PEST: Term-Propagation over Wiki Structures as Eigenvector Computation

Klara Weiand, Fabian Kneißl, Tim Furche, and François Bry

Institute for Informatics, University of Munich,
Oettingenstraße 67, D-80538 München, Germany
`http://www.pms.ifi.lmu.de/`

Abstract. We present PEST, a novel approach to approximate querying of structured wiki data that exploits the structure of that data to propagate term weights between related wiki pages and tags. Based on the PEST matrix, eigenvectors representing the distribution of a term after propagation are computed. The result is an index which takes the document structure into account and can be used with standard document retrieval techniques. This article gives a detailed outline of the approach and gives first experimental results showing its viability.

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"*. In the semantic wiki KiWi, this can be achieved by the KWQL [5] query `ci(java lucene tag(introduction))`, where `ci` indicates wiki pages, see Section 3.2.

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 conventions (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"* (rather than being tagged with it). However, in this case, documents that follow the convention are not necessarily ranked higher than other matching documents.

(2) Some projects use the rich annotation and structuring mechanisms 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, and 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

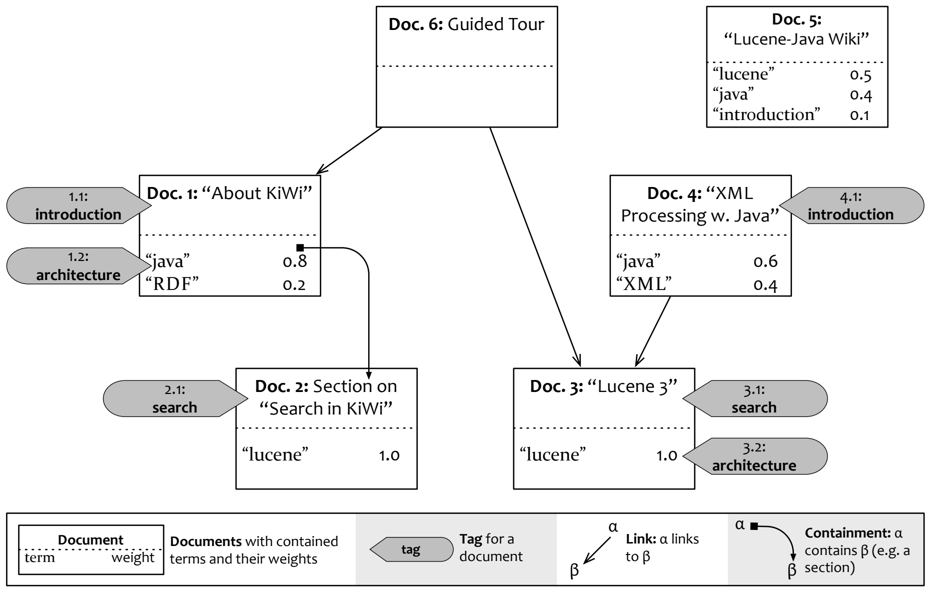

Fig. 1. Link and containment graph for a sample wiki

complex (even in a flexible query language such as KWQL). 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.

Fuzzy matching over words by means of, for example, stemming is an established technique widely used in Information Retrieval applications such as web search engines. Fuzzy matching over structure however, is only recently gaining attention as the amount of (semi-)structured data on the web increases. When a query explicitly imposes structural constraints on the selection, fuzzy matches are also returned where the structural constraints hold only approximately (e.g., a direct link is approximated by a chain of links).

In this article, we present PEST, short for term-**p**ropagation using **e**igenvector computation over wiki-**st**ructures, a novel approach to approximate or **fuzzy matching over structured data**. PEST 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 a wiki using eigenvector computation. The eigenvector computation is inspired by, but differs significantly from, Google's PageRank [4].

In contrast to many other fuzzy matching approaches (see Section 2), PEST relies solely on modifying term weights in the document index and requires no runtime query expansion, but can use existing document retrieval technologies such as Lucene. Nevertheless, it is able to solve all above described problems in the context of the semantic wiki KiWi.

