PEST: Fast Approximate Keyword Search in Semantic Data using Eigenvector-based Term Propagation

Klara Weiand[a,*], Fabian Kneißl[a], Tim Furche[b,a], François Bry[a]

[a] *Institute for Informatics, Ludwig-Maximilian University of Munich, 80538 Munich, Germany*
[b] *Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK*

Abstract

We present PEST, a novel approach to approximate querying of graph-structured data such as RDF that exploits the data's structure to propagate term weights between related data items. We focus on data where meaningful answers are given through the application semantics, e.g., pages in wikis, persons in social networks, or papers in a research network such as Mendeley. The PEST matrix generalizes the Google Matrix used in PageRank with a term-weight dependent leap 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 a term 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 document retrieval techniques. In extensive experiments including a user study on a real life wiki, we show how PEST improves the quality of the ranking over a range of existing ranking approaches.

Keywords: keyword search, indexing methods, approximate matching, eigenvector, semantic data, wikis

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 [1] 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:

*Corresponding author

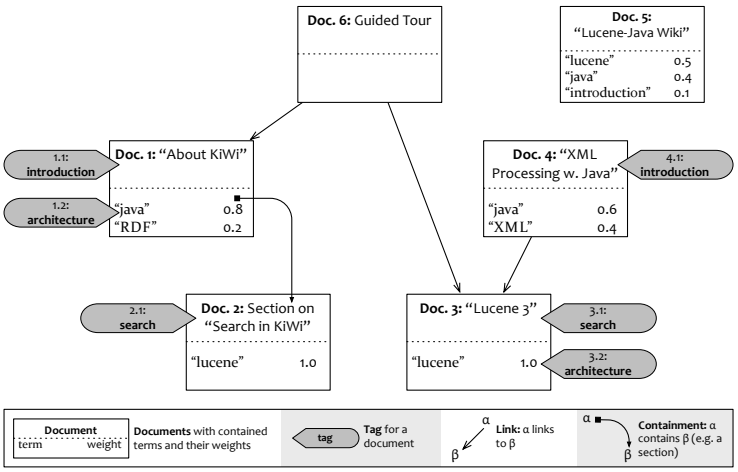

Figure 1: Link and containment graph for a sample wiki

(1) Some projects may not follow the wiki's conventions (or the convention might 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 that are not 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 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 documents, but such queries quickly become rather complex (even in a flexible query language such as KWQL) 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 publication repository. The common characteristic of these applications is that relevant answers (e.g., a wiki page or a person in a social network) are not fully self-contained documents as in the case of standard web search, but obtain a big part 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.

Since data items such as wiki pages tend to be less self-contained than common

web documents, 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. PageRank propagates the anchor text of links to a page as if they are content of that page which is a first step in this direction.

In this article, we generalize the idea of term propagation over structured data: PEST, short for short for term-**p**ropagation using **e**igenvector computation over **st**ructural data, is a novel approach to **approximate 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 the data using eigenvector computation. It generalizes the principles of Google's PageRank [2] to 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 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. Furthermore, the modified term weights represent how well a data item is connected to others in the structured data and therefore one can omit a separate adjustment of the answer ranking as in PageRank.

PEST's 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 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 two steps:

(1) Weighted propagation graph: First, we extend and weight the graph of data items (here, wiki pages and tags): These insertions are used to enable direct propagation between tags. Therefore, we can configure how terms propagate between tags of related pages independently from term propagation between the pages. 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 edge specific weights.

(2) "Informed Leap": The weighted propagation graph 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. To preserve differences in over-

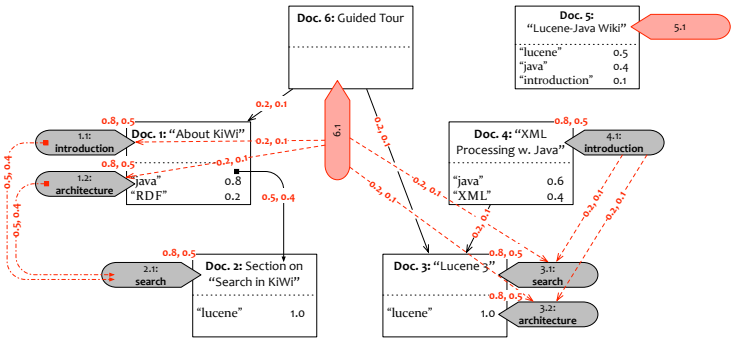

Figure 2: Edge weights and virtual nodes and edges for Figure 1

all edge weight between data items, we choose a normalization that is independent of the actual edge weights and the result is a sub-stochastic rather than stochastic matrix. Second, these remaining probabilities together with a fixed leap parameter α (e.g., 0.3) is used for an "informed leap" to an arbitrary node. The probability to leap to a node A in such an "informed leap" is not equal for all nodes (as in the case of PageRank), but depends on 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.

