.2. Approaches to Increase the Run Time Performance

Although delivering a good performance according to the benchmarks in Section 6.1.3, the implementation of the PEST algorithm can be further improved to make it more efficient both in run time and memory space:

- Currently, the whole implementation is strictly linear and does not execute code in parallel. It would not pose a problem to parallelize at least the code for computing the single PEST matrices and eigenvectors, and also other parts of the code are subject to these kind of improvements.
- One could also argue that the power method in its original version may not be the fastest way to compute the eigenvectors of a matrix. Especially the execution on large matrices can be improved using methods described by Chen et al. [17].
- Adding to the discussions about the data structures in Section 5.3, not only the use of a different matrix representation or storage can be regarded, but also the possibility of splitting the PEST matrix into a sparse matrix plus some additional information

about the entries containing a value below a custom threshold may be considered. This could result in less used memory and perhaps an improvement in efficiency.

Furthermore, ideas from other methods similar to PageRank could be adapted to improve PEST's efficiency:

Many approaches to implementing personalized PageRank in a less computationally expensive way have been suggested by Haveliwala et al. [28]. They reduce the number of PageRank calculations necessary by limiting the granularity of personalization and thereby the number of PageRank computations needed.

Topic-sensitive PageRank [27] offers query-dependent personalization on the basis of manipulating the prior probabilities of classes, that is, different topics, according to a user's interests. The personalization is restricted in that it operates at the level of topics, not individual web pages, calculating the score of a query as the sum of all pre-computed topic-dependent document scores multiplied by the likelihood of class membership.

Jeh and Widom [29] present an approach where personalized PageRank vectors are approximated as a linear combination of a number of partial vectors, so-called basis vectors representing certain highly-ranked web pages, pre-computed using a scalable dynamic programming approach. This method limits personalization through the choice of basis vectors: The surfer can only teleport to pages represented in the basis vectors.

BlockRank [30] combines individual web pages by their host or block, computing local PageRank vectors on a per host basis and weighting them by the overall importance of the host. Personalization here is realized at the granularity of blocks meaning that a user can only express his preference for a host, encoded in the weighting of local PageRank vectors, not for an individual web page.

While none of these approaches is directly applicable to PEST, it is likely that along similar lines approximate, but faster versions of PEST can be designed. The obvious way to achieve this is to limit the number of terms indexed by PEST, e.g., to only the most prominent terms, by merging synonymous or semantically close terms, or by merging terms with similar frequency distributions.

7.1.3. Extension for a Document Collection with Frequent Changes

PEST makes use of the structure of the data to implement approximate matching. The algorithm needs to be run once for each term that is to be propagated in the graph. This means that at first sight it might be considered to be very costly and this is one aspect that further research must take into careful considerations. However, we are optimistic that the approach is viable since methods for the efficient calculation of eigenvalue problems exist and continue to be developed. Further, the calculations do not have to be run upon query evaluation, but after updating a node in the graph, so the process can run in the background or be scheduled to a later time. Upon updating, not all text changes, so efficiency could be improved by running the process only on the terms which changed and simply updating the indices. This is possible since the terms are independent from each other. Finally, it is likely that not all words of the text should or need to be propagated through the graph as terms. Stopwords are an obvious candidate for exclusion from the process.

7.2. Conclusion

In this thesis, we have described PEST as a unique approach to approximate matching and provided a theoretical model supporting it's aims. PEST combines the principles of structural relevance from approaches such as PageRank with the standard vector space model. A structured datasource is converted into a graph representation and appropriate weights to the edges are set. Then, a fuzzy distribution of term weights across the nodes is created by propagating the term weights intelligently through the graph using eigenvector computations. PEST's particular strength is that it runs entirely at index time and results in a modified index representation.

We further analyzed PEST's applicability for a real-world wiki. The experiment results impressively show that PEST's approach is capable of improving the search results in order to get a better approximate result ranking. This is done in a way where not only the ranking of strict matches is changed, but also new matches are included that have not been part of the ranking before.

Aside from the effect on the results, the run time performance has been examined. It has been shown that even a relatively simple implementation without sophisticated efficiency tuning is easily capable of handling several thousands of documents. Besides, several hints for improving PEST's efficiency have been given.

Furthermore, a second dataset in the area of social networks has been examined using a variety of different network encodings which led to partly good results promoting important linked web pages and on the other hand some less promising ones. The main findings from these experiments – using data types several times degrade results, the number of connections should be balanced – were incorporated into a guideline what properties a dataset must possess.

In addition, the implementation of PEST showed the intelligent combination of several existing libraries together with custom exchangable data structures. The focus has been put on a maintainable and easy to extend code base which has been achieved.

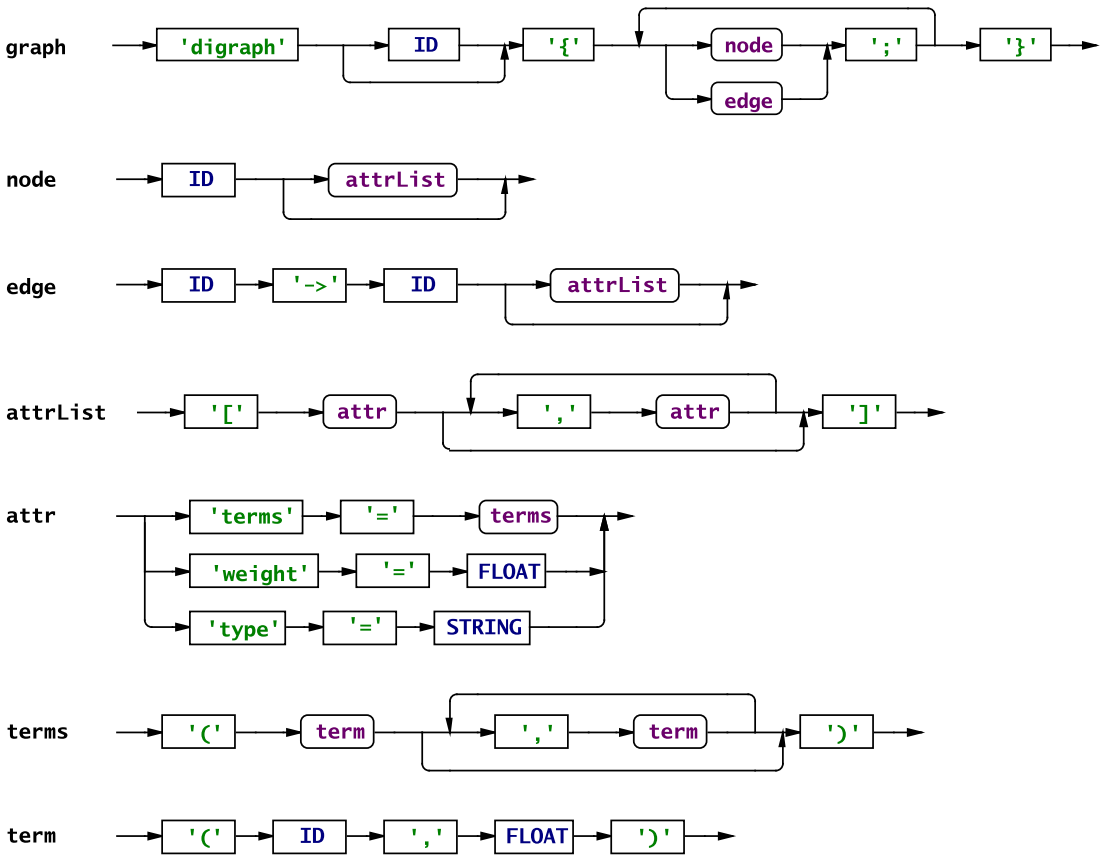
Altogether, this thesis provides a solid foundation for exploring the potential and research issues on term-**p**ropagation using **e**igenvector computation over **st**ructured data.

A. Appendix

A.1. Command Line Syntax

```
Usage: ./pest.sh [OPTION] [Input file]
 -b,--benchmark <arg>              only run benchmark, do not print
                                   results, write into given file
 -h,--help                         shows this help message
 -l,--useSelfLoop                  uses the self-loop strategy (default:
                                   false)
 -m,--mysql                        use mysql as source for calculations
 -n,--iterations <arg>             maximum number of iterations for the
                                   power method (default: 100)
 -o,--output <arg>                 output format, can be 'csv' or 'plain'
                                   (default: plain)
 -p,--precision <arg>              precision for output of numbers
                                   (default: 4)
 -q,--query <arg>                  query term for which the vector should
                                   be computed, separated by colons
                                   (default: all terms)
 -r,--randomLeap <arg>             range 0-1, higher value means more
                                   random leaps (default: 0.15)
 -s,--settings <arg>               load settings from file, refer to the
                                   manual for an example
 -t,--tolerance <arg>              tolerance to be used for the power
                                   method as convergence criterion
                                   (default: 1.0E-6)
 -u,--uniformDistribution <arg>    what percentage of the remaining
                                   probability will be distributed
                                   uniformly, u=0 distributes only
                                   according to term frequencies, u=1
                                   distributes totally uniform (default:
                                   0.25)
 -v,--verbosity <arg>              how much output is printed on console
                                   (default: INFO)
```

A.2. ANTLR Grammar for DOT-Like Syntax



A.3. Results of the Simpsons Experiments

A.3.1. MediaWiki-Style Ranking for Query "Bart"

	PEST score	Page title	tf rank	PEST change
1	1.272594	Bart Simpson	2	+1
2	0.222006	List of guest stars	95	+93
3	0.184705	Homer Simpson	590	+587
4	0.120561	Lisa Simpson	186	+182
5	0.117131	Bart Gets an F	7	+2
6	0.111575	Bart Simpson (comic book series)	3	-3
7	0.101464	List of one-time characters	29	+22
8	0.092190	Bart the General	4	-4
9	0.088538	List of Bart Episodes in The Simpsons	1	-8
10	0.078576	List of Simpsons releases by date	45	+35
11	0.078457	Springfield	1112	+1101
12	0.070372	Marge Simpson	1116	+1104
13	0.064391	Made-up words	18	+5
14	0.056234	The Bart Book	5	-9
15	0.049190	Bart Sells His Soul	15	0
16	0.048416	Bart Gets Hit by a Car/Full Synopsis	6	-10
17	0.048175	Bart vs. Thanksgiving	9	-8
18	0.047780	Bart vs. Lisa vs. the Third Grade	8	-10
19	0.047760	Charles Montgomery Burns	1113	+1094
20	0.045904	Radio Bart	14	-6
21	0.045050	Bart the Genius	33	+12
22	0.044868	Bart's Girlfriend	10	-12
23	0.043799	Bart Star	21	-2
24	0.043231	Maggie Simpson	442	+418
25	0.042586	Bart Gets an F/Quotes	11	-14
26	0.042124	Bart vs. Australia	24	-2
27	0.041227	Bart to the Future	16	-11
28	0.041223	Barting Over	13	-15
29	0.040935	Springfield's State	77	+48
30	0.040596	Non-English versions	241	+211
31	0.039500	Bart the General/Quotes	12	-19
32	0.037820	Bart Gets Famous	17	-15
33	0.035426	Seymour Skinner	258	+225
34	0.034144	Bart the Murderer	20	-14
35	0.033610	Milhouse Van Houten	176	+141
36	0.032626	The Simpsons: Hit and Run	42	+6
37	0.032351	Bart Simpson's Treehouse of Horror	28	-9
38	0.032032	Bart on the Road	22	-16
39	0.031902	Simpsons Comics in the US	54	+15
40	0.031579	Virtual Bart	19	-21