To illustrate how PEST propagates term weights, consider again Figure 1. As for 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 two steps:

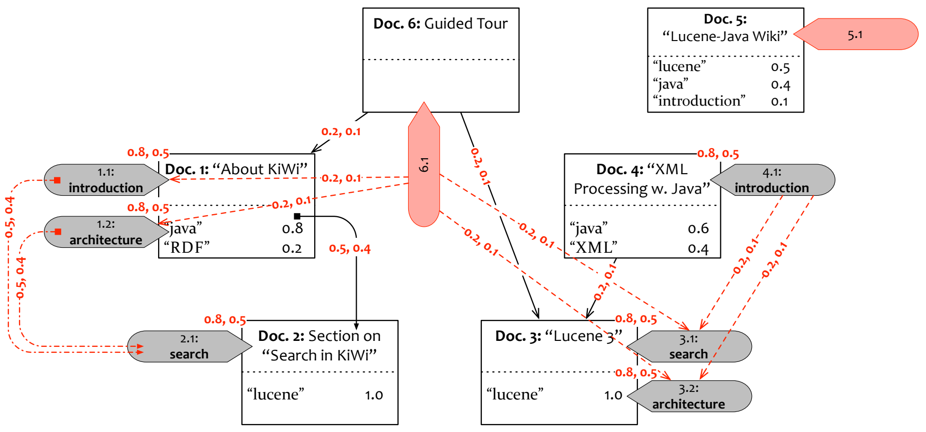

Fig. 2. Edge weights and virtual nodes and edges for Figure 1

(1) Weighted propagation graph: First, we extend and weight the graph of the wiki pages and their tags: 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.

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).

(2) "Informed Leap": The weighted propagation graph, however, 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": First, we transpose the weighted adjacency matrix of the weighted propagation graph and normalize it by the highest edge-weight sum over all documents (rather than for each document individually) to preserve differences in overall edge weight between documents. Second, the remaining probability together with a fixed leap parameter α (e.g., 30%) is used for an "informed leap" to an arbitrary node. The probability to go to a specific node A in such an "informed leap" is not random (as in the case of the original PageRank), but informed by the original term weight distribution: A page with a high term weight for term τ is more likely to be the target of a leap than a page with low term weight for τ.

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. A formal description of the PEST matrix computation is given in Section 5.

Finally, the eigenvectors of the PEST matrix for each term τ are combined to form the vector space representation (i.e., the document-term matrix) for the wiki pages and their tags. Section 6 presents, as an example, the computation and the resulting term weights for the wiki from Figure 1.

Keyword *queries* can be evaluated on this representation with any of the existing IR engines using a vector space model (e.g., Lucene). Queries mixing keywords and structure require an engine capable of combining keyword matches with structural constraints such as the KWQL engine.

Contributions

To summarize, PEST improves on existing fuzzy matching approaches for structured data (briefly summarized in Section 2) in the following aspects:

- It is based on a *simple, but flexible model for structured content* that captures a wide range of knowledge management systems and applications. We introduce the model in Section 4 and discuss how it can represent the core concepts of the semantic wiki KiWi, briefly recalled in Section 3.1. We also briefly recall KWQL (Section 3.2) to illustrate the need for a combination of structure and keyword queries.
- The main contribution of PEST is the PEST matrix for propagating term weights over structured data. The computation of that matrix for a given graph of structured content is formalized in Section 5.
  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 IR engine based on the vector space model (e.g., Lucene).
  Section 6 gives an extended example of the PEST matrix computation on the sample wiki from Figure 1.
- We prove formally in Section 5.3 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.

Though the results from Section 6 as well as further internal testing validate the PEST approach, there are a number of open issues summarized in Section 7.

2 Related Work: Fuzzy Matching on Structured Data

PEST differs from the majority of fuzzy matching approaches including those reviewed in the following in two important ways:

- It is designed for *graph-shaped data* rather than purely hierarchical data as most of the XML-based approaches discussed in the following.
- In essence, 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 vector space model. 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*.

Before we consider specific approaches, it is worth recalling that *fuzzy matching*—approaches to include not only strict matches, but also other results which are relevant but do not match the strict interpretation of the query—and *ranking* are closely related. Though they do not have to be used in conjunction, this is often the case, in particular to allow a fuzzy matching engine to differentiate looser results from results that adhere more strictly to the query.

While fuzzy matching is widely used in web search and other IR applications, conventional query languages for (semi-)structured data such as XQuery, SQL or SPARQL do not usually employ fuzzy matching or rank results. These languages have been applied to probabilistic data, but this is a distinct area from fuzzy querying: In probabilistic data management the data itself introduces uncertainty, in fuzzy matching uncertainty is introduced under the assumption that the user is also interested in matches that do not quite match her 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 to introducing IR methods to querying, for example in the form of extensions to conventional query languages, visual tools for exploratory search, extension of web keyword search to include (some) structure and keyword search over structured data. With the arrival of these techniques, the need for fuzzy querying that does not apply solely to individual terms or phrases but takes the data structure into account arises.