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. 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 data items (here, wiki pages and their tags). 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 [3].

It is worth emphasizing that only the second step is term-dependent and, as shown in the experimental evaluation in Section 6, the time needed for the calculation of the term-dependent part of the evaluation is in the low seconds on a single core for each term, even without sophisticated optimizations, and can be computed independently for each term. 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).

Contributions

To summarize, PEST improves on approximate keyword search approaches for structured data (summarized in Section 2) in the following aspects:

(1) 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. We introduce the model in Section 4 and discuss how it can represent the core concepts of the KiWi wiki, briefly recalled in Section 3.1. We also introduce KWQL (Section 3.2) to illustrate the benefit of a combination of structure and keyword queries.

(2) 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).

(3) We prove 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.

(4) 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 users preferences, however, with different goal and outcome. 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).

(5) In an **extensive experimental evaluation** of PEST on an entire real-life wiki (Section 6), we compare PEST with three existing keyword search approaches: a simple TF-based ranking, the ranking used by Wikipedia, and the ranking returned by Google. The experimental evaluation demonstrates

- that PEST significantly improves the ranking for each of the compared approaches over a range of keyword queries.
- that users generally prefer the ranking returned by PEST,
- that 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.

Though Section 6 clearly shows that PEST as discussed here can significantly improve existing keyword search in wikis, there are a number of open issues and further challenges to PEST summarized in Section 7.

2. Related Work: Approximate Matching on Structured Data

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

(1) 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 are new matches introduced, but strict matches may also be re-ranked depending on their structural connections.

(2) PEST is designed for *graph-shaped data* rather than purely hierarchical data like the XML-based approaches discussed in the following.

(3) 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 can be computed at *index time*. No additional computations such as query transformations are needed during query evaluation.

Before we consider specific approaches, it is worth recalling that *approximate matching* and *ranking* 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, SQL or SPARQL 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 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 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 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 been researched mainly in the context of XML [4]. The majority of work in this area can be divided into two main classes of approaches:

Tree edit distance: Tree edit distance [5, 6, 7] 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 [8]. 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 [9].

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 [10, 11].

Amer-Yahia et al. [12, 13] 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. [14] 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 a (often quite considerable) number of relaxed queries, or, a considerable 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, as easily treated by PEST, affect tree edit distance and pattern relaxation.

Adapting the vector space model: Another 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 consisting of a weight for each term. Their similarity is then computed using for example the cosine angle between the two vectors. Different schemes can be used for calculating the term weights, the most popular one being tf-idf.

Pokorny et al. [15] represent 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 Carmel et al.'s work [16], 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 et al. [17] 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 data set. 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 compare subtree and query vectors.

Activation propagation is used in Anh et al. [18] for approximate matching over XML 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 and the vector space are used to realize approximate matching over structure. However, here, propagation happens upon query evaluation and is unidirectional. Like the other approaches in this class, it is also limited to tree-shaped data.

PEST **and PageRank:** 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 [2] 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 paper where the only kind of propagation is that from anchor tags to linked pages and where links weights are ignored. The application of this approach is limited to anchor tags and does not apply to general graphs or generalize to different link types. However, there are a number of extensions [19] of the original PageRank that share more of the characteristics of PEST.