A.3.2. Ranking for Query "Bart"

	PEST score	Page title	tf rank	PEST change
1	0.009604	Bart Simpson	325	+324
2	0.004732	Homer Simpson	1667	+1665
3	0.004445	List of Bart Episodes in The Simpsons	1	-2
4	0.003611	Lisa Simpson	1085	+1081
5	0.003293	Bart's Bike	2	-3
6	0.002656	Marge Simpson	1773	+1767
7	0.002607	Bart Junior	3	-4
8	0.002260	Bart the Mother/Quotes	4	-4
9	0.002229	Ticket Bouncer	5	-4
10	0.002218	Chirpy Boy and Bart Junior	6	-4
11	0.002100	Spree for All	7	-4
12	0.001995	Charlie	8	-4
13	0.001964	Congressman 3	9	-4
14	0.001920	Bart the Genius	84	+70
15	0.001862	Bart Goes to the Movies	10	-5
16	0.001848	Bike Track	12	-4
17	0.001834	Maggie Simpson	1237	+1220
18	0.001830	Bart Cops Out	11	-7
19	0.001797	Bart Junior (frog)	17	-2
20	0.001785	Bart Jumps/Credits	14	-6
21	0.001785	Bart's Dog Gets an F/References	13	-8
22	0.001760	Bart Gets an F	257	+235
23	0.001752	First Ever Sled	16	-7
24	0.001748	Bart's Moon Party	15	-9
25	0.001672	Bart the Murderer/Quotes	18	-7
26	0.001656	Bart's Imaginary Hell	20	-6
27	0.001645	Bart Simpson Comics 30	19	-8
28	0.001638	Bart Jumps/Quotes	21	-7
29	0.001614	Bart's Haircut	22	-7
30	0.001580	Bart Simpson Comics 1	23	-7
31	0.001547	Police Officer (Bart the Hero)	25	-6
32	0.001522	Homer Simpson-Bart Simpson relationship	24	-8
33	0.001518	Old Red	26	-7
34	0.001515	Skinner's Sense of Snow/References	27	-7
35	0.001515	Bart Junior (person)	28	-7
36	0.001477	Bart vs. Australia/Appearances	32	-4
37	0.001473	Evan Conover	29	-8
38	0.001456	Bart Simpson Comics 24	50	+12
39	0.001435	Bart Simpson (comic book series)	127	+88
40	0.001434	List of Simpsons releases by date	704	+664

A.3.3. MediaWiki-Style Ranking for Query “Moe beer”

	PEST score	Page title	tf rank	PEST change
1	25.261261	Moe Szyslak	1	0
2	20.934569	Duff Beer	3	+1
3	20.323489	Moe’s Tavern	2	-1
4	18.901237	List of guest stars	13	+9
5	13.748875	List of one-time characters	9	+4
6	13.550532	Homer Simpson	108	+102
7	12.710784	Springfield’s State	21	+14
8	10.67065	Made-up words	6	-2
9	8.677318	Moe ’N’ a Lisa	4	-5
10	8.202666	Flaming Moe’s	7	-3
11	7.935875	The Seven-Beer Snitch	15	+4
12	7.935286	Eeny Teeny Maya Moe	5	-7
13	7.811267	Springfield	135	+122
14	7.070485	Bart Simpson	263	+249
15	7.027493	Charles Montgomery Burns	41	+26
16	5.956218	Non-English versions	37	+21
17	5.751666	List of Simpsons releases by date	43	+26
18	5.079298	The Simpsons Movie	88	+70
19	4.714443	Homer vs. the Eighteenth Amendment	19	0
20	4.651780	Lisa Simpson	-	*new*
21	4.537188	Barney Gumble	36	+15
22	4.527837	The Seemingly Never-Ending Story	11	-11
23	4.496271	Homer the Moe	10	-13
24	4.134657	Welcome to Moe’s	12	-12
25	4.087322	In-show deaths	23	-2
26	3.877459	Marge Simpson	-	*new*
27	3.745833	Ned Flanders	116	+89
28	3.702389	Mommie Beerest	20	-8
29	3.595084	Lenny Leonard	47	+18
30	3.476984	The Simpsons: Hit and Run	26	-4
31	3.393440	Songs in the Key of Springfield	102	+71
32	3.354291	Duffless	48	+16
33	3.354037	It Was a Very Good Beer	35	+2
34	3.323219	Eeny Teeny Maya Moe/Appearances	27	-7
35	3.275619	Fudd Beer	38	+3
36	3.210045	Mmm...	39	+3
37	3.178919	Moe Baby Blues	14	-23
38	3.079227	Homer’s Odyssey Full Synopis	45	+7
39	3.035344	List of Simpson Episodes by Production Code	89	+50
40	3.033796	List of Homer’s Jobs	59	+19

A.3.4. Ranking for Query "Moe beer"

	PEST score	Page title	tf rank	PEST change
1	0.474352	Colonel Homer/References	1	0
2	0.451702	Billy beer	2	0
3	0.407410	Fudd Beer	6	+3
4	0.399300	The Joy of Sect/Quotes	7	+3
5	0.382369	Duff Beer Advertiser	8	+3
6	0.380859	It Was a Very Good Beer	9	+3
7	0.347153	Homer Simpson	662	+655
8	0.340615	Channel 4 Simpsons Advert/Quotes	11	+3
9	0.319573	Duff Beer	29	+20
10	0.315634	Frank Gehry	12	+2
11	0.313839	Stacey Keach	13	+2
12	0.313296	Homer vs. the Eighteenth Amendment	14	+2
13	0.306982	The Aristocrat's Bartender	19	+6
14	0.277014	The Aristocrat	34	+20
15	0.266384	Unlucky	35	+20
16	0.256664	Duff Brewery	36	+20
17	0.251594	Duff Gardens	40	+23
18	0.251136	Titania	48	+30
19	0.246690	Moe Syzslak Connection	3	-16
20	0.244073	Channel 4 Simpsons Advert	44	+24
21	0.243186	Little Moe Syzslak	4	-17
22	0.239346	Selma's Choice/References	49	+27
23	0.237612	Duff Beer Krusty Burger Buzz Cola Costington's Department Store Kwik-E-Mart Stupid Flanders Park	50	+27
24	0.227599	Flaming Moe's	24	0
25	0.226966	Moe Szyslak	60	+35
26	0.226349	Moe Letter Blues	5	-21
27	0.223281	So It's Come to This: A Simpsons Clip Show	61	+34
28	0.219710	You Only Move Twice/Credits	63	+35
29	0.219090	Springfield Concert Hall	67	+38
30	0.217805	Charles Napier (actor)	64	+34
31	0.215642	Moe's Tavern	41	+10
32	0.213856	Lionel Richie	68	+36
33	0.213833	Marge's Demons	69	+36
34	0.210067	Boy Meets Curl/Quotes	71	+37
35	0.209389	Flaming Homer	46	+11
36	0.208465	The Seven Duffs	89	+53
37	0.207069	Selma's Choice/Quotes	78	+41
38	0.203708	Charles Napier	82	+44
39	0.200783	Fritz	84	+45
40	0.191443	Peanut Street	10	-30

A.3.5. MediaWiki-Style Ranking for Query "Bart" and Normalization 1

	PEST score	Page title	tf rank	PEST change
1	0.097134	Bart Simpson	2	+1
2	0.095738	List of Bart Episodes in The Simpsons	1	-1
3	0.081932	Bart Simpson (comic book series)	3	0
4	0.066322	Bart the General	4	0
5	0.060171	The Bart Book	5	0
6	0.053981	Bart Gets Hit by a Car/Full Synopsis	6	0
7	0.052693	Bart Gets an F	7	0
8	0.049372	Bart vs. Lisa vs. the Third Grade	8	0
9	0.049193	Bart vs. Thanksgiving	9	0
10	0.049020	Bart's Girlfriend	10	0
11	0.048513	Bart Gets an F/Quotes	11	0
12	0.044998	Bart the General/Quotes	12	0
13	0.044239	Radio Bart	14	+1
14	0.044135	Barting Over	13	-1
15	0.041949	Bart Sells His Soul	15	0
16	0.041723	Bart to the Future	16	0
17	0.040647	Bart Gets Famous	17	0
18	0.040214	Made-up words	18	0
19	0.035262	Virtual Bart	19	0
20	0.034786	Bart the Murderer	20	0
21	0.034181	Bart Star	21	0
22	0.033472	List of one-time characters	29	+7
23	0.031812	Bart on the Road	22	-1
24	0.031423	Bart the Genius/References	23	-1
25	0.031108	Bart Simpson's Treehouse of Horror	28	+3
26	0.030766	Bart vs. Australia	24	-2
27	0.030298	Bart Gets Hit by a Car	26	-1
28	0.030058	Bart Simpson-Sideshow Bob conflict	25	-3
29	0.029894	Do the Bartman	27	-2
30	0.029385	Bart the Daredevil/Quotes	30	0
31	0.028238	Homer Simpson-Bart Simpson relationship	31	0
32	0.028194	Bart the Daredevil	34	+2
33	0.028071	Bart Jumps	32	-1
34	0.027890	Bart-Mangled Banner	35	+1
35	0.027889	Bart the Genius	33	-2
36	0.027044	Bart the General/References	36	0
37	0.026627	O Brother, Where Bart Thou?	37	0
38	0.025807	Ten Commandments of Bart	41	+3
39	0.025613	The Simpsons: Hit and Run	42	+3
40	0.025313	Bart's Dog Gets an F/Quotes	38	-2