Approximate matching on data structure has been researched mainly in the context of XML data similarity [15]. A wide body of work in this area can be divided into three main classes of approaches:

Tree edit distance: Tree edit distance approaches, e.g., [10, 1, 2] extend the edit distance in such a way that not strings but trees are compared. A number of types of edit operations may be applied repeatedly in order to transform one XML document into another. The similarity between the documents can then be quantified through a cost function taking into account the number of steps and types of operations required.

In contrast to PEST, these approaches are hard to generalize to graph data, require a relaxation loop at query time, and require the evaluation of a (often quite considerable) number of relaxed queries whereas PEST's computation can be performed entirely at index time. The last effect is slightly ameliorated by novel top-k algorithms in [2]. Also it is not obvious how different edge types, as easily treated by PEST, affect tree edit distance.

Approximate tree matching: A small number of approaches modify existing matching algorithms to introduce certain degrees of freedom. In [13], direct relations in the query are allowed to be matched with indirect relations in the document. In [14], a document is considered a good approximate match if it and the query have few paths that are not common (a mismatching).

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 [14] 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.

Adapting the vector space model: Finally, the largest class of approaches aims, like PEST, to adapt the vector space model, a well-established IR technique, to the application on XML data. In the vector space model, documents and queries are represented as vectors of weights for each term; similarity is computed as the cosine angle between two vectors.

Pokorny et al. [12] represent paths and terms in an XML tree in a matrix instead of a vector, assigning weights to each combination of path and term. 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 [6] (and similarly [9]) document vectors are modified such that their elements are not weights for terms but rather weights for term and context pairs—the context of a term is the path in which it occurs. The vector then consists of a weight for each combination of term and context. Further, the cosine similarity measure is relaxed by computing context similarities which are integrated in the vector similarity measure.

Similarly, [13] and, later, [11] use tree embeddings combined with a vector space representation of XML elements.

Activation propagation is used in [3] for fuzzy matching over structure. 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 is used to realize approximate matching over structure. However, in this approach, propagation happens upon query evaluation and is unidirectional. Like the other approaches in this class, it is also limited to tree-shaped data.

Outside of XML, one widely-used method where structural relationship is used for fuzzy matching is the use of anchor-tags in web search [4]. The anchor text of a link to a web page is treated as if it was part of the text of that web page even if it does not appear there. However, the application of this approach is limited to anchor tags and does not apply to general graphs or generalize to different link types.

3 Preliminaries

3.1 KiWi

KiWi[1] is a semantic wiki with extended functionality in the areas of information extraction, personalization, reasoning, and querying. KiWi relies on a simple, 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

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

a conventional structuring of 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 of cycles, and thus a content item can be seen as a directed acyclic graph (of content items). **Links** are simple hypertext links and can be used for relating content items to each other or to external web sites.

Annotations 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. Though KiWi supports various types of annotations ranging from informal, freely chosen tags, to semi-formal tags selected from a pre-defined vocabulary, to RDF triples and relationships from an ontology, we consider only tags consisting of phrases (one or several words) in this paper.

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"*.

Structure, within as well as between resources, thus plays an important role for expressing knowledge in the wiki, ranging from simple tags to complex graphs of links or content item containment.

3.2 KWQL

KWQL [5], KiWi's label-keyword query language [16], allows for combined queries over full-text, annotations and content structure, fusing approaches from conventional query languages with information retrieval techniques for search.

KWQL aims to make data contained in a Semantic Wiki accessible to all users—not only those who have experience with query languages. Queries have little syntactic overhead and aim at being only as complex as necessary. The query language is designed to be close to the user experience, allowing queries over the elements of the conceptual model described in the previous section.

Further, KWQL has a flat learning curve and the complexity of queries increases with the complexity of the user's information need. Simple KWQL queries consist of a number of keywords and are no more complicated to write than search requests in web search engines. On the other hand, advanced KWQL queries can impose complex selection criteria and even reformat and aggregate the results into new wiki pages, giving rise to a simple form of reasoning.