PageRank is based on the intuition that a link from one webpage to another can be seen as an endorsement of the linked page. A page then is important if many important pages link to it, even more so if the number of pages linked to by these important pages is low. This idea is implemented by transforming the link graph into a transition matrix which is augmented by a random leap component that ensures that the probability to transition from any state, that is, page, to any other state is nonzero. The contribution of the random leap relative to the transition values is determined by the factor α. As a consequence of introducing the random leap and setting α to a non-zero value, the normalized matrix is stochastic and strictly positive and the principal eigenvector (called PageRank vector) for eigenvalue 1 exists and is unique. In standard PageRank, the random leap operates using a uniform distribution, that is, the likelihood to transition to a state is identical for all states. For a set of N pages, the corresponding leap or teleportation vector can be seen to consist of N entries with value $\frac{1}{N}$. Brin and Page [2] point out the possibility to realize a personalized version of PageRank by using a non-uniform leap vector.

Several schemes for improving the scalability of personalized PageRank have been presented in recent years [20, 21, 22, 23]. They are discussed in this section as well as in section 6.4.

Topic-sensitive PageRank [22] 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 each page 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 generally important pages getting highly ranked despite not containing information relevant to the query.

Query-dependent PageRank [23] is another extension of PageRank which is based on the idea that webpages matching a query that are connected to other matching webpages should be ranked higher. The PageRank algorithm is adapted in such a way that the probability of a transition from one webpage to another is determined by how relevant the target webpage is to the query. Towards this end, both 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 webpage is given as the proportion between that webpage's relevance score and the sum of all matching pages' relevance scores. When a webpage has no non-zero outlinks, the leap vector is used to jump to another webpage. 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.

PEST differs from the approaches described 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.

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

- None of the approaches implement approximate matching over structured data or generally add additional relevant results; they are purely approaches to *ranking* sets of webpages.
- The *assignment of edge weights 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 webpages 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 by PEST 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.

ObjectRank [24] is a system for authority-based ranking for keyword search in databases that, like PEST, uses PageRank to exploit the connections between data items for propagating authority with respect to a keyword across a data graph. Given a database modeled as a labeled graph and a schema graph that assigns bidirectional authority transfer rates to the different types of edges, an Authority Transfer Data Graph is derived. The weight of a node with respect to a given keyword is then established by a modified version of PageRank where a random surfer walks across the graph either traversing edges or jumping to any of the nodes that literally contain the keyword. The probability for following any outgoing edge or jumping to one of the keyword nodes is steered by the damping factor d: The probability to follow an edge is given as the product of d and the authority transfer rate of the edge, while the probability to randomly jump to one of the nodes containing the keyword is $(1 - d)$.

While ObjectRank clearly shares characteristics with PEST, it differs in various significant ways and has various drawbacks:

- ObjectRank uses a binary measure to represent literal keyword containment and randomly decides which of the nodes containing the keyword to jump to. Differences in the frequency of the keyword between documents are ignored and do not factor into the ranking. However, term frequencies are an important factor for ranking, particularly in text-heavy areas of application where it is of high relevance whether a term only occurs once or is frequently used.

- There are no jumps to nodes that do not contain the keyword and those nodes can only be reached through an edge traversal. Therefore, ObjectRank cannot be applied to graphs that are not strongly connected. While this may be of less concern in the area of databases, this constraint severely limits the possibility of applying ObjectRank to other types of graph-shaped data such as web or wiki pages which are frequently not strongly connected.

- Unlike PEST, ObjectRank is not a simple modification of term frequency distributions. As such, it cannot be directly used with standard information retrieval engines and easily used in conjunction with the vector space model.

- Queries are limited to simple keywords combined using disjunction and conjunction and it remains unclear whether ObjectRank could be combined with more powerful query languages.

- The probability to make a leap is steered only by the damping factor d and thus remains constant regardless of whether there is a promising edge that could be followed or not. Moreover, due to the way that authority transfer weights are assigned and normalized, the probability of all possible events may not add up to 1, which is unintuitive in terms of the random surfer model and the idea of spreading authority.

There is a significant body of research [25, 26, 27] on ranking entities in, e.g., entity-relationship graphs. These approaches share with PEST the aim to rank connected items by considering not only local, but also global properties, viz. what other items they are related to. However, they differ from PEST in two main aspects: (1) They require a new query engine with sophisticated, multi-part ranking functions, where PEST computes a modified vector space model that can be used with any existing vector-space IR system. (2) They are tailored to ranking of entities such as dates, prices etc. where PEST is tailored to domains such as semantic wikis or concept search in ontologies where the items of interest are self-contained and clearly identified.