A.3.6. MediaWiki-Style Ranking for Query "Bart" and Normalization 2

	PEST score	Page title	tf rank	PEST change
1	1.272594	Bart Simpson	2	+1
2	0.222006	List of guest stars	95	+93
3	0.184705	Homer Simpson	590	+587
4	0.120561	Lisa Simpson	186	+182
5	0.117131	Bart Gets an F	7	+2
6	0.111575	Bart Simpson (comic book series)	3	-3
7	0.101464	List of one-time characters	29	+22
8	0.092190	Bart the General	4	-4
9	0.088538	List of Bart Episodes in The Simpsons	1	-8
10	0.078576	List of Simpsons releases by date	45	+35
11	0.078457	Springfield	1112	+1101
12	0.070372	Marge Simpson	1116	+1104
13	0.064391	Made-up words	18	+5
14	0.056234	The Bart Book	5	-9
15	0.049190	Bart Sells His Soul	15	0
16	0.048416	Bart Gets Hit by a Car/Full Synopsis	6	-10
17	0.048175	Bart vs. Thanksgiving	9	-8
18	0.047780	Bart vs. Lisa vs. the Third Grade	8	-10
19	0.047760	Charles Montgomery Burns	1113	+1094
20	0.045904	Radio Bart	14	-6
21	0.045050	Bart the Genius	33	+12
22	0.044868	Bart's Girlfriend	10	-12
23	0.043799	Bart Star	21	-2
24	0.043231	Maggie Simpson	442	+418
25	0.042586	Bart Gets an F/Quotes	11	-14
26	0.042124	Bart vs. Australia	24	-2
27	0.041227	Bart to the Future	16	-11
28	0.041223	Barting Over	13	-15
29	0.040935	Springfield's State	77	+48
30	0.040596	Non-English versions	241	+211
31	0.039500	Bart the General/Quotes	12	-19
32	0.037820	Bart Gets Famous	17	-15
33	0.035426	Seymour Skinner	258	+225
34	0.034144	Bart the Murderer	20	-14
35	0.033610	Milhouse Van Houten	176	+141
36	0.032626	The Simpsons: Hit and Run	42	+6
37	0.032351	Bart Simpson's Treehouse of Horror	28	-9
38	0.032032	Bart on the Road	22	-16
39	0.031902	Simpsons Comics in the US	54	+15
40	0.031579	Virtual Bart	19	-21

A.3.7. MediaWiki-Style Ranking for Query "Bart" and Normalization 3

	PEST score	Page title	tf rank	PEST change
1	0.135520	Bart Simpson	2	+1
2	0.095127	List of Bart Episodes in The Simpsons	1	-1
3	0.081928	Bart Simpson (comic book series)	3	0
4	0.067628	Bart the General	4	0
5	0.059784	The Bart Book	5	0
6	0.054222	Bart Gets an F	7	+1
7	0.053631	Bart Gets Hit by a Car/Full Synopsis	6	-1
8	0.049235	Bart vs. Lisa vs. the Third Grade	8	0
9	0.049155	Bart vs. Thanksgiving	9	0
10	0.048876	Bart's Girlfriend	10	0
11	0.048081	Bart Gets an F/Quotes	11	0
12	0.044597	Bart the General/Quotes	12	0
13	0.044446	Radio Bart	14	+1
14	0.044003	Barting Over	13	-1
15	0.042621	Made-up words	18	+3
16	0.042076	Bart Sells His Soul	15	-1
17	0.041842	Bart to the Future	16	-1
18	0.040580	Bart Gets Famous	17	-1
19	0.037236	List of one-time characters	29	+10
20	0.035015	Virtual Bart	19	-1
21	0.034769	Bart the Murderer	20	-1
22	0.034413	Bart Star	21	-1
23	0.031908	Bart on the Road	22	-1
24	0.031189	Bart the Genius/References	23	-1
25	0.031042	Bart Simpson's Treehouse of Horror	28	+3
26	0.030964	Bart vs. Australia	24	-2
27	0.030413	Bart Gets Hit by a Car	26	-1
28	0.029832	Do the Bartman	27	-1
29	0.029808	Bart Simpson-Sideshow Bob conflict	25	-4
30	0.029124	Bart the Daredevil/Quotes	30	0
31	0.028518	Bart the Genius	33	+2
32	0.028384	Bart the Daredevil	34	+2
33	0.027995	Homer Simpson-Bart Simpson relationship	31	-2
34	0.027951	Bart-Mangled Banner	35	+1
35	0.027885	Bart Jumps	32	-3
36	0.026963	List of Simpsons releases by date	45	+9
37	0.026804	Bart the General/References	36	-1
38	0.026601	O Brother, Where Bart Thou?	37	-1
39	0.026412	List of guest stars	95	+56
40	0.026317	The Simpsons: Hit and Run	42	+2

A.3.8. MediaWiki-Style Ranking for Query "Bart" and Normalization 4

	PEST score	Page title	tf rank	PEST change
1	3.022971	Bart Simpson	2	+1
2	0.812955	List of guest stars	95	+93
3	0.659529	Homer Simpson	590	+587
4	0.429255	List of one-time characters	29	+25
5	0.416012	Lisa Simpson	186	+181
6	0.391276	Springfield	1112	+1106
7	0.290718	Marge Simpson	1116	+1109
8	0.285218	Charles Montgomery Burns	1113	+1105
9	0.277147	Made-up words	18	+9
10	0.232601	List of Simpsons releases by date	45	+35
11	0.194012	Springfield's State	77	+66
12	0.176782	Ned Flanders	295	+283
13	0.153750	Bart Gets an F	7	-6
14	0.148877	Maggie Simpson	442	+428
15	0.129789	Bart the General	4	-11
16	0.122974	Seymour Skinner	258	+242
17	0.119709	The Simpsons Movie	100	+83
18	0.118524	Krusty the Clown	527	+509
19	0.117437	Character Gallery	403	+384
20	0.116273	Clancy Wiggum	591	+571
21	0.116043	Abraham Simpson	457	+436
22	0.114024	Milhouse Van Houten	176	+154
23	0.107061	Moe Szyslak	794	+771
24	0.096619	The Simpsons: Hit and Run	42	+18
25	0.095324	Lenny Leonard	1111	+1086
26	0.089421	Nelson Muntz	213	+187
27	0.085253	Character Guide	604	+577
28	0.079660	The Simpsons Movie/Credits	1114	+1086
29	0.073799	List of Simpson Episodes by Production Code	311	+282
30	0.073492	In-show deaths	151	+121
31	0.068473	Bart Simpson (comic book series)	3	-28
32	0.068174	The Simpsons Game	108	+76
33	0.067891	Non-English versions	241	+208
34	0.064228	Bart the Genius	33	-1
35	0.063400	Patty Bouvier	916	+881
36	0.062402	There's No Disgrace Like Home	237	+201
37	0.062178	Barney Gumble	1820	+1783
38	0.061313	Martin Prince	103	+65
39	0.061034	742 Evergreen Terrace	1107	+1068
40	0.059050	Julius Hibbert	601	+561

A.3.9. MediaWiki-Style Ranking for Query "Bart" and Normalization 5

	PEST score	Page title	tf rank	PEST change
1	2.689119	Bart Simpson	2	+1
2	1.284926	List of guest stars	95	+93
3	0.639687	Homer Simpson	590	+587
4	0.606073	List of one-time characters	29	+25
5	0.390519	Springfield	1112	+1107
6	0.377072	Lisa Simpson	186	+180
7	0.302606	Made-up words	18	+11
8	0.301590	Charles Montgomery Burns	1113	+1105
9	0.278858	List of Simpsons releases by date	45	+36
10	0.270991	Marge Simpson	1116	+1106
11	0.225322	Non-English versions	241	+230
12	0.215936	Springfield's State	77	+65
13	0.184719	Ned Flanders	295	+282
14	0.155056	Bart Gets an F	7	-7
15	0.132003	Maggie Simpson	442	+427
16	0.129969	Julius Hibbert	601	+585
17	0.125921	Bart the General	4	-13
18	0.123387	List of Simpson Episodes by Production Code	311	+293
19	0.122418	The Simpsons Movie	100	+81
20	0.120586	Krusty the Clown	527	+507
21	0.117899	Seymour Skinner	258	+237
22	0.115788	Abraham Simpson	457	+435
23	0.114631	Clancy Wiggum	591	+568
24	0.109950	Milhouse Van Houten	176	+152
25	0.108299	Moe Szyslak	794	+769
26	0.106699	Lenny Leonard	1111	+1085
27	0.104229	Character Gallery	403	+376
28	0.103859	The Simpsons: Hit and Run	42	+14
29	0.099851	Simpsons Comics in the US	54	+25
30	0.095976	Bart Simpson (comic book series)	3	-27
31	0.092119	In-show deaths	151	+120
32	0.088615	Character Guide	604	+572
33	0.087848	Simpsons Comics in the UK	82	+49
34	0.081496	The Simpsons Movie/Credits	1114	+1080
35	0.080299	Nelson Muntz	213	+178
36	0.069687	Locations	-	*new*
37	0.067253	The Simpsons Game	108	+71
38	0.064173	Bart Has Two Mommies	75	+37
39	0.062532	Barney Gumble	1820	+1781
40	0.062343	Patty Bouvier	916	+876

A.4. Results of the Delicious Experiments

A.4.1. Method 1, Query "art"

	PEST score	Page title	tf rank	PEST change
1	0.011931	http://modinidesing.deviantart.com/art/power-rangers-70988278	6	+5
2	0.011602	http://www.newingtoncropsey.com/publications/americanarts.htm	4	+2
3	0.010996	http://ichigobleachcake.deviantart.com/art/Orange-Ranger-70995964	7	+4
4	0.010728	http://community.livejournal.com/jr__nal/3606652.html#cutid1	2	-2
5	0.010693	http://www.hirschlandadler.com/view_3.html?type=MODERN&id=94&num=6&artist=true	10	+5
6	0.010548	http://burnsyroxx.deviantart.com/art/Yellow-Overdrive-71025396	8	+2
7	0.010291	http://community.livejournal.com/jr__nal/3598752.html#cutid1	3	-4
8	0.010261	http://www.sculpturetools.com/sculptplace/cart/indexsculptplace.html	5	-3
9	0.010258	http://www.artofillusion.org/downloads	1	-8
10	0.010257	http://burnsyroxx.deviantart.com/art/Clare-White-Mystic-Ranger-71026202	9	-1
11	0.010257	http://www.etsy.com/shop.php?user_id=64040&order=&section_id=&page=3	12	+1
12	0.010257	http://www.franklinbowlesgallery.com/Shared_Elements/ArtistPages/Frolova/pages/frolova-home.html	11	-1
13	0.010257	http://www.parawire.com/copper.html	13	0
14	0.010257	http://www.goddess-dreams.com/	14	0
15	0.005249	http://www.manyaudiobooks.com/list.aspx?catID=2&gen=1	19	+4
16	0.005216	http://www.thismustbetheplace.org/	16	0
17	0.005189	http://www.manyaudiobooks.com/list.aspx?catID=2&page=2	21	+4
18	0.005152	http://www.nicolecardiff.com/	15	-3
19	0.005140	http://www.avoision.com/experiments/youare/	18	-1
20	0.005140	http://flickr.com/photos/samsa1973/	17	-3
21	0.005140	http://winstonsmith.com/	20	-1
22	0.005140	http://www.manyaudiobooks.com/list.aspx?catID=2&page=3	22	0
23	0.004604	http://bugartbysteven.com/	23	0
24	0.004289	http://www.photolucida.org/cm_winners.aspx?CMYear=2006	24	0
25	0.004118	http://www.jasonhackenwerth.com/pastprojects/index.html	25	0
26	0.004117	http://www.photolucida.org/cm_winners.aspx?CMYear=2007	26	0
27	0.003745	http://www.nathanhuang.com/index.html	27	0
28	0.003654	http://www.abcgallery.com/R/rivera/rivera74.html	36	+8
29	0.003547	http://whoisdan.com/	38	+9
30	0.003532	http://en.wikipedia.org/wiki/Plastic_People_of_the_Universe	32	+2
31	0.003494	http://myworld.ebay.co.uk/irish_art_group/	37	+6
32	0.003439	http://www.ultralab-paris.org/	40	+8
33	0.003435	http://www.publik.dk/hotsummer/resources/links/	34	+1
34	0.003435	http://artsdesignblog.com/page/5/	31	-3
35	0.003435	http://www.pushtoyproject.com/	42	+7
36	0.003435	http://www.zompist.com/howto2.htm	33	-3
37	0.003435	http://www.youtube.com/user/FREEwareDELAware	39	+2
38	0.003435	http://www.guardian.co.uk/music/gallery/2007/nov/14/dylan?picture=331263237	35	-3
39	0.003435	http://www.djtees.com/tshop/store/index.asp?gclid=CKrqtOHxhZACFRM6sgodehAwsw	41	+2
40	0.003435	http://www.library.txstate.edu/swwc/exhibits/koth.html	28	-12