Some examples of KWQL queries are given in the following table:

`Java`	Content items containing *"java"* directly or in any of its tags or other meta data
`ci(author:Mary)`	Content items authored by Mary (using `author` meta-data)
`ci(Java OR (tag(XML) AND author:Mary))`	

Content items that either contain *"java"* or have a tag containing *"XML"* and are authored by Mary

```
ci(tag(Java) link(target:ci(Lucene)))
```

Content items with a tag containing *"java"* that contain a link to a content item containing *"lucene"*

4 A Formal Model for Wiki Content: Content Graphs

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 KiWi.

Definition 1 (Content graph). *A **content graph** is a tuple $G = (V_d, V_t, E_l, E_n, \mathcal{T}, w_t)$ where V_d and V_t are sets of vertices and $E_l, E_n \subseteq (V_d \cup V_t) \times (V_d \cup V_t)$. V_d and V_t represent documents (content items) and tags. E_l and E_n describe the directed linking and nesting among documents and tags.*

The textual content of documents and tags is represented by a set $\mathcal{T}$ of terms and a function $w_t : (V_d \cup V_t) \times \mathcal{T} \to \mathbb{R}$ that assigns a weight to each pair of a vertex and a term. We assume that the term weights for each vertex v are a stochastic vector (i.e., $\sum_{\tau \in \mathcal{T}} w_t(v, \tau) = 1$).

We denote the type of an edge e with $type(e) \in \{l, n\}$ and the type of a vertex v with $type(v) \in \{d, t\}$.

The above is an instance of a generic model, that allows for an arbitrary number of vertex and edge sets for flexible typing. Tags can be used to represent any property of a document other than its textual content. Here, we limit ourselves to two vertex and edge types each for sake of clarity. The model allows for different types of links and nestings exist depending on the types of linked and nested nodes. For example, an edge in $E_l \cap (V_d \times V_t)$ represents a link from a document to a tag, whereas an edge $E_l \cap (V_d \times V_d)$ represents a link between documents.

For the sample wiki from Figure 1, the six documents 1 to 6 form V_d, $V_t = \{1.1, 1.2, 2.1, 3.1, 3.2, 4.1\}$, $E_l = \{(6,1), (6,3), (4,3), (1,1.1), (1,1.2), \ldots, (4,4.1)\}$, $E_n = \{(1,2)\}$, $\mathcal{T}$ the set of all terms in the wiki and $w_t = \{(1, \text{"java"}, 0.8), \ldots, (2.1, \text{"search"}, 1), \ldots\}$.

Nesting of tags in documents, $E_n \cap (V_d \times V_t)$, do not occur in our model of a semantic wiki, but may do so in other applications.

5 Computing the PEST Propagation Matrix

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

A document's tag is descriptive of the content of the text 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 data structure 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 or term-weight propagation.

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 where in the graph τ occurs literally. 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 knowledge management system, 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. Let $\mathbf{H}$ be the transposed, normalized adjacency matrix of the resulting graph. 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)

The overall PEST matrix $\mathbf{P}_\tau$ is therefore computed as (where α is the leap factor)

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

Each entry $m_{i,j} \in \mathbf{P}_\tau$, that is, the probability of transitioning from vertex j to vertex i, is thus determined primarily by two factors, the normalized edge weights of any edge from j to i, the term weight of τ in j.

5.1 Weighted Propagation Graph

To be able to control the choices the semi-random reader performs when following edges in the content graph, we first extend the content graph with a number of additional edges and vertices and, second, assign weights to all edges in that graph.

Definition 2 (Weighted propagation graph). *A* ***weighted propagation graph*** *is a content graph extended with a function $w_e : (E_l \cup E_n) \to \mathbb{R}^2$ for assigning weights to edges that fulfills the following conditions:*

- *For each document $v \in V_d$, there is a tag $t_v \in V_t$ with $(v, t_v) \in E_l$.*

- *For each pair of documents $v, w \in V_d$ with $(u, v) \in E_l$ (E_n), if t_v and t_w are tags of v and w respectively, then there is an edge $(t_v, t_w) \in E_l$ (E_n).*

Edge weights are given as pairs of numbers, one for traversing the edge in its direction, one for traversing it against its direction.

The first condition 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.