PEST can be applied to any type of structured data, including RDF ontologies. Ranking of RDF query results is discussed, e.g., in [28] and [29]. However, these works differ from PEST by focusing on general RDF data and by using statistical language models with limited propagation.

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 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.

Tags and RDF 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. Two kinds of annotations

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

are available: tags and RDF triples. Tags allow to express knowledge informally, that is, without having to use a pre-defined vocabulary, while RDF triples are used for formal knowledge representation, possibly using an ontology or some other application-dependent pre-defined vocabulary.

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, content item (document) 1 "About KiWi" contains the content item 2 representing a section on "Search in KiWi" and is linked to by 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 tags to complex graphs of links or content item containment.

3.2. KWQL

KWQL [1], KiWi's label-keyword query language [30], 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 Structured Data: 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 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). A content graph is defined based on a **type structure** $\mathcal{T} = (\mathcal{D}, \mathcal{A}, \mathcal{E})$ 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.

For KiWi, $\mathcal{D}$ contains a single type, "content item", $\mathcal{A}$ contains the annotation types "tag", "RDF class", "RDF literal", and "RDF other". The latter three annotation types are used to represent RDF class, literals and all other resources. It is straightforward to add further annotation types, but in our experiments the above annotation types suffice. $\mathcal{E}$ contains two types, link (l) and containment edges (c). This suffices to represent the primary representation mechanisms of KiWi. E.g., a link edge from a content item to a tag represents a tagging, a link edge between two documents a (hypertext) link, a link edge between a content item and an RDF class a type relation etc. We choose to omit the RDF edge labels in this representation as distinguishing between edges with different labels in the propagation did not show a marked improvement and introduces considerable complexity.

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

For the sample wiki from Figure 1, 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 $w_t = \{(1, \textit{"java"}, 0.8), \ldots, (2.1, \textit{"search"}, 1), \ldots\}$.

5. Computing the PEST 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 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 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 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 node j to node i, is thus determined primarily by two factors, the normalized edge weights of any edge from j to i and the term weight of τ in j.

5.1. Weighted Propagation Graph

To be able to control the choices the semi-random reader makes 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 $w_e : E \to \mathbb{R}^2$ for assigning weights to edges that fulfills the following conditions:*

- *Each primary data item carries at least one annotation from each annotation type: For each node $v_d \in V$ with type$(v_d) \in \mathcal{D}$ and each annotation type $t_a \in \mathcal{A}$, there is a node $v_a \in V$ with type$(v_a) \in \mathcal{A}$ and $(v_d, v_a, t) \in E$ for some edge type e.*
- *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 $v_d, w_d \in \mathcal{D}, v_a, w_a \in \mathcal{A}$ with $(v_d, w_d, t), (v_d, v_a, t_a), (w_d, w_a, t_a) \in E$ and type(v_a) = type(w_a), there is an edge $(v_a, w_a, t) \in E$.*

Edge weights are given as pairs 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 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.

Proposition 1. *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.*

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 $e_1 = (v_1, w_1, t), e_2 = (v_2, w_2, t) \in E$ it holds that, if $type(v_1) = type(v_2)$ and $type(w_1) = type(w_2)$, 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 (where $outdegree(v)$ denotes the number of outgoing edges of v):

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

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, for other applications, a different choice of normalization may be advisable.

5.2. Informed Leap

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

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

A leap may be *random* or *informed*. In a random leap, the probability of jumping to some other node is uniformly distributed and calculated as $l^{\text{rnd}}(i, j) = \frac{1}{|V_d \cup V_t|}$ 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 $\tau \in \mathcal{T}$. Thus, when the probability of following any of the outgoing edges of a node decreases, in turn a leap becomes more likely.

In our 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.

Higher values of ρ mean that, after the propagation, the term occurrences are more evenly distributed between the nodes in the graph, while a low value of ρ, means that a higher proportion of leaps are informed and therefore leads to term occurrences being focused around the nodes which already contain the term before propagation.

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

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

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) for $\mathbf{H}$ and term τ with random leap factor $\rho \in (0,1]$. Then the* PEST *matrix $\mathbf{P}_\tau$ is the matrix*

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

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 node to each other node.

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

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

□

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).

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. Validating PEST: Simpsons Wiki

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`.