A.4.2. Method 1, Query "photography"

	PEST score	Page title	tf rank	PEST change
1	0.023217	http://jaapscheeren.nl/	6	+5
2	0.021398	http://martinfengel.de/	8	+6
3	0.020525	http://www.cherryandmartin.com/artistDetail.php?id=12	9	+6
4	0.019960	http://www.manandeve.co.uk/exhib_media.php?id=13&im_id=274	10	+6
5	0.019545	http://www.hannover.de/madeingermany/madeingermany-en/artists/Annette_Kelm/index.html	11	+6
6	0.019226	http://www.tim-beach.com/f111.htm	3	-3
7	0.019220	http://www.suntimes.com/realchicago/	14	+7
8	0.019219	http://www.creativepro.com/story/feature/26148.html	1	-7
9	0.019218	http://www.nytimes.com/indexes/2007/12/02/style/t/index.html#pageName=02iconc	4	-5
10	0.019218	http://agm-foto.de/main.php?g2_itemId=711&g2_enterAlbum=0	7	-3
11	0.019218	http://jeanlecrenier.com/	12	+1
12	0.019218	http://ab-photoaday2007.blogspot.com/	2	-10
13	0.019218	http://www.photoaxe.com/how-to-light-portraits-in-photography-part-ii/	5	-8
14	0.019218	http://www.ephotozine.com/article/Photography-at-night	13	-1
15	0.011254	http://www.photoaxe.com/how-to-light-portraits-in-photography/	21	+6
16	0.010882	http://www.jenniferpearsonphotography.com/index2.php	22	+6
17	0.010030	http://www.snugabugportraits.com/about/productline.php?category=2&sc=28	23	+6
18	0.009623	http://www.zonezero.com/magazine/essays/diegotime/time.html#	18	0
19	0.009621	http://digital-photography-school.com/blog/14-tips-for-great-candlelight-photography/	15	-4
20	0.009621	http://www.stockxpert.com/info.phtml?f=help&s=11_2	20	0
21	0.009621	http://www.tsewhat.com/	17	-4
22	0.009621	http://blog.bluefire.tv/?p=15#more-15	16	-6
23	0.009621	http://www.imagecatalog.com/sell.php?icsid=7fc3770d2f2bae8c95929ac15aa32814	19	-4
24	0.009621	http://www.macymills.com/index2.php	24	0
25	0.009621	http://luminous-landscape.com/forum/index.php?act=idx	25	0
26	0.006719	http://photo.net/bboard/q-and-a-fetch-msg?msg_id=00NQEv	33	+7
27	0.006428	http://www.searchles.com/search/tag/photography	28	+1
28	0.006425	http://www.dpreview.com/reviews/FujifilmS8000fd/	37	+9
29	0.006423	http://www.corinneday.co.uk/home.php	27	-2
30	0.006423	http://www.flickr.com/groups/thenatureconservancy/pool/page40/	34	+4
31	0.006423	http://urbanisolation.livejournal.com/16925.html#cutid1	30	-1
32	0.006423	http://digitalphotography.inthenewstonight.com/2007/11/29/corbis-offers-bloggers-free-photos-with-ads-reuters-via-yahoo-news/	35	+3
33	0.006422	http://www.photobetty.com/jeanchung	29	-4
34	0.006422	http://www.focuscontemporary.co.za/	26	-8
35	0.006422	http://community.livejournal.com/foto_decadent/1651103.html	31	-4
36	0.006422	http://blog.xuite.net/miau/zgundam	32	-4
37	0.006422	http://photo.net/bboard/q-and-a?topic_id=1550&category=Portrait%2C+Technique	36	-1
38	0.006422	http://www.outdoorphotographer.com/content/2007/nov/dh_magic.shtml	38	0
39	0.005368	http://community.livejournal.com/foto_decadent/1650907.html	41	+2
40	0.005232	http://nikas-culinaria.com/food-photo-101/	47	+7

A.4.3. Method 2, Query "art"

	PEST score	Page title	tf rank	PEST change
1	0.009183	http://www.artofillusion.org/downloads	1	0
2	0.008891	http://community.livejournal.com/jr__nal/3606652.html#cutid1	2	0
3	0.008858	http://community.livejournal.com/jr__nal/3598752.html#cutid1	3	0
4	0.008748	http://www.newingtoncropsey.com/publications/americanarts.htm	4	0
5	0.008701	http://www.sculpturetools.com/sculptplace/cart/indexsculptplace.html	5	0
6	0.008520	http://modinidesing.deviantart.com/art/power-rangers-70988278	6	0
7	0.008512	http://ichigobleachcake.deviantart.com/art/Orange-Ranger-70995964	7	0
8	0.008501	http://burnsyroxx.deviantart.com/art/Yellow-Overdrive-71025396	8	0
9	0.008494	http://burnsyroxx.deviantart.com/art/Clare-White-Mystic-Ranger-71026202	9	0
10	0.008480	http://www.hirschlandadler.com/view_3.html?type=MODERN&id=94&num=6&artist=true	10	0
11	0.008425	http://www.franklinbowlesgallery.com/Shared_Elements/ArtistPages/Frolova/pages/frolova-home.html	11	0
12	0.008400	http://www.etsy.com/shop.php?user_id=64040&order=&section_id=&page=3	12	0
13	0.008386	http://www.parawire.com/copper.html	13	0
14	0.008377	http://www.goddess-dreams.com/	14	0
15	0.004538	http://bugartbysteven.com/	23	+8
16	0.004386	http://www.nicolecardiff.com/	15	-1
17	0.004271	http://www.thismustbetheplace.org/	16	-1
18	0.004267	http://flickr.com/photos/samsa1973/	17	-1
19	0.004250	http://www.avoision.com/experiments/youare/	18	-1
20	0.004242	http://www.manyaudiobooks.com/list.aspx?catID=2&gen=1	19	-1
21	0.004241	http://winstonsmith.com/	20	-1
22	0.004194	http://www.manyaudiobooks.com/list.aspx?catID=2&page=2	21	-1
23	0.004192	http://www.manyaudiobooks.com/list.aspx?catID=2&page=3	22	-1
24	0.003580	http://www.photolucida.org/cm_winners.aspx?CMYear=2006	24	0
25	0.003462	http://www.jasonhackenwerth.com/pastprojects/index.html	25	0
26	0.003393	http://www.photolucida.org/cm_winners.aspx?CMYear=2007	26	0
27	0.003116	http://www.library.txstate.edu/swwc/exhibits/koth.html	28	+1
28	0.003072	http://www.nathanhuang.com/index.html	27	-1
29	0.003066	http://www.pinktentacle.com/2007/09/kiri-origami-creatures/	29	0
30	0.003059	http://www.eastsidearts.org/newsletter/music	30	0
31	0.003017	http://www.ashleyhopeart.com/	43	+12
32	0.003011	http://artsdesignblog.com/page/5/	31	-1
33	0.003007	http://en.wikipedia.org/wiki/Plastic_People_of_the_Universe	32	-1
34	0.002981	http://www.zompist.com/howto2.htm	33	-1
35	0.002964	http://www.publik.dk/hotsummer/resources/links/	34	-1
36	0.002924	http://www.guardian.co.uk/music/gallery/2007/nov/14/dylan?picture=331263237	35	-1
37	0.002917	http://www.abcgallery.com/R/rivera/rivera74.html	36	-1
38	0.002913	http://myworld.ebay.co.uk/irish_art_group/	37	-1
39	0.002902	http://whoisdan.com/	38	-1
40	0.002891	http://www.youtube.com/user/FREEwareDELAware	39	-1

A.4.4. Method 2, Query "photography"

	PEST score	Page title	tf rank	PEST change
1	0.017971	http://www.creativepro.com/story/feature/26148.html	1	0
2	0.016119	http://ab-photoaday2007.blogspot.com/	2	0
3	0.015999	http://www.tim-beach.com/f111.htm	3	0
4	0.015964	http://www.nytimes.com/indexes/2007/12/02/style/t/index.html#pageName=02iconc	4	0
5	0.015923	http://www.photoaxe.com/how-to-light-portraits-in-photography-part-ii/	5	0
6	0.015813	http://agm-foto.de/main.php?g2_itemId=711&g2_enterAlbum=0	7	+1
7	0.015768	http://jaapscheeren.nl/	6	-1
8	0.015733	http://martinfengel.de/	8	0
9	0.015725	http://www.cherryandmartin.com/artistDetail.php?id=12	9	0
10	0.015717	http://www.manandeve.co.uk/exhib_media.php?id=13&im_id=274	10	0
11	0.015707	http://www.hannover.de/madeingermany/madeingermany-en/artists/Annette_Kelm/index.html	11	0
12	0.015696	http://jeanlecrenier.com/	12	0
13	0.015643	http://www.ephotozine.com/article/Photography-at-night	13	0
14	0.015625	http://www.suntimes.com/realchicago/	14	0
15	0.009978	http://digital-photography-school.com/blog/14-tips-for-great-candlelight-photography/	15	0
16	0.008620	http://blog.bluefire.tv/?p=15#more-15	16	0
17	0.008538	http://www.tsewhat.com/	17	0
18	0.008392	http://www.zonezero.com/magazine/essays/diegotime/time.html#	18	0
19	0.008303	http://www.imagecatalog.com/sell.php?icsid=7fc3770d2f2bae8c95929ac15aa32814	19	0
20	0.008284	http://www.stockxpert.com/info.phtml?f=help&s=11_2	20	0
21	0.008225	http://www.photoaxe.com/how-to-light-portraits-in-photography/	21	0
22	0.008069	http://www.jenniferpearsonphotography.com/index2.php	22	0
23	0.007892	http://www.snugabugportraits.com/about/productline.php?category=2&sc=28	23	0
24	0.007888	http://www.macymills.com/index2.php	24	0
25	0.007822	http://luminous-landscape.com/forum/index.php?act=idx	25	0
26	0.006528	http://www.focuscontemporary.co.za/	26	0
27	0.006304	http://www.corinneday.co.uk/home.php	27	0
28	0.006201	http://www.searchles.com/search/tag/photography	28	0
29	0.005905	http://www.photobetty.com/jeanchung	29	0
30	0.005878	http://urbanisolation.livejournal.com/16925.html#cutid1	30	0
31	0.005836	http://community.livejournal.com/foto_decadent/1651103.html	31	0
32	0.005539	http://blog.xuite.net/miau/zgundam	32	0
33	0.005504	http://www.flickr.com/groups/thenatureconservancy/pool/page40/	34	+1
34	0.005503	http://photo.net/bboard/q-and-a-fetch-msg?msg_id=00NQEv	33	-1
35	0.005332	http://digitalphotography.inthenewstonight.com/2007/11/29/corbis-offers-bloggers-free-photos-with-ads-reuters-via-yahoo-news/	35	0
36	0.005237	http://photo.net/bboard/q-and-a?topic_id=1550&category=Portrait%2C+Technique	36	0
37	0.005232	http://www.dpreview.com/reviews/FujifilmS8000fd/	37	0
38	0.005211	http://www.outdoorphotographer.com/content/2007/nov/dh_magic.shtml	38	0
39	0.004865	http://smad.jmu.edu/dof/	39	0
40	0.004706	http://www.dpchallenge.com/lens.php	40	0