Proposition 1. *For every content graph, a weighted propagation graph can be constructed by (1) adding an empty tag ("dummy tag") to each document that is not tagged at all and (2) copying any relation between two documents to its tags (if not already present).*

Consider again the sample wiki from Figure 1, the resulting weighted propagation graph is shown in Figure 2. It 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 $e_1 = (v_1, w_1), e_2 = (v_2, w_2) \in E_l \cup E_n$ it holds that, if $\mathit{type}(e_1) = \mathit{type}(e_2)$, $\mathit{type}(v_1) = \mathit{type}(v_2)$, 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.

Let $\mathbf{A}_w$ be the weighted adjacency matrix of a weighted propagation graph G. Then we normalize and transpose $\mathbf{A}_w$ to obtain the transition matrix $\mathbf{H}$ for G as follows:

$$\mathbf{H} = \frac{1}{\max\left(\sum_i w_e((i,j))\right)} \mathbf{A}_w^T$$

Note that we normalize the columns for all vertices with the same maximum sum of outgoing term weights. This preserves differences in weights between nodes with the same number of outgoing edges, but also yields only a sub-stochastic matrix.

5.2 Informed Leap

Given a leap factor $\alpha \in (0, 1]$, a leap from vertex j occurs with a probability

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

A leap may be *random* or *informed*. In a random leap, the probability of jumping to some other vertex is uniformly distributed and calculated as $l^{\text{rnd}}(i, j) = \frac{1}{|V_d \cup V_t|}$ 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 $l_\tau^{\text{inf}}(i,j) = \frac{w_t(i,\tau)}{\sum_k w_t(k,\tau)}$ for a $\tau \in \mathcal{T}$.

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 for 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

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

5.3 Properties of the PEST Matrix

Definition 3 (PEST matrix). *Let $\alpha \in (0,1]$ be a leap factor, $\mathbf{H}$ be the normalized transition matrix of a given content graph (as defined in Section 5.1) and $\mathbf{L}_\tau$ the leap matrix (as defined in Section 5.2) to $\mathbf{H}$ and term τ with random leap factor $\rho \in (0,1]$. Then the* PEST *matrix $\mathbf{P}_\tau$ is the matrix*

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

Theorem 1. *The* PEST *matrix $\mathbf{P}_\tau$ for any content graph and term τ is column-stochastic and strictly positive (all entries > 0).*

Proof. It is easy to see that $\mathbf{P}_\tau$ is strictly positive as both α and ρ are > 0 and thus there is a non-zero random leap probability from each vertex to each other vertex.

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

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

Corollary 1. *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 1).

	1	2	1.1	1.2	2.1	4	3	4.1	3.1	3.2
1	0.1463	0.4091	0.4848	0.4848	0.1054	0.2556	0.1873	0.1873	0.2146	0.2146
2	0.1630	0.0109	0.0088	0.0088	0.3165	0.0130	0.0095	0.0095	0.0109	0.0109
1.1	0.2019	0.0109	0.0088	0.0088	0.1998	0.0130	0.0095	0.0095	0.0109	0.0109
1.2	0.2019	0.0109	0.0088	0.0088	0.1998	0.0130	0.0095	0.0095	0.0109	0.0109
2.1	0.0074	0.2054	0.1644	0.1644	0.0054	0.0130	0.0095	0.0095	0.0109	0.0109
4	0.1116	0.1637	0.1324	0.1324	0.0804	0.1949	0.1817	0.4540	0.1637	0.1637
3	0.0074	0.0109	0.0088	0.0088	0.0054	0.0908	0.0095	0.0095	0.3220	0.3220
4.1	0.0074	0.0109	0.0088	0.0088	0.0054	0.2074	0.0095	0.0095	0.0498	0.0498
3.1	0.0074	0.0109	0.0088	0.0088	0.0054	0.0130	0.2040	0.0873	0.0109	0.0109
3.2	0.0074	0.0109	0.0088	0.0088	0.0054	0.0130	0.2040	0.0873	0.0109	0.0109

Table 1. Excerpt of PEST matrix for *"java"* with $\alpha = 0.3$ and $\rho = 0.25$

The vector space representation of the content graph *after term-weight propagation* is the document-term matrix using the propagation vectors $\mathbf{p}_\tau$ for each term τ as columns.

6 Structure Propagation with PEST Matrix: An Example

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. The implementation is available from `http://www.pms.ifi.lmu.de/pest`.

Here, we present the results for the sample wiki from Figure 1. We use 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 Figure 2.

The resulting matrix for the term *"java"* is shown in Table 1, omitting Documents 5 and 6 and their tags for space reasons.