6.1. Experiment: Setup and Parameters

As an example we choose a real world wiki, the Simpsons Wiki at `http://simpsons.wikia.com`, which is self-contained, focused on a specific topic and at the same time small enough to make it easy to 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).

Wiki pages are first stripped of any markup. The resulting text is normalized, stop word filtered[2] and stemmed[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[4]. 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)[5]: It extends the Luc ranking mechanism by compensating 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 of the wiki page contains a query keyword, this page is boosted by a certain factor, in our case by 3.

- Goo: Where possible, we also compare with the ranking returned by a Google query restricted to the Simpsons wiki.

[2] `http://www.ranks.nl/resources/stopwords.html`

[3] Using the English snowball filter `http://snowball.tartarus.org/algorithms/english/stemmer.html`

[4] `http://lucene.apache.org/java/3_0_1/api/all/org/apache/lucene/search/Similarity.html`

[5] `http://www.mediawiki.org/wiki/Extension:Lucene-search`

	PEST Score	Page title	Wik		Goo	
1	0.1348	Bart Simpson	4	+3	1	0
2	0.0340	Homer Simpson	980	+978	36	+34
3	0.0245	Lisa Simpson	281	+278	181	+178
4	0.0183	Bart the Genius	2	-2	4	0
5	0.0180	Marge Simpson	1321	+1316	548	+543
6	0.0148	Bart Gets an F	19	+13	3	-3
7	0.0115	Bart's Bike	1	-6	-	*new*
8	0.0112	Maggie Simpson	497	+489	678	+670
9	0.0105	Bart the General	11	+2	20	+11
10	0.0089	List of Bart Episodes in The Simpsons	3	-7	216	+206
11	0.0084	Bart Simpson (comic book series)	25	+14	18	+7
12	0.0081	Springfield	1669	+1657	-	*new*
13	0.0077	Chirpy Boy and Bart Junior	5	-8	-	*new*
14	0.0078	Bart vs. Australia	31	+17	12	-2
15	0.0074	The Bart Wants What It Wants	7	-8	30	+15
16	0.0073	Bart's Haircut	6	-10	84	+68
17	0.0070	Milhouse Van Houten	248	+231	67	+50
18	0.0069	Charlie	8	-10	-	*new*
19	0.0069	Bart Gets Famous	12	-7	6	-13
20	0.0069	Bart Junior	9	-11	-	*new*

Table 1: Top-20 ranking for query *"Bart"* (each first column in Wik and Goo gives the respective rank and each second column the PEST change, "-": page is not returned as answer at all)

6.2. 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, viz. *"Bart"*. Second *"moe beer"* is used to illustrate the effect of PEST in the presence of queries consisting of multiple keywords.

Table 1 shows the top 20 answers for the single keyword query *"Bart"* using PEST with MediaWiki-style term weights (Wik ranking). In the last two columns, we compare that ranking with the ranking returned by MediaWiki without PEST and with the ranking returned by Google (Goo ranking, with search restricted to the domain `simpsons.wikia.com`).

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), Bart's hometown, or Bart's best friend as highly relevant for the query "Bart". This is clearly due to the fact that these pages contain the term "Bart" infrequently. 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 either do not occur at all in the Goo ranking or at a rank below 500.

Applying PEST to a basic tf-idf ranking such as Luc yields even greater improvements as shown in Table 2, where we compare the ranking of the top 20 pages using PEST with

PEST rank	Page title	Luc rank & PEST change	
1	Bart Simpson	325	+324
2	Homer Simpson	1667	+1665
3	List of Bart Episodes ...	1	-2
4	Lisa Simpson	1085	+1081
5	Bart's Bike	2	-3
6	Marge Simpson	1773	+1767
7	Bart Junior	3	-4
8	Bart the Mother/Quotes	4	-4
9	Ticket Bouncer	5	-4
10	Chirpy Boy and Bart Junior	6	-4
11	Spree for All	7	-4
12	Charlie	8	-4
13	Congressman 3	9	-4
14	Bart the Genius	84	+70
15	Bart Goes to the Movies	10	-5
16	Bike Track	12	-4
17	Maggie Simpson	1237	+1220
18	Bart Cops Out	11	-7
19	Bart Junior (frog)	17	-2
20	Bart Jumps/Credits	14	-6

Table 2: Top-20 ranking for query *"Bart"*

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 to 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 Table 3 (due to space limitations we give only the top 10 answers). Here, we start with the Wik ranking. For example, *Homer Simpson*, a frequent visitor of *Moe's Tavern* (rank 4) and big consumer of duff beer (rank 3), is ranked 2^{nd} compared to rank 122 without PEST. For completeness, the relevance measures are given in Table 4.