A.4.5. Method 3, Query "art"

	PEST score	Page title	tf rank	PEST change
1	0.021000	http://modinidesing.deviantart.com/art/power-rangers-70988278	1	0
2	0.021000	http://ichigobleachcake.deviantart.com/art/Orange-Ranger-70995964	2	0
3	0.021000	http://burnsyroxx.deviantart.com/art/Yellow-Overdrive-71025396	3	0
4	0.021000	http://burnsyroxx.deviantart.com/art/Clare-White-Mystic-Ranger-71026202	4	0
5	0.021000	http://www.franklinbowlesgallery.com/Shared_Elements/ArtistPages/Frolova/pages/frolova-home.html	5	0
6	0.008412	http://www.jasonhackenwerth.com/pastprojects/index.html	6	0
7	0.007650	http://www.nathanhuang.com/index.html	7	0
8	0.005837	http://www.ashleyhopeart.com/	8	0
9	0.005759	http://weburbanist.com/2007/11/29/painting-the-town-3-more-types-of-unusually-legal-urban-street-art/	26	+17
10	0.005742	http://theflowerdrumsong.blogspot.com/	9	-1
11	0.005435	http://www.treeshark.com/Persptut.html	10	-1
12	0.005382	http://www.feric.com/	11	-1
13	0.005266	http://www.amyruppel.com/evanbharris_preview/index.html	12	-1
14	0.005266	http://www.riveramural.org/panel.asp?section=mural&key=1001&language=english	13	-1
15	0.005266	http://www.systemax.jp/sai/	14	-1
16	0.005266	http://www.animalky.org/	15	-1
17	0.005266	http://www.losaltos-downtown.org/overview_artswine.htm	16	-1
18	0.005266	http://www.mambo-bologna.org/file-sito/ita/index.html	17	-1
19	0.005263	http://dailypoetics.typepad.com/photos/business_cards_and_other_/p1010063_1.html	53	+34
20	0.005016	http://www.mangarevolution.com/	18	-2
21	0.004546	http://www.pirellical.com/thecal/calendar.html	19	-2
22	0.004394	http://aleksandradomanovic.com/anhedonia.htm	52	+30
23	0.004217	http://www.makoto-komatsu.com/	20	-3
24	0.004217	http://www.ubermorgen.com/CATALOGUE/	21	-3
25	0.003906	http://inventiontocompletehumanbeing.blogspot.com/	22	-3
26	0.003835	http://www.art-dept.com/artists/borsodi/	23	-3
27	0.003835	http://www.altx.com/thebody/	24	-3
28	0.003835	http://www.3vitv.com/media/great-ways-to-advertise/	25	-3
29	0.003826	http://www.unitseven.co.nz/	43	+14
30	0.003792	http://www.zefrank.com/snowflake/	46	+16
31	0.003721	http://webclipart.about.com/od/webclipartatoz/l/blclpexp.htm	27	-4
32	0.003571	http://mrtoledano.com/	28	-4
33	0.003517	http://gallery.artofgregmartin.com/	29	-4
34	0.003517	http://www.shepmedia.com/Paul'sCorner/Lite1.html	30	-4
35	0.003517	http://www.bumstein.com/art/	31	-4
36	0.003358	http://www.maninthedark.com/	32	-4
37	0.003253	http://www.clker.com/	57	+20
38	0.003248	http://www.zissou.com/main.html	33	-5
39	0.003248	http://linkoln.net/netartparade/	34	-5
40	0.003245	http://salavon.com/	88	+48

A.4.6. Method 3, Query "photography"

	PEST score	Page title	tf rank	PEST change
1	0.055842	http://jaapscheeren.nl/	1	0
2	0.055842	http://agm-foto.de/main.php?g2_itemId=711&g2_enterAlbum=0	2	0
3	0.055842	http://www.manandeve.co.uk/exhib_media.php?id=13&im_id=274	3	0
4	0.055842	http://www.hannover.de/madeingermany/madeingermany-en/artists/Annette_Kelm/index.html	4	0
5	0.027932	http://digital-photography-school.com/blog/14-tips-for-great-candlelight-photography/	5	0
6	0.027932	http://www.imagecatalog.com/sell.php?icsid=7fc3770d2f2bae8c95929ac15aa32814	6	0
7	0.027932	http://www.stockxpert.com/info.phtml?f=help&s=11_2	7	0
8	0.027932	http://martinfengel.de/	8	0
9	0.018629	http://blog.xuite.net/miau/zgundam	9	0
10	0.018629	http://www.outdoorphotographer.com/content/2007/nov/dh_magic.shtml	10	0
11	0.016872	http://www.goldprints.com/	15	+4
12	0.013977	http://www.geocities.jp/deal_steal/bui2	11	-1
13	0.011186	http://www.focuscontemporary.co.za/	12	-1
14	0.011186	http://pobox.upenn.edu/~davidtoc/calvin.html	13	-1
15	0.009325	http://www.phapak.net/	14	-1
16	0.007996	http://smad.jmu.edu/dof/	16	0
17	0.007996	http://www.callalillie.com/	17	0
18	0.007000	http://www.butzi.net/articles/toning.htm	18	0
19	0.007000	http://www.kurtstallaert.be/	19	0
20	0.006224	http://www.macymills.com/index2.php	20	0
21	0.005604	http://danielaglunz.com/?mod=commercial&page=2	21	0
22	0.005441	http://www.photoaxe.com/how-to-light-portraits-in-photography/	37	+15
23	0.005097	http://theflowerdrumsong.blogspot.com/	22	-1
24	0.005097	http://www.tonicbound.com/wordpress/	23	-1
25	0.005069	http://www.photoaxe.com/how-to-light-portraits-in-photography-part-ii/	36	+11
26	0.004674	http://www.photoethnography.com/ClassicCameras/index-frameset.html?Hasselblad500.html~mainFrame	24	-2
27	0.004674	http://www.altphotos.com/	25	-2
28	0.004009	http://www.trevor-jackson.com/	26	-2
29	0.004009	http://www.techroam.com/getflickr/	27	-2
30	0.004009	http://mjolk.com.au/	28	-2
31	0.003974	http://www.photographyknowhow.com/5/free-course/	29	-2
32	0.003872	http://www.tsewhat.com/	30	-2
33	0.003511	http://www.cherryandmartin.com/artistDetail.php?id=12	31	-2
34	0.003306	http://www.oliviabeasley.com/	32	-2
35	0.002960	http://wiki.photoblogs.org/wiki/Main_Page	33	-2
36	0.002913	http://www.flickr.com/photos/daveward/sets/241262/	34	-2
37	0.002559	http://www.jenniferpearsonphotography.com/index2.php	35	-2
38	0.002106	http://ab-photoaday2007.blogspot.com/	72	+34
39	0.002075	http://en.wikipedia.org/wiki/Joseph_Niepce	82	+43
40	0.001962	http://di63.toandonot.com/weddingcamerafortable.html	38	-2

A.4.7. Method 4, Query "art"

	PEST score	Page title	tf rank	PEST change
1	0.026831	http://modinidesing.deviantart.com/art/power-rangers-70988278	1	0
2	0.024729	http://ichigobleachcake.deviantart.com/art/Orange-Ranger-70995964	2	0
3	0.023721	http://burnsyroxx.deviantart.com/art/Yellow-Overdrive-71025396	3	0
4	0.023067	http://burnsyroxx.deviantart.com/art/Clare-White-Mystic-Ranger-71026202	4	0
5	0.023067	http://www.franklinbowlesgallery.com/Shared_Elements/ArtistPages/Frolova/pages/frolova-home.html	5	0
6	0.009241	http://www.jasonhackenwerth.com/pastprojects/index.html	6	0
7	0.008402	http://www.nathanhuang.com/index.html	7	0
8	0.006472	http://www.ashleyhopeart.com/	8	0
9	0.006319	http://theflowerdrumsong.blogspot.com/	9	0
10	0.006113	http://www.losaltos-downtown.org/overview_artswine.htm	16	+6
11	0.005982	http://www.treeshark.com/Persptut.html	10	-1
12	0.005942	http://www.feric.com/	11	-1
13	0.005802	http://www.amyruppel.com/evanbharris_preview/index.html	12	-1
14	0.005793	http://www.animalky.org/	15	+1
15	0.005785	http://www.systemax.jp/sai/	14	-1
16	0.005784	http://www.mambo-bologna.org/file-sito/ita/index.html	17	+1
17	0.005784	http://www.riveramural.org/panel.asp?section=mural&key=1001&language=english	13	-4
18	0.005512	http://www.mangarevolution.com/	18	0
19	0.005009	http://www.pirellical.com/thecal/calendar.html	19	0
20	0.004634	http://www.makoto-komatsu.com/	20	0
21	0.004631	http://www.ubermorgen.com/CATALOGUE/	21	0
22	0.004304	http://inventiontocompletehumanbeing.blogspot.com/	22	0
23	0.004237	http://www.art-dept.com/artists/borsodi/	23	0
24	0.004214	http://www.altx.com/thebody/	24	0
25	0.004213	http://www.3vitv.com/media/great-ways-to-advertise/	25	0
26	0.004184	http://weburbanist.com/2007/11/29/painting-the-town-3-more-types-of-unusually-legal-urban-street-art/	26	0
27	0.004087	http://webclipart.about.com/od/webclipartatoz/l/blclpexp.htm	27	0
28	0.003940	http://mrtoledano.com/	28	0
29	0.003866	http://gallery.artofgregmartin.com/	29	0
30	0.003863	http://www.shepmedia.com/Paul'sCorner/Lite1.html	30	0
31	0.003863	http://www.bumstein.com/art/	31	0
32	0.003689	http://www.maninthedark.com/	32	0
33	0.003626	http://www.zissou.com/main.html	33	0
34	0.003581	http://linkoln.net/netartparade/	34	0
35	0.003577	http://misprintedtype.com/v3/	36	+1
36	0.003530	http://www.thetemples.org/	35	-1
37	0.003481	http://www.mlaproductions.com/	37	0
38	0.003413	http://www.tenbyten.net/actions.html	38	0
39	0.003384	http://www.paulsermon.org/	40	+1
40	0.003354	http://www.eigelb.at/HP/Links/SpecialEffects/Grappa/DelayedTrace/	39	-1