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 2.

To verify the veracity of our approach, let us consider a number of desirable properties an approach to fuzzy 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

	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 2. Document-term matrix after term-weight computation

guarantee that direct search results (that would have been returned without fuzzy matching) are retained.

Indeed Documents 1, 4, and 5, all containing *"java"* are highest ranked for that term, though the tags of Document 1 come fairly close. This is desired, as Document 1 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 a KWQL query such as `ci(architecture introduction)` should also rank highly documents 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).

Though this evaluation can, by design, only illustrate the effectiveness of the proposed term-weight propagation approach for fuzzy matching, we believe that it is a strong indication that it will prove efficient and effective also for larger and more diverse document collections.

7 Conclusion and Open Questions

PEST is a unique approach to fuzzy matching that combines the principles of structural relevance from approaches such as PageRank with the standard vector space model. Its particular strength is that it runs entirely at index time and results in a modified vector space representation.

However, the present paper is just the first step in exploring the potential and research issues on term-weight propagation as eigenvector computation over structured data.

First, and most obvious, extensive *experimental evaluation* of the approach including a comparison with existing methods is called for. In particular, we are currently estimating the values for α and ρ as well as for the edge weights "by the seat of our pants" rather than empirical observation. A guide to choosing these values might be possible to derive from studying the behavior of PEST on various kinds of data. Edge values, in particular, could also be amenable to various machine learning approaches, using, for example, average semantic relatedness as a criterion, or to semi-automatic approaches through user-feedback.

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 [7, 8] might be applicable and a comparison study is called for. Furthermore, the computation of the PEST matrix is just one of several alternatives for finding a stochastic propagation matrix.

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

1. In PEST, propagation between documents and between tags and documents influence each other: E.g., a document with many tags will propagate only a relatively smaller amount to its children than a document with few children. For extreme cases, a model where each of these kinds of propagations is at least each given a minimal amount might prove superior to the basic version of PEST described here.

2. The model in this paper does not address the representation of tagged links. One simple way to do this would be to represent a tagged link between two documents as a direct link and in addition a tag that is connected via links to both documents. Alternatively, *typed links* could be introduced. They create the possibility of dynamically determining the weight of a connection based on the link type and term being propagated, for example depending on their semantic similarity as determined through their Google distance or distance in an *ontology.*

3. 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.

4. 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.

5. Any fuzzy 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 as much as 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, automatically computing good, minimal explanations is far from a solved issue.

Acknowledgments

The research leading to these results is part of the project *"KiWi—Knowledge in a Wiki"* and has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 211932.

References

1. S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *EDBT*, 2002.
2. S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. FleXPath: flexible structure and full-text querying for XML. In *SIGMOD*, 2004.
3. V. N. Anh and A. Moffat. Compression and an IR approach to XML retrieval. In *INEX Workshop*, pages 99–104, 2002.
4. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW*, 1998.
5. F. Bry and K. A. Weiand. Flavors of KWQL, a keyword query language for a semantic wiki. In *SOFSEM*, 2010.
6. 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*, pages 14–25, 2002.
7. A. Collins and E. Loftus. A spreading-activation theory of semantic processing. *Psychological review*, 82(6):407–428, 1975.
8. F. Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6):453–482, 1997.
9. T. Grabs and H.-J. Schek. Flexible information retrieval on XML documents. In *Intelligent Search on XML Data*, 2003.
10. S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate xml joins. In *SIGMOD*, 2002.
11. V. Kakade and P. Raghavan. Encoding xml in vector spaces. In *ECIR*, 2005.
12. J. Pokorný. Vector-oriented retrieval in XML data collections. In *DATESO*, 2008.
13. T. Schlieder and H. Meuss. Querying and ranking xml documents. *J. Am. Soc. Inf. Sci. Technol.*, 53(6):489–503, 2002.
14. D. Shasha, J. T.-L. Wang, H. Shan, and K. Zhang. Atreegrep: Approximate searching in unordered trees. In *SSDBM*, 2002.
15. J. Tekli, R. Chbeir, and K. Yetongnon. An overview on XML similarity: Background, current trends and future directions. *Computer Science Review*, 3(3):151 – 173, 2009.
16. K. Weiand, T. Furche, and F. Bry. Quo vadis, web queries? In *Int'l. Workshop on Semantic Web Technologies (Web4Web)*, 2008.