In addition to the top 20 ranking, we also considered the top 100 answers for each of the above rankings for *"Bart"*. The results of this comparison are summarized in Table 4:

The number of relevant pages (manually evaluated) that are introduced by PEST into the top 100 answers is significant. 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. Nearly no irrelevant pages are introduced.

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.

6.3. User Study

To determine whether users consider PEST's search results superior to the ranking used by Wikipedia, a user study involving a forced decision task was carried out. To

	Page title	Wik rank & PEST change	
1	Moe Szyslak	1	0
2	Homer Simpson	122	+120
3	Duff Beer	4	+1
4	Moe's Tavern	2	-2
5	Flaming Moe's	3	-2
6	Fudd Beer	5	-1
7	Duff Beer Advertiser	9	+2
8	Bart Simpson	365	+357
9	Homer vs. the Eighteenth Amendment	15	+6
10	It Was a Very Good Beer	14	+4
11	Marge Simpson	-	*new*
12	Lisa Simpson	-	*new*
13	Billy beer	18	+5
14	The Seven-Beer Snitch	36	+22
15	Barney Gumble	29	+14
16	Eeny Teeny Maya Moe	6	-10
17	Springfield	382	+365
18	Moe Baby Blues	7	-11
19	Homer the Moe	8	-11
20	Duff Beer Krusty Burger Buzz ...	+5	

Table 3: Top-20 ranking for query *"Moe beer"*

	"Bart"	
	Wik rank	**Luc rank**
Relevant	17	17
Irrelevant	4	2

Table 4: (Ir-) relevant pages added by PEST compared to the Wik and Luc ranking

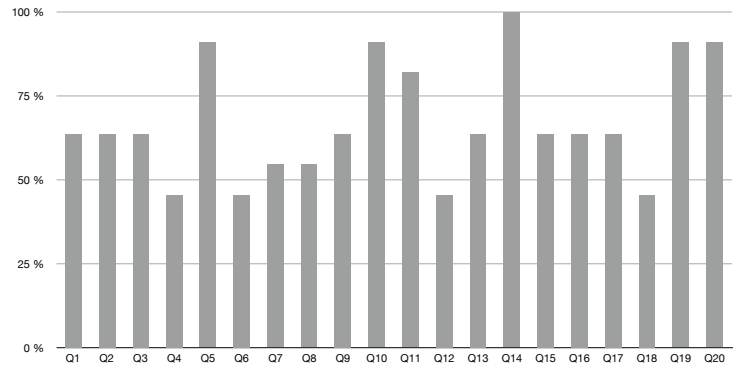

Figure 3: Percentage of participants who preferred the result returned by a PEST-enhanced Wik ranking over the a simple Wik ranking, per query

specifically evaluate the effect of PEST when used together with a state-of-the-art ranking technique, we chose to compare search results obtained using the Wik ranking with PEST to those of a simple Wik ranking without the application of PEST.

Experimental setup. Both types of result rankings, Wik with and without PEST, were generated for twenty single-keyword queries consisting of names of Simpsons characters (for example "Bart", pictured in table 2 and "Milhouse"), locations in the Simpsons universe (such as "tavern" and "brewery") and other concepts relevant to the TV-show (for example "skateboarding"). The titles of the top 20 matching wiki pages for each query were then placed in individual files which did not contain the name of the ranking used and randomizing the ranking identifier. These files were then presented to eleven participants together with the corresponding queries, all of which had at least basic knowledge of the Simpsons. The participants were asked to indicate for each query which search result they preferred. Though the result rankings only gave the titles of the matching wiki pages, participants were encouraged to look up more information about the individual results if they felt they could not make a clear decision based on the titles alone.

Results and Discussion. Across all queries and users, the PEST-enhanced ranking was preferred 67.23 percent of the time. A chart showing for each query how many users preferred the PEST-enhanced results is given in Figure 3. Overall, participants liked the regular Wik ranking better for four of the queries, but only by a slight margin of about 5%. Two other queries prompted an equally divided reaction, their PEST-enhanced results were preferred by about 5%. The PEST-enhanced results of fourteen queries were more clearly preferred with scores between 63 and 100 percent.