A.4.8. Method 4, Query "photography"

	PEST score	Page title	tf rank	PEST change
1	0.063562	http://jaapscheeren.nl/	1	0
2	0.058893	http://www.manandeve.co.uk/exhib_media.php?id=13&im_id=274	3	+1
3	0.057667	http://www.hannover.de/madeingermany/madeingermany-en/artists/Annette_Kelm/index.html	4	+1
4	0.057650	http://agm-foto.de/main.php?g2_itemId=711&g2_enterAlbum=0	2	-2
5	0.032026	http://martinfengel.de/	8	+3
6	0.028837	http://digital-photography-school.com/blog/14-tips-for-great-candlelight-photography/	5	-1
7	0.028836	http://www.stockxpert.com/info.phtml?f=help&s=11_2	7	0
8	0.028836	http://www.imagecatalog.com/sell.php?icsid=7fc3770d2f2bae8c95929ac15aa32814	6	-2
9	0.019232	http://blog.xuite.net/miau/zgundam	9	0
10	0.019232	http://www.outdoorphotographer.com/content/2007/nov/dh_magic.shtml	10	0
11	0.017418	http://www.goldprints.com/	15	+4
12	0.014430	http://www.geocities.jp/deal_steal/bui2	11	-1
13	0.011552	http://pobox.upenn.edu/~davidtoc/calvin.html	13	0
14	0.011548	http://www.focuscontemporary.co.za/	12	-2
15	0.009698	http://www.phapak.net/	14	-1
16	0.008256	http://www.callalillie.com/	17	+1
17	0.008255	http://smad.jmu.edu/dof/	16	-1
18	0.007226	http://www.kurtstallaert.be/	19	+1
19	0.007226	http://www.butzi.net/articles/toning.htm	18	-1
20	0.006536	http://www.cherryandmartin.com/artistDetail.php?id=12	31	+11
21	0.006426	http://www.macymills.com/index2.php	20	-1
22	0.005785	http://danielaglunz.com/?mod=commercial&page=2	21	-1
23	0.005262	http://theflowerdrumsong.blogspot.com/	22	-1
24	0.005262	http://www.tonicbound.com/wordpress/	23	-1
25	0.005196	http://www.photoaxe.com/how-to-light-portraits-in-photography/	37	+12
26	0.004930	http://synergy2.sourceforge.net/	-	*new*
27	0.004825	http://www.altphotos.com/	25	-2
28	0.004825	http://www.photoethnography.com/ClassicCameras/index-frameset.html?Hasselblad500.html~mainFrame	24	-4
29	0.004185	http://www.trevor-jackson.com/	26	-3
30	0.004140	http://www.techroam.com/getflickr/	27	-3
31	0.004140	http://mjolk.com.au/	28	-3
32	0.004103	http://www.photographyknowhow.com/5/free-course/	29	-3
33	0.003997	http://www.tsewhat.com/	30	-3
34	0.003432	http://www.oliviabeasley.com/	32	-2
35	0.003311	http://www.photoaxe.com/how-to-light-portraits-in-photography-part-ii/	36	+1
36	0.003066	http://www.jenniferpearsonphotography.com/index2.php	35	-1
37	0.003056	http://wiki.photoblogs.org/wiki/Main_Page	33	-4
38	0.003007	http://www.flickr.com/photos/daveward/sets/241262/	34	-4
39	0.002331	http://nikas-culinaria.com/food-photo-101/	79	+40
40	0.002044	http://www.kengarland.co.uk/	42	+2

A.4.9. Method 5, Query "art"

	PEST score	Page title	tf rank	PEST change
1	0.024398	http://www.franklinbowlesgallery.com/Shared_Elements/ArtistPages/Frolova/pages/frolova-home.html	5	+4
2	0.024398	http://modinidesing.deviantart.com/art/power-rangers-70988278	1	-1
3	0.024398	http://ichigobleachcake.deviantart.com/art/Orange-Ranger-70995964	2	-1
4	0.024398	http://burnsyroxx.deviantart.com/art/Yellow-Overdrive-71025396	3	-1
5	0.024398	http://burnsyroxx.deviantart.com/art/Clare-White-Mystic-Ranger-71026202	4	-1
6	0.009774	http://www.jasonhackenwerth.com/pastprojects/index.html	6	0
7	0.008888	http://www.nathanhuang.com/index.html	7	0
8	0.006781	http://www.ashleyhopeart.com/	8	0
9	0.006672	http://theflowerdrumsong.blogspot.com/	9	0
10	0.006314	http://www.treeshark.com/Persptut.html	10	0
11	0.006253	http://www.feric.com/	11	0
12	0.006119	http://www.systemax.jp/sai/	14	+2
13	0.006118	http://www.amyruppel.com/evanbharris_preview/index.html	12	-1
14	0.006118	http://www.riveramural.org/panel.asp?section=mural&key=1001&language=english	13	-1
15	0.006118	http://www.animalky.org/	15	0
16	0.006118	http://www.losaltos-downtown.org/overview_artswine.htm	16	0
17	0.006118	http://www.mambo-bologna.org/file-sito/ita/index.html	17	0
18	0.005829	http://www.mangarevolution.com/	18	0
19	0.005281	http://www.pirellical.com/thecal/calendar.html	19	0
20	0.004899	http://www.makoto-komatsu.com/	20	0
21	0.004899	http://www.ubermorgen.com/CATALOGUE/	21	0
22	0.004538	http://inventiontocompletehumanbeing.blogspot.com/	22	0
23	0.004456	http://www.art-dept.com/artists/borsodi/	23	0
24	0.004456	http://www.altx.com/thebody/	24	0
25	0.004456	http://www.3vitv.com/media/great-ways-to-advertise/	25	0
26	0.004424	http://weburbanist.com/2007/11/29/painting-the-town-3-more-types-of-unusually-legal-urban-street-art/	26	0
27	0.004421	http://webclipart.about.com/od/webclipartatoz/l/blclpexp.htm	27	0
28	0.004182	http://africanpainters.blogspot.com/	284	+256
29	0.004149	http://mrtoledano.com/	28	-1
30	0.004087	http://www.shepmedia.com/Paul'sCorner/Lite1.html	30	0
31	0.004087	http://www.bumstein.com/art/	31	0
32	0.004086	http://gallery.artofgregmartin.com/	29	-3
33	0.003922	http://vagos.es/showthread.php?t=126518	602	+569
34	0.003902	http://www.maninthedark.com/	32	-2
35	0.003805	http://www.creativetechs.com/iq/build_animated_gifs_in_photoshop.html	-	*new*
36	0.003791	http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html	-	*new*
37	0.003789	http://www.parsleysoup.co.uk/	-	*new*
38	0.003774	http://linkoln.net/netartparade/	34	-4
39	0.003774	http://www.zissou.com/main.html	33	-6
40	0.003733	http://www.thetemples.org/	35	-5

A.4.10. Method 5, Query "photography"

	PEST score	Page title	tf rank	PEST change
1	0.060654	http://agm-foto.de/main.php?g2_itemId=711&g2_enterAlbum=0	2	+1
2	0.060502	http://www.manandeve.co.uk/exhib_media.php?id=13&im_id=274	3	+1
3	0.060502	http://www.hannover.de/madeingermany/madeingermany-en/artists/Annette_Kelm/index.html	4	+1
4	0.060501	http://jaapscheeren.nl/	1	-3
5	0.030265	http://digital-photography-school.com/blog/14-tips-for-great-candlelight-photography/	5	0
6	0.030264	http://www.imagecatalog.com/sell.php?icsid=7fc3770d2f2bae8c95929ac15aa32814	6	0
7	0.030263	http://www.stockxpert.com/info.phtml?f=help&s=11_2	7	0
8	0.030263	http://martinfengel.de/	8	0
9	0.020203	http://blog.xuite.net/miau/zgundam	9	0
10	0.020183	http://www.outdoorphotographer.com/content/2007/nov/dh_magic.shtml	10	0
11	0.015149	http://www.geocities.jp/deal_steal/bui2	11	0
12	0.012120	http://www.focuscontemporary.co.za/	12	0
13	0.012119	http://pobox.upenn.edu/~davidtoc/calvin.html	13	0
14	0.010104	http://www.phapak.net/	14	0
15	0.009012	http://www.goldprints.com/	15	0
16	0.008726	http://www.callalillie.com/	17	+1
17	0.008664	http://smad.jmu.edu/dof/	16	-1
18	0.007584	http://www.butzi.net/articles/toning.htm	18	0
19	0.007584	http://www.kurtstallaert.be/	19	0
20	0.006744	http://www.macymills.com/index2.php	20	0
21	0.006072	http://danielaglunz.com/?mod=commercial&page=2	21	0
22	0.005522	http://theflowerdrumsong.blogspot.com/	22	0
23	0.005522	http://www.tonicbound.com/wordpress/	23	0
24	0.005064	http://www.photoethnography.com/ClassicCameras/index-frameset.html?Hasselblad500.html~mainFrame	24	0
25	0.005064	http://www.altphotos.com/	25	0
26	0.004344	http://www.trevor-jackson.com/	26	0
27	0.004344	http://www.techroam.com/getflickr/	27	0
28	0.004344	http://mjolk.com.au/	28	0
29	0.004306	http://www.photographyknowhow.com/5/free-course/	29	0
30	0.004195	http://www.tsewhat.com/	30	0
31	0.003923	http://africanpainters.blogspot.com/	211	+180
32	0.003804	http://www.cherryandmartin.com/artistDetail.php?id=12	31	-1
33	0.003724	http://www.creativetechs.com/iq/build_animated_gifs_in_photoshop.html	566	+533
34	0.003694	http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html	-	*new*
35	0.003692	http://www.parsleysoup.co.uk/	-	*new*
36	0.003692	http://vagos.es/showthread.php?t=126518	-	*new*
37	0.003604	http://www.oliviabeasley.com/	32	-5
38	0.003207	http://wiki.photoblogs.org/wiki/Main_Page	33	-5
39	0.003181	http://www.flickr.com/photos/daveward/sets/241262/	34	-5
40	0.002773	http://www.jenniferpearsonphotography.com/index2.php	35	-5

A.4.11. Method 6, Query "art"

	PEST score	Page title	tf rank	PEST change
1	0.027711	http://modinidesing.deviantart.com/art/power-rangers-70988278	1	0
2	0.025540	http://ichigobleachcake.deviantart.com/art/Orange-Ranger-70995964	2	0
3	0.024499	http://burnsyroxx.deviantart.com/art/Yellow-Overdrive-71025396	3	0
4	0.023824	http://www.franklinbowlesgallery.com/Shared_Elements/ArtistPages/Frolova/pages/frolova-home.html	5	+1
5	0.023824	http://burnsyroxx.deviantart.com/art/Clare-White-Mystic-Ranger-71026202	4	-1
6	0.009544	http://www.jasonhackenwerth.com/pastprojects/index.html	6	0
7	0.008678	http://www.nathanhuang.com/index.html	7	0
8	0.006652	http://www.ashleyhopeart.com/	8	0
9	0.006525	http://theflowerdrumsong.blogspot.com/	9	0
10	0.006172	http://www.treeshark.com/Persptut.html	10	0
11	0.006134	http://www.feric.com/	11	0
12	0.006034	http://www.losaltos-downtown.org/overview_artswine.htm	16	+4
13	0.005985	http://www.amyruppel.com/evanbharris_preview/index.html	12	-1
14	0.005979	http://www.animalky.org/	15	+1
15	0.005975	http://www.systemax.jp/sai/	14	-1
16	0.005974	http://www.mambo-bologna.org/file-sito/ita/index.html	17	+1
17	0.005974	http://www.riveramural.org/panel.asp?section=mural&key=1001&language=english	13	-4
18	0.005693	http://www.mangarevolution.com/	18	0
19	0.005163	http://www.pirellical.com/thecal/calendar.html	19	0
20	0.004786	http://www.makoto-komatsu.com/	20	0
21	0.004784	http://www.ubermorgen.com/CATALOGUE/	21	0
22	0.004443	http://inventiontocompletehumanbeing.blogspot.com/	22	0
23	0.004366	http://www.art-dept.com/artists/borsodi/	23	0
24	0.004351	http://www.altx.com/thebody/	24	0
25	0.004351	http://www.3vitv.com/media/great-ways-to-advertise/	25	0
26	0.004315	http://webclipart.about.com/od/webclipartatoz/l/blclpexp.htm	27	+1
27	0.004298	http://weburbanist.com/2007/11/29/painting-the-town-3-more-types-of-unusually-legal-urban-street-art/	26	-1
28	0.004067	http://mrtoledano.com/	28	0
29	0.003992	http://gallery.artofgregmartin.com/	29	0
30	0.003990	http://www.shepmedia.com/Paul'sCorner/Lite1.html	30	0
31	0.003990	http://www.bumstein.com/art/	31	0
32	0.003810	http://www.maninthedark.com/	32	0
33	0.003718	http://www.zissou.com/main.html	33	0
34	0.003690	http://linkoln.net/netartparade/	34	0
35	0.003676	http://misprintedtype.com/v3/	36	+1
36	0.003646	http://www.thetemples.org/	35	-1
37	0.003571	http://www.mlaproductions.com/	37	0
38	0.003525	http://www.tenbyten.net/actions.html	38	0
39	0.003474	http://www.paulsermon.org/	40	+1
40	0.003457	http://www.eigelb.at/HP/Links/SpecialEffects/Grappa/DelayedTrace/	39	-1

A.4.12. Method 6, Query "photography"

	PEST score	Page title	tf rank	PEST change
1	0.063541	http://jaapscheeren.nl/	1	0
2	0.060198	http://www.manandeve.co.uk/exhib_media.php?id=13&im_id=274	3	+1
3	0.059763	http://agm-foto.de/main.php?g2_itemId=711&g2_enterAlbum=0	2	-1
4	0.059634	http://www.hannover.de/madeingermany/madeingermany-en/artists/Annette_Kelm/index.html	4	0
5	0.031099	http://martinfengel.de/	8	+3
6	0.029824	http://digital-photography-school.com/blog/14-tips-for-great-candlelight-photography/	5	-1
7	0.029823	http://www.imagecatalog.com/sell.php?icsid=7fc3770d2f2bae8c95929ac15aa32814	6	-1
8	0.029822	http://www.stockxpert.com/info.phtml?f=help&s=11_2	7	-1
9	0.019909	http://blog.xuite.net/miau/zgundam	9	0
10	0.019889	http://www.outdoorphotographer.com/content/2007/nov/dh_magic.shtml	10	0
11	0.014926	http://www.geocities.jp/deal_steal/bui2	11	0
12	0.011944	http://pobox.upenn.edu/~davidtoc/calvin.html	13	+1
13	0.011943	http://www.focuscontemporary.co.za/	12	-1
14	0.010012	http://www.phapak.net/	14	0
15	0.008881	http://www.goldprints.com/	15	0
16	0.008578	http://www.callalillie.com/	17	+1
17	0.008538	http://smad.jmu.edu/dof/	16	-1
18	0.007473	http://www.kurtstallaert.be/	19	+1
19	0.007473	http://www.butzi.net/articles/toning.htm	18	-1
20	0.006646	http://www.macymills.com/index2.php	20	0
21	0.005984	http://danielaglunz.com/?mod=commercial&page=2	21	0
22	0.005442	http://theflowerdrumsong.blogspot.com/	22	0
23	0.005442	http://www.tonicbound.com/wordpress/	23	0
24	0.004990	http://www.altphotos.com/	25	+1
25	0.004990	http://www.photoethnography.com/ClassicCameras/index-frameset.html?Hasselblad500.html~mainFrame	24	-1
26	0.004965	http://www.cherryandmartin.com/artistDetail.php?id=12	31	+5
27	0.004317	http://www.trevor-jackson.com/	26	-1
28	0.004282	http://www.techroam.com/getflickr/	27	-1
29	0.004281	http://mjolk.com.au/	28	-1
30	0.004243	http://www.photographyknowhow.com/5/free-course/	29	-1
31	0.004134	http://www.tsewhat.com/	30	-1
32	0.003554	http://www.oliviabeasley.com/	32	0
33	0.003160	http://wiki.photoblogs.org/wiki/Main_Page	33	0
34	0.003131	http://www.flickr.com/photos/daveward/sets/241262/	34	0
35	0.002826	http://www.jenniferpearsonphotography.com/index2.php	35	0
36	0.002573	http://www.photoaxe.com/how-to-light-portraits-in-photography-part-ii/	36	0
37	0.002536	http://africanpainters.blogspot.com/	211	+174
38	0.002521	http://www.photoaxe.com/how-to-light-portraits-in-photography/	37	-1
39	0.002340	http://www.creativetechs.com/iq/build_animated_gifs_in_photoshop.html	566	+527
40	0.002318	http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html	-	*new*

A.4.13. Method 7, Query "art"

	PEST score	Page title	tf rank	PEST change
1	0.033146	http://africanpainters.blogspot.com	41	+40
2	0.031947	http://www.photoethnography.com	-	*new*
3	0.031899	http://myevent.com	-	*new*
4	0.031729	http://www.parsleysoup.co.uk	-	*new*
5	0.031727	http://www.nmcnet.edu	-	*new*
6	0.031717	http://vagos.es	-	*new*
7	0.009082	http://twitter.com	1575	+1568
8	0.008446	http://www.facebook.com	1602	+1594
9	0.006032	http://pingmag.jp	1175	+1166
10	0.004587	http://www.goddess-dreams.com	8	-2
11	0.004256	http://www.sculpturetools.com	2	-9
12	0.004253	http://www.newingtoncropsey.com	1	-11
13	0.004253	http://www.hirschlandadler.com	6	-7
14	0.004253	http://www.franklinbowlesgallery.com	7	-7
15	0.004232	http://modinidesing.deviantart.com	3	-12
16	0.004232	http://ichigobleachcake.deviantart.com	4	-12
17	0.004232	http://burnsyroxx.deviantart.com	5	-12
18	0.003944	http://www.youtube.com	1634	+1616
19	0.003756	http://www.flickr.com	1411	+1392
20	0.003586	http://feeds.feedburner.com	1494	+1474
21	0.002728	http://www.amazon.com	1353	+1332
22	0.002692	http://beinart.org	39	+17
23	0.002620	http://www.google.com.mx	828	+805
24	0.002564	http://www.google.com	1630	+1606
25	0.002347	http://www.usbgadgets.org	-	*new*
26	0.002338	http://mckdh.net	-	*new*
27	0.002331	http://www.addthis.com	-	*new*
28	0.002311	http://www.nicolecardiff.com	10	-18
29	0.002288	http://alinanimation.blogspot.com	9	-20
30	0.002174	http://www.klompching.com	13	-17
31	0.002159	http://www.thismustbetheplace.org	11	-20
32	0.002149	http://www.suzascaloraphotography.com	14	-18
33	0.002122	http://winstonsmith.com	12	-21
34	0.001967	http://en.wikipedia.org	1552	+1518
35	0.001929	http://www.farlowstudios.com	16	-19
36	0.001832	http://substrom.livejournal.com	15	-21
37	0.001830	http://www.mambo-bologna.org	17	-20
38	0.001782	http://www.photolucida.org	18	-20
39	0.001746	http://painting.about.com	96	+57
40	0.001725	http://www.myspace.com	1600	+1560

A.4.14. Method 7, Query "photography"

	PEST score	Page title	tf rank	PEST change
1	0.028591	http://www.photoethnography.com	171	+170
2	0.028301	http://africanpainters.blogspot.com	-	*new*
3	0.028288	http://myevent.com	-	*new*
4	0.027836	http://www.parsleysoup.co.uk	-	*new*
5	0.027835	http://www.nmcnet.edu	-	*new*
6	0.027826	http://vagos.es	-	*new*
7	0.027573	http://jeanlecrenier.com	6	-1
8	0.015533	http://agm-foto.de	4	-4
9	0.014831	http://ab-photoaday2007.blogspot.com	1	-8
10	0.013551	http://www.tim-beach.com	2	-8
11	0.013511	http://jaapscheeren.nl	3	-8
12	0.013511	http://martinfengel.de	5	-7
13	0.009207	http://urbanisolation.livejournal.com	17	+4
14	0.009086	http://publicenergy.co.uk	12	-2
15	0.008187	http://sergiorecabarren.com	13	-2
16	0.007838	http://twitter.com	-	*new*
17	0.007264	http://www.macymills.com	9	-8
18	0.007264	http://www.littlecupcakephotography.com	10	-8
19	0.007246	http://www.jenniferpearsonphotography.com	7	-12
20	0.007092	http://pingmag.jp	689	+669
21	0.006911	http://bbs.youthvision.cn	23	+2
22	0.006780	http://www.snugabugportraits.com	8	-14
23	0.006780	http://www.cynthiadamarphotography.com	11	-12
24	0.005924	http://www.facebook.com	873	+849
25	0.005163	http://www.usbgadgets.org	-	*new*
26	0.005153	http://mckdh.net	-	*new*
27	0.004564	http://smad.jmu.edu	19	-8
28	0.004539	http://www.photobetty.com	16	-12
29	0.004530	http://www.focuscontemporary.co.za	14	-15
30	0.004530	http://www.corinneday.co.uk	15	-15
31	0.004511	http://digitalphotography.inthenewstonight.com	18	-13
32	0.004264	http://creativecommons.org	672	+640
33	0.003817	http://www.microsoft.com	967	+934
34	0.003730	http://blog.bluefire.tv	20	-14
35	0.003673	http://www.worksongs.com	22	-13
36	0.003555	http://www.macromedia.com	-	*new*
37	0.003459	http://www.flickr.com	984	+947
38	0.003405	http://www.nowilaymedowntosleep.org	24	-14
39	0.003386	http://wallspaceseattle.com	21	-18
40	0.003381	http://www.amazon.com	978	+938