While the results computed using PEST were not considered to be more relevant for all of the twenty queries by the majority, participants showed a palpable preference for the PEST-enhanced rankings for fourteen queries. Further, the regular Wik ranking results were not unequivocally judged to be better for any of the queries, while on the other hand six of the results using PEST were preferred by more than 75% of participants with the results for one query even reaching a perfect score of 100 percent.

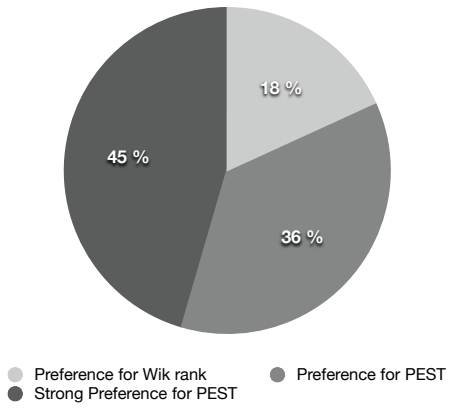

Figure 4: Percentage of users who preferred the PEST-enhanced ranking for less than half of the queries ("Preference for Wik rank"), 50-70% of the queries ("Preference for PEST") and 75% and more of the queries ("Strong Preference for PEST")

Figure 4 shows the results broken down into groups of participants formed based on how often they indicated a preference for the PEST-enhanced query results. 82% of users overall prefer the results generated using PEST, more than half of them do so for 75% or more of the queries.

The finding that the percentage of users who overall strongly prefer the PEST-enhanced ranking is higher than that of the queries that received similarly high scores suggests that PEST is to some extent divisive in that its effects are perceived as very positive by a large amount of users, while a small group of users mostly prefers the unchanged Wik ranking.

Overall, the results of this study indicate that users considers PEST to substantially improve the quality of result rankings.

6.4. Performance Evaluation

The comparative evaluation of the quality of the rankings with and without PEST shows the significant improvements PEST can contribute to keyword search. What about the cost?

To quantify the performance of PEST, we run 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 6.

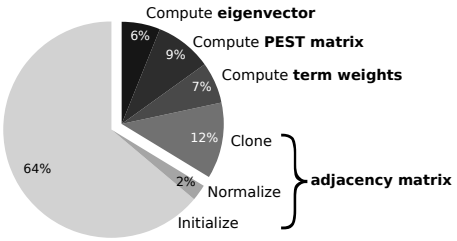

Figure 5: Indexing a single term

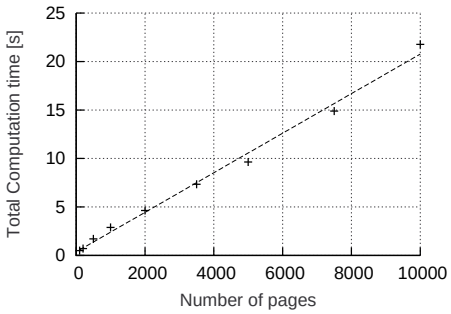

Figure 6: Indexing time over dataset size

Figure 5 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 **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 are indexed at once, this part needs to be executed once 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 7 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 8 further shows 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, 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

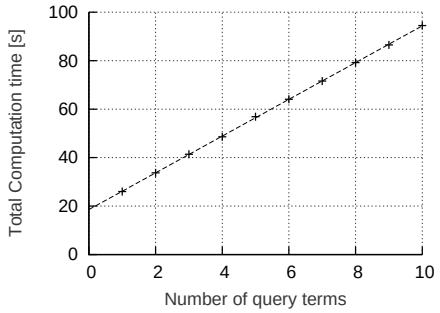

Figure 7: Indexing time over number of indexed terms

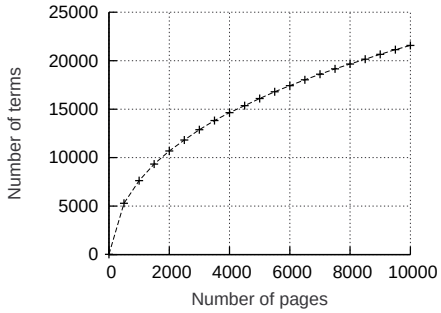

Figure 8: Number of unique terms over dataset size

re-calculated.