A.4.15. Method 8, Query "art"

	PEST score	Page title	tf rank	PEST change
1	0.017467	http://africanpainters.blogspot.com	41	+40
2	0.015475	http://myevent.com	-	*new*
3	0.015459	http://www.photoethnography.com	-	*new*
4	0.015434	http://www.nmcnet.edu	-	*new*
5	0.015386	http://www.parsleysoup.co.uk	-	*new*
6	0.015341	http://vagos.es	-	*new*
7	0.007418	http://www.goddess-dreams.com	8	+1
8	0.007211	http://modinidesing.deviantart.com	3	-5
9	0.007013	http://ichigobleachcake.deviantart.com	4	-5
10	0.006883	http://www.sculpturetools.com	2	-8
11	0.006877	http://www.newingtoncropsey.com	1	-10
12	0.006877	http://www.hirschlandadler.com	6	-6
13	0.006877	http://www.franklinbowlesgallery.com	7	-6
14	0.006867	http://burnsyroxx.deviantart.com	5	-9
15	0.003690	http://www.nicolecardiff.com	10	-5
16	0.003653	http://alinanimation.blogspot.com	9	-7
17	0.003458	http://www.klompching.com	13	-4
18	0.003455	http://www.suzascaloraphotography.com	14	-4
19	0.003454	http://www.thismustbetheplace.org	11	-8
20	0.003443	http://winstonsmith.com	12	-8
21	0.003014	http://www.youtube.com	1634	+1613
22	0.002969	http://www.farlowstudios.com	16	-6
23	0.002954	http://substrom.livejournal.com	15	-8
24	0.002905	http://www.mambo-bologna.org	17	-7
25	0.002782	http://www.facebook.com	1602	+1577
26	0.002659	http://www.photolucida.org	18	-8
27	0.002554	http://www.jasonhackenwerth.com	19	-8
28	0.002493	http://www.eastsidearts.org	21	-7
29	0.002493	http://www.palmalouca.com.br	33	+4
30	0.002470	http://www.projecttosurface.com	26	-4
31	0.002451	http://www.library.txstate.edu	20	-11
32	0.002416	http://www.kevart.com	22	-10
33	0.002388	http://www.pushtoyproject.com	29	-4
34	0.002366	http://www.mooserental.com	36	+2
35	0.002320	http://www.cubegallery.ca	35	0
36	0.002315	http://www.artndeed.com	31	-5
37	0.002313	http://twitter.com	1575	+1538
38	0.002312	http://www.riveramural.org	27	-11
39	0.002312	http://www.publik.dk	24	-15
40	0.002312	http://www.ktfgallery.com	32	-8

A.4.16. Method 8, Query "photography"

	PEST score	Page title	tf rank	PEST change
1	0.022932	http://jeanlecrenier.com	6	+5
2	0.022512	http://jaapscheeren.nl	3	+1
3	0.022044	http://ab-photoaday2007.blogspot.com	1	-2
4	0.021930	http://agm-foto.de	4	0
5	0.021594	http://martinfengel.de	5	0
6	0.021273	http://www.tim-beach.com	2	-4
7	0.014532	http://www.photoethnography.com	171	+164
8	0.014488	http://urbanisolation.livejournal.com	17	+9
9	0.014107	http://myevent.com	-	*new*
10	0.014080	http://africanpainters.blogspot.com	-	*new*
11	0.014000	http://www.nmcnet.edu	-	*new*
12	0.013964	http://www.parsleysoup.co.uk	-	*new*
13	0.013928	http://vagos.es	-	*new*
14	0.012196	http://publicenergy.co.uk	12	-2
15	0.011225	http://www.jenniferpearsonphotography.com	7	-8
16	0.011069	http://www.macymills.com	9	-7
17	0.011002	http://www.littlecupcakephotography.com	10	-7
18	0.010937	http://sergiorecabarren.com	13	-5
19	0.010805	http://www.snugabugportraits.com	8	-11
20	0.010653	http://www.cynthiadamarphotography.com	11	-9
21	0.007109	http://www.focuscontemporary.co.za	14	-7
22	0.007109	http://www.photobetty.com	16	-6
23	0.007109	http://www.corinneday.co.uk	15	-8
24	0.007100	http://digitalphotography.inthenewstonight.com	18	-6
25	0.005825	http://smad.jmu.edu	19	-6
26	0.005479	http://blog.bluefire.tv	20	-6
27	0.005456	http://bbs.youthvision.cn	23	-4
28	0.005425	http://www.worksongs.com	22	-6
29	0.005338	http://www.nowilaymedowntosleep.org	24	-5
30	0.005329	http://wallspaceseattle.com	21	-9
31	0.004566	http://www.cherryandmartin.com	28	-3
32	0.004277	http://www.tsewhat.com	25	-7
33	0.004276	http://www.elannashville.com	27	-6
34	0.004276	http://www.caple.co.uk	26	-8
35	0.004270	http://photoacute.com	29	-6
36	0.003661	http://www.mooserental.com	35	-1
37	0.003571	http://www.imagecatalog.com	33	-4
38	0.003569	http://www.phapak.net	34	-4
39	0.003568	http://www.m-peoplephotography.com	36	-3
40	0.003559	http://danielaglunz.com	32	-8

References

[1] Another tool for language recognition. URL `http://www.antlr.org`.

[2] The DOT language. URL `http://www.graphviz.org/doc/info/lang.html`.

[3] Apache Lucene. URL `http://lucene.apache.org`.

[4] Zitgist DataViewer. URL `http://zitgist.com/products/dataviewer/dataviewer.html`.

[5] S. Abdali and D. Wise. Experiments with quadtree representation of matrices. In P. Gianni, editor, *Symbolic and Algebraic Computation*, volume 358 of *Lecture Notes in Computer Science*, pages 96–108. Springer Berlin / Heidelberg, 1989.

[6] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *Int'l Conf. on Extending Database Technology*, 2002.

[7] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. FleXPath: Flexible structure and full-text querying for XML. In *ACM SIGMOD Int'l Conf. on Management of Data*, 2004.

[8] V. N. Anh and A. Moffat. Compression and an IR approach to XML retrieval. In *INEX Workshop*, 2002.

[9] P. Berkhin. A survey on PageRank computing. *Internet Mathematics*, 2(1), 2005.

[10] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3), 2005.

[11] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. Technical report, W3C, January 2007. URL `http://www.w3.org/TR/xquery/`.

[12] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Int'l World Wide Web Conf.*, 1998.

[13] F. Bry and K. Weiand. Flavors of KWQL, a keyword query language for a semantic wiki. In *Int'l Conf. on Current Trends in Theory and Practice of Computer Science*, 2010.

[14] K. Bryan and T. Leise. The $25,000,000,000 eigenvector: The linear algebra behind Google. *SIAM Review*, 48(3):569–581, 2006.

[15] D. Carmel, Y. Maarek, Y. Mass, N. Efraty, and G. Landau. An extension of the vector space model for querying XML documents via XML fragments. In *SIGIR Workshop on XML and Information Retrieval*, 2002.

[16] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *ACM SIGMOD Int'l Conf. on Management of Data*, 1996.

[17] Y. Y. Chen, Q. Gan, and T. Suel. I/O-efficient techniques for computing PageRank. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 549–557, New York, NY, USA, 2002. ACM.

[18] A. Collins and E. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, 82(6), 1975.

[19] F. Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6), 1997.

[20] A. Eisenberg and J. Melton. SQL: 1999, formerly known as SQL3. *SIGMOD Rec.*, 28 (1):131–138, 1999.

[21] G. H. Golub and C. F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition)*. The Johns Hopkins University Press, 3rd edition, October 1996.

[22] O. Gospodnetic and E. Hatcher. *Lucene in Action*. Manning, Greenwich, January 2005.

[23] D. A. Grossman and O. Frieder. *Information Retrieval: Algorithms and Heuristics*. Springer, 2nd edition, December 2004.

[24] R. Guha, R. McCool, and E. Miller. Semantic search. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 700–709, New York, NY, USA, 2003. ACM.

[25] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML joins. In *ACM SIGMOD Int'l Conf. on Management of Data*, 2002.

[26] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: ranked keyword search over XML documents. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 16–27, New York, NY, USA, 2003. ACM.

[27] T. Haveliwala. Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4), 2003.

[28] T. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing PageRank. Technical report, Stanford, 2003.

[29] G. Jeh and J. Widom. Scaling personalized web search. In *Int'l World Wide Web Conf.*, 2003.

[30] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing PageRank. Technical report, Stanford, 2003.

[31] A. N. Langville and C. D. Meyer. Deeper inside PageRank. *Internet Mathematics*, 1 (3):335–380, 2003.

[32] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*, volume 10, 1966.

[33] G. Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, April 2006.

[34] J.-L. Marignier. *Nicephore Niepce 1765-1833: L'invention de la photographie (Un savant, une epoque)*. Belin, 1999.

[35] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[36] J. Pokorný. Vector-oriented retrieval in XML data collections. In *Databáze, Texty, Specifikace a Objekty*, 2008.

[37] L. Prechelt. Technical opinion: comparing Java vs. C/C++ efficiency differences to interpersonal differences. *Commun. ACM*, 42(10):109–112, 1999.

[38] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. Technical report, W3C, January 2008. URL `http://www.w3.org/TR/rdf-sparql-query/`.

[39] M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in PageRank. *Advances in Neural Information Processing Systems*, 2, 2002.

[40] G. Salton. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[41] S. Schaffert, J. Eder, S. Grünwald, T. Kurz, R. Sint, and S. Stroka. KiWi – A platform for semantic social software. In C. Lange, S. Schaffert, H. Skaf-Molli, and M. Völkel, editors, *Fourth Workshop on Semantic Wikis*, 2009.

[42] T. Schlieder. Similarity search in XML data using cost-based query transformations. In *ACM SIGMOD Web and Databases Workshop*, 2001.

[43] T. Schlieder and H. Meuss. Querying and ranking XML documents. *Journal of the American Society for Information Science and Technology*, 53(6), 2002.

[44] D. Shasha, J. T.-L. Wang, H. Shan, and K. Zhang. Atreegrep: Approximate searching in unordered trees. In *Int'l Conf. on Scientific and Statistical Database Management*, 2002.

[45] K. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3), 1979.

[46] J. Tekli, R. Chbeir, and K. Yetongnon. An overview on XML similarity: Background, current trends and future directions. *Computer Science Review*, 3(3), 2009.

[47] R. Wetzker, C. Zimmermann, and C. Bauckhage. Analyzing social bookmarking systems: A del.icio.us cookbook. In *Proceedings of the ECAI 2008 Mining Social Data Workshop*, pages 26–30. IOS Press, 2008.

[48] R. Yang, P. Kalnis, and A. Tung. Similarity evaluation on tree-structured data. In *ACM SIGMOD Int'l Conf. on Management of Data*, 2005.