The scalability challenge that PEST faces is also less severe than that of personalized PageRank discussed in Section 2 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 documents; 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 for a set of thematically homogeneous set of documents. Figure 8 shows the change in the number of unique terms as the number of documents in the dataset increases.

Many approaches to implementing personalized PageRank in a less computationally expensive way have been suggested [31]. 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, introduced in Section 2, 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 [20] 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 [21] 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 a limit on the number of terms index 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. Conclusion and Open Questions

PEST is a unique approach to approximate 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 index representation.

In this article, we have analyzed PEST's performance on a wiki and shown that it improves search results not only by including new matches, but also by changing the result rankings of strict matches.

There is a wide body of further work to refine and extend PEST.

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, 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 [32, 33] 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. The model for structured data described in this paper assumes that edge weights are uniform for all terms. 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.

2. 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.

3. 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 weight based on the position of the text in the document exists.

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 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 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.

6. In PEST, the term weights propagated along an edge are normalized by the number of edges incident to the same node. Thus, e.g., a document with many tags propagates only a relatively smaller amount to its children than a document with few tags. A model where the propagation along each type can not drop below a given minimum might prove superior to the basic version of PEST described here.

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.

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. The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 246858 (DIADEM, `diadem-project.info`).

[1] F. Bry, 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.

[2] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, in: Int'l World Wide Web Conf., 1998.

[3] K. Weiand, S. Hausmann, T. Furche, F. Bry, KWilt: A semantic patchwork for flexible access to heterogeneous knowledge, in: RR, Vol. 6333 of Lecture Notes in Computer Science, 2010.

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

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

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

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

[8] V. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, in: Soviet Physics Doklady, Vol. 10, 1966.

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

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

[11] R. Yang, P. Kalnis, A. Tung, Similarity evaluation on tree-structured data, in: ACM SIGMOD Int'l Conf. on Management of Data, 2005.

[12] S. Amer-Yahia, S. Cho, D. Srivastava, Tree pattern relaxation, in: Int'l Conf. on Extending Database Technology, 2002.

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

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

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

[16] D. Carmel, Y. Maarek, Y. Mass, N. Efraty, 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.

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

[18] V. N. Anh, A. Moffat, Compression and an IR approach to XML retrieval, in: INEX Workshop, 2002.

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

[20] G. Jeh, J. Widom, Scaling personalized web search, in: Int'l World Wide Web Conf., 2003.

[21] S. Kamvar, T. Haveliwala, C. Manning, G. Golub, Exploiting the block structure of the web for computing PageRank, Tech. rep., Stanford (2003).

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

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

[24] H. Hwang, V. Hristidis, Y. Papakonstantinou, Objectrank: a system for authority-based search on databases, in: SIGMOD Conference, 2006, pp. 796–798.

[25] T. Cheng, X. Yan, K. C.-C. Chang, Entityrank: searching entities directly and holistically, in: VLDB '07: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment, 2007, pp. 387–398.

[26] S. Chakrabarti, Dynamic personalized pagerank in entity-relation graphs, in: WWW '07: Proceedings of the 16th international conference on World Wide Web, ACM, New York, NY, USA, 2007, pp. 571–580.

[27] H. Rode, P. Serdyukov, D. Hiemstra, Combining document- and paragraph-based entity ranking, in: SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, ACM, New York, NY, USA, 2008, pp. 851–852.

[28] S. Elbassuoni, M. Ramanath, R. Schenkel, M. Sydow, G. Weikum, Language-model-based ranking for queries on rdf-graphs, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM 2009), ACM, Hongkong, China, 2009.

[29] S. Elbassuoni, M. Ramanath, R. Schenkel, G. Weikum, Searching RDF Graphs with SPARQL and Keywords, IEEE Data Enginerring Bulletin 33 (1).

[30] K. Weiand, T. Furche, F. Bry, Quo vadis, web queries?, in: Int'l Workshop on Semantic Web Technologies (Web4Web), 2008.

[31] T. Haveliwala, S. Kamvar, G. Jeh, An analytical comparison of approaches to personalizing PageRank, Tech. rep., Stanford (2003).

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

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